

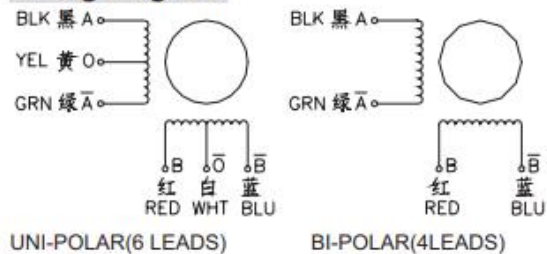
דו"ח מעבדה

מעבדה 2 – בקר מנוע צעד

1. סקירה על מנוע צעד



Wiring Diagram:



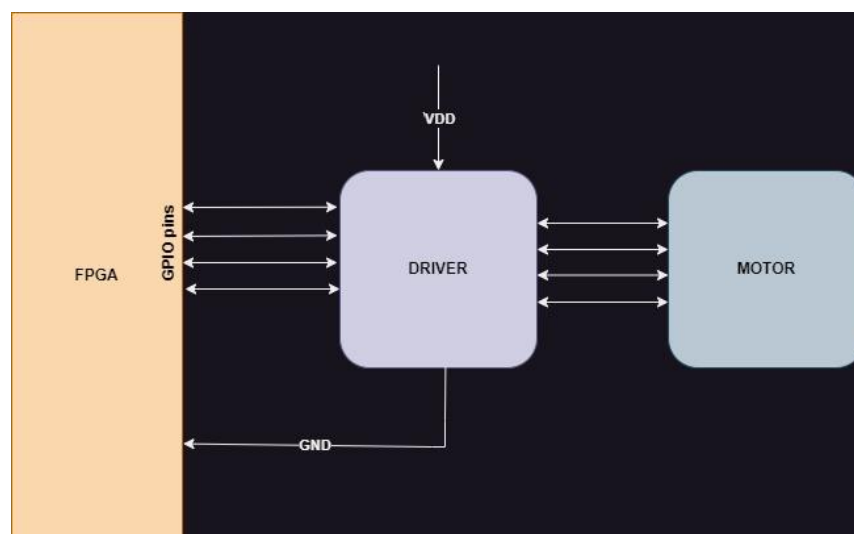
מנוע צעד הוא סוג של מנוע המופעל על ידי מתח DC. ההזזה של המנוע מתקיימת על ידי הפעלת מתח על הסלילים אשר מזיזים את המנוע בצעדים קבועים. הסלילים מקבלים מתח וכתוצאה מכך נוצר שדה מגנטי אשר מזיז את ציר המנוע בצעד. על ידי הפעלת מתח על הסלילים השונים בסדר מסוים ניתן להזיז את המנוע בסיבובים שלמים בהתאם לפרמטרים (מהירות המנוע, כיוון התנועה, גודל הצעד וכדומה).
אנחנו עובדים עם מנוע bipolar, שמורכב משני סלילים הפועלים יחד כדי לסובב את ציר המנוע. קיים גם מנוע unipolar שמורכב מ-4 סלילים.

הפעלת המנוע: המנוע זז ב- 1.8° בצעד מלא וב- 0.9° בחצי צעד. לכן להשלמת סיבוב מלא נדרשים בהתאם 200 ו-400 צעדים. הסלילים מופעלים על ידי הספקת סדרות הבאות:

- צעד מלא עם כיוון השעון: 1000, 0010, 0100, 0001
- צעד מלא נגד כיוון השעון: 0001, 0100, 0010, 1000
- חצי צעד עם כיוון השעון: 1000, 1010, 0010, 0110, 0100, 0101, 0001, 1001
- חצי צעד נגד כיוון השעון: 1001, 0001, 0101, 0100, 0110, 0010, 1010, 1000

בצעד מלא הסלילים מופעלים אחד-אחד, לעומת זאת עבור צעד חלקי מפעילים סליל ראשון, אז מפעילים את השני בלי לכבות את הראשון ולבסוף מפסיקים הספקת מתח על הראשון ומפעילים רק על השני. אופן פעולה של חצי צעד מאפשר תנועה יותר חלקה.

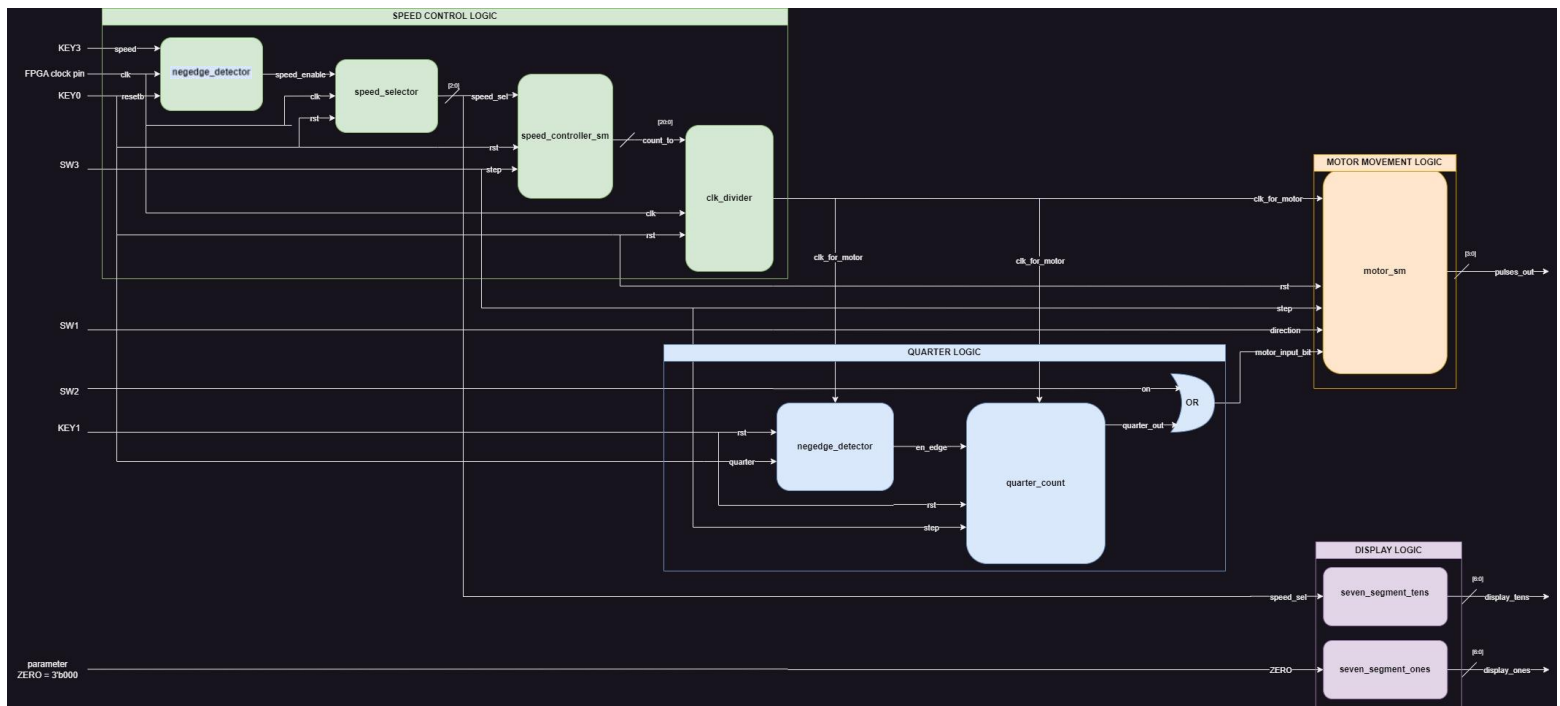
חיבור המנוע לכרטיס: בחיבור ל-FPGA השתמשנו בדרייבר חיצוני על מנת לספק את המתח הדרוש. להלן מוצגת דיאגרמת החיבור:



2. דיאגרמת הבלוקים של המעגל

להלן מוצגת דיאגרמת הבלוקים של המעגל. בה ניתן לראות את החלוקה לאזורים לוגיים:

1. לוגיקה של שינוי מהירויות
2. לוגיקה של רבע סיבוב
3. הצגת מהירות למסך 7-segment
4. מכונת מצבים האחראית על הזזת המנוע



המימוש שבחרנו מאפשר מודולריות גבוהה של המערכת, כלומר ניתן בקלות להוסיף/להוסיף פעולה כלשהי למנוע בלי לפגוע בשאר המערכת.

3. הסבר של המימוש + סימולציות

כעת נעבור על כל המודולים ונראה מהי הפעולה של כל אחד מהם.

- step_motor_top

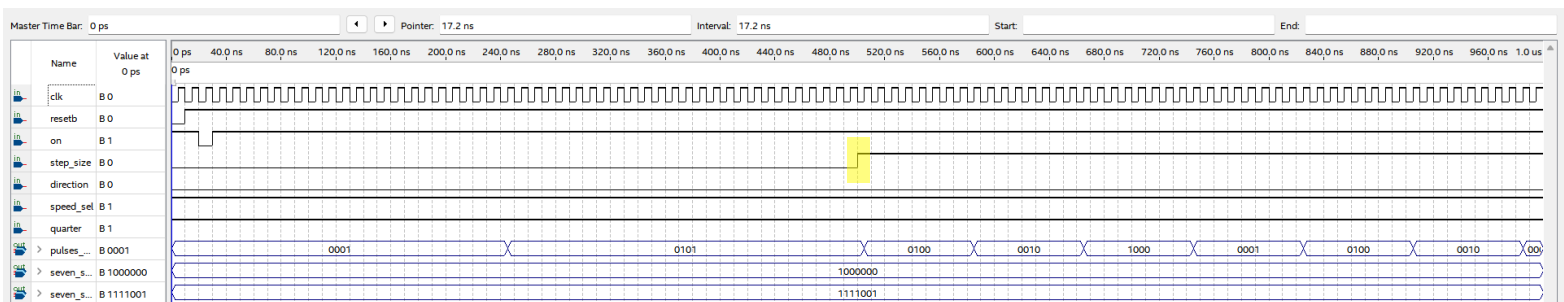
```

1 module step_motor_top(clk, resetb, direction, speed_sel, on, quarter, step_size, seven_seg_o, seven_seg_t, pulses_out);
2
3 // IO declaration
4 input wire clk;
5 input wire resetb;
6 input wire direction;
7 input wire step_size;
8 input wire speed_sel;
9 input wire quarter;
10 input wire on;
11 output wire [6:0] seven_seg_o; // for 7 seg
12 output wire [6:0] seven_seg_t; // for 7 seg
13 output wire [3:0] pulses_out; // signal for the driver
14
15 // inner regs
16 wire clk_for_motor;
17 wire [2:0] speed;
18 wire [20:0] count_to;
19 wire speed_enable;
20 wire en_edge;
21 wire quarter_out;
22 wire motor_input_bit;
23
24 // parameters
25 parameter ZERO = 3'b000;
26
27 //posedge detector for velocity changes
28 negedge_detector negedge_detector_inst1(.clk(clk), .rst(resetb), .sig(speed_sel), .out(speed_enable));
29
30 //speed selector
31 speed_selector speed_selector_inst(.clk(clk), .rst(resetb), .en(speed_enable), .speed_out(speed));
32
33 //speed controller state machine
34 speed_controller_sm speed_controller_sm_inst(.rst(resetb), .step(step_size), .speed_sel(speed), .count_to(count_to));
35
36 //clock divider
37 clk_divider clk_divider_inst(.clk(clk), .rst(resetb), .clk_out(clk_for_motor), .count_to(count_to));
38
39 // negedge detector for quarter
40 negedge_detector negedge_detector_inst2(.clk(clk_for_motor), .rst(resetb), .sig(quarter), .out(en_edge));
41
42 //quarter counter
43 quarter_count quarter_count_inst(.clk(clk_for_motor), .rst(resetb), .step(step_size), .en_edge(en_edge), .quarter_out(quarter_out));
44
45 // OR gate to define whether the motor is in quarter state or in ON state
46 assign motor_input_bit = (on || quarter_out);
47
48 //motor controller
49 motor_sm motor_sm_inst(.clk(clk_for_motor), .nrst(resetb), .input_bit(motor_input_bit), .direction(direction), .step(step_size), .out(pulses_out));
50
51 //seven segment display output for ones
52 seven_segment seven_segment_inst_ones(.number_in(ZERO), .hex_out(seven_seg_o));
53
54 //seven segment display output for tens
55 seven_segment seven_segment_inst_tens(.number_in(speed), .hex_out(seven_seg_t));
56
57 endmodule

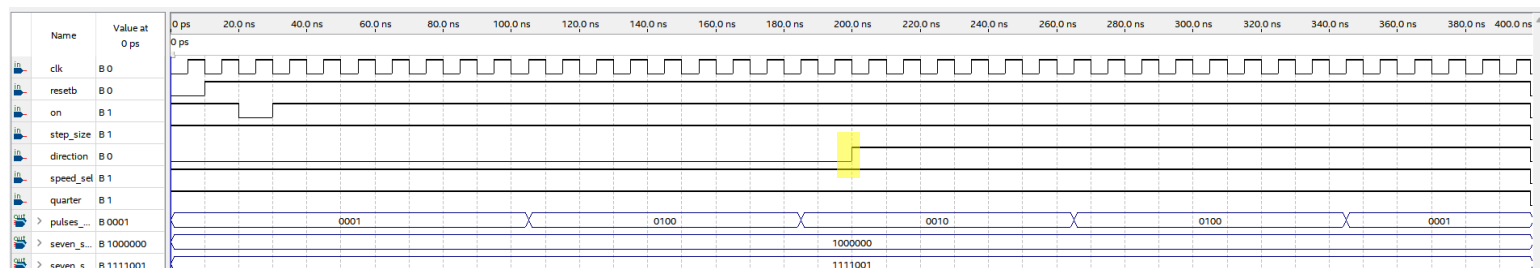
```

זהו מודול ה-top בו מתקיימות הקריאות למודולים אחרים עם הכניסות המתאימות. הלוגיקה היחידה שנמצאת במודול (שורה 48) היא בחירה בין סיגנל מהמתג on ובין quarter_out – ביט המוצא של מודול של רבע סיבוב. על ידי שער OR אנו קובעים איזה סיגנל יהיה ה-enable של מכונת המצבים של המנוע. להלן סימולצית גלים של המודול:

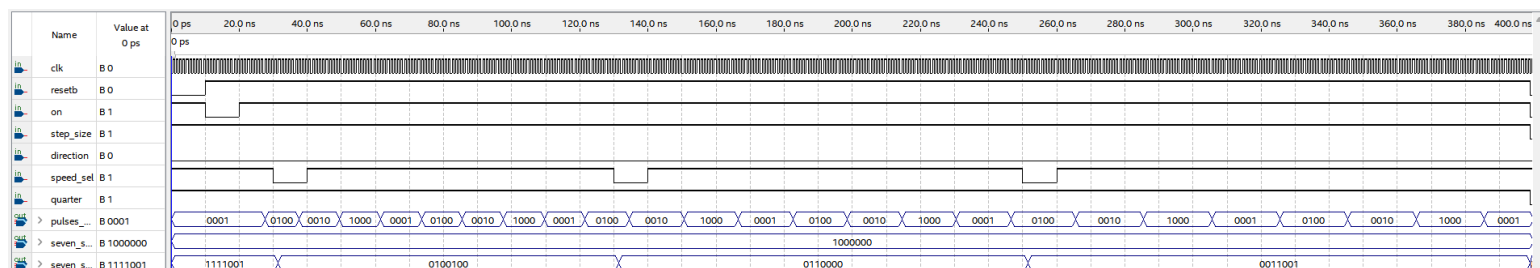
1. שינוי גודל הצעד – התדירות של שינוי המצבים גדלה פי 2 – כפי שציפינו מ-half step



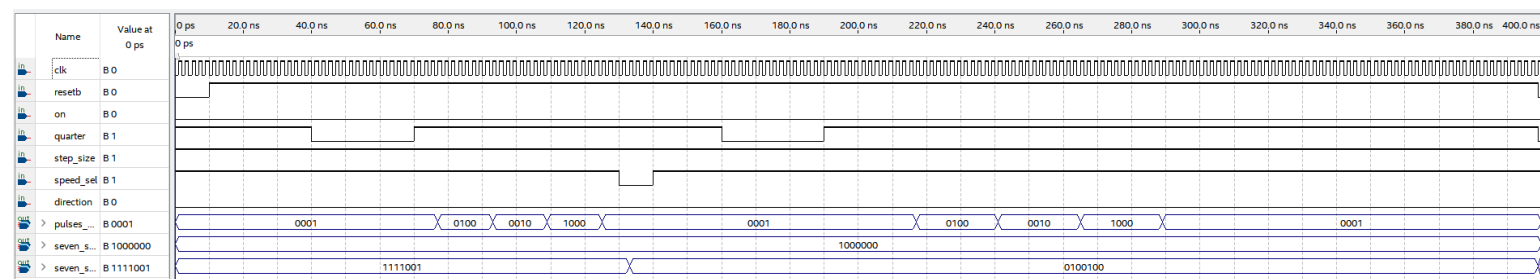
2. שינוי כיוון התנועה – בשינוי כיוון התנועה המנוע נשאר במצב הנוכחי הנדרש ובמצב הבא עובר למצב של הכיוון ההפוך.



3. שינוי המהירות – ניתן לראות שינוי בהחלפת המהירות



4. רבע סיבוב – ניתן לראות שסופרים עד 4 מצבים (הערך לבדיקה – בפועל סופרים ל-100/200) ואז נשארים במצב האחרון



speed control logic בלוק

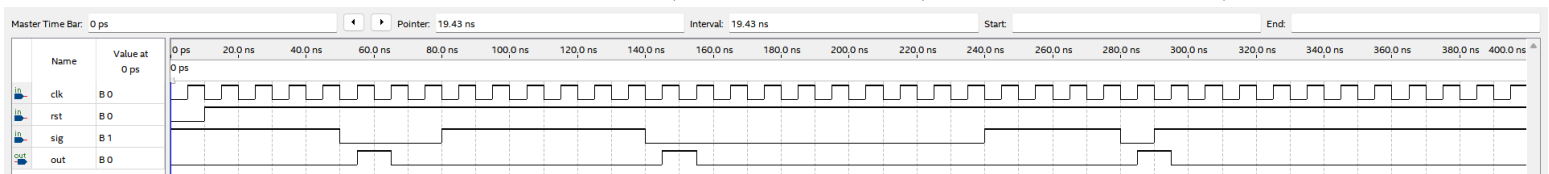
הסבר אופן הפעולה: ברגע של לחיצה על כפתור המהירות (KEY3), `negedge_detector` מוציא ביט באורך מחזור שעות אחד אשר פועל בתור `enable` ל-`speed selector`. הוא מוציא את המהירות המתאימה למצב. המהירות המתאימה נכנסת ל-`speed_controller_sm` שמוציא את כמות מחזורי שעות שצריך לספור בהתאם למהירות. ה-`clock_divider` מקבל את המספר הזה ומוציא את השעות החדש המחולק בהתאם לכמה שצריך לספור. נסתכל כעת על כל אחד מהמודולים ונראה סימולציות.

- `negedge_detector`

```
1 module negedge_detector(  
2     // io declaration  
3     input wire clk,  
4     input wire rst,  
5     input wire sig,  
6     output wire out);  
7  
8     // inner reg  
9     reg [1:0] sig_del;  
10  
11     // logic as in posedge detector  
12     always @(posedge clk or negedge rst)  
13     begin  
14         if (~rst) begin  
15             sig_del <= 2'b11;  
16         end else begin  
17             sig_del <= {sig_del[0], sig};  
18         end  
19     end  
20  
21     assign out = ((~sig_del[0]) & (sig_del[1]));  
22  
23 endmodule  
24
```

המודול אחראי על זיהוי ה-`negedge` של לחיצה על כפתור שינוי המהירות. מכיוון שהלחיצה היא הרבה יותר ארוכה ממחזור השעות של ה-FPGA, רצינו לזהות את תחילת הלחיצה ולאחר מכן לשלוח פולס באורך מחזור שעות אחד אשר ייקבע לדרגה הבאה שהתרחשה הלחיצה.

הסימולציה: ניתן לראות שבירידת סיגנל מתקבל פולס של מחזור שעות אחד



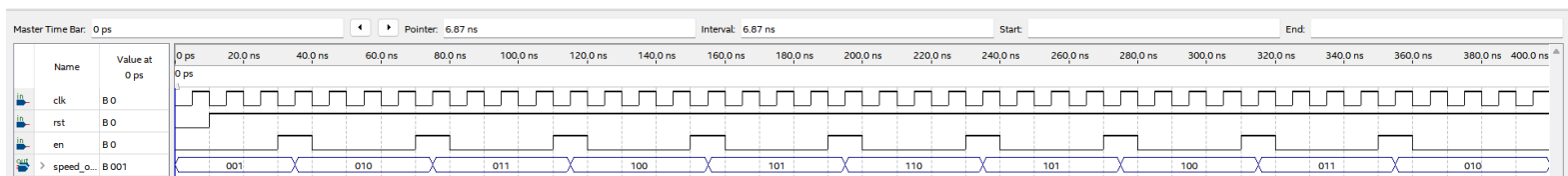
```

1 module speed_selector(
2     // IO declaration
3     input wire clk,
4     input wire rst,
5     input wire en,
6     output wire [2:0] speed_out);
7
8     // inner regs
9     reg [2:0] cs;
10    reg [2:0] ns;
11    reg flag;
12
13    // parameters: states that are passed to motor
14    parameter state10 = 3'b001;
15    parameter state20 = 3'b010;
16    parameter state30 = 3'b011;
17    parameter state40 = 3'b100;
18    parameter state50 = 3'b101;
19    parameter state60 = 3'b110;
20
21    // update current state to the next state logic
22    always @(posedge clk or negedge rst)
23    begin
24        if (~rst)
25        begin
26            cs <= state10;
27            flag <= 1'b0;
28        end
29        else if (en)
30        begin
31            cs <= ns;
32            if ((cs == state10) | (cs == state60))
33                flag <= (~flag); // change the flag when at 10/60 rpm to go to the opposite side
34            end
35        end
36    end
37
38    // state machine
39    always @(*)
40    begin
41        case (cs)
42            state10: ns = state20;
43            state20: ns = (flag) ? state30 : state10;
44            state30: ns = (flag) ? state40 : state20;
45            state40: ns = (flag) ? state50 : state30;
46            state50: ns = (flag) ? state60 : state40;
47            state60: ns = state50;
48            default: ns = state10;
49        endcase
50    end
51
52    assign speed_out = cs;
53
54 endmodule

```

המודול כולל את מכונת המצבים האחראית על החלפת מהירויות. הוא מקבל פולס של מחזור שעון אחד מהמודול הקודם, ובהתאם מחליף את המצב שלו. ישנם 6 מצבים בהתאם למהירויות. כשאר מגיעים למצב של 10/60 rpm, מחליפים את משתנה ה-flag וחוזרים חזרה בהתאם לתנאי בכל מצב (flag=1 – קדימה, אחרת אחורה).

הסימולציה: ניתן לראות שמגיעים ממהירות 10 ל-60 וחוזרים אחורה



```

1 module speed_controller_sm(
2     input wire rst,
3     input wire step,
4     input wire [2:0] speed_sel,
5     output reg [20:0] count_to
6 );
7
8
9
10 // all the values must be *4 but in reality we got the right speed values by dividing by 4
11
12
13 // REAL VALUES
14 //parameter full_10rpm =21'd375000; // 1500000
15 //parameter full_20rpm =21'd187500; // 750000
16 //parameter full_30rpm =21'd125000; // 500000
17 //parameter full_40rpm =21'd93750; // 375000
18 //parameter full_50rpm =21'd75000; // 300000
19 //parameter full_60rpm =21'd62500; // 250000
20 //parameter half_10rpm =21'd187500; // 750000
21 //parameter half_20rpm =21'd93750; // 375000
22 //parameter half_30rpm =21'd62500; // 250000
23 //parameter half_40rpm =21'd46750; // 187500
24 //parameter half_50rpm =21'd37500; // 150000
25 //parameter half_60rpm =21'd31250; // 125000
26
27
28 // VALUES FOR WAVEFORM TEST
29 parameter full_10rpm =21'd1;
30 parameter full_20rpm =21'd2;
31 parameter full_30rpm =21'd3;
32 parameter full_40rpm =21'd4;
33 parameter full_50rpm =21'd5;
34 parameter full_60rpm =21'd6;
35 parameter half_10rpm =21'd7;
36 parameter half_20rpm =21'd8;
37 parameter half_30rpm =21'd9;
38 parameter half_40rpm =21'd10;
39 parameter half_50rpm =21'd11;
40 parameter half_60rpm =21'd12;
41
42 // logic
43 always @(*)
44 begin
45     if (~rst)
46         count_to <= full_10rpm;
47     if ((speed_sel == 3'b001) && (step))
48         count_to <= full_10rpm;
49
50     else if ((speed_sel == 3'b001) && (~step))
51         count_to <= half_10rpm;
52
53     else if ((speed_sel == 3'b010) && (step))
54         count_to <= full_20rpm;
55
56     else if ((speed_sel == 3'b010) && (~step))
57         count_to <= half_20rpm;
58
59     else if ((speed_sel == 3'b011) && (step))
60         count_to <= full_30rpm;
61
62     else if ((speed_sel == 3'b011) && (~step))
63         count_to <= half_30rpm;
64
65     else if ((speed_sel == 3'b100) && (step))
66         count_to <= full_40rpm;
67
68     else if ((speed_sel == 3'b100) && (~step))
69         count_to <= half_40rpm;
70
71     else if ((speed_sel == 3'b101) && (step))
72         count_to <= full_50rpm;
73
74     else if ((speed_sel == 3'b101) && (~step))
75         count_to <= half_50rpm;
76
77     else if ((speed_sel == 3'b110) && (step))
78         count_to <= full_60rpm;
79
80     else if ((speed_sel == 3'b110) && (~step))
81         count_to <= half_60rpm;
82     else
83         count_to <= full_10rpm;
84 end
85
86
87 endmodule
88

```

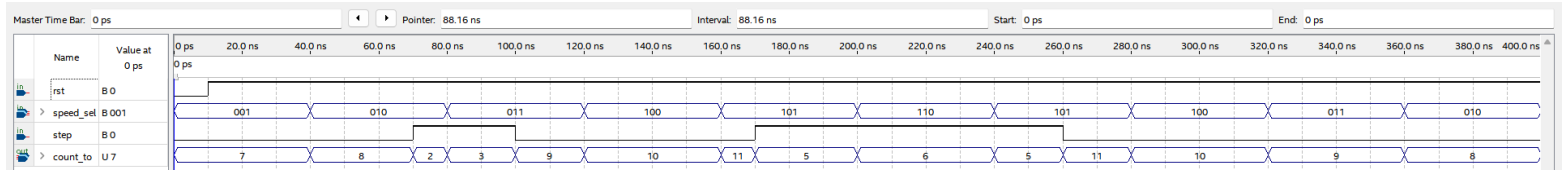
המודול מקבל את ערך המהירות מהמודול הקודם (3 ביט) ובהתאם לגודל הזה וגודל הצעד (input step) מוציא את מספר עד עליו נצטרך לספור. המספר הזה מתקבל אח"כ במחלק השעון (ראה עמוד הבא).

הערה: חישבנו את המספרים האמיתיים לפי פרמטרי המנוע, אך בהצבה של המספרים האלה לא קיבלנו את המהירות הרצויה (מדדנו בפועל את כמות הסיבובים לדקה שהמנוע עושה). לכן חילקנו את כל אחד מהמספרים ב-4, שזה נותן בדיוק את המהירויות הרצויות.

הערה: חישבנו את הערך שעליו צריך לספור לפי הנוסחה הבאה:

$$count = \frac{clock\ of\ FPGA}{\frac{turns}{sec} * (\frac{360}{1.8}\ for\ full\ step\ OR\ \frac{360}{0.9}\ for\ half\ step)}$$

הסימולציה : ניתן לראות ש-count_to משתנה בהתאם לפרמטרי הכניסה כנדרש



הערה : על מנת להבחין בשינויים שינינו את הערכים (בקוד זה מופיע בתור values for test)

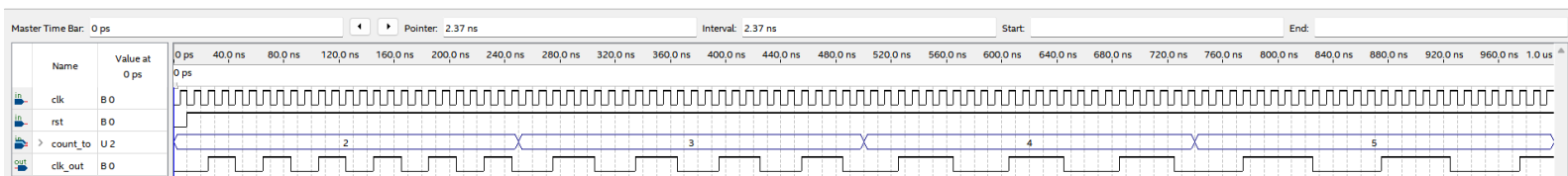
```

1  module clk_divider(clk, rst, count_to, clk_out);
2
3  // IO declaration
4  input wire clk;
5  input wire rst;
6  input wire [20:0] count_to;
7  output reg clk_out;
8
9  // inner reg
10 reg [20:0] count;
11
12 // clock divider logic - create clock with half period of count_to
13 always @(posedge clk or negedge rst)
14 begin
15     if (~rst) begin
16         count <= 21'b0;
17         clk_out = 0;
18     end else if (count >= count_to - 1'b1) begin
19         count <= 21'b0;
20         clk_out = (~clk_out); // change the clock
21     end else begin
22         count <= count + 21'b1;
23     end
24 end
25
26 endmodule

```

זהו המודול של מחלק שעות. הוא מקבל את המספר עד עליו צריך לספור (count_to) ובהתאם למהירות שהתקבלה מייצר את השעות המחולק אשר ישמש במכונת המצבים של המנוע. המודול בנוי כמו counter רגיל שראינו בקורס, רק אם הבדל אחד של החלפת ערך במקום הוצאת אותו חוצה (שורה 20). השינוי הזה גורם לכך שהמוצא של מודול לא יראה כמו רצף פולסים אלא שיהיה שעות עם מחזור שעות ארוך מהזה של FPGA.

הסימולציה: ניתן לראות כי בהתאם לכניסה מחזור השעות המחולק משתנה כנדרש



בלוק quarter logic

הסבר אופן הפעולה: ברגע של לחיצה על כפתור רבע סיבוב (KEY1), negedge_detector מוציא ביט באורך מחזור שעון אחד אשר פועל בתור enable ל-quarter_count. הוא מוציא 1 לוגי בהתאם לכמות מחזורי שעון שצריך לספור (50 לצעד מלא, 100 לחצי צעד) ואז מוציא 0 לוגי.

נסתכל כעת על כל אחד מהמודולים ונראה סימולציות.

- negedge_detector

```
1 module negedge_detector(  
2     // Io declaration  
3     input wire clk,  
4     input wire rst,  
5     input wire sig,  
6     output wire out);  
7  
8     // inner reg  
9     reg [1:0] sig_del;  
10  
11     // logic as in posedge detector  
12     always @(posedge clk or negedge rst)  
13     begin  
14         if (~rst) begin  
15             sig_del <= 2'b11;  
16         end else begin  
17             sig_del <= {sig_del[0], sig};  
18         end  
19     end  
20  
21     assign out = ((~sig_del[0]) & (sig_del[1]));  
22  
23 endmodule  
24
```

הראנו מקודם את אופן הפעולה של המודול ואת הסימולציה

```

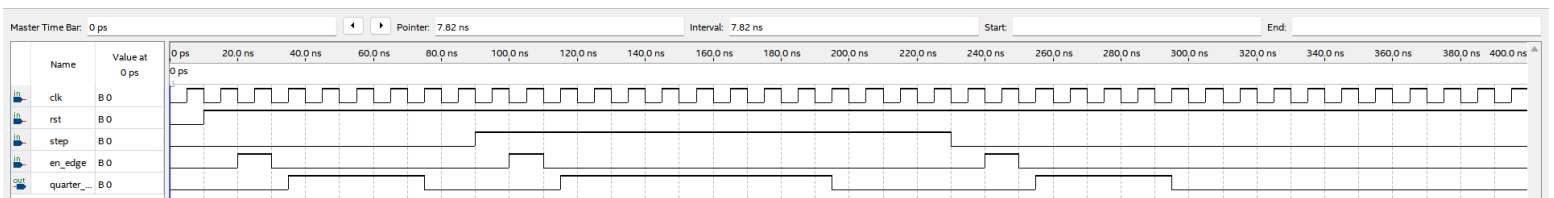
1 module quarter_count(
2     // IO declaration
3     input wire clk, //clk_for_motor
4     input wire rst,
5     input wire step,
6     input wire en_edge,
7     output reg quarter_out);
8
9     // parameters: according to step size
10    parameter HS_COUNT = 8'd200;
11    parameter FS_COUNT = 8'd100;
12
13    // inner regs
14    reg [7:0] count;
15    reg start_count;
16
17    // counter logic
18    always @(posedge clk or negedge rst)
19    begin
20        if (~rst)
21        begin
22            count <= 8'd0;
23            quarter_out <= 1'b0;
24        end
25        else if (en_edge) // if enabled
26        begin
27            start_count <= en_edge;
28        end
29        else if (start_count)
30        begin
31            if (step == 1'b1) // if in full step
32            begin
33                if (count >= FS_COUNT)
34                begin
35                    quarter_out <= 1'b0;
36                    start_count <= 1'b0;
37                    count <= 8'b0;
38                end
39                else begin
40                    count <= count + 8'b1;
41                    quarter_out <= 1'b1;
42                end
43            end
44            else if (step == 1'b0) // the same logic but if in half step
45            begin
46                if (count >= HS_COUNT)
47                begin
48                    quarter_out <= 1'b0;
49                    start_count <= 1'b0;
50                    count <= 8'b0;
51                end
52                else begin
53                    count <= count + 8'b1;
54                    quarter_out <= 1'b1;
55                end
56            end
57        end
58    end
59 end
60
61
62 endmodule

```

המודול שולט על פעולה של רבע סיבוב. המודול מקבל את השעון המחולק בהתאם למהירות הנוכחית, את הפולס מהמודול הקודם שקובע את הלחיצה על הכפתור ואת גודל הצעד. בתוך המודול יש counter עם enable, כלומר אם זיהינו את הלחיצה, המודול מופעל, לפי גודל הצעד נקבע עד כמה לספור וה-counter מאותחל. המודול מוציא החוצה ביט באורך של כמות המחזורים (בהתאם לגודל הצעד) ואז 0.

הערה: בגלל שחילקנו את כל המספרים במודול של speed_controller_sm פי 4, בהתאם נצטרך לספור לא עד 100 (חצי צעד) ו-50 מחזורים (צעד מלא), אלא ל-200 ו-100 בהתאם. בבדיקה המעשית ראינו שעם המספרים האלה המנוע אכן עושה רבע סיבוב כנדרש.

הסימולציה: ניתן לראות שהמוצא הוא כנדרש



הערה: על מנת לבחון בשינויים בסימולציה, הקטנו את הפרמטרים מ-100 ו-200 ל-4 ו-8 בהתאם.

בלוק motor movement logic

הסבר אופן הפעולה: הבלוק כולל את מודול של מכונת מצבים האחראית על שליחת ביטים להזזת המנוע.

```
1 module motor_sm (clk,  
2     nrst,  
3     input_bit,  
4     direction,  
5     step,  
6     out);  
7  
8 // io definition  
9 input wire clk;  
10 input wire nrst;  
11 input wire input_bit;  
12 input wire direction;  
13 input wire step; // full step=1, half step=0  
14 output wire [3:0] out;  
15  
16 // inner variables  
17 reg [3:0] cs;  
18 reg [3:0] ns;  
19  
20 // parameters  
21 parameter A = 4'b1000,  
22     AB = 4'b1010,  
23     B = 4'b0010,  
24     BC = 4'b0110,  
25     C = 4'b0100,  
26     CD = 4'b0101,  
27     D = 4'b0001,  
28     DA = 4'b1001;  
29  
30 // state machine  
31 always @(posedge clk or negedge nrst)  
32 begin  
33     if (~nrst)  
34     begin  
35         cs <= cs;  
36     end  
37     else if (input_bit == 1'b1)  
38     begin  
39         cs <= ns;  
40     end  
41     case (cs)  
42     A:  
43         if (direction == 1'b1 && step == 1'b1)  
44             ns = B;  
45         else if (direction == 1'b0 && step == 1'b1)  
46             ns = D;  
47         else if (direction == 1'b1 && step == 1'b0)  
48             ns = AB;  
49         else if (direction == 1'b0 && step == 1'b0)  
50             ns = DA;  
51     AB:  
52         if (direction == 1'b1 && step == 1'b0)  
53             ns = B;  
54         else if (direction == 1'b0 && step == 1'b0)  
55             ns = A;  
56     B:  
57         if (direction == 1'b1 && step == 1'b1)  
58             ns = C;  
59         else if (direction == 1'b0 && step == 1'b1)  
60             ns = A;  
61         else if (direction == 1'b1 && step == 1'b0)  
62             ns = BC;  
63         else if (direction == 1'b0 && step == 1'b0)  
64             ns = AB;  
65     BC:  
66         if (direction == 1'b1 && step == 1'b0)  
67             ns = C;  
68         else if (direction == 1'b0 && step == 1'b0)  
69             ns = B;  
70     C:  
71         if (direction == 1'b1 && step == 1'b1)  
72             ns = D;  
73         else if (direction == 1'b0 && step == 1'b1)  
74             ns = B;  
75         else if (direction == 1'b1 && step == 1'b0)  
76             ns = CD;  
77         else if (direction == 1'b0 && step == 1'b0)  
78             ns = BC;  
79     endcase  
80     out <= cs;  
81 end
```

motor_sm -

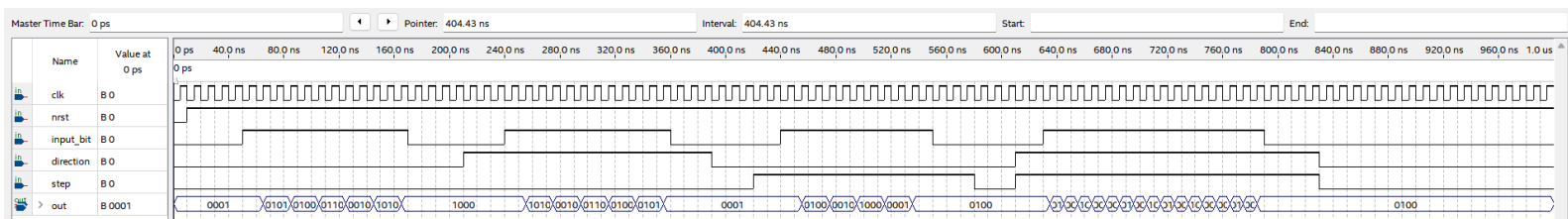
```

82
83
84         if (direction == 1'b1 && step == 1'b0)
85             ns = D;
86         else if (direction == 1'b0 && step == 1'b0)
87             ns = C;
88
89
90     D:
91         if (direction == 1'b1 && step == 1'b1)
92             ns = A;
93         else if (direction == 1'b0 && step == 1'b1)
94             ns = C;
95         else if (direction == 1'b1 && step == 1'b0)
96             ns = DA;
97         else if (direction == 1'b0 && step == 1'b0)
98             ns = CD;
99
100     DA:
101         if (direction == 1'b1 && step == 1'b0)
102             ns = A;
103         else if (direction == 1'b0 && step == 1'b0)
104             ns = D;
105         default:
106             ns = A;
107     endcase
108 end
109
110
111 assign out = ns;
112
113 endmodule

```

זהו המודול אשר קובע את המצב הנוכחי של המנוע ושולח לו את 4 ביט. המודול מכיל את מכוונת המצבים אשר לפי גודל הצעד וכיוון התנועה (עם/נגד כיוון השעון) מחליף בין הצמבים. יש סה"כ 8 מצבים אפשריים – 4 ראשיים לצעד מלא ו-4 מצבי ביניים לחצי צעד. המנוע מקבל את השעון המחולק בהתאם למהירות מה-
clock_divider.

הסימולציה: ניתן לראות שהמצבים משתנים בהתאם לכניסות כנדרש



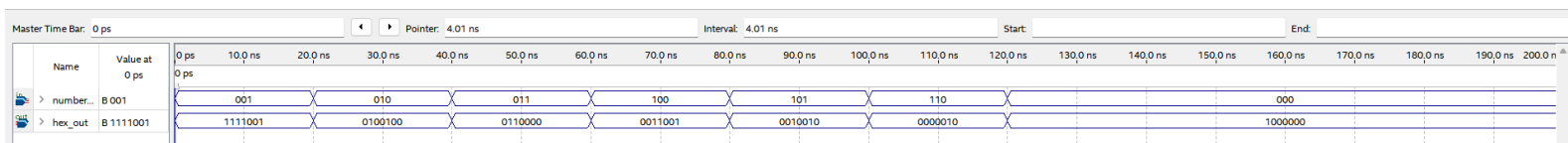
בלוק logic display

- seven_segment

```
1 module seven_segment(number_in, hex_out);
2
3     // in/out declaration
4     input wire [2:0] number_in;
5     output wire [6:0] hex_out;
6
7     // inner reg declaration
8     reg [6:0] hex;
9
10    // logic
11    always @(*)
12    begin
13        case(number_in)
14            3'b000: hex = 7'b1000000; // number = 0
15            3'b001: hex = 7'b1111001; // number = 1
16            3'b010: hex = 7'b0100100; // number = 2
17            3'b011: hex = 7'b0110000; // number = 3
18            3'b100: hex = 7'b0011001; // number = 4
19            3'b101: hex = 7'b0010010; // number = 5
20            3'b110: hex = 7'b0000010; // number = 6
21
22        endcase
23    end
24
25    assign hex_out = hex;
26
27 endmodule
```

זהו המודול אשר מעביר את הערך של המהירות למספר אשר נשלח ל-7-segment display. יש שתי קריאות למודול – אחת לעשרות ואחת לאחדות. הקריאה לאחדות מקבלת פרמטר 0 תמיד (הרי המהירויות שלנו הם כפולות של 10) והקריאה לעשרות מקבלת את הערך מהמודול speed_controller (אשר מוציא את הערכים 1 עד 6 בהתאם למהירות הנוכחית).

הסימולציה:



4. סרטון של המנוע

להלן מצורף קישור לסרטון המוכיח את פעולת המנוע בכל המצבים הנדרשים שלו:

<https://drive.google.com/drive/folders/1NjUFtS9h3SJfQyZAFKnY2IZ3iQ2w-Z4H?usp=sharing>