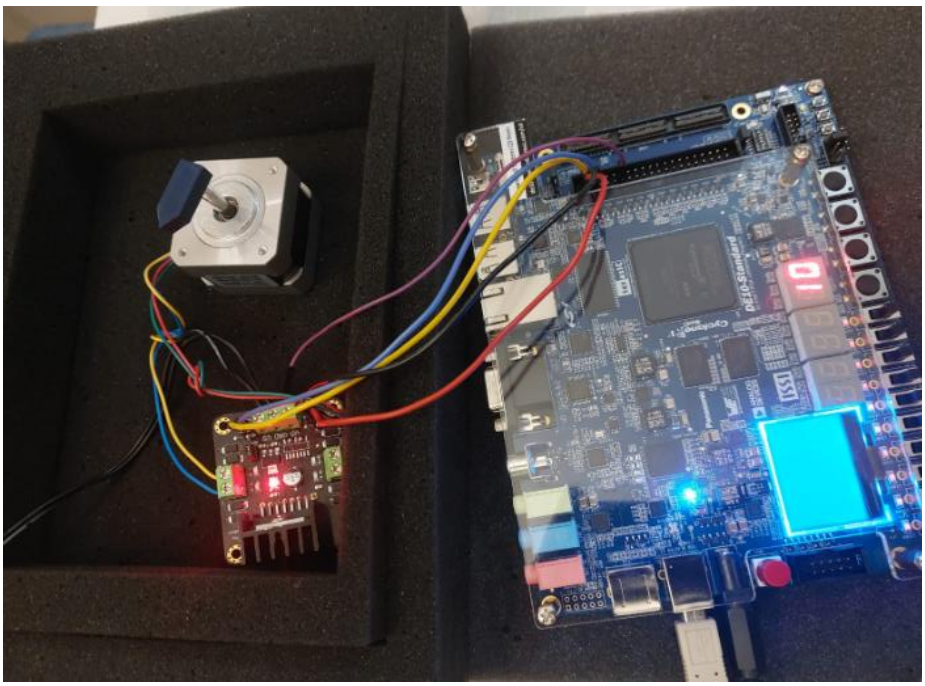
Lab #2 – מנוע צעד

עמיחי בלום ויצחק כהן

תוכן עניינים

תהליך ביצוע המעבדה 2
יתרונות וחסרונות התכנון 2
שני סוגי המנוע הנפוצים 2
Unipolar ו-Bipolar 2
המערכת 4
שרטוט דיאגרמת הבלוקים 4
הסברים על הבלוקים – וטיפה על המימוש שלהם בVerilog 4
המערכת מבחינת החומרה והסברים: 7
סימולציות 8
Direction, On, reset: 8
Half step and Full step: 9
Speed_sel: 9
quarter: 9
סרטון – קישור לדרייב, ופירוט על הקבצים 10
סרטון 10
שמות הקבצים ותפקידם 10



תהליך ביצוע המעבדה

1. בתחילת המעבדה בדקנו קודם כל מהו הרכיב המסופק – גילינו שהוא מנוע Bipolar, וראינו איך ניתן להפעיל אותו[1]. לעצמנו הגדרנו אילו אותות מגיעים מיציאות הGPIO שבFPGA לדרייבר, ומשם למנוע עצמו.
2. לאחר מכן פנינו לתכנון דיאגרמת הבלוקים הבסיסית (ללא הפונקציה רבע סיבוב). התחלנו מלתכנן את מכונת המצבים והתצוגה ב7seg שבכרטיס, המשכנו למחלק תדרים, משם היציאה נכנסה כenable למכונת המצבים שהעבירה תדרים למנוע.
3. לאחר מכן הוספנו מונה, שתפקידו הוא להדליק את הon_switch למשך 100 enables. בצורה זו, מכיוון שכל חצי צעד המנוע מתקדם 0.9° , כאשר נספור 100 צעדים, נגיע ל90 מעלות, שהוא כמובן רבע סיבוב.
4. בשלב זה היו לנו תקלות בפונקציה של רבע הסיבוב. ניסינו לסדר את זה ולראות איפה התקלה בעזרת הוספה של מעט יותר מ100 צעדים, ניסיון להוסיף flags למונה של רבע הסיבוב ולבדוק מה הפער וכו'. בסוף הבנו שהתקלה היא בגלל הדרישה של המעבדה להכניס 0 לכל הסלילים כשהמנוע לא עובד – דרישה שגורמת למנוע להיות כל פעם במצב אחר כשלוחצים שוב על רבע הסיבוב וזה גרם למעברים לא חוקיים בתוך מכונת המצבים – שפגעו בדיוק של 90 המעלות. לאחר הבנת הבעיה, הורדנו את הדרישה, ואכן נפתרה הבעיה.

יתרונות וחסרונות התכנון

- את המערכת ניסינו לבנות כך שכל בלוק יהיה בנוי עם פונקציונליות אחת מוגדרת, צורת הבנייה הזו, אפשרה לנו לעשות Debugging במקרה הצורך, בצורה נוחה. בנוסף, כתיבת בלוקים מסודרת לפני התחלת כתיבת הקוד, ממעטת כפל קוד כך שהקוד יהיה ברור וחסכוני.
- חריגה יחידה שעשינו הייתה שהוספנו בלוק פנימי בתוך מודול req_sm שתפקידו לcounter בשביל רבע סיבוב. למה עשינו את זה?
  - אמנם יותר מסובך לבדוק ולקרוא את הקוד ככה. אבל במימוש בצורה הזו יכולנו בקלות להגדיר למנוע - מנוע עובד או לא[2]. במחשבה שניה כנראה יכולנו לחבר הכל בקישוריות תחת הבלוק ״Top״ אבל כבר לא נגיע לזה.

שני סוגי המנוע הנפוצים

Unipolar ו-Bipolar

- <u>מבנה כללי של מנוע צעד</u>: ציר הסיבוב של המנוע נמצא בתוך גלגל שיניים (מתפקד כRotor) שמגיב לשדות האלקטרומגנטיים. מסביב לגלגל השניים יש לנו מספר סלילים (מתפקדים כStator) שבזמן העברת זרם חשמלי באחד מהם, נוצר שדה מגנטי[3] שגורם לגלגל השיניים להגיב לשדה המגנטי ולהתיישר לסליל בו עבר הזרם. בשונה ממנועים חשמליים רגילים אשר לרוב נעים בצורה סיבובית רצופה, כל "צעד" של מנוע צעדים הוא סיבוב של הציר

---

[1] הסבר מפורט על ההבדל בין מנוע ביפולרי לחד-פולרי בהמשך.
[2] השורה בקוד נראית ככה : motor_on = quarter_on || on_switch.
[3] לפי חוק ביו סבר, תנועת מטענים(זרם) יוצרת שדה מגנטי.

שלו בזווית קטנה מסוימת וקבועה[4]. עם זאת, המנגנון כולו יוצר בסופו של דבר תנועה, אשר יכולה להיות או סביב הציר (תנועה בזווית רצויה) או תנועה ישרה.

- <u>ההבדלים בין מנוע חד/דו-קוטבי</u>: מנוע **חד-קוטבי** בנוי בצורה הבאה:[5]

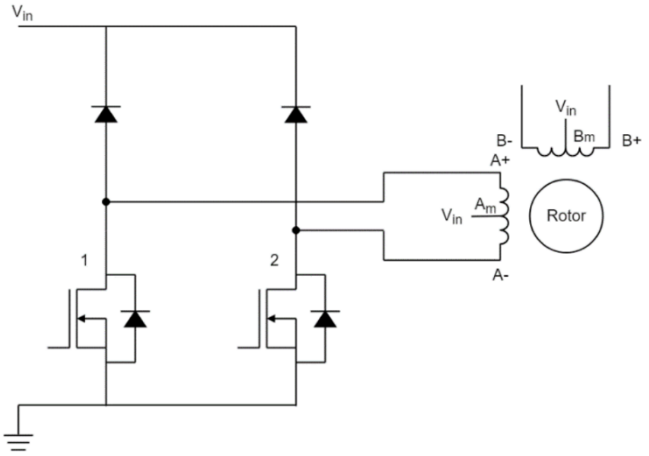


Figure 9: Unipolar Stepper Motor Driving Circuit

בהפעלת טרנזיסטורים 1 או 2 (בצד שמאל), אנחנו מפעילים את חצי הסליל Am-A+ או Am-A-, שגורמים לשדות מגנטיים הפוכים. מאוד קל לשלוט בצורה הזו בזרמים ומספיקים 2 טרנזיסטורים לשני חצאי הסליל. כתוצאה מכיוון הסליל כל פעם חצי מהסליל בכל פעולה, מה שמוביל למומנט כוח נמוך יותר ויעילות פחותה. המנוע מחובר בדרך כלל לשלושה חוטים לכל פאזה, ולעיתים קרובות הפאזות מחוברות פנימית, כך שיש למנוע רק חמישה חוטים. לעומת זאת, **מנוע דו-קוטבי** מפעיל את כל הסליל בכל פעולה, מה שמאפשר מומנט כוח גבוה יותר ויעילות רבה יותר. המנוע מחובר לשני חוטים לכל פאזה, ואין לו חוט משותף. המעגל המניע צריך להיות מורכב יותר כדי להפוך את הקוטביות של הסליל. הוא בנוי כך:

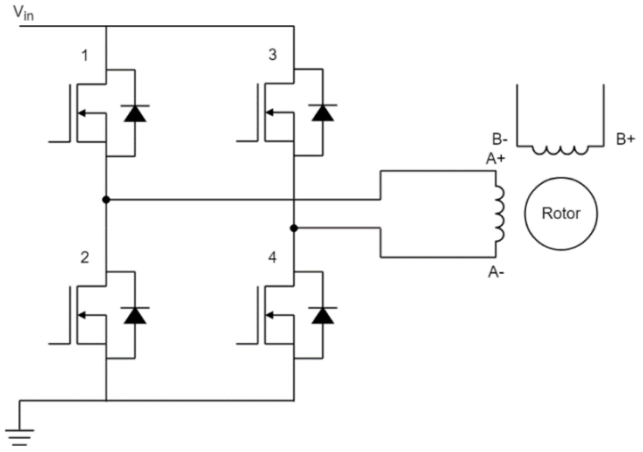


Figure 10: Bipolar Stepper Motor Driving Circuit

- <u>איך מייצרים צעד בודד לעומת חצי צעד</u>? צעד בודד בעזרת הדלקה כל פעם של סליל בודד, וחצי צעד נקבל בעזרת הדלקה של סליל בודד ולאחר מכן שני סלילים סמוכים, סליל בודד ושני סלילים סמוכים וכן הלאה, כך שנקבל את הזוויות בין צעדים שלמים.

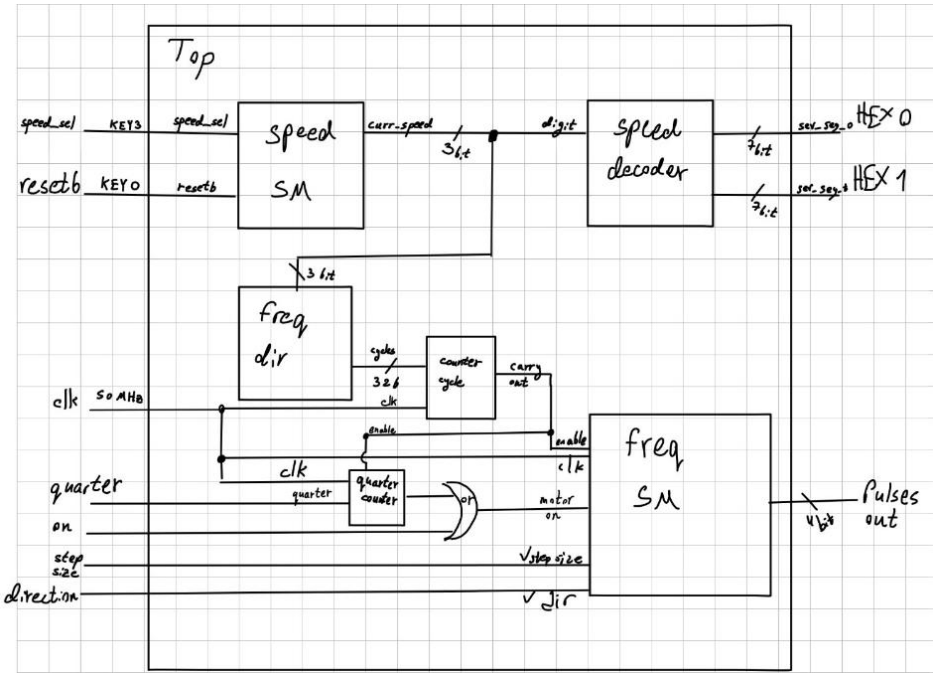
---

[4] במקרה שלנו, צעד שלם הוא 1.8 מעלות, וחצי צעד הוא 0.9 מעלות.

[5] השרטוטים של מנועי הצעד הם מהאתר https://www.monolithicpower.com/en/learning/resources/stepper-motors-basics-types-uses?srsltid=AfmBOop9McTHCIcckgr05vjeBVOknuz-7vRCtxqT5_wujt5Z4coD8ONk

המערכת

שרטוט דיאגרמת הבלוקים



הסברים על הבלוקים – וטיפה על המימוש שלהם בVerilog

Top *- הבלוק שמכיל את כל הקישוריות*[6].

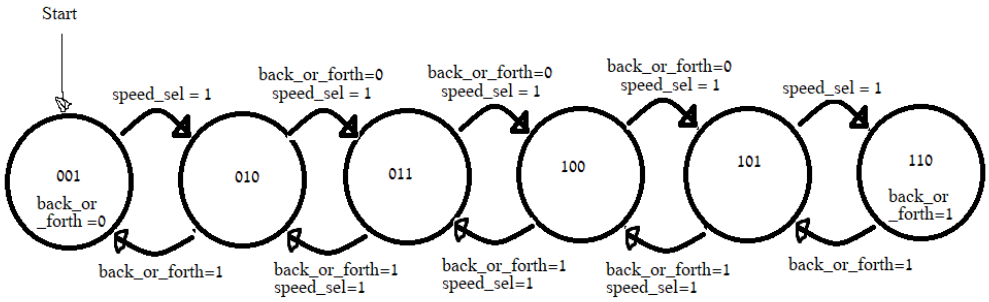
כניסות		**יציאות**	
Speed_Sel	בלחיצה על הכפתור הזה מתקדמת מכונת מצבים ממהירות של 10 סיבובים לדקה עד 60 בקפיצות של 10 וחזרה.	**Pulses _Out**	4 ביטים שמהווים את היציאות למנוע ('1' סליל דולק, '0' כבוי)
resetb	כפתור שמאתחל את המהירות ל10 סיבובים בדקה, ובזמן הלחיצה עליו מאפס את היציאה למנוע.	**HEX_0**	7segments – 7 נוריות מסודרות בצורת 8, קובעות את ספרת **האחדות** במהירות הנתונה
clk	כניסה של שעון מובנה (בתדר של 50MHz)	**HEX_1**	7segments – 7 נוריות מסודרות בצורת 8, קובעות את ספרת **העשרות** במהירות הנתונה
on	Switch שקובע האם להפעיל את המנוע או לא		
Direction	Switch שתפקידו לקבוע האם המנוע יסתובב בכיוון השעון או כנגדו.		
Step size	Switch נוסף שקובע האם כל "צעד" המנוע יתקדם 0.9 מעלות או 1.8 מעלות (חצי צעד/צעד שלם, בהתאמה).		
quarter	כפתור שבלחיצה עליו, במידה והמנוע כבוי, גורם למנוע לבצע רבע סיבוב		

[6] נקרא בקובץ "Lab2.v".

Speed_sm

בלוק זה הוא מכונת מצבים של המהירויות (10, ... ,60).

- <u>כניסות</u>: speed_sel (כפתור KEY3), ו resetb (לאיפוס המהירות)
- <u>יציאה</u>: curr_speed (מצב מבין 6 מצבים ולכן 3 ביטים ביציאה)
- <u>אופן הפעולה</u>: שמרנו 6 מצבים, עבור כל אחד מהמהירויות, ועוד ביט אחד שאומר האם אנחנו עולים במהירויות (מ10 עד 60) או יורדים (מ60 ל10). בצורה זו ידענו לאיזה מצב להתקדם.
- <u>דיאגרמה</u>:



- <u>הסבר הדיאגרמה</u>: מתחילים במצב הראשון (001). כל פעם שלוחצים על ה-speed-select מתקדמים מצב. מה שקובע אם חוזרים או מתקדמים זה הביט הנוסף שסימנו בback_or_forth. קדימה זה 0, אחורה זה 1. הביט עצמו משתנה רק כשמגיעים לסוף ההתקדמות/החזרה, כמו שניתן לראות בדיאגרמה.
  כאשר speed_select=0, כלומר לא לוחצים על הכפתור לשינוי המהירות, (חסר בדיאגרמה שנהייתה קצת צפופה) - כל מצב צריך להיות עם חץ לעצמו – כלומר נשארים באותה המהירות.

Speed_decoder

דיקודר שמטרתו להמיר את המצב הנוכחי של מכונת המצבים - לתצוגה על הספרות Hex0,1.

- <u>כניסה</u>: curr_speed (כנ״ל)
- <u>יציאות</u>: 7_seg_o,7_seg_t שמגיעות לHEX0,1 בהתאמה.
- <u>אופן פעולה</u>: כיוון שהקפיצות הן של 10 בין המצבים השונים, ספרת האחדות היא תמיד 0, ולכן היציאה של HEX_0 קבועה.
  מה שמשתנה זה כמובן העשרות, ולכן עשינו טבלה שממירה את המהירות הנוכחית לנורות הספציפיות שאמורות לדלוק בHEX_1 לפי המצב הנוכחי וזה הoutput.

Freq_div

בלוק שמטרתו לקבוע לcounter_cycle עד כמה cycles של השעון צריך לספור, לפי המהירות הנוכחית.

- <u>כניסות</u>: curr_speed (כנ״ל)
- <u>יציאות</u>: cycles (כמה מחזורי שעון בפועל יש לספור לקבלת המהירות).
- <u>אופן פעולה</u>:
  - עבור כל מהירות חישבנו (הסבר להלן) את מספר מחזורי השעון שיש לספור כדי להגיע למהירות הרצויה, בהינתן clk מובנה של 50MHz. את המספר הזה העברנו כoutput לCounter_cycle כמו שניתן לראות בדיאגרמה.

- <u>חישוב המהירויות</u>:
  - את כל המהירויות חישבנו רק עבור חצי צעד[7]. בdata_sheet של המנוע כתוב שחצי צעד המנוע מבצע רק 0.9° מעלות.
  - כדי לקבל סיבוב שלם אנחנו צריכים $\frac{360}{0.9} = 400$, כלומר,400 enable-ים להעביר למנוע.
  - אנחנו רוצים X סיבובים לדקה כלומר צריכים $400 * X$ enable-ים לדקה.
  - כשנעביר לסיבובים **לשניה** נחלק ב60 כלומר $\frac{400}{60} * X = 6\frac{2}{3} * X$ enable-ים לשניה.
  - יוצא שכדי לקבל X סיבובים לדקה, נחלק במספר הenable שחישבנו שצריך כלומר:

$$\frac{50{,}000{,}000\left[\frac{1}{sec}\right]}{6\frac{2}{3} * X\left[\frac{circles}{sec}\right]} = \frac{7{,}500{,}000}{X}$$

ולכן: עבור 10 סיבובים לדקה נקבל שיש לספור 750000 מחזורי שעון, עבור 20 סיבובים נקבל 375000 מחזורי שעון וכך הלאה.

Counter_cycle

מונה עד מספר הcycles שנקבע בבלוק Freq_div, ומוציא enable כoutput כשמגיע למספר ומתחיל לספור מחדש ללא עצירה.

- <u>כניסות</u>: cycles (כנ"ל), ו clk.
- <u>יציאות</u>: enable (מתפקד ברעיון כשעון, אלא כשעון יותר יציב אז משלבים).
- <u>אופן הפעולה</u>: זהו counter רגיל שמונה עד שמגיעים למספר מהFreq_divider. כאשר המונה מגיע למספר המבוקש הוא מוציא 1, ומתחיל לספור מחדש.

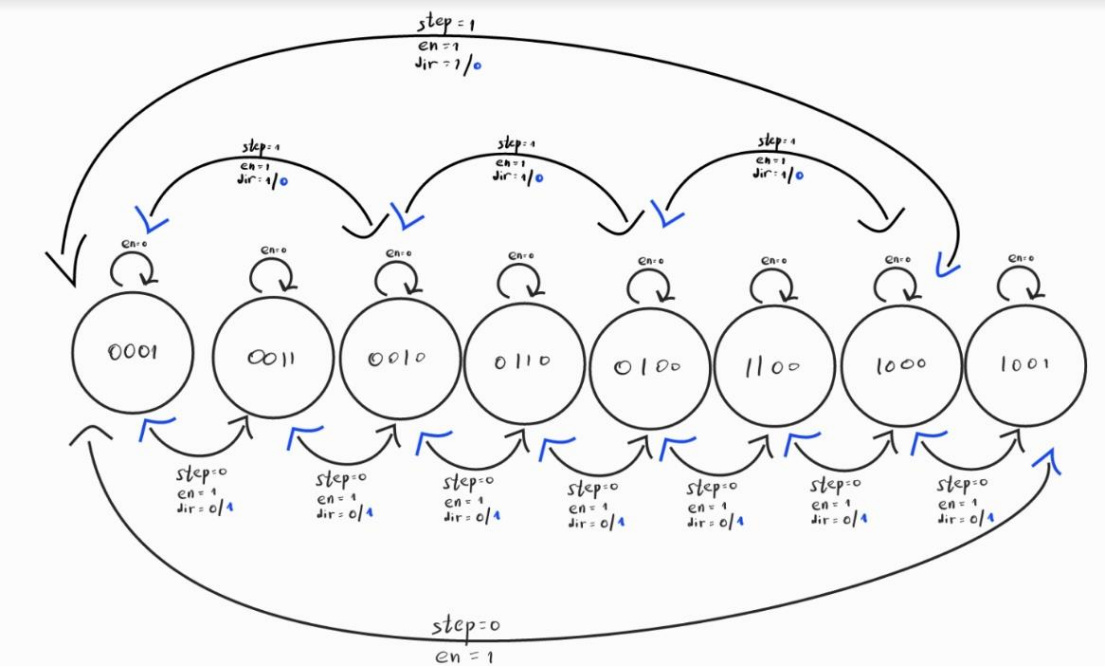
Freq_sm

בלוק זה מכיל מכונת מצבים שקובעת את האותות שיוצאים למנוע (ובעצם קובע דרך הדרייבר איזה סליל במנוע צעד יעבוד).
במכונת המצבים הזו המעברים תלויים בכיוון, צעד/חצי צעד.
לגבי המימוש של צעד/חצי צעד, השתמשנו באותה התדירות של חצי צעד גם לצעד שלם, ובגלל שבחצי צעד התדירות מהירה פי שניים, קבענו בצעד שלם שפעמיים נעביר אותו המצב, ולכן לא נצטרך להוסיף שינוי בתדירויות.

- **<u>Counter_quarter</u>** – בשביל החלק של הquarter הוספנו עוד counter_quarter – שנועד לוודא שהמנוע עובד למשך רבע סיבוב, ואז חוזר למצב הכבוי שלו. בעיקרון הוא נמצא בתוך freq_sm אבל הוצאתי אותו בדיאגרמה על מנת שיהיה ברור מה תפקידו.
  - הכניסות אליו: start_count (כניסה שמגיעה מהquarter), clk – שעון יציב, enable – אומר בפועל האם התקדם אות במנוע ולכן צריך לקדם את המונה.
  - יציאות: End_count – אומר מתי לכבות את המנוע בFreq_sm.
- <u>כניסות</u>: direction, step size, on, quarter, enable, clock

[7] וכשנרצה לעשות צעד שלם – נשלח פעמיים את אותו הפולס למנוע, אבל נפרט על זה בהמשך בבלוק "Freq_sm"

- <u>יציאות</u>: pulses_out
- דיאגרמה של מכונת המצבים:



- <u>הסבר על מכונת המצבים</u>:
  - ישנם 8 מצבים, שכל אחד מבטא איזה סליל יעבוד (סליל שמקבל 1 לוגי – יעבוד ו-0 יהיה כבוי).
  - במעבר ממצב למצב ישנם 3 קריטריונים: צעד/חצי צעד, כיוון, ו-enable.
  - אם ה- en כבוי, נשארים באותו המצב. אחרת – מתקדמים בהתאם לקריטריונים הבאים:
  - אם הכיוון (״dir״) הוא 0 או 1, מתקדמים קדימה או אחורה, בהתאם. (סימנתי חץ כחול לכיוון ההפוך).
  - אם step=0 אז עושים חצי צעד, כלומר מתייחסים לתיאור שמתחת למצבים (עוברים בין כל שמונת המצבים), ואם זה ׳1׳ אז קופצים בין 4 המצבים.

i. לשם השלמת התמונה נעיר שהיה צריך להוסיף עוד שלב ביניים בין כל שני מצבים בצעד שלם, כיוון שאמרנו שבצעד שלם כדי שהתדירות תהיה שווה, אנחנו פעמיים שולחים למנוע את אותו האות. לכן היה צריך עוד מצב זהה – עבור צעד שלם בלבד – שההבדל היחיד הוא קריטריון נוסף האם כבר שלחנו את האות הזהה פעם נוספת.

המערכת מבחינת החומרה והסברים:

- בחומרה אנחנו כמובן משתמשים באותו הרכיב FPGA שניתן לנו CycloneV.
- בכרטיס driver[8] שנועד להמיר אותות מהFPGA למנוע.
- במנוע צעד (mercury-motor-2-phase-SM42BYG011-25).
- השתמשנו ב4 חיבורים של נקבה-נקבה (להעברת האותות מהכרטיס לdriver).

[8] כרטיס הדרייבר הוא DFROBOT-DF-MD V1.3.

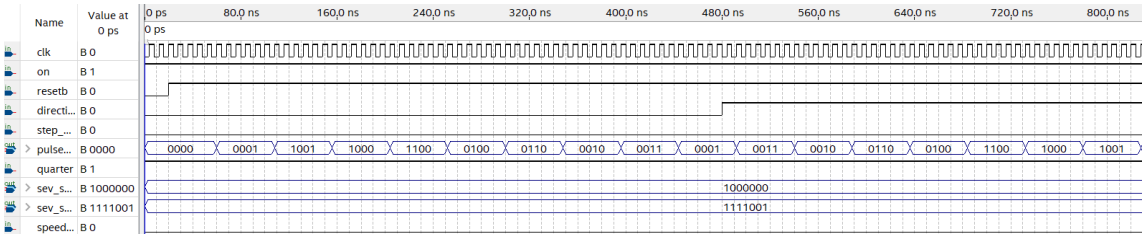
- חיבור אחד זכר-נקבה (על מנת לוודא שהאדמה בכרטיס ובדרייבר משותפת שלא ייווצרו פערים בנקודת ייחוס לאדמה – דבר שיכול להוביל להבדלים- אות בכרטיס יכול להיות '0', ובדרייבר יכול להיות '1').
- מגבלות על החומרה – על מנת לא לשרוף את המנוע שמרנו על מהירות מקסימלית של 60 סיבובים לדקה (מימשנו את הפונקציה של רבע סיבוב ולא של ולכן לא היינו צריכים לעלות מעל מהירות זו). בנוסף כל פעם שלא היה צורך ווידאנו בקוד שהסלילים מקבלים '0' למערכת. (וגם ניתקנו מהחשמל – מחוץ לפעילות).
- מבחינת הגנה על הdebouncing וקפיצות באות שנובעות מהחומרה – בכפתורים יש מנגנון מובנה ולכן לא היינו צריכים לממש. בswitchs לא הגענו לזה אבל ראינו שהמערכת עובדת חלק ולכן לא נזקקנו לזה.

סימולציות[9]

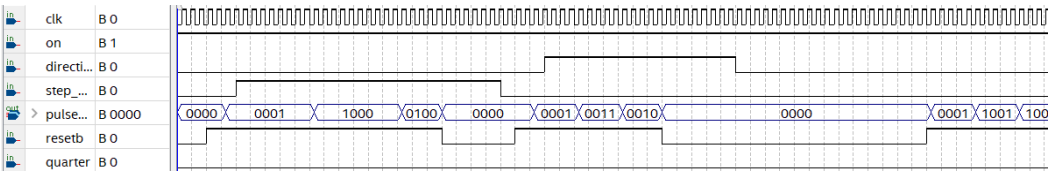
Direction, On, reset:

- בסימולציה הבאה המערכת במצב פעולה רגיל (ללא רבע סיבוב) במהירות הראשונה, ביחד עם החלפת כיוון באמצע. אפשר לראות איך המצבים מתקדמים מאחד לשני. בשורה של pulse אפשר לראות איזה פולסים מגיעים למנוע. בסימולציה הראשונה אנחנו ב"חצי צעד", לכן כל פעם משתנה רק ביט אחד במעבר ממצב למצב.

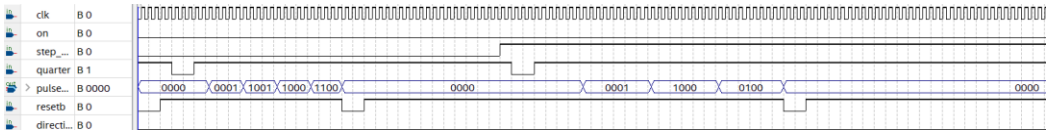
בנוסף, אפשר לראות את כיוון התקדמות המצבים ביחס לdirection – בהתחלה הוא 0 כלומר חוזר אחורה, ובהמשך הוא במצב '1' כלומר מתקדם קדימה:



- סימולציה של כפתור הreset בשני מצבים – בפעולה רגילה (בכיוונים שונים וצעד/חצי צעד), וסימולציה בזמן פעולת פונקציית רבע סיבוב:
  - במצב פעולה רגיל, on על '1', ולחצנו פעמיים על reset – והמנוע התאפס כמצופה:



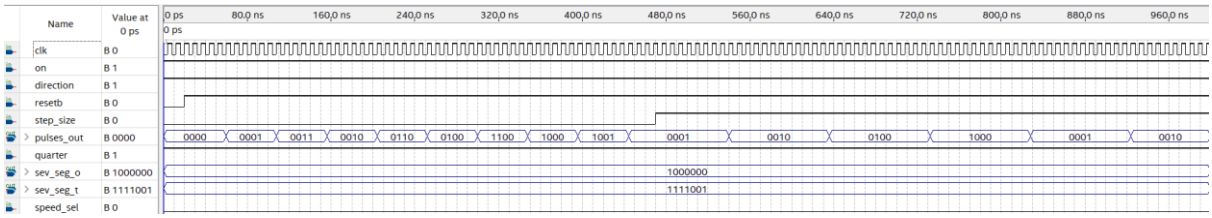
  - במצב פעולת רבע סיבוב, מצב on על '0'. reset מאפס את סלילי המנוע במקרים של צעד/חצי צעד (ולכן ההפרש באורכי האותות למנוע), כשאנחנו באמצע רבע סיבוב:



[9] לאורך כל הסימולציות שינינו את מספר הclk-ים שצריך לספור אחרת לא היינו רואים כלום. עשיתי את זה עבור המהירות הרגילה, במעבדה יואב ראה שהמספרים של הseg7 מתקדמים בצורה טובה כשמקדמים מהירות.

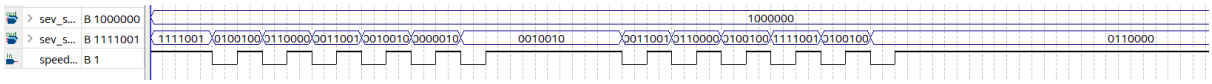
Half step and Full step:

- בתמונה הבאה אפשר לראות את ההבדל בין חצי צעד לצעד שלם. עבור חצי צעד אפשר לראות שהחצי צעד מתקדם במהירות פי 2, אבל הוא עושה את זה יותר לאט מבחינת מי שמסתכל מבחוץ, כך שהמהירות נשמרת:



Speed_sel:

- בתמונה הבאה אפשר לראות את קידום הseg-7. ספרת האחדות נשארת קבועה, כיוון שה-0 לא משתנה כמובן. מה שמשתנה זה העשרות מאחד עד שש(מקודד לפי הHEX):



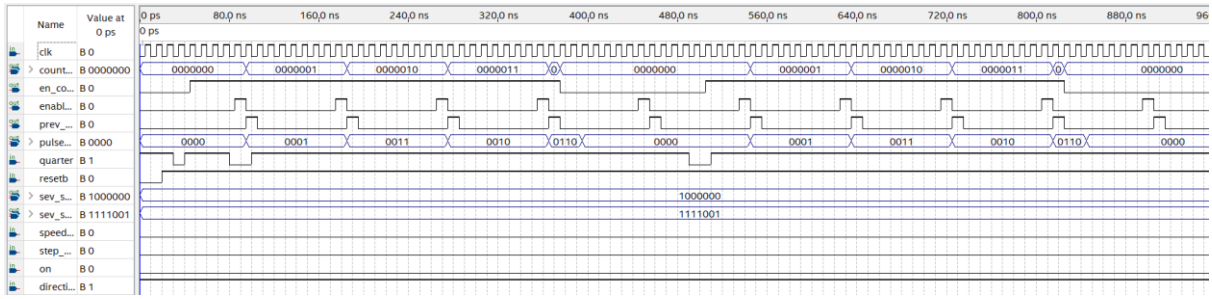
quarter:

- בסימולציה הבאה אפשר לראות שכשלוחצים על כפתור הquarter מתחילה ספירה (הראינו שאם לוחצים שוב – זה לא משנה את הספירה וממילא לא משנה את הפולסים שיוצאים למנוע).

השורה השניה – זה בשביל בדיקה של הcounter – אבל אפשר לראות שהוא סופר עד 4 (זה שהוא סופר רק חלק קטנטן זה לא משנה – העיקר הוא שכל הפולסים עוברים למנוע).

שורות 3-5 הן גם בשביל בדיקת הcounter.

שורת הpulse היא בעצם הפולסים שיוצאים למנוע – בשביל הבדיקה עשינו שרבע סיבוב יחשב שיעברו 4 פולסים. לאחר ארבעה פולסים ניתן לראות שה counter וה pulses מתאפסים.



סרטון – קישור לדרייב, ופירוט על הקבצים

סרטון

סרטון שמוכיח שהמנוע עובד כנדרש נמצא בקישור[10]

- https://drive.google.com/file/d/1_lo0D38MQgzepWxvUHLTL-IAGa5Ubluu/view?usp=drive_link

שמות הקבצים ותפקידם

- **Lab2.v** – הקובץ שמכיל את כל הקישוריות (קובץ הגג)
- **Speed_sm.v** – מכונת מצבים של המהירויות (10,…,60)
- **Speed_decoder.v** – דיקורדר לטובת התצוגה של הHex0,1.
- **Freq_div.v** – קובע לcounter_cycle לפי המהירות – עד כמה cycles של השעון צריך לספור, לפי המהירות הנוכחית.
- **Counter_cycle**.v – מונה עד מספר הcycles שנקבע, ומוציא enable כoutput כשמגיע למספר וממשיך לספור כל הזמן.
- **Freq_sm.v** – מכונת מצבים שמוציאה למנוע את האותות איזה סליל צריך לעבוד, בנוסף ללוגיקה מתי להפעיל את המנוע ואיך. בשביל החלק של הquarter הוספנו עוד counter_quarter – שנועד לוודא שהמנוע עובד למשך רבע סיבוב, ואז חוזר למצב הכבוי שלו.
- **Counter_quarter**.V – כתבתי לעיל מה השימוש שלו.

[10] אעיר שהייתה בעיה קטנה שניסיתי לפתור הרבה מאוד זמן - המנוע עושה כמעט רבע סיבוב מלא, אבל צובר חוסר בזווית קטנה. בדקתי את הלוגיקה עשרות פעמים, מימשתי ב2 דרכים שונות ועדיין התוצאה נשארה אותו הדבר. שאלתי את יואב ואחרים במעבדה ולא הצלחתי לסדר את זה. כשהגעתי למעבדה סידרתי את זה ויואב ראה שזה עובד כמו שצריך. מברוק!
הבעיה היתה בדרישה של המעבדה לאפס את המצב של הסלילים ובעקבות זה כל פעם המנוע התחיל ממצב אחר וזה גרם לרבע סיבוב להיות תקול..