20/06/2020
דוד קאופמן 308304476
יואב זילברשטיין 305628125

מעבדה ספרתית 83394

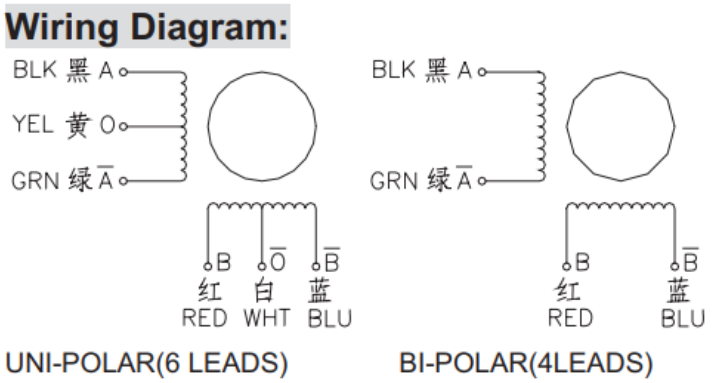
דו"ח 2 - Step Motor Controller

מבוא

מנוע צעד

מנוע המופעל ע"י מתח DC, וניתן לסובב אותו סיבוב שלם בסדרה של צעדים שגודלם קבוע מראש.
המנוע בנוי מסלילים המסודרים בקבוצות שנקראות "פאזות". כאשר מסופק לפאזה מסוימת מתח נוצר שדה מגנטי שמסובב את ציר המנוע צעד אחד. בכדי להשלים סיבוב שלם נצטרך לספק מתח לכל אחת מהפאזות על פי סדר מיקומן. סדר ההדלקה קובע את כיוון הסיבוב.
מנוע צעד מאפשר שליטה מדויקת על מספר הצעדים שהמנוע עושה וכן על מהירות הסיבוב. משום כך, משתמשים בו במכשירים שונים כמו: מדפסות תלת ממד, מכונות CNC, כוננים קשיחים (למיקום ראש הקריאה/כתיבה של הכונן) ועוד. בדוח זה נממש שליטה דיגיטלית במנוע צעד bipolar בעזרת FPGA.

מנוע unipolar ומנוע bipolar



ניתן לראות כי מנוע ה unipolar מורכב מ 4 סלילים ואילו מנוע ה bipolar מורכב רק מ 2 סלילים.
במנוע ה unipolar החיבורים האמצעיים $O, \bar{O}$ בכל סליל מחוברים למקור המתח. בכדי לבצע צעד מקצרים לאדמה את אחד מקצות הסלילים. לדוגמא נחבר את ההדקים $O, \bar{O}$ למתח חיובי, נקצר את $\bar{B}$ לאדמה לאחר מכן את $\bar{A}$ ואז את B ולבסוף את A. באופן זה ייווצר שדה מגנטי שיסובב את ציר המנוע.
לעומתו במנוע ה bipolar הסלילים פועלים יחד כדי לסובב את ציר המנוע. על מנוע ה bipolar נרחיב כעת.

אופן פעולת המנוע

במעבדה השתמשנו במנוע מסוג bipolar.
בכדי לסובב את המנוע מפעילים מתח בסלילים שונים.
<u>בצעד רגיל:</u> בכל פעם נפעיל רק סליל אחד. גודל כל צעד הוא 1.8 מעלות.
תחילה נפעיל מתח רק בסליל הראשון. השדה המגנטי שנוצר מסובב את ציר המנוע עד שיתיישר עם הסליל הראשון. כעת נפסיק את המתח בסליל הראשון ונפעיל מתח רק בסליל השני. השדה המגנטי משתנה וציר המנוע מסתובב שוב בכדי להתיישר עם הסליל השני.
<u>בחצי צעד:</u> מפעילים את שני הסלילים בחלק מהמצבים. גודל כל צעד הוא 0.9 מעלות. מפעילים תחילה מתח רק בסליל הראשון, לאחר מכן מפעילים מתח גם בסליל השני ולבסוף מפסיקים את המתח בסליל הראשון ומשאירים מתח רק בסלילי השני. שימוש בחצי צעד מאפשר לנו לסובב את הציר בצורה חלקה ויותר מדויקת מאשר בצעד רגיל, החיסרון בכך הוא שנדרש יותר מתח במצב פעולה זה.

בכדי לסובב את הציר סיבוב שלם צריך ליצור רצף של הדלקה והפסקת המתח בסלילים כך שבכל פעם המנוע יסתובב בכיוון הרצוי. אם נהפוך את סדר ההדלקה והכיבוי, נוכל לסובב את המנוע בכיוון הנגדי.
בכדי לעצור את המנוע במיקום מדויק קיימות מספר תושבות שעוצרות את הציר ומאפשרות לנו דיוק בתזוזה של המנוע.

בטבלאות הבאות מופיעים הערכים שמסובבים את המנוע בכיוון ובגודל הצעד הרצוי:

צעד מלא בכיוון השעון:

	שחור	ירוק	אדום	כחול	Hex Value
A	1	0	0	0	8
B	0	0	1	0	2
C	0	1	0	0	4
D	0	0	0	1	1

צעד מלא נגד כיוון השעון:

	שחור	ירוק	אדום	כחול	Hex Value
D	0	0	0	1	1
C	0	1	0	0	4
B	0	0	1	0	2
A	1	0	0	0	8

חצי צעד בכיוון השעון:

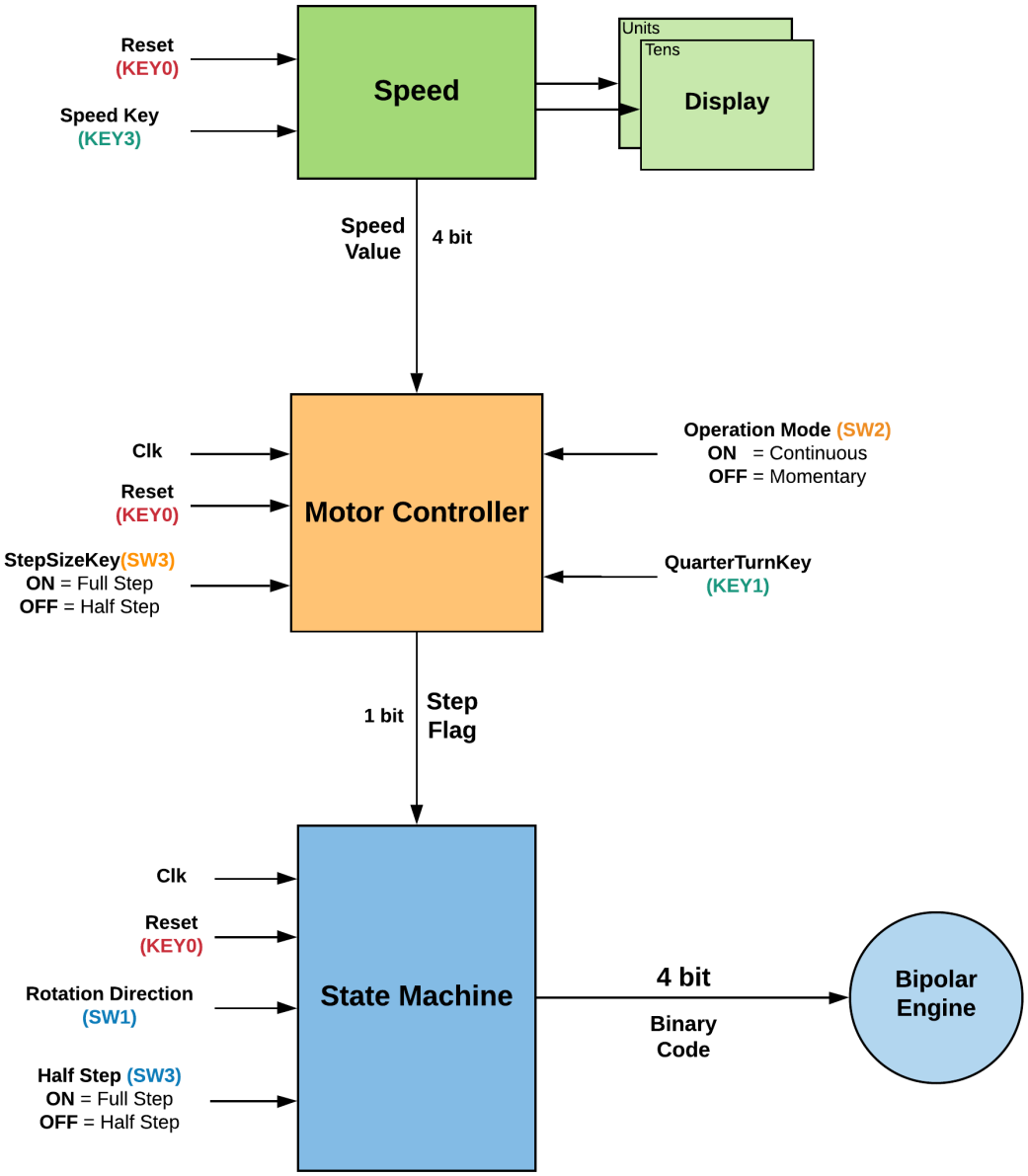
	שחור	ירוק	אדום	כחול	Hex value
A	1	0	0	0	8
AB	1	0	1	0	A
B	0	0	1	0	2
BC	0	1	1	0	6
C	0	1	0	0	4
CD	0	1	0	1	5
D	0	0	0	1	1
DA	1	0	0	1	9

חצי צעד נגד כיוון השעון:

	שחור	ירוק	אדום	כחול	Hex value
DA	1	0	0	1	9
D	0	0	0	1	1
CD	0	1	0	1	5
C	0	1	0	0	4
BC	0	1	1	0	6
B	0	0	1	0	2
AB	1	0	1	0	A
A	1	0	0	0	8

תכנון וסכמת מערכת

בשרטוט הבא מופיעה סכמת בלוקים המתארת את המבנה הכללי של המערכת, והסיגנלים היוצאים והנכנסים לכל בלוק.



בשלב התיכנון בחרנו לחלק את הבקר ל 3 יחידות מרכזיות:

1. speed – יחידה שתפקידה לקבוע את המהירות הסיבובית של המנוע. המהירות נקבעת לפי מספר הלחיצות על כפתור המהירות. בנוסף היחידה תציג את המהירות הנוכחית בתצוגה 7 ספרות.
2. motorController – יחידה שתפקידה לקבוע את קצב ומשך (סיבוב רציף או רבע סיבוב) מעבר המצבים במכונת המצבים של המנוע.
3. MotorStateMachine - מכונת מצבים שבהינתן אות לביצוע צעד, תשלח למנוע את הקוד הבינארי המתאים ביחס לכיוון הסיבוב וגודל הצעד.

שלושת החלקים הללו מאפשרים לנו לפרוט את הבעיה הגדולה לבעיות קטנות, אותן נפרוט בהמשך למודולים קטנים יותר. כמו-כן כל חלק יכול לעמוד בפני עצמו ולא תלוי בפעולה של הרכיבים הקודמים לו.

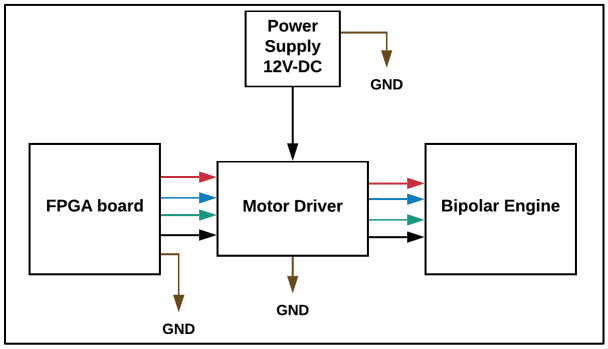
בצורה זו אנחנו יכולים לבודד כל חלק ולבדוק שהוא פועל באופן תקין (כך גם עשינו כאשר הגענו לבדוק במעבדה). בכל שלב ניתן להסתכל על החלקים הקודמים כקופסאות שחורות שמה שמעניין אותנו זה רק הסיגנלים שנכנסים ויוצאים מהן.

<u>כפתורים ומתגים:</u>

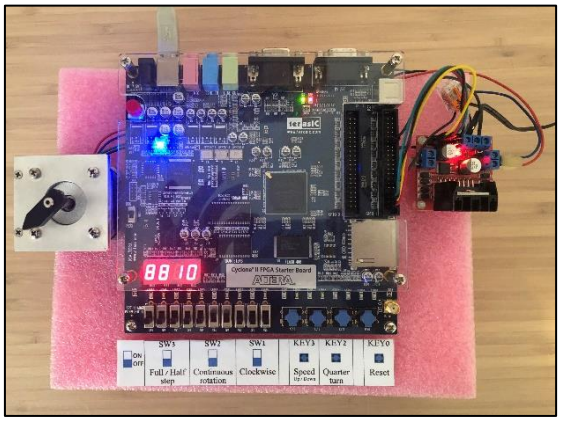
- rst – כפתור איפוס (Key0).
- quarterTrunKey – כפתור רבע סיבוב (Key1).
- speedKey – לחיצה על כפתור זה תשנה את מהירות הסיבוב של המנוע (Key3).
- rotateDirectionKey – קובע את כיוון הסיבוב של המנוע (SW1).
- operationModekey – קובע את אופן הפעולה של המנוע, פעולה רציפה או רבע סיבוב (SW2).
- stepSizeKey – קובע את גודל הצעד שהמנוע יעשה (SW3).

חיבור המנוע לכרטיס

חיברנו את המנוע לכרטיס באופן הבא:



1. את קוד ה Verilog צרבנו ללוח ה FPGA. הקוד משתמש במתגים ובלחצנים שעל הלוח בכדי לשלוט במנוע. אותות הבקרה למנוע (אדום, כחול, ירוק ושחור) מתחברים לדוחף הזרם.
2. Motor Driver – לוח הFPGA לא מסוגל לספק את הזרם הנדרש בכדי לסובב את המנוע ולכן, אנחנו משתמשים בדוחף זרם שמקבל את אותות הבקרה מה FPGA, מגביר אותם ומעביר אותם למנוע. בכדי להגביר את האותות דוחף הזרם מחובר לספק מתח חיצוני.
3. Power Supply – ספק כוח שמזין את דוחף הזרם. כמו כן הספק מהווה הארקה לאדמה עבור הרגליים המתאימות בלוח ה FPGA ודוחף הזרם Motor Driver.
4. Bipolar Motor – המנוע עצמו.



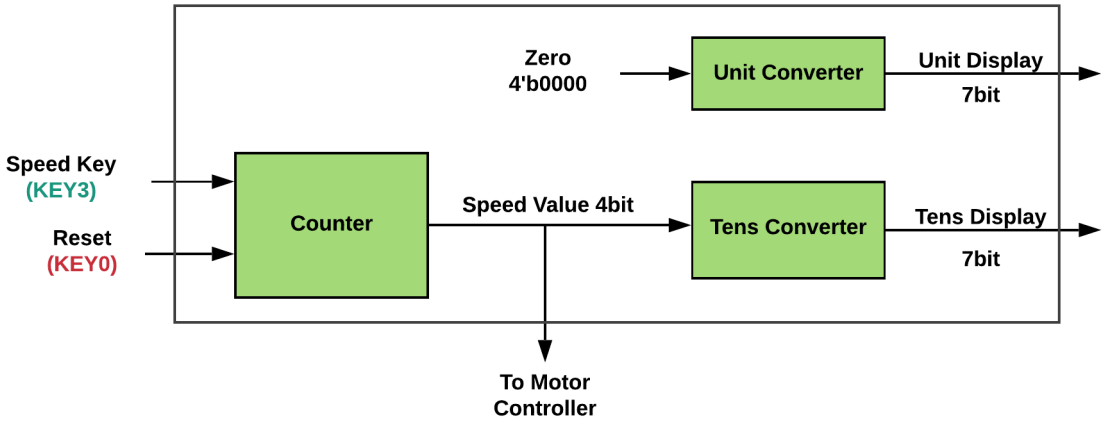
רכיבי המערכת

הערה כללית: בכל הסימולציות Timing תדירות השעון מוגדרת לתדר 50MHz התדר שבחרנו לעבוד איתו מבין שני התדרים הקיימים בכרטיס.

מודול speed

א. קביעת מהירות הסיבוב של המנוע.

ב. הצגת המהירות בתצוגת 7 הספרות.



<u>כניסות</u>

- rst – כפתור איפוס, מאפס את המהירות והתצוגה למהירות של 10 סיבובים בדקה.
- speedKey – בעת לחיצה על הכפתור המהירות משתנה.

<u>יציאות</u>

- unitsDisplay – באס של 7 ביטים המיצגים את ספרת היחידות בקוד בינארי של תצוגת 7 ספרות.
- tensDisplay – באס של 7 ביטים המיצגים את ספרת העשרות בקוד בינארי של תצוגת 7 ספרות.
- speedValue – באס של 4 ביטים המציין את המהירות שבה המנוע צריך לפעול.

<u>קביעת המהירות</u>

המהירות ההתחלתית מוגדרת ל10 סיבובים בדקה, כל לחיצה משנה את המהירות בקפיצות של 10 סיבובים בדקה, בסדר הבא:

10 -> 20 -> 30 -> 40 -> 50 -> 60 -> 50 -> 40 -> 30-> 20 -> 10

וחוזר חלילה.

speedValue הוא הערך של המהירות הנוכחית שבה המנוע צריך להסתובב. כאשר מתקבל אות ריסט ערך זה חוזר לברירת המחדל – 10 סיבובים לדקה.

<u>הצגת המהירות</u>

המהירות מוצגת בתצוגת 7 הספרות באופן הבא:

- ספרת היחידות מציגה תמיד אפס ולכן מועבר לממיר תצוגה 7 ספרות ערך אפס קבוע.
- ספרת העשרות היא הערך speedValue ולכן ממיר התצוגה של העשרות מקבל את ערך זה.

קביעת המהירות מתבצעת בקובץ speed.v :

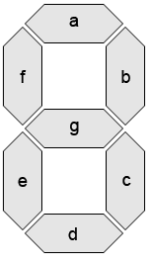
```
module speed(rst, speedKey, unitsDisplay, tensDisplay, speedValue);

    // input declarations
    input wire rst ,speedKey;

    // output declarations
    output wire [6:0] unitsDisplay;
    output wire [6:0] tensDisplay;
    output reg  [3:0] speedValue;

    // internal registers
    reg direction;                  //1 - up ; 0 - down

    // internal parameters
    parameter   up      = 1'b1,
                zero    = 4'b0000,
                speed10 = 4'b0001,
                speed60 = 4'b0110;

    always @ (posedge speedKey or negedge rst)
        begin
            if (~rst) begin                 // restart. reset enable at 0
                speedValue = speed10;       // initial speed = 10
                direction   = ~up;          // initial direction = down
            end

            else begin
                    if(speedValue == speed10 || speedValue == speed60)
                        direction = ~direction;         //change direction

                    if(direction == up)
                        speedValue = speedValue + 1;    // increase speed by 1
                    else
                        speedValue = speedValue - 1;    // decrease speed by 1
                end
        end

    //output logic
    seven_segment unitsConvert(.in(zero),       .out(unitsDisplay)); // output to display
    seven_segment tensConvert(.in(speedValue), .out(tensDisplay));  // output to display

endmodule
```

פעולת הממיר 7 ספרות :

תצוגת 7 הספרות בנויה משבע נורות לד, שנדלקות באפס לוגי. לתצוגה יש 7 כניסות כך שכל כניסה אחראית להדליק לד מסוים.

משום שהקוד הבינארי שמציג ספרה מסוימת איננו שווה לייצוג הבינארי של המספר, קיים צורך במודול המרה מייצוג בינארי של המספר לייצוג תצוגה 7 ספרות.

בטבלה הבאה מוצגים הערכים שיש לשלוח לתצוגה, בכדי להציג את המספר הרצוי



Decimal Digit	Input Lines				Output Lines							Hex Value
	In [0]	In [1]	In [2]	In [3]	g	f	e	d	c	b	a	
0	0	0	0	0	1	0	0	0	0	0	0	40
1	0	0	0	1	1	1	1	1	0	0	1	79
2	0	0	1	0	0	1	0	0	1	0	0	24
3	0	0	1	1	0	1	1	0	0	0	0	30
4	0	1	0	0	0	0	1	1	0	0	1	19
5	0	1	0	1	0	0	1	0	0	1	0	12
6	0	1	1	0	0	0	0	0	0	1	0	2
7	0	1	1	1	1	1	1	1	0	0	0	78
8	1	0	0	0	0	0	0	0	0	0	0	0
9	1	0	0	1	0	0	1	0	0	0	0	10

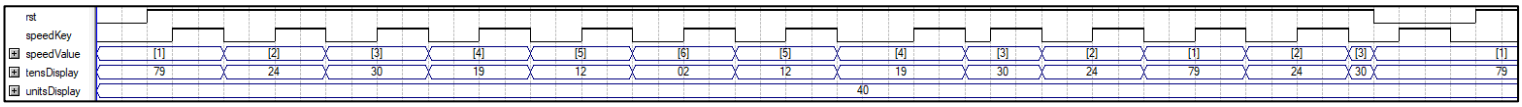
טבלת המרה מבינארי לתצוגת 7 ספרות

ממיר לתצוגת 7 ספרות ממומש בקובץ sevenSegment.v :

```
// Convert 4bit binary number to seven segment representation
module sevenSegment(in, out);

input wire [3:0] in;
output wire [6:0] out;

assign out = (in == 4'b0000) ? 7'h40 :
             (in == 4'b0001) ? 7'h79 :
             (in == 4'b0010) ? 7'h24 :
             (in == 4'b0011) ? 7'h30 :
             (in == 4'b0100) ? 7'h19 :
             (in == 4'b0101) ? 7'h12 :
             (in == 4'b0110) ? 7'h02 :
             (in == 4'b0111) ? 7'h78 :
             (in == 4'b1000) ? 7'h00 :
             (in == 4'b1001) ? 7'h10 :
             //default
             7'h40;
endmodule
```

סימולציות

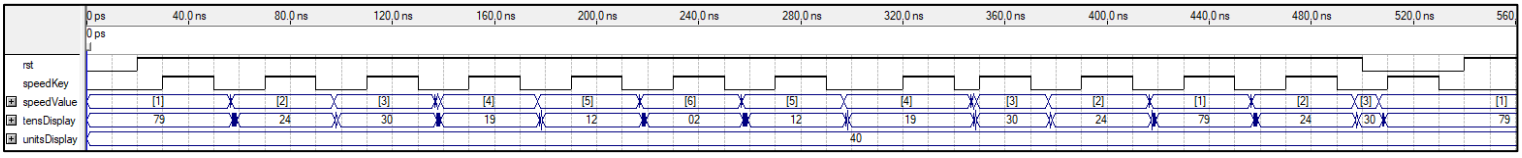
סימולציית Functional



מהסימולציה ניתן לראות שפעולה הלוגית של המודול תקינה, הספירה (speedValue) מתחיל מ1 ועולה בכל לחיצת כפתור (speedKey) עד 6, ומ6 הספירה יורדת בחזרה ל1 וחוזר חלילה.
לחיצה על כפתור rst, מאפסת את המונה ל1.

הערה : כיוון שמהירויות המנוע מוגדרות לתחום מ10 עד 60 בקפיצות של 10, סיפרת היחידות מקובעת, והספירה מבוצעת רק על ספרת העשרות.

סימולציית Timing



מהסימולציה ניתן לראות שלא קיימת בעיות תזמונים והפעולה הלוגית של המודול נשמרת.

בסימולציות ניתן לראות גם את tensDisplay ו unitsDisplay יציאות של שני הממרים 7 ספרות הפחות נוחים לקריאה.

בקר המנוע motorController

קובע את מספר הצעדים הנדרשים ותזמונם ע״י הרמת דגל במוצא, התואם את המהירות הסיבובית המוגדרת בכניסה, ומצבי הכפתורים: גודל הצעד, מצב פעולה (רציף או רבע סיבוב), וכפתור רבע סיבוב.

יחידה זו מורכבת מ3 תתי מודולים ורכיב mux2:1.

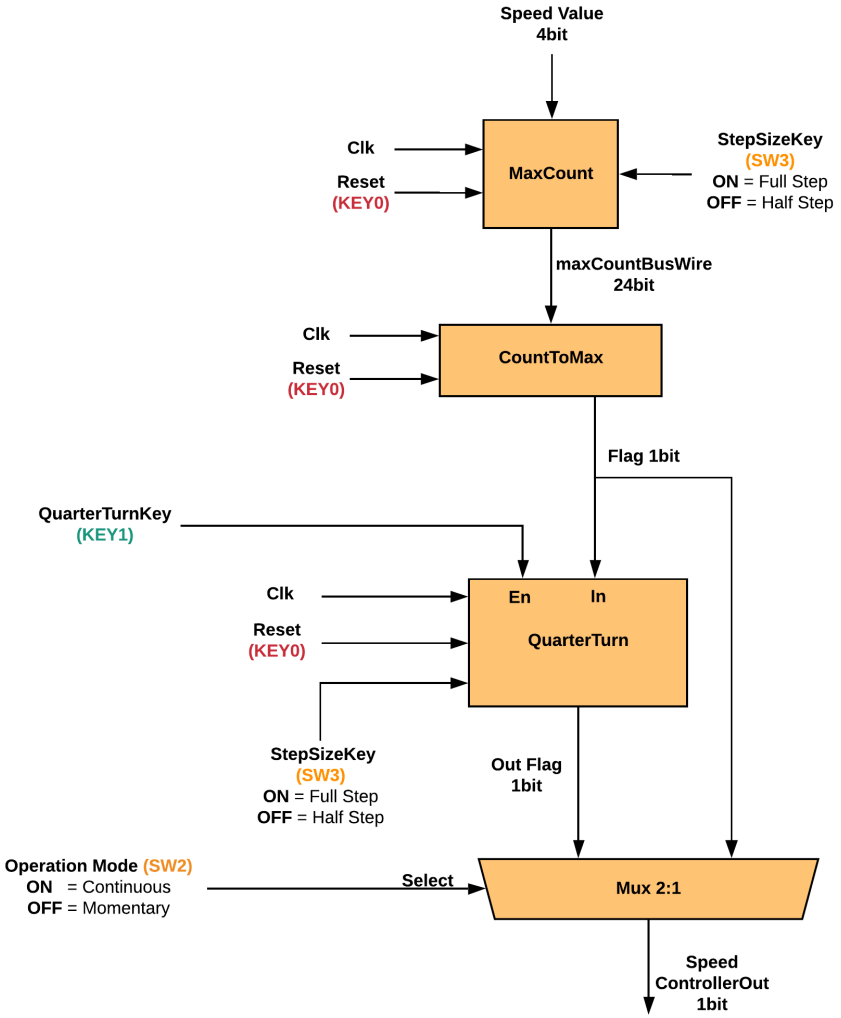
<u>כניסות</u>

- speedValue – המהירות בה המנוע צריך לפעול. מתקבל ממודול בקרת המהירות (speed).
- stepSizeKey – גודל הצעד.
- clk, rst - שעון המערכת 50MHz ואות איפוס (reset).
- operationModekey - קביעת מצב רציף או רבע סיבוב.
- quarterTurnKey - כפתור רבע סיבוב

<u>יציאות</u>

- speedControllerOut - דגל

<u>שרטוט סכמתי של בקר המנוע</u>



קוד קובץ motorController.v

```
module motorController(clk, rst, speedValue, stepSizeKey, quarterTurnKey, operationModekey, speedControllerOut);

    input wire clk, rst, stepSizeKey, quarterTurnKey, operationModekey;
    input wire [3:0] speedValue;


    output wire speedControllerOut;

    wire countMaxOut;
    wire quarterTrunOut;
    wire [23:0]maxCountBusWire;

    MaxCount maxCount(.clk(clk), .rst(rst), .speedValue(speedValue), .stepSizeKey(stepSizeKey),
                      .maxCountOut(maxCountBusWire));

    CountToMax CountToMax(.clk(clk), .rst(rst), .endCountValue(maxCountBusWire), .countMaxOutOut(countMaxOut));

    quarterTurn quarterTurn(.clk(clk), .rst(rst), .quarterTurnKey((quarterTurnKey)), .in(countMaxOut),
                            .stepSizeKey(stepSizeKey), . quarterTurnOut(quarterTrunOut));

    assign speedControllerOut = (operationModekey)? countMaxOut: quarterTrunOut; // MUX2:1

endmodule
```

תתי מודולים של בקר המנוע motorController

מודול MaxCount

מודול המוציא מספר המציין כמה עליות שעון (clk) נדרשות לספור, בשביל שהמנוע יסתובב במהירות סיבובית מסוימת, ובהתאם לגודל הצעד והמהירות הסיבובית הנדרשת בכניסה.

כניסות

- speedValue – המהירות בה המנוע צריך לפעול. מתקבל ממודול בקרת המהירות (speed).
- stepSizeKey – גודל הצעד.
- clk, rst – שעון המערכת 50MHz ואות איפוס.

יציאות

- maxCountOut – מספר 24bit המציין כמה עליות שעון נדרשות לספור לצורך סיבוב המנוע במהירות הנדרשת.

אופן הפעולה

בכל מחזור שעון clk, מוצא maxCountOut נקבע לערך המתאים לפי המהירות הנדרשת התקבלת בכניסה למודול, ולפי המתג stepSizeKey הקובע את גודל הצעד. כיוון שהמהירויות הנדרשות ידועות, את הערך המתאים לכל מהירות חישבנו מראש ושמרנו בפרמטר. חישוב הערכים נעשה ע"פ הנוסחה הבאה:

$$value = \frac{clk * 60[\sec per\ minute]}{speedValue * num\_of\_pulses * step}$$

- clk – שעון המערכת 50MHz.
- $speedValue$ - מספר הסיבובים שהמנוע עושה בדקה. בכדי לקבל מהירות נמוכה יותר עלינו לספור יותר עליות שעון. ולכן אנחנו מחלקים בערך של המהירות.
- num_of_pulses - מספר הפולסים שיש לשלוח למנוע. לדוגמא, בצעד מלא המנוע מסתובב 1.8 מעלות וכדי לבצע סיבוב מלא עלינו לשלוח למנוע $\frac{360}{1.8} = 200$ פולסים.
- $step$ – פקטור שכופל את מספר הפולסים שיש לשלוח למנוע בהתאם לגודל הצעד.
  - צעד מלא: $step = 1$ ולכן מקבלים 200 פולסים.
  - חצי צעד: $step = 2$ ולכן מקבלים $\frac{360}{1.8} \cdot 2 = 200 \cdot 2 = 400$ פולסים.

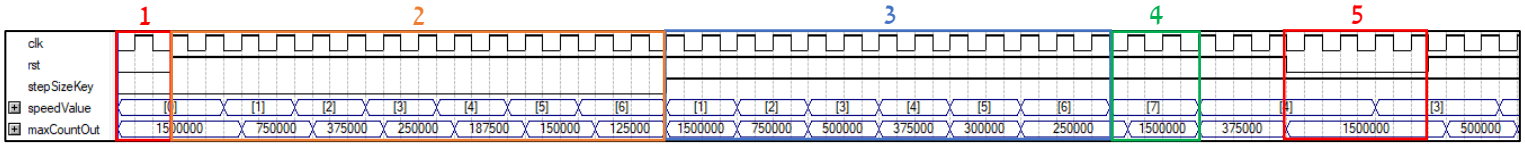
בעת לחיצה על rst המערכת מקבלת את הערך הדיפולטיבי של צעד מלא במהירות 10 סיבובים לדקה.

קוד קובץ MaxCount.v

```
module MaxCount(clk, rst, speedValue, stepSizeKey, maxCountOut);

    // input declarations
    input wire  clk,
                rst,
                stepSizeKey;    // ON(1) = full step ; OFF(0) = half step

    input wire [3:0] speedValue;

    //output declarations
    output reg [23:0] maxCountOut;

    //speed values
    parameter   speed10 = 4'b0001,
                speed20 = 4'b0010,
                speed30 = 4'b0011,
                speed40 = 4'b0100,
                speed50 = 4'b0101,
                speed60 = 4'b0110;

    // 50MHz clk
    parameter   count10_full_step = 24'h16e360,
                count20_full_step = 24'hb71b0,
                count30_full_step = 24'h7a120,
                count40_full_step = 24'h5b8d8,
                count50_full_step = 24'h493e0,
                count60_full_step = 24'h3d090;

    parameter   count10_half_step = 24'hb71b0,
                count20_half_step = 24'h5b8d8,
                count30_half_step = 24'h3d090,
                count40_half_step = 24'h2dc6c,
                count50_half_step = 24'h249f0,
                count60_half_step = 24'h1e848;

    always @ (posedge clk or negedge rst)
    begin
        if (~rst)                               // restart. reset enable at 0
            maxCountOut <= count10_full_step;   // on reset set maxCountOut to speed10
        else
            case(speedValue)
                speed10:  maxCountOut <= stepSizeKey ? count10_full_step : count10_half_step;
                speed20:  maxCountOut <= stepSizeKey ? count20_full_step : count20_half_step;
                speed30:  maxCountOut <= stepSizeKey ? count30_full_step : count30_half_step;
                speed40:  maxCountOut <= stepSizeKey ? count40_full_step : count40_half_step;
                speed50:  maxCountOut <= stepSizeKey ? count50_full_step : count50_half_step;
                speed60:  maxCountOut <= stepSizeKey ? count60_full_step : count60_half_step;
                default:  maxCountOut <= count10_full_step;
            endcase
    end

endmodule
```

סימולציות

<u>סימולציית Functional</u>

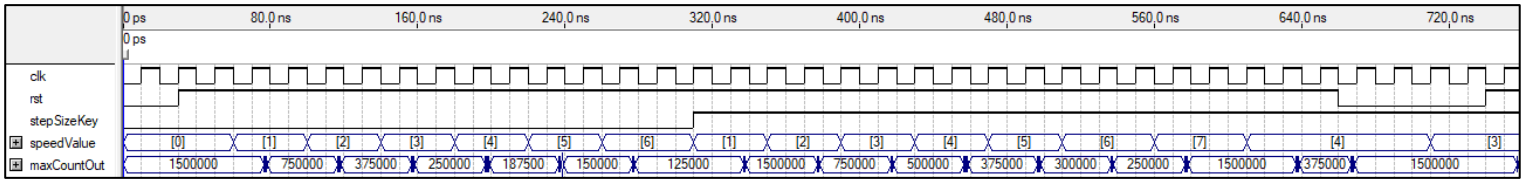


step	speed	Clock cycles
Full	10	1500000
	20	750000
	30	500000
	40	375000
	50	300000
	60	250000
Half	10	750000
	20	375000
	30	250000
	40	187500
	50	150000
	60	125000

- 1 + 5 בדיקת reset, הערך במוצא כנדרש.
- 2 בדיקה עבור חצי צעד, ערכים תקינים.
- 3 בדיקת צעד שלם, ערכים תקינים.
- 4 בדיקת ערך ברירת מחדל עבור מהירות לא מוגדרת, תקין

נשים לב, שהמוצא מתעדכן בעת עליית שעון, ולכן בחלק מהמקרים, הערך במוצא מתעדכן חצי מחזור שעון לאחר שינוי המהירות בכניסה.

- <u>סימוצליית Timing</u>



מהסימולציה ניתן לראות שאין בעיות תזמונים, והלוגיקה נשמרת.

מודול CountToMax

מודול המונה את מספר עליות השעון של המערכת ובסיום הספירה המוגדרת בהכניסה endCountValue, מעלה דגל במוצא.

<u>כניסות</u>

- clk, rst – שעון המערכת 50Mhz ואות ריסט.
- endCountValue – מספר 24bit שאומר לנו כמה עליות שעון עלינו לספור.

<u>יציאות</u>

- countToMaxOut – דגל שאומר שסיימנו לספור Max_Count עליות שעון.

<u>אופן הפעולה</u>

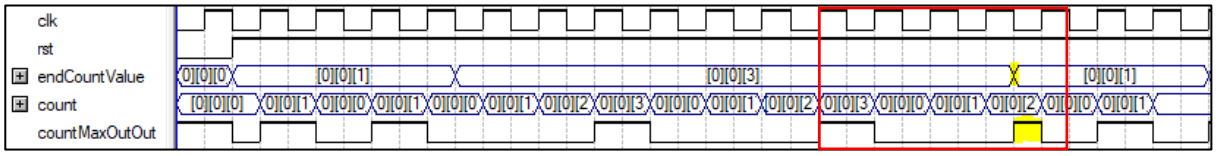
בכל עליית שעון המערכת clk נבדק האם הרגיסטר הפנימי count שווה לערך המקסימלי endCountValue. אם כן, count מתאפס ודגל המציין סיום ספירה עולה במוצא ומתחילה ספירה חדשה. אם לא, count גדל ב1. אם מתקבל אות ריסט count מתאפס.

<u>קוד קובץ CountToMax.v</u>

```
module CountToMax(clk, rst, endCountValue, countMaxOutOut);

    // inputs declarations
    input wire clk, rst;
    input wire [23:0] endCountValue;

    // output declarations
    output wire countMaxOutOut;

    // internal registers
    reg [23:0] count;

    always @ (posedge clk or negedge rst) begin

        if (~rst)                           // restart. reset enable at 0
            count <= 24'b0;
        else if (count >= endCountValue)    // counting is done
            count <= 24'b0;
        else
            count <= count + 24'b1;         // increase count by 1
    end

    assign countMaxOutOut = (count >= endCountValue)? 1'b1: 1'b0;

endmodule
```

סימולציות

<u>סימולציית Functional</u>



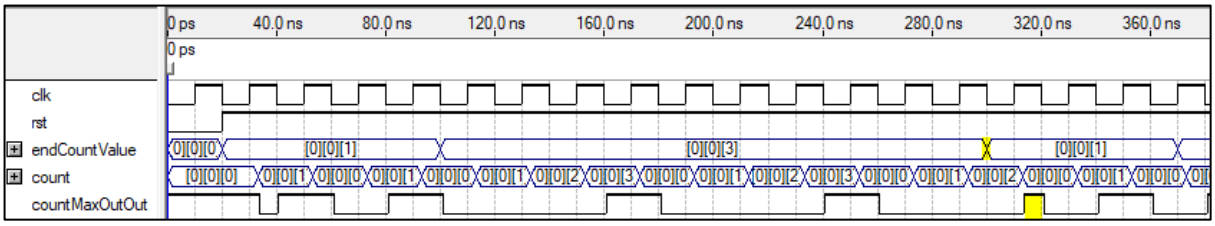
בסימולציה הגדרנו מקטעים שונים של ערכי סיום ספירה.

ניתן לראות שהספירה במקטעים מבוצעת כנדרש.

קיימות שתי נקודות בהן המוצא מתעדכן בצורה שונה.

1. לאחר שחרור כפתור הrst, ניתן לראות שהמוצא מתעדכן לאחר חצי מחזור ולא מחזור שלם. דנו בבעיה זו בדו"ח הקודם, והגענו למסכנה שבעיה זו היא זניחה.
2. במעבר בין ספירה עד 3, לספירה עד 1. כיוון שערך המונה count גדול מ1, המוצא מתעדכן ל "1" לוגי, לרגע קצר (חצי מחזור שעון) כשהשעון נמצא למטה. במצב זה ככול הנראה המוצא לא ידגם ע"י המודל הבא בשרשרת (quarterTrun או motorStateMachin) שדוגמים את הכניסה בעליית שעון. ולכן עשוי להיווצר מצב שתהיה השהיה בסיבוב המנוע. גם בעיה זו זניחה יחסית כי המהירות בה הפולסים נשלחים היא גבוה יחסית ולכן יהיה קשה להבחין בבעיה זו בעת סיבוב המנוע.

<u>סימולציית Timing</u>



בסימולציה ניתן לראות שהספירה מבוצעת כנדרש במקטעים הרצופים.

ניתן להבחין בנקודה בעייתית במעבר מספירה עד 3 לספירה עד 1, המוצא מתעדכן ל"1" לוגי לזמן קשר, ולא מספיק זמן לפני עליית השעון, ככה שבמצבים מסוג זה מוצא המודול עשוי שלא להידגם ותהיה השהיה (זניחה) בסיבוב המנוע.

מודול QuarterTurn

מודול האחראי על סיבוב המנוע בקפיצות של רבע סיבוב.

<u>כניסות</u>

- clk, rst – שעון המערכת 50Mhz ואות איפוס.
- quarterTurnKey – כפתור רבע סיבוב
- stepSizeKey – מציין את גודל הצעד.
- In – כניסת עבור דגל.

<u>יציאות</u>

- QuarterTrunOut – מוצא – דגל (הכניסה).

<u>אופן הפעלה</u>

בעת לחיצה על כפתור quarterTurnKey, כניסת המודול In מעוברת למוצא quarterTrunOut, למשך מספר מוגדר של עליות In, הנקבעות בהתאם לגודל הצעד (חצי או מלא) stepSizeKey.

עבור צעד שלם נדרשות $\frac{200}{4} = 50$ עליות דגל בכניסה לצורך השלמת רבע סיבוב.

עבור חצי צעד נדרשות $\frac{400}{4} = 100$ עליות דגל הכניסה לצורך השלמת רבע סיבוב.

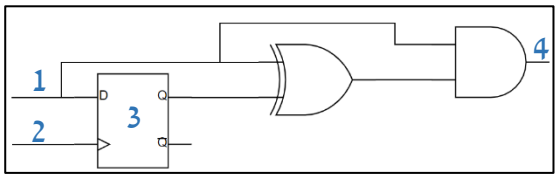
כל עוד המודול נמצא בהמתנה ללחיצה על כפתור רבע סיבוב, המוצא מוחזק ב"0" לוגי.

<u>הערות כלליות על אופן המימוש</u>

1. הגדרנו דגל המציין שהמונה צעדים (דגל כניסה) נמצא בספירה. רוחב הדגל 2 ביטים בשביל לקודד 3 מצבים אפשריים: 00 לא בספירה, 10 ספירה של צעד מלא, 01 ספירה של חצי צעד. בעזרת הדגל אנחנו מונעים התחלת ספירה חדשה במקרה של לחיצות על כפתור רבע סיבוב תוך כדי שהמנוע כבר מבצע רבע סיבוב.

   הבחירה להשתמש בדגל ברוחב של 2 ביטים ולא בביט 1, הייתה במטרה לחסוך 5 רגיסטרים. במימוש חלופי היה ניתן להשתמש בדגל ברוחב ביט 1, יחד עם 7 רגיסטרים שבהם נשמר הערך סיום ספירה (צעד שלם/ חצי צעד). יכול להיות שחיסכון ברגיסטרים אינו היה שיקול הכרחי בפרויקט זה, והיה עדיף מימוש אינטואיטיבי יותר.

2. בשורות 27-19 בקוד, מימשנו מנגנון שמטרתו למנוע פירוש של לחיצה רציפה כמספר לחיצות. המימוש מבוסס על הסכמת בלוקים משמאל.



במימוש הראשון ממשנו את המודול ללא מנגנון זה ובכל סיום רבע סיבוב, במידה והכפתור רבע סיבוב היה לחוץ, המנוע התחיל רבע סיבוב חדש. כיוון שהכפתור נדגם בעליות שעון, וברגע שהדגל enableCount==2'b0 (לא בספירה), הכפתור הלחוץ היה נדגם בעלית שעון והמנוע היה מתחיל רבע סיבוב.

<u>טבלת אמת</u>

QuarterTurn (1)	Clock cycle (2)	LongToShortPressReg (3)	longToShortPressWire (4)
לחיצה רציפה	0	0	1
	1	1	0
	…	1	0
שחרור לחיצה	n	1	0

קוד קובץ quarterTrun.v

```
module quarterTurn(clk, rst, in, quarterTurnKey, stepSizeKey, quarterTurnOut);

    // input and output declaration
    input wire clk, rst, in, quarterTurnKey, stepSizeKey;
    output wire  quarterTurnOut;

    reg [6:0] count;
    reg [1:0] enableCount;
    reg longToShortPressReg;

    wire longToShortPressWire;

    parameter MaxFullStep = 7'h32,
              MaxHalfStep = 7'h64,

              enableCountFullStep = 2'b10, // Enable full step
              enableCountHalfStep = 2'b01; // Enable half step

    always @ (posedge clk or negedge rst) begin
        if(~rst)
            longToShortPressReg <= 1'b0;

        else
            longToShortPressReg <= ~quarterTurnKey;
    end

    assign longToShortPressWire = (~quarterTurnKey & ((~quarterTurnKey)^longToShortPressReg));

    always @ (posedge clk or negedge rst) begin
        if(~rst) begin
            enableCount <= 2'b0;
            count <= 7'b0;
        end

        else if (in && (enableCount != 2'b0)) begin
            count <= count + 7'b1;                                          // increase counter by 1

        end
        else if ((count == MaxFullStep && enableCount == enableCountFullStep) ||
         (count == MaxHalfStep && enableCount == enableCountHalfStep)) begin    // if end count
            enableCount <= 2'b0;

        end

        else if (longToShortPressWire && enableCount == 2'b0) begin         // start to count
            count <= 7'b0;
            if(stepSizeKey)                                                 // full or half step
                enableCount = enableCountFullStep;
            else
                enableCount = enableCountHalfStep;
        end

    end

    assign  quarterTurnOut = ((enableCount == enableCountFullStep) || (enableCount == enableCountHalfStep)) ? in: 1'b0;

endmodule
```

סימולציות

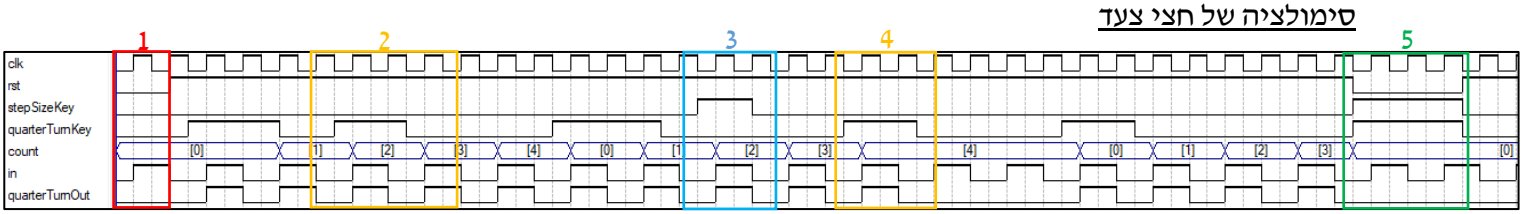
הערה: זמני הלחיצה על הכפתורים בסימולציה קצרים משמעותית ביחס ללחיצה על כפתורים בכרטיס.

סימולציית Functional

כדי לבצע סימולציה הגדרנו בקוד ערכי סיום ספירה קטנים יותר:

עבור צעד מלא סיום הספירה מוגדר ל2, וסיום הספירה של חצי צעד מוגדר ל4.

סימולציה של חצי צעד



1 - כפתור reset לחוץ, המוצא מוחזק ב"0" לוגי.

2+4 - התעלמות מלחיצה על כפתור רבע סיבוב בזמן ספירה.

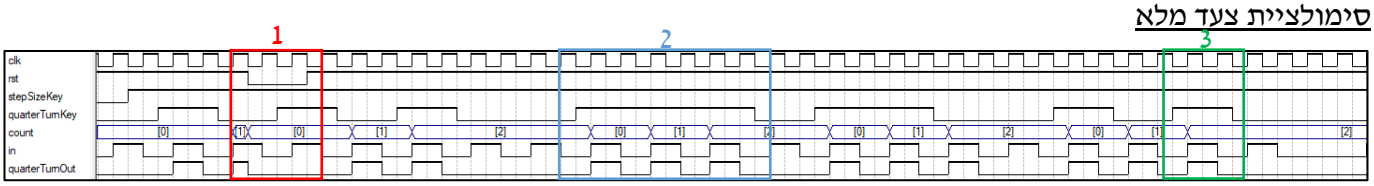
3 - התעלמות מלחיצה על שינוי גודל הצעד תוך כדי ספירה.

5 - כפתורים reset, גודל צעד, רבע סיבוב לחוצים, reset לוקח (כנדרש).

מהסימולציה ניתן לראות שהבדיקות תקינות למעט הספירה.

הסבר אפשרי לך הוא שבסימולציה functional המוצא מעביר את הכניסה באופן מיידי ולא במחזור שעון הבא. נוודא זאת בסימולציית timing הספירה תקינה.

סימולציית צעד מלא



1 - כפתור reset לחוץ, המוצא מוחזק ב"0" לוגי.

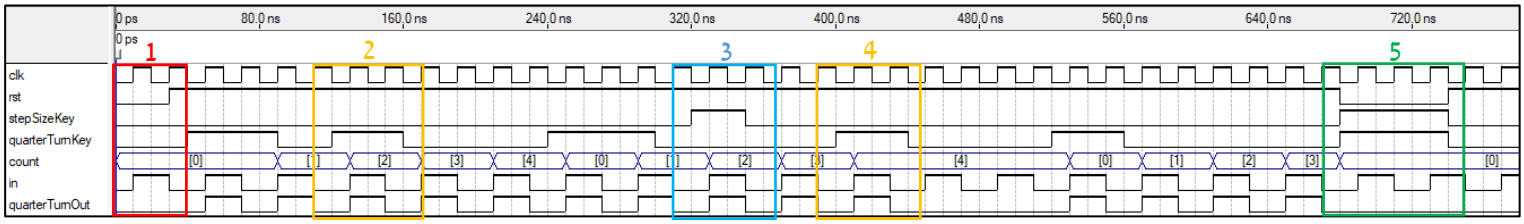
2 - לחיצה ארוכה על כפתור רבע סיבוב.

3 - התעלמות מלחיצה על כפתור גב סיבוב.

מהסימולציה ניתן לראות שהבדיקות תקינות למעט הספירה, בעיה שנתקלנו גם בסימולציה של חצי צעד.

סימולציית Timing

סימולציה של חצי צעד



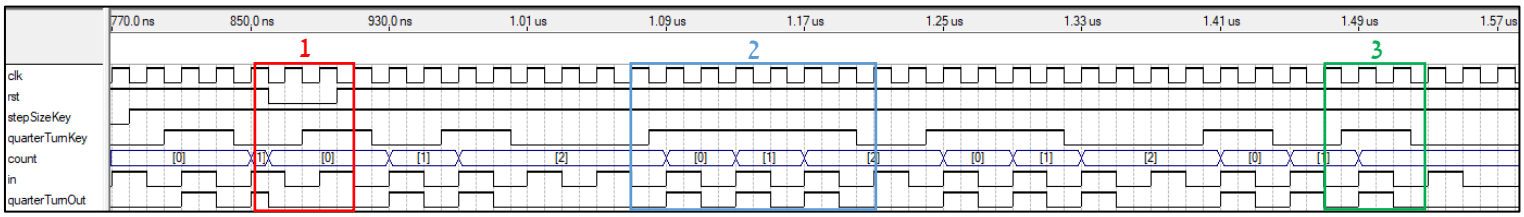
1 - כפתור reset לחוץ, המוצא מוחזק ב"0" לוגי.

2+4 - התעלמות מלחיצה על כפתור רבע סיבוב בזמן ספירה.

3 - התעלמות מלחיצה על שינוי גודל הצעד תוך כדי ספירה.

5 - כפתורים reset, גודל צעד, רבע סיבוב לחוצים, reset לוקח (כנדרש).

סימולציה של צעד שלם



1 - כפתור reset לחוץ, המוצא מוחזק ב"0" לוגי.

2 - לחיצה ארוכה על כפתור רבע סיבוב.

3 - התעלמות מלחיצה על כפתור רבע סיבוב.

בסימולציות timing ניתן לראות הלוגיקה והספירה תקינות.

מכונת מצבים State Machine

יחידה זו אחראית לשלוח למנוע את הקוד הבינארי המתאים ע״פ כיוון הסיבוב וגודל הצעד. היחידה בנויה מ4 מכונות מצבים, כל מכונה מיועדת לאופן פעולה שונה.

<u>כניסות</u>

- clk - כניסת שעון המערכת (50MHz)
- rst – איפוס מכונת המצבים.
- make_step – אות שמציין מתי מכונת המצבים צריכה להחליף מצב בכדי לבצע צעד של המנוע.
- rotateDirectionKey – חיבור למתג 1 שקובע את כיוון הסיבוב.
  - 1 – בכיוון השעון
  - 0 – נגד כיוון השעון
- stepSizeKey – חיבור למתג 3 שקובע את גודל הצעד
  - 1 – צעד רגיל
  - 0 – חצי צעד

<u>יציאות</u>

- Out – באס של 4 ביטים שהם הקוד הבינארי שנשלח למנוע. הקוד הבינארי מפעיל את הסלילים המתאימים וכך המנוע מבצע צעד (או חצי).

<u>אופן פעולה</u>

בכל מחזור שעון של המערכת (clk) אנחנו בודקים האם התקבל סיגנל לביצוע צעד של המנוע (make_step). ברגע שהתקבל סיגנל לביצוע צעד אנחנו בודקים את מצב המתגים rotateDirectionKey וstepSizeKey ובהתאם לכך אנחנו פונים למכונת המצבים הרלוונטית, ומבצעים את הצעד ע״י שליחת הקוד הבינארי המתאים למנוע. כפי שניתן לראות בקוד, יש לנו 4 מכונות מצבים נפרדות. הסיבה, כפי שהוסבר במבוא, היא שבכדי לסובב את המנוע אנחנו נדרשים לשלוח אליו סדרת קודים שונה בהתאם למצב הפעולה. בכדי לפשט את הבעיה החלטנו ליצור מכונת מצבים נפרדת לכל אופן פעולה הנדרש.

להלן רשימת מכונות המצבים, ואופן בחירתן לפי מצב המתגים:

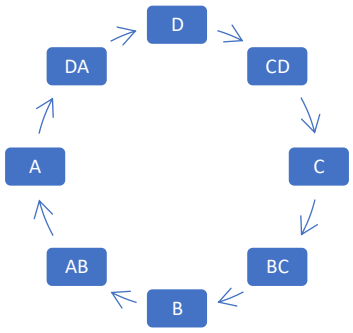
rotateDirectionKey	stepSizeKey	Operation Mode
0	0	חצי צעד - נגד כיוון השעון
0	1	צעד מלא - נגד כיוון השעון
1	0	חצי צעד - בכיוון השעון
1	1	חצי צעד - בכיוון השעון

אופן הפעולה בהתאם למצב המתגים

בסרטוטים הבאים מופיע המעבר בין המצבים עבור כל מכונה. את רשימת הקודים עבור כל מצב ניתן לראות בטבלאות שבמבוא.

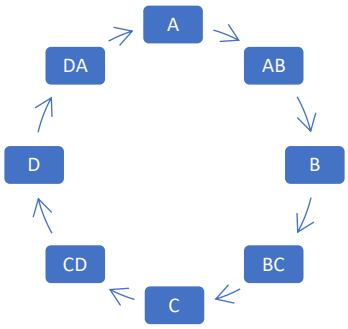
חצי צעד- נגד כיוון השעון :

המכונה מתחילה במצב D



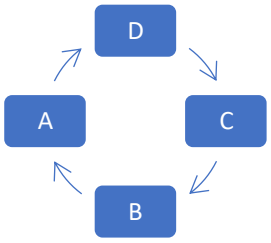
חצי צעד – בכיוון השעון :

המכונה מתחילה במצב A



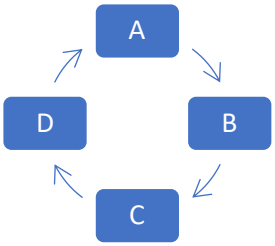
צעד מלא – נגד כיוון השעון :

המכונה מתחילה במצב D



צעד מלא – בכיוון השעון :

המכונה מתחילה במצב A



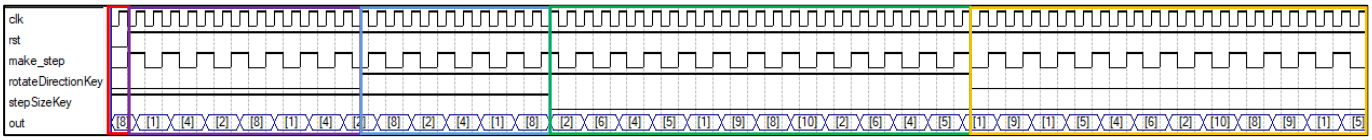
קוד קובץ motorStateMacine.v

```
module motorStateMachine(clk ,rst, make_step, out, rotateDirectionKey, stepSizeKey);
    input wire  clk,
                rst,
                make_step,
                rotateDirectionKey,         // ON(1) = clockwise ; OFF(0) = counter_clockwise;
                stepSizeKey;                // ON(1) = full step ; OFF(0) = half step;

    output wire [3:0] out;
    //wire make_step;

    reg [3:0] cs;
    reg [3:0] ns;

    parameter   A  = 4'b1000,   //8
                B  = 4'b0010,   //2
                C  = 4'b0100,   //4
                D  = 4'b0001,   //1
                AB = 4'b1010,   //A
                BC = 4'b0110,   //6
                CD = 4'b0101,   //5
                DA = 4'b1001;   //9

    //next state logic
    always @ (posedge clk or negedge rst)
        begin
            if (~rst)   // restart. reset enable at 0
                cs = A;

            else if (make_step) begin

                    cs = ns;

                    // clockwise - full step
                    // SW1 = 1 && SW3 = 1
                    if(rotateDirectionKey && stepSizeKey == 1'b1)   begin
                        case(cs)
                            A: ns = B;
                            B: ns = C;
                            C: ns = D;
                            D: ns = A;
                            default: ns = A;
                        endcase
                        end
```

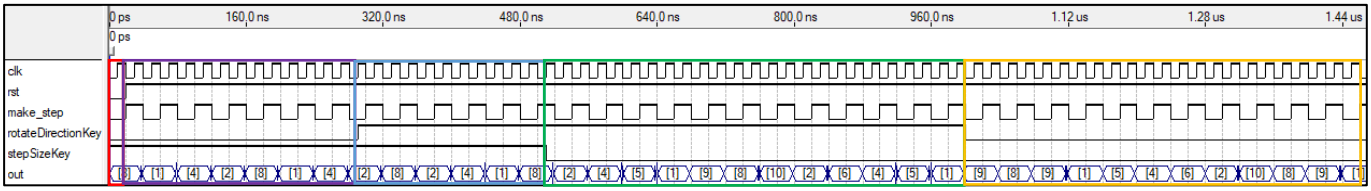
```
44
45                  // clockwise - half step
46                  // SW1 = 1 && SW3 = 0
47                  else if(rotateDirectionKey && stepSizeKey == 1'b0)  begin
48                          case(cs)
49                              A:  ns = AB;
50                              AB: ns = B;
51                              B:  ns = BC;
52                              BC: ns = C;
53                              C:  ns = CD;
54                              CD: ns = D;
55                              D:  ns = DA;
56                              DA: ns = A;
57                              default: ns = A;
58                          endcase
59                      end
60                  // counter clockwise - full step
61                  // SW1 = 0 && SW3 = 1
62                  else if((~rotateDirectionKey) && stepSizeKey == 1'b1)   begin
63                          case(cs)
64                              D: ns = C;
65                              C: ns = B;
66                              B: ns = A;
67                              A: ns = D;
68                              default: ns = D;
69                          endcase
70                      end
71                  // counter clockwise - half step
72                  // SW1 = 0 && SW3 = 0
73                  else if((~rotateDirectionKey) && stepSizeKey == 1'b0)   begin
74                          case(cs)
75                              D:  ns = CD;
76                              CD: ns = C;
77                              C:  ns = BC;
78                              BC: ns = B;
79                              B:  ns = AB;
80                              AB: ns = A;
81                              A:  ns = DA;
82                              DA: ns = D;
83                              default: ns = D;
84                          endcase
85                      end
86                  end //else if
87
88              else
89                  cs <= cs;
90          end
91      assign out = cs;
92
93  endmodule
```

סימולציות

סימולציית Functional



סימולציית Timing



- איפוס מכונת מצבים למצב A (8)
- מכונת מצבים נגד כיוון השעון בגודל צעד מלא.
- מכונת מצבים עם כיוון השעון בגודל צעד מלא
- מכונת מצבים עם כיוון השעון בגודל צעד של חצי צעד.
- מכונת מצבים נגד כיוון השעון בגודל צעד של חצי צעד.

מהסימולציות ניתן לראות שמכונות המצבים עובדות כנדרש, המעברים בין המכונות ובין המצבים תקינים ואין בעיות תזמונים.

הערה: שמנו לב שכאשר המנוע עומד במקום הוא מתחמם, אנחנו חושבים שהסיבה לקח היא שזורם זרם שמחזיקה את המנוע באותו מצב. לכן אנחנו חושבים שהיה נכון להוסיף מצב idle = 4'b0000 למכונות המצבים, מצב בו לא זורם זרם למנוע, וכאשר המנוע נמצא במצב רבע סיבוב המנוע יוחזק במצב זה בזמני המתנה ללחיצות. לא מספיק להוסיף את המצב למכונות המצבים, נדרש להתאים את התכנון הכללי למעבר למצב זה.

רכיב TOP:

```
module StepMotorControllerTop(clk, rst, speedKey, stepSizeKey, rotateDirectionKey, operationModekey, quarterTurnKey, unitsDisplay, tensDisplay, out);

    input wire clk, rst, speedKey, stepSizeKey, rotateDirectionKey, operationModekey, quarterTurnKey;

    output wire [3:0] out;
    output wire [6:0] unitsDisplay;
    output wire [6:0] tensDisplay;

    wire [3:0] speedValueBusWire;
    wire speedControllerOutWire;

    speed speed(.rst(rst), .speedKey(speedKey), .unitsDisplay(unitsDisplay), .tensDisplay(tensDisplay), .speedValue(speedValueBusWire));

    motorController motorController(.clk(clk), .rst(rst), .speedValue(speedValueBusWire), .stepSizeKey(stepSizeKey),
                                    .quarterTurnKey(quarterTurnKey), .operationModekey(operationModekey), .speedControllerOut(speedConvertorOutWire));

    motorStateMachine motorStateMachine(.clk(clk), .rst(rst), .make_step(speedConvertorOutWire), .rotateDirectionKey(rotateDirectionKey),
                                        .stepSizeKey(stepSizeKey), .out(out));

endmodule
```

ברכיב זה אנחנו יוצרים אינסטאנס של בקר המהירות, בקר המנוע ומכונת המצבים. לכל רכיב מחוברים הכניסות והיציאות הרלוונטיות. רכיב זה נצרב אל ה FPGA.

סימולציות

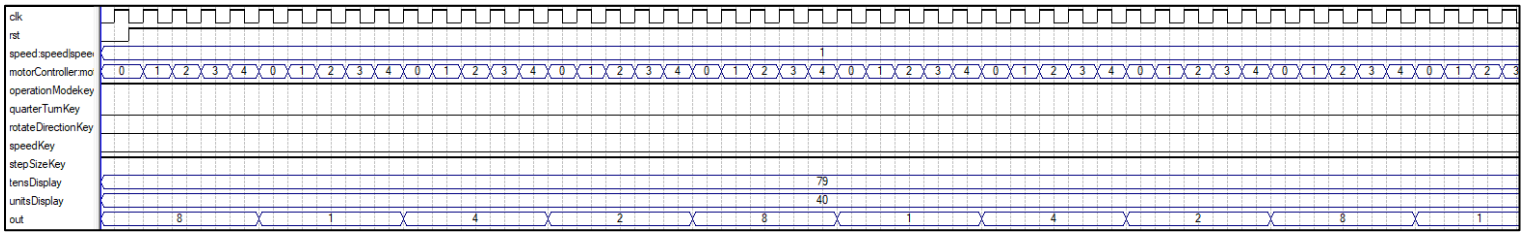
הערה: מודול motorStateMacine, משמש כמודול Top, בעל מספר מצבים אפשריים גדול. לכן נבדוק בסימולציות הבאות מצבים מדגמיים.

לצורך הסימולציות הגדרנו את המונים לערכים קטנים מהנדרש בכרטיס.

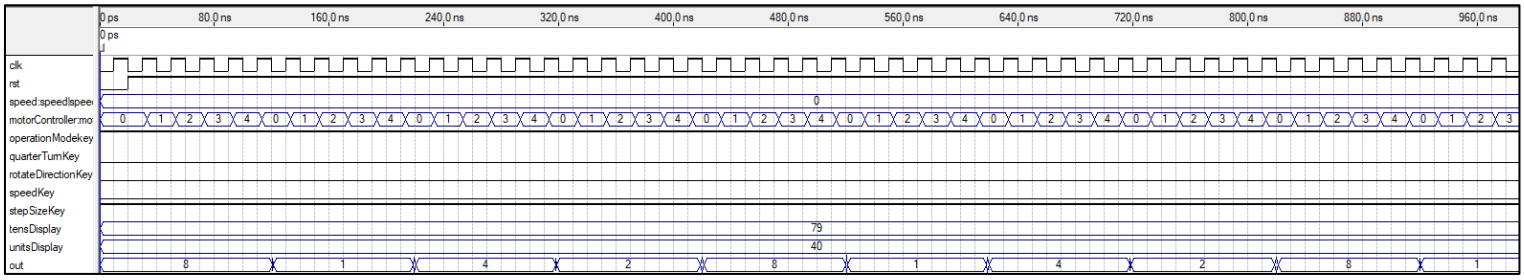
- סיום ספירה עבור מהירות 10 סיבובים בדקה בגודל צעד מלא, מוגדר לערך 4 לצורך הסימולציה.
- סיום ספירה עבור מהירות 10 סיבובים בדקה בגודל צעד חצי צעד, מוגדר לערך 2 לצורך הסימולציה.
- סיום ספירה עבור מהירות 20 סיבובים בדקה בגודל צעד מלא מוגדר לערך 2 לצורך הסימולציה.
- סיום ספירה עבור מהירות 20 סיבובים בדקה בגודל צעד חצי צעד, מוגדר לערך 1 לצורך הסימולציה.
- רבע סיבוב צעד מלא מוגדר לספירה 2.
- רבע סיבוב חצי צעד מוגדר לספירה עד 4.

1. סיבוב רציף נגד כיוון השעון בגודל צעד מלא, מהירות סיבוב של 10 סיבובים לדקה.

<u>סימולציית Functional</u>



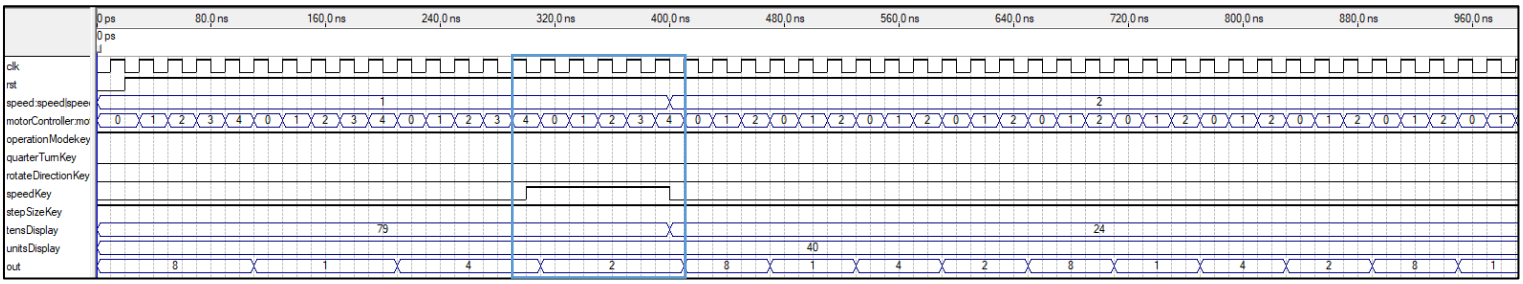
סימולציית Timing



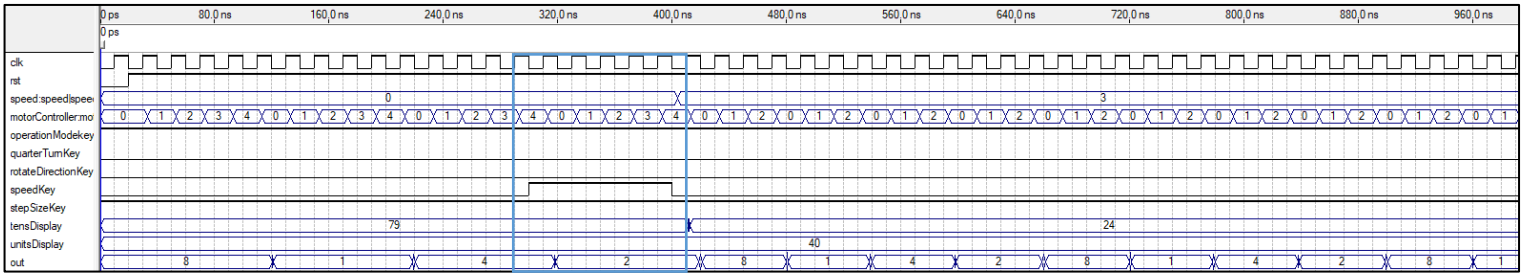
מהסימולציות ניתן לראות שהלוגיקה והתזמונים תקינים, מכונות המצבים והתצוגה תקינות.

2. מעבר למהירות 20 סיבובים בדקה

סימולציית Functional



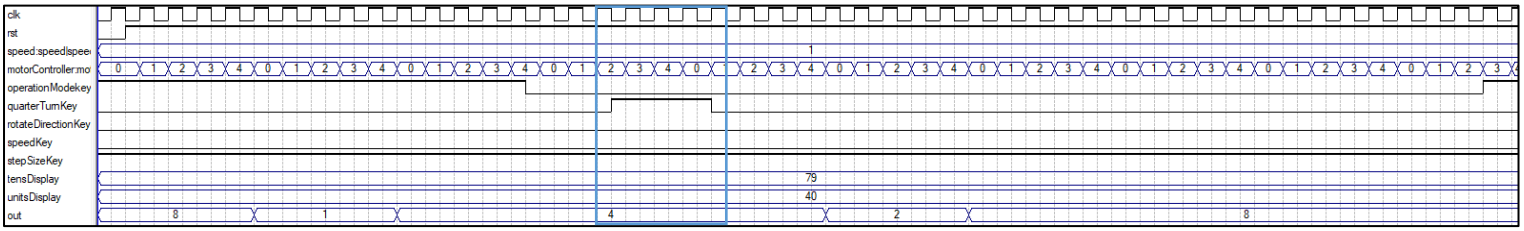
סימולציית Timing



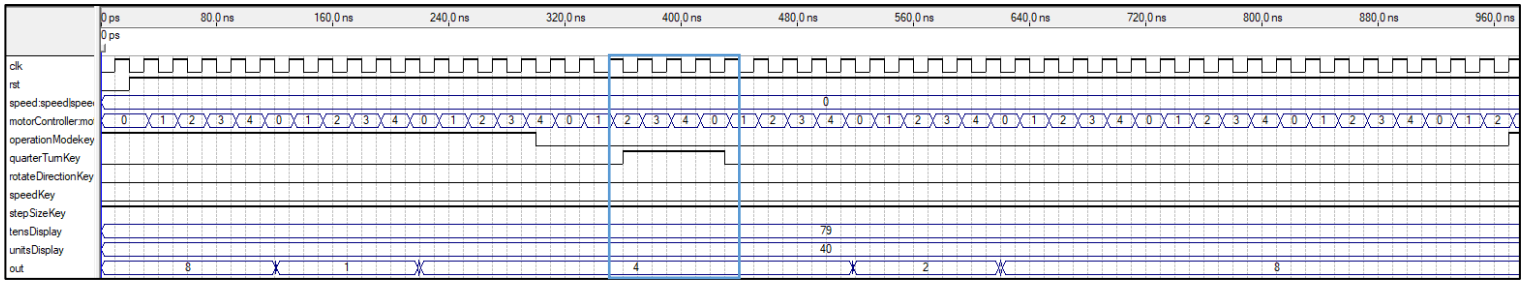
מהסימולציות ניתן לראות שהמעבר למהירות 20 סיבובים בדקה תקין, סיום הספירה של המונה מחזורים שעון השתנה ל2.

3. מעבר לרבע סיבוב במהירות של 10 סיבובים בדקה.

סימולציית Functional



סימולציית Timing



מהסימולציות ניתן לראות שהמעבר למצב רבע סיבוב, וביצוע רבע סיבוב עובד כנדרש.

קישור לצפייה בפעולת המנוע

https://www.youtube.com/watch?v=LQNS8mFqXJU&feature=youtu.be