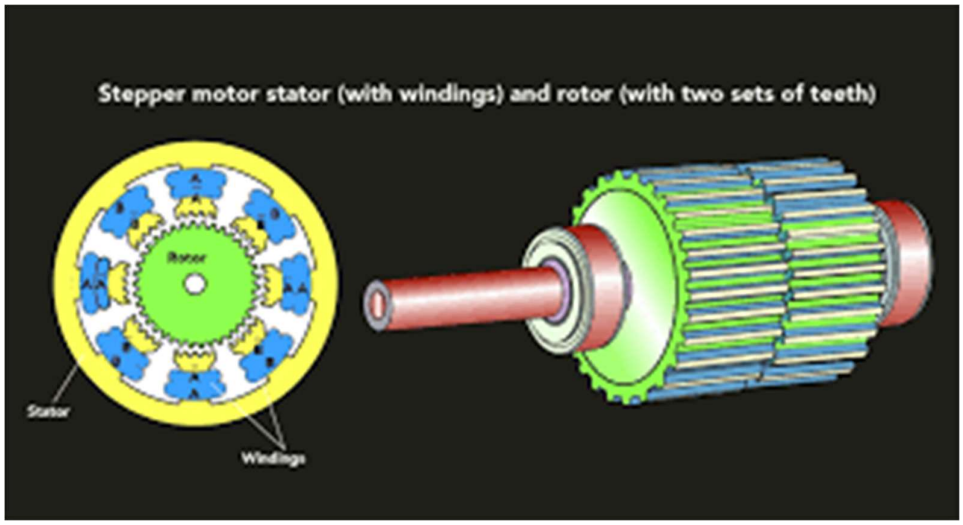
בס"ד

מנוע צעד



מגישים :

יוסף סוקוליק

שובי פסקו

במעבדה זו נדרשנו לתכנן מערכת בקרה דיגיטלית למנוע step-motor מסוג bi-polar. הזרם החשמלי מומר לצעד מנוע באופן הבא, ישנם שני סלילים ניצבים זה לזה ואנו מזרימים זרם דרכם. המנוע משלים מחזור תנועה ב-200 צעדים כאשר גודל כל צעד הוא 1.8°.

האופן בו המנוע פועל כך שהוא מתקדם 1.8° הוא ע״י סדרת הכניסות להדקי הסלילים הבאה (1000,0010,0100,0001), כאשר (a,~a,b,~b) זהו סדר הדקי הסלילים. כלומר A, B הם שני הסלילים דרכם זורם זרם משתנה הגורם ליצירת שדה מגנטי (לפי חוק פאראדיי) המניע את הרוטור. עבור כל אות כניסה בסדרה הרוטור מסתובב 1.8°. בכדי שהמנוע יתקדם 0.9° סדרת הכניסות מורכבת מ-8 שלבים כך: (1000,1010,0010,0110,0100,0101,0001,1001) כלומר, בין כל שני כניסות של הסדרה עבור צעד שלם אנו מוסיפים כניסה בה אנו מזרימים זרם בשני הסלילים יחד כך שהרוטור סב 0.9° לאחור. יוצא שעבור שתי אותות כניסה עוקבים בסדרה זו המנוע מסתובב 0.9°.

דרישות המערכת הן:

א. גודל הצעד – צעד שלם (1.8°) או חצי צעד (0.9°), בתור מהנדסי המערכת אנו יכולים להגדיר את סדרות המתח המחזוריות עבור הדקי הסלילים, כך שנקבל את גודל הצעד הרצוי.
המשתמש יוכל לשלוט בכך בעזרת הורדת המתג SW3 כאשר מתג לחוץ משמעותו צעד שלם.

ב. כיוון הסיבוב – עם כיוון השעון או נגד כיוון השעון.
המשתמש יוכל לשלוט בכך בעזרת הורדת המתג SW1 כאשר מתג לחוץ משמעותו סיבוב עם כיוון השעון.

ג. הפעלה רציפה/ בדידה – למנוע יהיו שני מצבי פעולה, הפעלה רציפה ללא הפסק והפעלה שתלויה בלחיצת המשתמש על הכפתור KEY1, כך שבכל לחיצה חדשה על KEY1 המנוע יסתובב רבע מחזור.
המשתמש יוכל לשלוט בכך בעזרת הורדת SW2 כאשר מתג לחוץ משמעותו הפעלה רציפה.

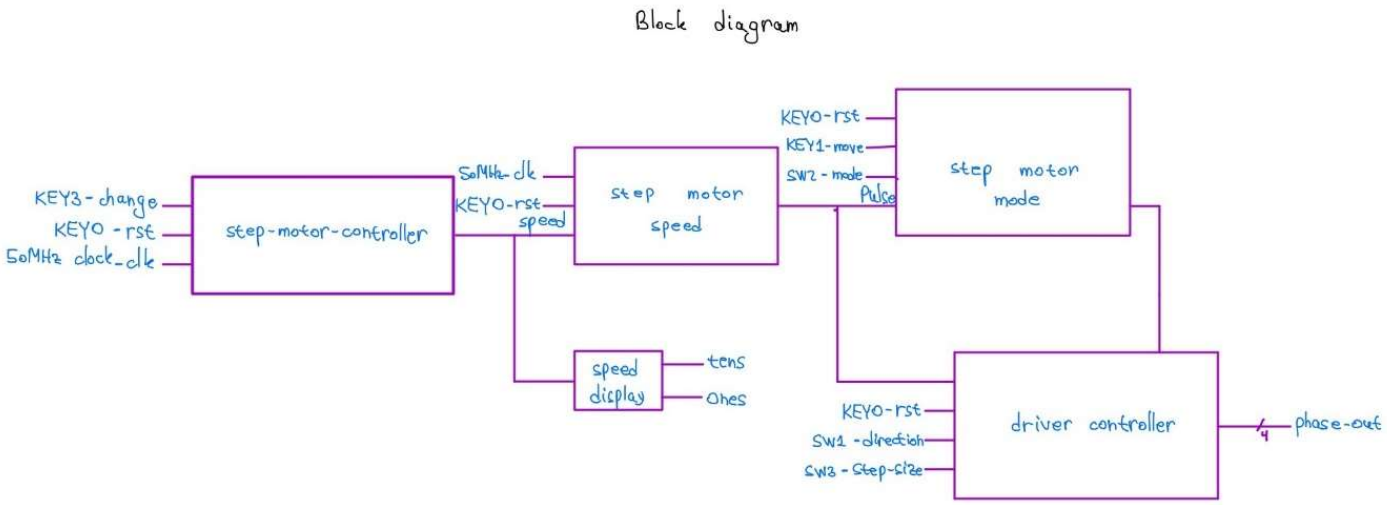
ד. למנוע יהיו מס׳ מהירויות בהן הוא יוכל להסתובב,
$\{10\frac{cycles}{minute}, 20\frac{cycles}{minute}, 30\frac{cycles}{minute}, 40\frac{cycles}{sminute}, 50\frac{cycles}{sminute}, 60\frac{cycles}{minute}\}$, והמעבר ביניהן הוא ע״י לחיצה על KEY3 בסדר עולה מ-$10\frac{cycles}{minute}$ ל-$60\frac{cycles}{minute}$ ולאחר מכן בסדר יורד מ-$60\frac{cycles}{minute}$ ל-$10\frac{cycles}{minute}$.

ה. למערכת יהיה לחצן (KEY0) reset שמאתחל את המערכת וכן מאתחל את מהירות המנוע ל-$10\frac{cycles}{minute}$.

ו. מהירות המנוע תוצג ע״י 7- segment display שבכרטיס בספרות HEX0, HEX1.

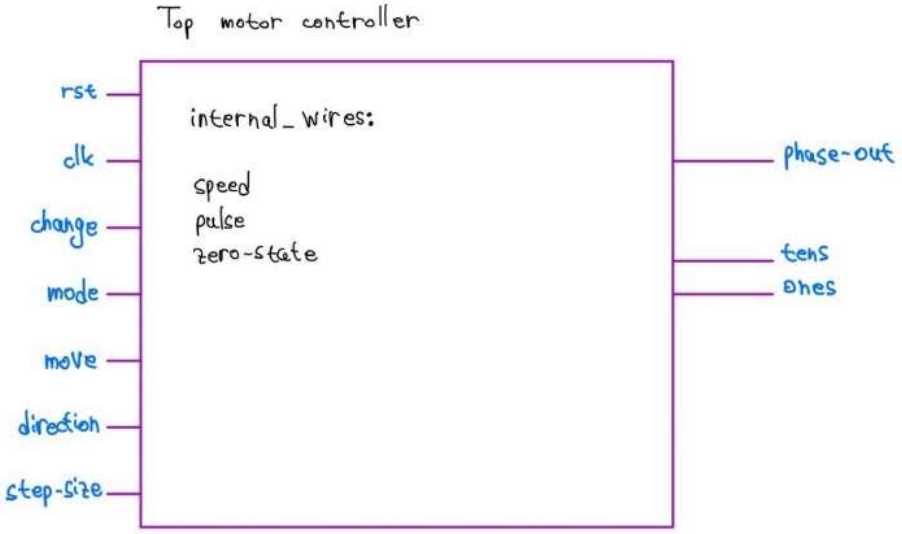
תכנון המערכת

על מנת לפשט את פעולת ה-debug והבנת הקוד, מימשנו את מערכת הבקרה ע״י מס׳ modules כאשר כל אחד מהם מממש דרישה יחידה מדרישות המערכת, ולבסוף איחדנו אותם ב-top level.



האינטגרציה של הmodules השונים במערכת היא כך :

הסכמה של המערכת הכוללת כ-black box היא :



step_motor_controller

מודל זה קובע את המהירות הרצויה של המנוע על ידי כניסה מכפתור KEY3 שבכרטיס, וכן כפתור KEY0 לאתחול למהירות התחלתית של $10\frac{cycles}{minute}$.

מימשנו את המודל על ידי מכונת מצבים, הגדרנו את מצבים 1-6 המתאימים למהירויות $10\frac{cycles}{minu}$ עד $60\frac{cycles}{minute}$.

הכניסות: המודל מקבל את אותות הכניסה שלו ישירות מהכרטיס.

שעון/clk: מחובר לשעון המערכת (שהגדרנו את תדירות העבודה שלו $f = 50[MHz]$ הוא משמש לבדיקת האתחול ותפעול מכונת המצבים.

אתחול/rst: מחובר לכפתור KEY0, כאשר המשתמש לוחץ עליו, אות זה משתנה ל-'0'.

שינוי המהירות/change: מחובר לכפתור KEY3 כאשר כל לחיצה חדשה (קצרה או ארוכה) מעלה את המהירות עד למצב 6 ולאחר מכן כל לחיצה מורידה את המהירות עד למצב 1 וחוזר חלילה.

המוצא: אות זה נכנס אל שני מודלים step_motor_speed, speed_display.

המהירות הרצויה/ speed : המוצא הוא המצב העכשווי של מכונת המצבים כלומר, מהי מהירות המנוע. זהו אות המורכב משלושה ביטים.

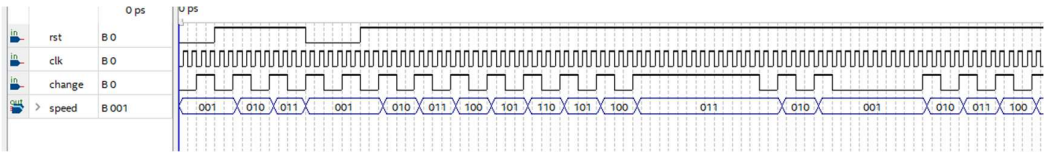
מצורף הקוד שכתבנו למימוש מודל זה:

```
module step_motor_controller(
                        input wire change,/*key 3- increase/decrease
                                            the motor speed*/
                        input wire rst,
                        input wire clk,//50MHz
                        output wire [2:0]speed
                        );

                        //State definition
                        parameter start = 3'b001,
                                  s20 = 3'b010,
                                  s30 = 3'b011,
                                  s40 = 3'b100,
                                  s50 = 3'b101,
                                  s60 = 3'b110;

                        reg change_o;
                        reg[2:0]cs;
                        reg[2:0]ns;
                        reg faster = 1'b1;/*If key 3 makes the motor faster
                                            or slower */


                        always  @(posedge clk)
                           begin
                           change_o <= change;
                           end
```

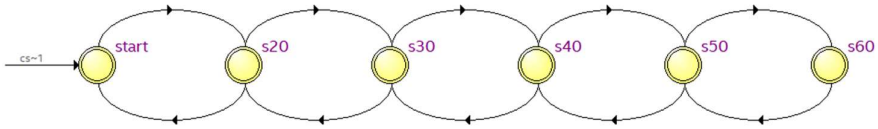
```
28
29                          //every clock insert ns to cs
30
31                          always @(posedge change_o or negedge rst)
32                             begin
33                                if (~rst)
34                                   begin
35                                   cs <= start;
36                                   faster <= 1'b1;
37                                   end
38                                else
39                                   begin
40                                   cs <= ns;
41
42                                   if (cs == start)
43                                      begin
44                                      faster <= 1'b1;
45                                      end
46
47                                   if (cs == s60)
48                                      begin
49                                      faster <= 1'b0;
50                                      end
51                                   end
52
53                             end
54
55                          //Next state logic
56
57                          always @(cs)//Will actevaite when cs or the in change
58                             begin
59                                case (cs)
60                                         start: ns = s20;
61                                         s20: ns = faster ? s30 : start;
62                                         s30: ns = faster ? s40 : s20;
63                                         s40: ns = faster ? s50 : s30;
64                                         s50: ns = faster ? s60 : s40;
65                                         s60: ns = s50;
66                                         default: ns = start;
67                                         endcase
68
69                                   end
70                       assign speed = cs;
71
72 endmodule
```

רואים בסימולציה שהמהירות עולה ויורדת בהתאם לדרישות וכן שהמהירות לא משתנית עד שלוחצים על הכפתור לחיצה חדשה (בפרט לא לחיצה ארוכה). וכן שכאשר מאתחלים את המערכת המערכת מאותחלת במהירות 10. כאשר לא לוחצים על הכפתור המהירות נשארת קבועה.

מצורף דיאגרמת הגלים:



כמו כן מצורפת סכמה של מכונת המצבים, ואכן רואים שהמהירות עולה ויורדת בהתאם לדרישות ושהאתחול מתחיל במהירות הראשונה:



Speed display

מודל זה אחראי לתצוגת המהירות בה המנוע מסתובב בעזרת ה7 segment display- שבכרטיס בספרות HEX0, HEX1.

מימשנו את המודל בעזרת יצירת אובייקט של המודל **bcd_7seg** (שיצרנו במעבדה 1) האובייקט מקבל מס׳ המיוצג בעזרת ספרות בינאריות וממיר זאת לרצף הביטים כך ש- HEX1 יציג את הספרה המתאימה. המס׳ שהאובייקט קיבל עבור כניסה זו הוא מצב המערכת)שנקבע ע״י מכונת המצבים שבמודל (**step motor controller** המתאים למהירות המנוע. לדוג׳ מהירות 10 סיבובים לדקה מיוצגת ע״י המספר 1, ומספר זה הוא ספרת העשרות של תצוגת המהירות.

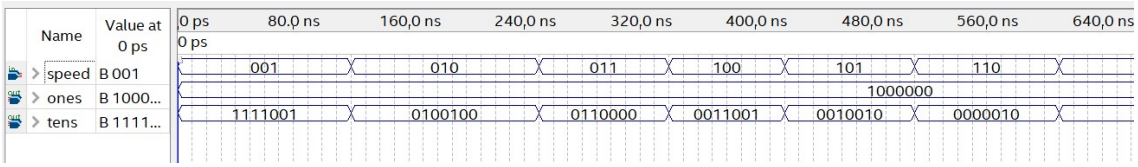
כניסות: speed- המודל מקבל את הכניסה מהיציאה של המודל **step motor controller** והיא מייצגת את מהירות המנוע.

יציאות: ones, tens- ספרת היחידות וספרת העשרות המרכיבות את גודל המהירות יוצאות אל ה-7 segment display- שבכרטיס בספרות HEX0, HEX1.

מצורף הקוד שכתבנו למימוש מודל זה:

```
module speed_display(input wire [2:0]speed,
                     output [6:0] tens,
                     output [6:0] ones);

                     bcd_7seg seg_tens (.num(speed), .seg_num(tens));


                     assign ones = 7'b1000000;

endmodule
```

לצורך בדיקת התצוגה ביצענו סימולציה בה שינינו את הכניסה speed ובחנו את היציאות בפרט את tens:



כפי שניתן לראות בדקנו עבור כל ערכי הכניסות המוגדרים במערכת, חלקן בסדר עולה וחלקן בסדר יורד וקיבלנו את המוצא המייצג את הספרות הרצויות ב- 7 segment display.

step_motor_speed

מודל זה מקבל את המהירות הרצויה מהמודל step_motor_controller ומוציא סיגנל שעון בתדירות המתאימה למהירות הרצויה כלומר, שיספק מס׳ פולסים לשנייה כך שהמנוע יסתובב במהירות הרצויה.

במודל זה אנחנו בעצם מכניסים למודל הפנימי counter את המספר שאליו צריך לספור עד למעבר שעון.

כדי לדעת מהו המספר שאליו צריך לספור עבור כל מהירות, חישבנו שכדי להגיע לN סיבובים לדקה צריך לתת פולס למנוע כל: $N\_cycles\_per\_minute, \frac{400N}{60} = \frac{20}{3}N$ (מכיוון שצריך לתת 400 פולסים למנוע כדי שישלים סיבוב). תדר העבודה של שעון המערכת הוא $50[MHz]$ ולכן צריך לספור עד $x = \frac{3}{20N} \cdot 50M$.

כדי שנוכל לדעת מתי משתנה כל פולס הגדרנו את המוצא שיתנהג כאל שעון, לכן סך הכל צריך לספור עד חצי זמן מחזור כלומר: $num\_to\_pulse = \frac{3}{20N} \cdot 50M \cdot \frac{1}{2}$.

הכניסות: המודל מקבל חלק מהכניסות ישירות מהכרטיס וחלק מהמודל step_motor_controller.

שעון/clk: מחובר לשעון המערכת ובשימוש לצורך המונה.

איתחול/rst: מחובר לכפתור KEY0.

המוצא: אות זה נכנס אל המודלים driver_controller, step_motor_mode.

שעון הפולסים/step_pulse : שעון שזמן המחזור שלו נקבע בהתאם למהירות הרצויה (שנבחרה ע״י המשתמש).

מצורף הקוד שכתבנו למימוש מודל זה:

```
module step_motor_speed(
                        input wire [2:0]speed,
                        input wire clk,
                        input wire rst,
                        output wire step_pulse
                        );

                        reg [19:0] num_to_pulse;
                        always @(speed)
                              begin
                              case(speed)
                                 3'b001: num_to_pulse = 20'hB5B8D8;// counts until 375000
                                 3'b010: num_to_pulse = 20'h2DC6C;// counts until 187500
                                 3'b011: num_to_pulse = 20'h1E848;// counts until 125000
                                 3'b100: num_to_pulse = 20'h16E36;// counts until 93750
                                 3'b101: num_to_pulse = 20'h124F8;// counts until 75000
                                 3'b110: num_to_pulse = 20'hF424;// counts until 62500
                                 default num_to_pulse = 20'hB5B8D8;// counts until 375000

                              endcase
                           end

                counter counter_inst(.clk(clk), .rst(rst), .num_to_pulse(num_to_pulse), .step_pulse(step_pulse));



endmodule
```

מודל פנימי -Counter

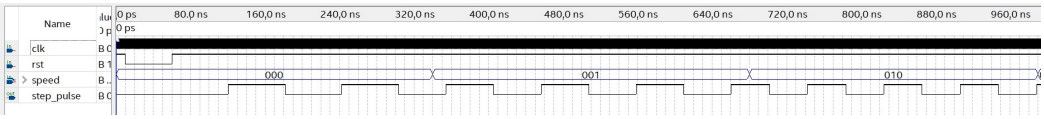
המונה סופר עד המספר הרצוי כפי שהסברנו לעיל ואז הופך את המוצא step_pulse וזהו בעצם השעון שלנו. הכניסות והמוצא זהים למודל הקודם למעט הכניסה num שזה המספר שאליו צריך לספור.

מצורף הקוד שכתבנו למימוש מודל זה:

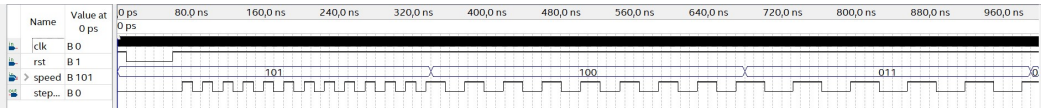
```
31  module counter(
32                  input wire clk,
33                  input wire rst,
34                  input wire [19:0]num_to_pulse,
35                  output reg step_pulse);
36
37                  reg [19:0]count;
38
39                  always @(posedge clk or negedge rst)
40                  begin
41                     if(~rst)
42                        begin
43                        count <= 20'h0;
44                        step_pulse<= 1'b1;
45                        end
46                     else
47                        begin
48                        count <= count + 1'h1;
49                        if (count >= num_to_pulse)
50                                 begin
51                                 count <= 20'h0;
52                                 step_pulse = ~step_pulse;
53                                 end
54
55                        end
56
57                  end
58
59  endmodule
```

בדקנו את שני המודלים ביחד מכיוון שהיציאות שלהם זהות. (והמודל step_motor_speed לא עומד בפני עצמו). לא יכולנו להריץ סימולציה עם המספרים שהמכונה באמת צריכה להריץ מכיוון שהסימולטור מוגבל לזמן ריצה של עד $960[ns]$. לכן לצורך הסימולציה הגדרנו מספרים קטנים ורואים שהעיקרון עובד כלומר, רואים שהמודל מייצר לנו שעון וכן שהמודל יודע גם לשנות מצבים באמצע הספירה.

3 מצבים ראשונים כאשר מעלים את המהירות :



מצבים אחרונים בסדר יורד 3:



step_motor_mode

מודל זה אחראי למצב הפעולה של המנוע האם הוא יפעל ברציפות או ברבעי מחזור בדידים. כאשר מצב הפעולה של המערכת הוא בדיד, סיבוב המנוע תלוי בלחיצת המשתמש על KEY1 באופן הבא, אין משמעות למשך הזמן בו KEY1 לחוץ אפילו אם נשלם רבע המחזור המנוע לא יסתובב עוד. כמו כן אין משמעות למספר הלחיצות על KEY1 במהלך הסיבוב של רבע המחזור. בכדי שהמנוע יסתובב רבע מחזור נוסף צריך להתקיים שני תנאים: 1) KEY1 השתחרר לאחר הלחיצה הקודמת, 2) KEY1 נלחץ שוב לאחר שהמנוע השלים את הסיבוב.

המוצא של המודל הוא 'מצב אפס' ('zero_state') וזה '1' אם המנוע צריך להיות במנוחה, אחרת '0'. כאשר אנחנו מאתחלים את המערכת

כאשר ה'מצב' ('mode') ב- '1' המנוע צריך לפעול ברציפות ולכן המוצא יהיה '0'.

כאשר המערכת במצב של רבע מחזור המנוע צריך להיות במנוחה כלומר ב-'1' אלא אם כן לחצנו על הכפתור ('move') KEY1 ואז המנוע צריך להסתובב רבע סיבוב. כדי שהמנוע יסתובב רבע מחזור אנחנו נותנים לו לרוץ 100 צעדים (שהם רבע מהצעדים הדרושים למנוע על מנת שיסתובב מחזור שלם) על ידי זה שאנחנו מאתחלים משתנה ('num2count') ל- 100 וכל מחזור שעון (של שעון pulse) המשתנה יורד באחד.

כדי לוודא שאכן הלחיצה היא חדשה הגדרנו דגל ('new_pressed') שמאותחל ב-1 ואחרי שלחצנו על הכפתור הוא יורד ל-0 ועולה רק אחרי שהלחיצה על הכפתור הופסקה. כדי לוודא שמספר לחיצות במהלך תזוזת המנוע לא ישפיעו על המנוע, המשתנה יאותחל ל-100 רק אם הוא היה 0, הדגל מורם ו KEY1 לחוץ.

הכניסות: המודל מקבל חלק מהכניסות ישירות מהכרטיס וחלק מהמודל step_motor_speed.

שעון/pulse : המוצא של step_motor_speed. שעון שזמן המחזור שלו נקבע בהתאם למהירות הרצויה של המנוע.

איתחול/rst: מחובר לכפתור KEY0.

מצב/mode: מחובר למתג SW1 שבכרטיס. קובע האם המנוע פועל ברציפות או ברבעי מחזור בדידים.

הזז/move: מחובר לכפתור KEY1 שבכרטיס. עבור מצב רבע מחזור לחיצה חדשה תסובב את המנוע רבע מחזור.

המוצא: אות זה נכנס אל המודל driver_controller.

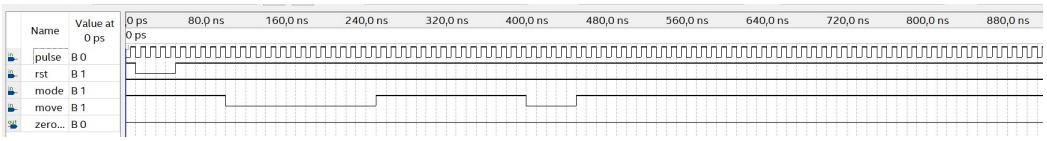
מצב אפס/'zero_state': המוצא של step_motor_mode. קובע האם יזרום זרם בסלילי המנוע.

מצורף הקוד שכתבנו למימוש מודל זה :

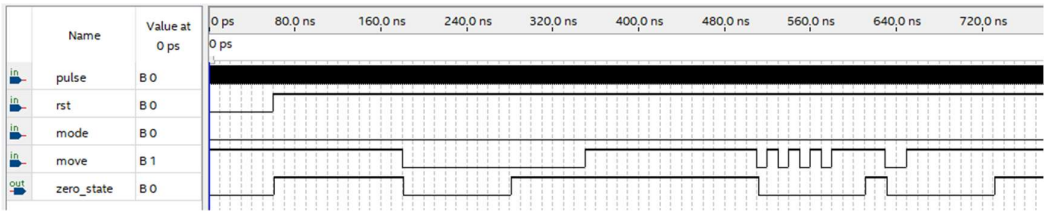
```
module step_motor_mode2(
                        input wire rst,
                        input wire pulse,//step_pulse
                        input wire mode,//sw2; 0 for quadrater cycle, 1 for continous
                        input wire move, //key1= move quadrater cycle

                        output reg zero_state
                        );

                        reg [6:0] num2count;
                        reg new_pressed;
                        always @(posedge pulse or negedge rst)//(pulse)
                           begin
                              if(~rst)
                              begin
                                 new_pressed <= 1'b1;
                                 num2count <= 7'h0;
                                    if(~mode)
                                       zero_state <= 1'b1;

                              end

                              else if (mode == 1'b1)
                                 begin
                                 zero_state <= 1'b0;
                                 end

                              else
                                 begin
```

```
                                 else
                                    begin
                                          if(~move & new_pressed)
                                             begin
                                             new_pressed <= 1'b0;

                                             if(num2count == 7'h0)

                                                num2count <= 7'h64; //count to 100
                                             end
                                          else if (move)
                                             new_pressed <= 1'b1;

                                          if (num2count > 7'h0)
                                             begin
                                             num2count <= num2count - 7'h1;

                                             zero_state <= 1'b0;
                                             end

                                          else if(num2count == 7'h0)
                                             zero_state <= 1'b1;
                                    end
                              end
endmodule
```

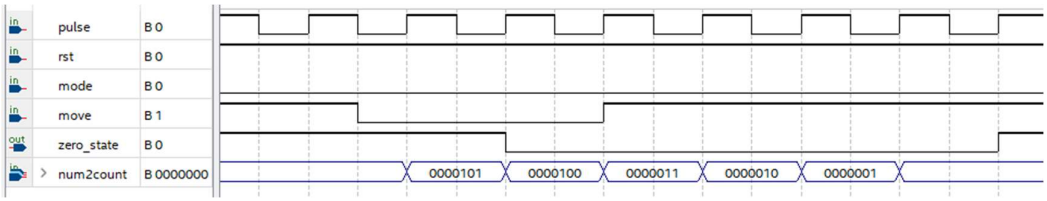
סימולציה לבדיקת 'הפעלה רציפה', רואים ש'מצב אפס' יהיה אפס ו'הזז' לא משפיע על זה :



כדי להריץ את הסימולציה למצב 'רבע סיבוב' הרצנו את השעון שלנו בננו שניה. ואנחנו רואים שהמנוע נמצא ב'מצב אפס' אלא אם כן יש לחיצה חדשה ואז הוא זז רק רבע סיבוב. .



הרצנו סימולציה שבה רואים שכאשר מגדירים למודל לספור עד חמש הוא אכן סופר 5 מחזורי שעון ואז מעלה בחזרה את מצב אפס. לא הראנו את זה על 100 כיון שזה לא כל כך אפשרי להראות את זה על תמונה במסך.



driver_controller

זהו המודל שבעצם מתפעל את המנוע בצורה ישירה, כאשר מקבלים מהמשתמש דרך הכרטיס את הכיוון וגודל הצעד' כמו כן הוא מקבל אות האם המנוע צריך לפעול מהמודל step_motor_mode, כאשר מהירות סיבוב המנוע נקבעת על ידי 'שעון pulse' שהוא מקבל מהמודל step_motor_speed.

הגדרנו 8 מצבים (phase_out) שהם הביטים שנכנסים לסלילים השונים של המנוע, כדי לרוץ על המצבים השונים הגדרנו משתנה 'state' שבכל מחזור שעון מתקדם לפי כיוון התנועה הרצוי. הוספנו גם מצב אפס כאשר המנוע לא אמור לפעול, אז לא יזרום שום זרך דרך הסלילים (הפרש המתחים בין הדקי הסלילים הוא אפס) על מנת שלא לבזבז הספק.

עד המודל הזה בעצם בנינו את כל המערכת ל-חצי צעד, בכדי שהמערכת תפעל ב-צעד שלם כל שעלינו לעשות זה לדלג על המצבים שהם חצי וכן לשנות מצב רק כל מחזור שעון שני (כדי שהמהירויות לא תשתנינה). בכדי לקיים את הדרישות האלו אנחנו מקדמים את 'state' כל פעם שניה, וכדי לדלג על מחזור שעון אנחנו נשתמש בדגל ('flag').

כאשר אנחנו נמצאים במצב של רבע מחזור והמנוע סיים להסתובב, השארנו את ה'state' במקום ורק הוצאנו פולס של אפסים. לא העברנו אותו ל'state' של אפס כדי שבפעם הבאה שהמנוע יתחיל לפעול הוא ימשיך להתקדם ולא יחזור על אותו 'מצב' בו סיים ברבע מחזור הקודם, (אם הוא היה חוזר על המצב אז המנוע לא היה מתקדם [שהרי הסליל כבר התקדם אליו בסיבוב הקודם] אבל הוא עדיין היה נכלל בספירה של 100 הפולסים שצריך לתת למנוע בשביל הרבע מחזור וכך היינו מקבלים 'באג' שאת הרבע מחזור הראשון הוא היה עושה מושלם ומאז הוא היה מתחיל לפגר).

הכניסות: המודל מקבל חלק מהכניסות ישירות מהכרטיס ואת חלקן מהמודלים step_motor_speed, step_motor_mode.

איתחול/rst: מחובר לכפתור KEY0.

שעון pulse/step_pulse : המוצא של step_motor_speed. שעון שזמן המחזור שלו נקבע בהתאם למהירות הרצויה של המנוע.

כיוון/:direction מחובר למתג SW1 בכרטיס. קובע את כיוון סיבוב המנוע.

מצב אפס/zero_state: המוצא של step_motor_mode. קובע האם המנוע יפעל או לא.

גודל הצעד/step_size: מחובר למתג SW3 בכרטיס וקובע האם המנוע יסתובב בחצי צעד או צעד שלם.

המוצא: מתחבר לכרטיס ומשם למנוע.

הביטים למנוע/phase_out: 'הביטים' שקובעים על איזה סליל ייפול מתח ובאיזה כיוון כך שיזרום דרכו זרם בכיוון הנגדי, ב'פולס' הנוכחי.

מצורף הקוד שכתבנו למימוש מודל זה:

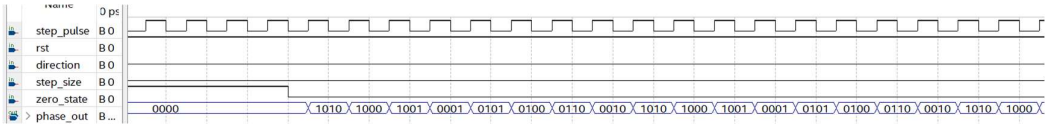
```
module driver_controller(
                        input wire rst,
                        input wire step_pulse,// a 'pulse clock'
                        input wire direction,//sw1: 0 for counter clockwise, 1 for clockwise
                        input wire step_size,//sw3: 0 for half a step, 1 for a full step
                        input wire zero_state,// 1 will "turn off" the motor
                        output reg [3:0] phase_out);

                        reg [3:0] state;// the output pulse
                        reg flag;//for a full step, it will ignore evrey other cycle
                        always @(posedge step_pulse or negedge rst)
                               begin
                               if (~rst)
                                  begin
                                  state <= 4'b1;
                                  flag <= 1'b0;
                                  phase_out <= 4'b0000;
                                  end

                               else
                               begin
                                     if(zero_state == 1'b1)
                                         begin
                                         phase_out <= 4'b0000;
                                         flag <= 1'b1;
                                         end
                                      else
                                      begin

                                     if (direction == 1'b1 & step_size == 1'b0)
                                     begin
                                          if (state == 4'b1000)
                                            begin
                                            state <= 4'b0001;
                                            end
                                          else
                                            begin
                                            state <= state + 4'b0001;
                                            end
                                     end

                                     else if (direction ==1'b0 & step_size == 1'b0)
                                      begin
                                            if (state == 4'b0001)
                                              state <= 4'b1000;
                                            else
                                              state <= state - 4'b0001;
                                      end
                                     else if (direction ==1'b1 & step_size == 1'b1)
                                      begin
```

```
                                     else if (direction ==1'b1 & step_size == 1'b1)
                                            begin
                                                if (flag)
                                                  flag <= ~flag;
                                                else if (state == 4'b0111)
                                                    begin
                                                    state <= 4'b0001;
                                                    flag <= ~flag;
                                                    end
                                                else
                                                    begin
                                                    state <= state + 4'b010;
                                                    flag <= ~flag;
                                                    end
                                            end
                                     else if (direction ==1'b0 & step_size == 1'b1)
                                        begin
                                            if (flag)
                                                 flag <= ~flag;
                                            else if (state == 4'b0001)
                                                begin
                                                state <= 4'b0111;
                                                flag <= ~flag;
                                                end
                                            else
                                                begin
                                                state <= state - 4'b010;
                                                flag <= ~flag;
                                                end
                                        end
                                     case (state)
                                            4'b0001: phase_out <= 4'b1000 ;
                                            4'b0010: phase_out <= 4'b1010 ;
                                            4'b0011: phase_out <= 4'b0010 ;
                                            4'b0100: phase_out <= 4'b0110 ;
                                            4'b0101: phase_out <= 4'b0100 ;
                                            4'b0110: phase_out <= 4'b0101 ;
                                            4'b0111: phase_out <= 4'b0001 ;
                                            4'b1000: phase_out <= 4'b1001 ;
                                            default  phase_out <= 4'b0000 ;
                                     endcase

                                     end

                                 end
                             end
 endmodule
```

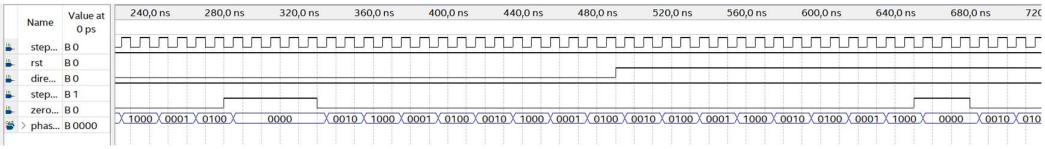
הרצנו סימולציות ורואים שכאשר 'מצב אפס' דלוק הוא מאפס את המוצא וכאשר הוא יורד המוצא ממשיך לספור בסדר הנכון, כלומר יוציא את הפולס הבא ולא יחזור על הפולס האחרון. וכן אנחנו רואים שהפולסים יוצאים בסדר הנכון בצורה מחזורית.

סימולציה של חצי צעד ונגד כיוון השעון :

סימולציה של חצי צעד ונגד כיוון השעון :



סימולציה של צעד שלם בשני הכיוונים :



top_motor_controller

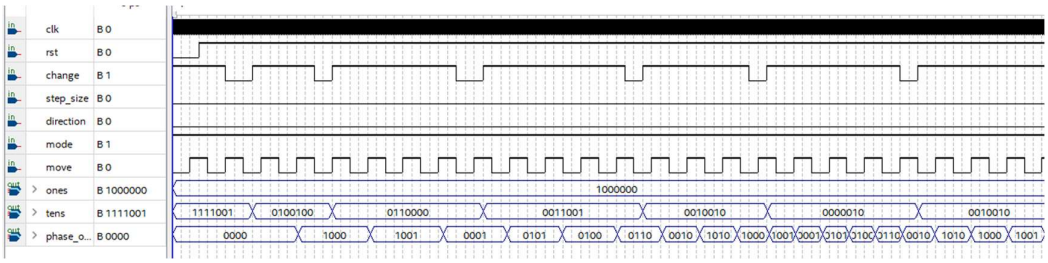
לבסוף יש לנו את המודל העליון שכל מטרתו לקרוא לשאר המודלים ולתאם בינהם.

מצורף הקוד שכתבנו:

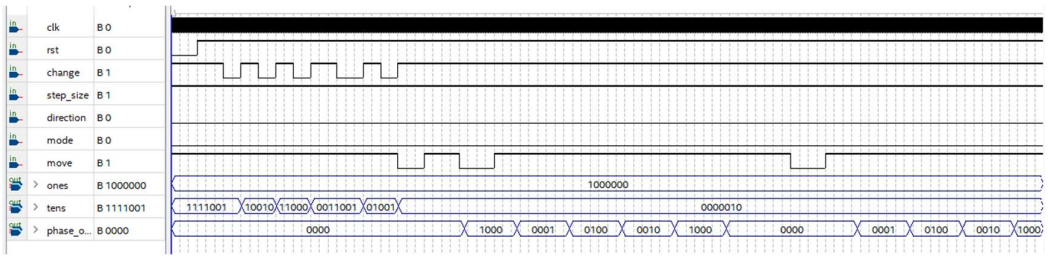
```
module top_motor_controller(
                            input wire rst,
                            input wire clk,
                            input wire change,
                            input wire mode,
                            input wire move,
                            input wire direction,
                            input wire step_size,
                            output wire [3:0] phase_out,
                            output wire [6:0] ones,
                            output wire [6:0] tens
                            );

                            //internal wires
                            wire [2:0] speed;
                            wire pulse;
                            wire zero_state;

                            step_motor_controller smc_inst(.change(change), .rst(rst), .clk(clk), .speed(speed));

                            step_motor_speed sms(.speed(speed), .rst(rst), .clk(clk), .step_pulse(pulse));

                            speed_display sd(.speed(speed), .ones(ones), .tens(tens));

                            step_motor_mode2 smm(.rst(rst), .mode(mode), .move(move),
                                                 .pulse(pulse), .zero_state(zero_state));

                            driver_controller dc(.rst(rst), .step_pulse(pulse), .direction(direction),
                              .step_size(step_size), .zero_state(zero_state), .phase_out(phase_out));

 endmodule
```

לצורך הסימולציה השתמשנו במספרים קטנים יותר כדי לראות את הסימולציה (ולרבע סיבוב אמרנו לא לספור עד 10 לחצי צעד כלומר 5 לצעד שלם). וכן רואים דילאיי שמגיע מה'שעון הפולסים' שהגדרנו (שלא רואים כאן כיון שהוא 'חוט' פנימי' של המערכת).

עבור הפעלה רציפה:



עבור רבע סיבוב.



מצורף קישור לסרטונים שבהם אנחנו מראים שהמנוע עובד.

https://drive.google.com/drive/folders/1Y6A6OZUF1mH_5I1zCEP_fZWYV5OWf--Y?usp=sharing