



תוכן עניינים:

<u>3</u>	הרעיון של הפרויקט
<u>4</u>	רציוונל
<u>5</u>	סכמה אלקטرونית
<u>6</u>	רכיבי הפרויקט
<u>7</u>	Quartus Prime
<u>8</u>	VHDL, MODELSIM
<u>9</u>	Arduino IDE
<u>10</u>	C++
<u>11</u>	תאוריה נדרשת – רשתות
<u>12</u>	תאוריה נדרשת – DHCP
<u>13</u>	תאוריה נדרשת – OSI LAYERS
<u>14</u>	תאוריה נדרשת – אות אנלוגי
<u>17</u>	הרכיבים
<u>20</u>	*כיצד אותן פועל?
<u>24</u>	PWM
<u>26</u>	UART Communication and RS232
<u>27</u>	Code: DRONETOP
<u>32</u>	Code: RECIEVELOGIC
<u>34</u>	Code: PWM_SIG
<u>39</u>	Reversing the motors
<u>42</u>	Code: TimeoutManager
<u>43</u>	Code: ESP Controller
<u>47</u>	תנועה ברחפן
<u>49</u>	תקשורת
<u>55</u>	הdfsות תלת מימד – מבוא
<u>56</u>	הלייר הבניה
<u>62</u>	הלייר הבניה – הרכבת הרחפן
<u>65</u>	System on a chip (SoC)
<u>71</u>	מחשובות עתידיות
<u>75</u>	קשיים בפרויקט
<u>76</u>	רפלציה אישית
<u>77</u>	מקורות מידע



הרענון של הפרויקט:

תיאור של הפרויקט: בפרויקט, אני מתכוון בקר רחפן אשר יוכל לאוזן את עצמו בעזרת שימוש בתורת המטריצות.

השתמשתי בVHDL ובסביבת ארדואינו ובסימולציות שונות על מנת לבנות את קטעי הקוד שמאפיינים את הרחפן.

הכנתי תהליך FLOW שפועל עם מוקם כר שעובר מה-FPGA אוטות דיגיטליים שלפיהם נקבעת מהירות המנוע(RPM - סל"ד). באמצעות שימוש בWMW, כאשר אני משנה את רוחב הפולס (כלומר את duty cycle) כדי לתפעל את מהירות המנוע(/סל"ד).

אני משתמש בFPGA כבקור טישה - ד"א שולט על כל המנועים ומבצע את החישובים ההכרחיים לאייזון ולמטרות נוספות. הוא מחובר לבקר אלחוטי esp32 בשבייל שיוכל לתקשר עם ממשק אינטגרני לצורכי שליטה ברחפן(שליטה בכל מנוע בנפרד) ובקרה.

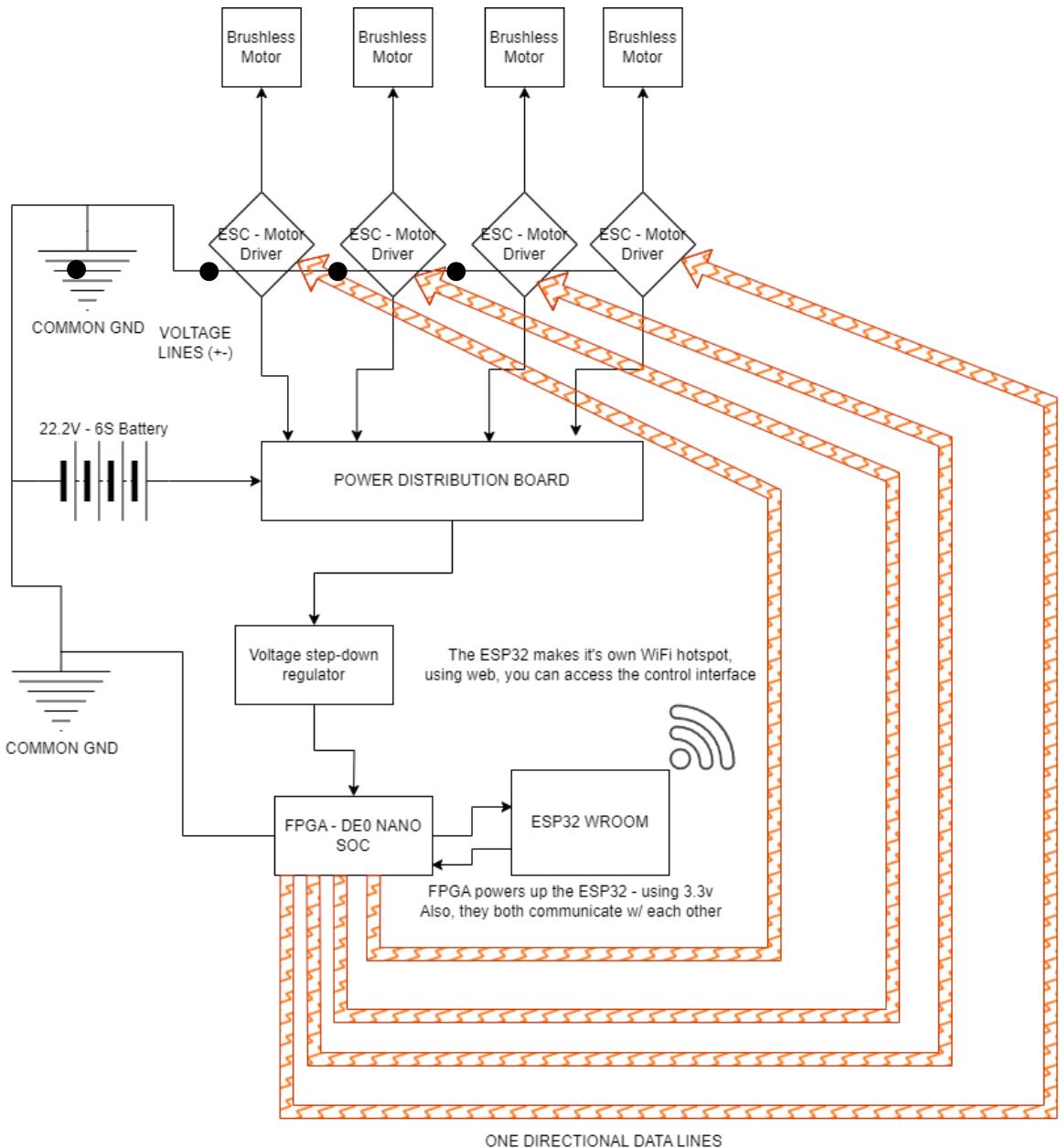
רצוֹנָל:

בחרתי בפרויקט זה מכיוון שאנו מאוד מסוקרנו בתחום הרחפנים, מימוש המתמטיקה בשימושים פיזיקליים (המרת מטריצות) ואופטימיזציה חומרית בהדפסות תלת ממד. בנוסף, המנחה שלי (רפוי) המליץ לי בחום על הנושא הזה מכיוון שציגתי בין נושאים ולקח לי הזמן זמן להיסגר סוף סוף על פרויקט, שכן קבוצות לפני ביצעו חלקים ספציפיים אך לא באמת הצליחו להעלות רחפן לאויר וכן גם לקיים תיקונים מתמטיים או שילוב של בינה מלאכותית.

הרצוֹןל בפרויקט זה הוא ביצוע של מס' דברים, חלק הצלחתי למשח וחלק אני עדין בתהליכיים (אני ממשיר את הפיתוח גם לאחר הבגרות ☺);

1. **תפעול שליטה מרוחק** - מבוצע על ידי **ESP32WROOM**, משק שליטה אינטראקטיבי
2. **העברת המידע בין ESP32 SOC לBIN DE0 NANO FGPA** - כעט מבוצע באמצעות **UART**, פרוטוקול **RS232**, תהליך הפיתוח מפורט בהמשך.
3. **קליטת המידע מממשק השליטה ותרגום לאותות PWM תואמים.**
4. **שליטה על כל מנוע בנפרד**, בסיס לתיקון ייצוב הרחפן כתגובה למשבי רוח חזקים – **בפיתוח**
5. **תיקון ייצוב הרחפן באמצעות טרנספורמציה מטריצות ומישוריים תלת ממדיים** – **בפיתוח**

סכמה אלקטרוניות:



כמו כן, ישנו ממשק השליטה האינטראקטיבי המתקשר באמצעות WIFI והוא קיים על כל מכשיר שמחובר לנקודת הגישה של ESP32.



רכיבי הפרויקט:

לוח FPGA - בקר הטיסה - **DE0 NANO SOC**
סוללה - **READYEDI 6S 22.2V 1400mAh** או כל סוללה 6תאים אחרת.

בקרים מהירות - **BLHELI 40A**

מנוע - **T-Motor Velox V3 2207 2050KV**

רגולטור - **LM2596**

בקר תקשורת - **ESP32 WROOM**

כמו כן לוחות **perf** ומלחם למען גימור סופי.

Quartus Prime

Intel Quartus Prime היא תוכנת תכנון לוגיקה ניתנת לתוכנות המיצרת על ידי אינטל. Quartus Prime מאפשר ניתוח וסינטזה של עיצובי HDL, מה שמאפשר למפתח להרכיב את העיצובים שלהם, לבדוק דיאגרמות RTL ולדמאות תגובת עיצוב לגירויים שונים. Quartus Prime כולל יישום של VHDL ו-Verilog לתיאור החומרה, ועריכה חזותית של מעגלים לוגיים.



VHDL

שפת תיאור חומרה המשמשת בתכנון מעגלים דיגיטליים ובתוכנות לוחות FPGA. זהה לשפה סטנדרטית המאפשרת למהנדסים ולמעצבים לתאר את התנהלות והמבנה של מערכות דיגיטליות ברמות שונות של הפשטה. את קוד VHDL אני כותבים בתוכנה בשם Quartus ובסביל לבדוק את התוכנית שכתבנו, אני מבצעים סימולציות בתוכנת ModelSim.

ModelSim

תוכנת MODELSIM היא כלי חשוב עבור מהנדסי אלקטרוני. תוכנה זו מאפשרת לבצע סימולציה של התנהלות מעגלים לוגיים ולבזק את התפקיד שלהם לפני הייצור הפיזי דרור שימוש בשפת תכנות ייעודית לתיאור חומרה כמו VHDL או Verilog. באמצעות כלי זה ניתן לתכנן ולבזק מעגלים מורכבים ולפוזד שאון בהם תקלות וכל זאת בסביבה יירוטואלית המאפשרת חישכון בזמן ובמשאים לפני יצוא המוצר הסופי. התוכנה מעניקה למשתמשים את היכולת לראות בזמן אמת איך המודול מתנהג ומאפשרת לבצע תיקונים ושיפורים בקלות.



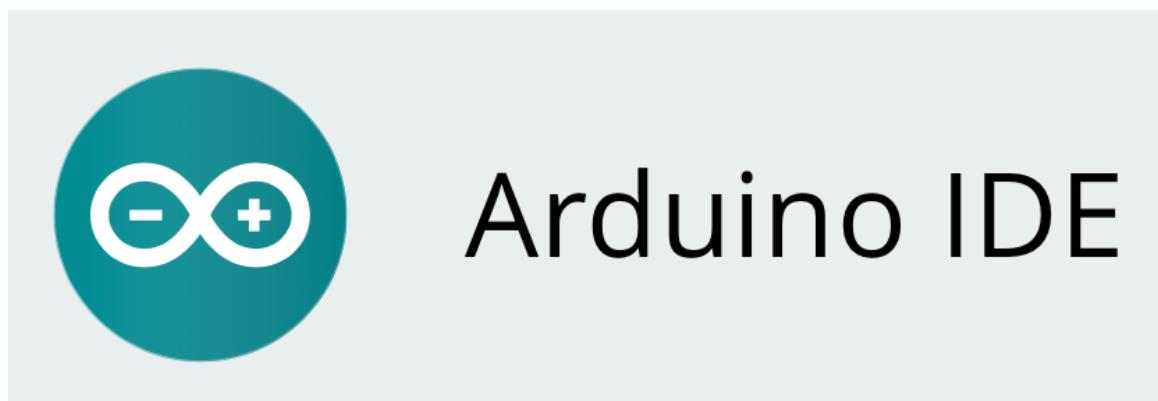
Arduino IDE

Arduino IDE (Integrated Development Environment) הוא סביבת פיתוח משלבת המיעדת לתוכנות ופיתוח פרויקטים מבוססי סולינויד. המטרה המרכזית של Arduino IDE היא לספק למשתמשים כלים פשוטים וידידותיים לפיתוח תוכנה עבור מיקרו-בקרים של סולינויד.

מטרת הסביבה היא לאפשר סביבת פיתוח ועבודה על מיקרו בקרים שכן גם כוללת בתוכה קומפайлר (מַהְדֵר) ותוכנת צריבה (flasher) מתאימה. הסביבה מאפשרת פיתוח על כל רכיב ארדואינו ובנוסף גם על רכיבים פופולריים אחרים – esp8266, esp32, lilyPad וcdcמה.

באמצעות סביבת הפיתוח המתקדמת ניתן להוריד כל מיני ספריות יעילות לניהול אפקטיבי של רכיבים. בפרויקט של הורדתי ספרייה אחת והוא למען ניהול של הקס, הידור וצריבה לכרטיס. כמו כן קיים מסך Serial monitor שמאפשר כדי לניטור נוח של הקוד, כמו כן גישה ישירה לתקשורת UART עם רכיבים שונים.

לרוב משתמשים בסביבה זו בשפת C++ . אך קיימת אפשרות להשתמש במַהְדֵר מיוחד שנבנה לשפת C, וכן גם לשפת פיתון (Python). אך מעטים האנשים משתמשים בשפות אלו למטרות פיתוח על הכרטיסים.



C++

שפת C++ היא שפת תכנות אובייקטיבית וקרובה לשפת C. השפה מציעה כלים ואפשרויות מתקדמות לתכנות מודולרי ומורכב.

בעולם של ESP, Arduino, C++ משמשת כשפה התכנות הראשית. רוב הקוד שנכתב לマイקו-בקרים אלו מתווכח בשפה זו. בזכות תמייה מובנית ב-C++, והוספת ספריות ותיקונים מתקדמים, ניתן ליצור פרויקטים מורכבים ומתקדמים במיוחד עם שפת CPP והマイקו-בקרים האלו. שפת C++ מאפשרת כתיבה נוחה של קוד, התmeshקota עם החומרה של הרכטיים ושימוש קל בספריות הקיימות המתמחות בפונקציונליות שונה כמו תקשורת רשת, חישנים, וממשקים חיצוניים.

כל התוכנות האלואפשרים לפתח פרויקטים רביעוצה, כולל מערכות מורכבות של OS וAPPLICATIONS תוכנה משולבת.

תאוריה נדרשת

רשתות

רשתות הן תשתיות טכנולוגיות המאפשרות למכשירים להתחבר וلتתקשר זה עם זה באמצעות חיבורים פיזיים או אלחוטיים. הרשתות מאפשרות שיתוף מידע, משא ומתן של נתונים, ושימוש בשירותים שונים, כגון גישה באינטרנט, שיתוף קבצים, או הודעות אימיל.

ישנן מגוון רחב של סוגים של רשתות:

1. **רשתות מקומיות (LAN)**: רשתות שנוצרות בתחום גוף המוסד או בתחום מבנה מסוים, כמו בית, משרד, או ארגון. הן מאפשרות שיתוף קבצים, מדפסות, ושירותי רשת בין מכשירים שונים בתחום הרשת.

2. **רשתות מרוחק (WAN)**: רשתות שמקשרות בין תשתיות נפרדות למרחבים גדולים יותר, כגון רשת האינטרנט. הן מאפשרות תקשורת בין משתמשים מרוחק, גישה לשירותים מרוחק, והעברת נתונים בין אתרים שונים ברחבי העולם.

3. **רשתות אלחוטיות (Wi-Fi, Bluetooth)**: רשתות שמאפשרות תקשורת באמצעות גלי רדיו, ללא צורך בחיבור פיזי. הן משמשות ברוב המכשירים הניידים ובצד המתקדם, ומאפשרות תקשורת גם בין מכשירים שונים בסביבה מקומית.

4. **רשתות קוויות (Wired Networks)**: רשתות שבןן התקשרות מתבצעת באמצעות כבילים פיזיים, כמו כבלי Ethernet או כבלי טלפון. הן נפוצות בסביבות מוסדות עסקיים ודירות.

רשתות מהוות יסוד חשוב בתשתיות המידע והתקשורת בעולם המודרני, ומספקות את היסודות לתקשורת ולשיתוף מידע בין מכשירים שונים באופן יעיל ומהיר.

בפרויקט זה יש שימוש ברשתות LAN (בתוכו נק' הגישה של ESP) ורשתות אלחוטיות (נק' הגישה עצמה)

DHCP

DHCP הוא ראשי תיבות של "Dynamic Host Configuration Protocol". זהו פרוטוקול תקשורת המשמש ברשות מחשבים להפצת כתובות IP והגדירות רשות אחרות אוטומטית למכשורים ברשות. בעזרת DHCP, מכשורים ברשות יכולים לקבל אוטומטית כתובות IP תקף, שעת השירות, שם התחום (DNS), שעת הגישה לרשות, והגדירות אחרות, מה שמקל על ניהול הרשות ומחית את הצורך בתפקיד ידני. תהליך זה מתבצע דרך כמה שלבים –

1. DHCP Discover (חיפוש):

- המכשור שמחפש כתובות IP ברשות שולח הודעה DHCP Discover באמצעות כתובות ה- MAC שלו.
- ההודעה משודרת כשיידור מופץ (broadcast) ונשלחת לכל המכשורים ברשות.

2. DHCP Offer (הצעה):

- השירות DHCP שומע את הבקשה ושולח הודעה DHCP Offer עם כתובות IP זמינה.
- ההודעה מתארת את הכתובת IP המוצעת ואת ההגדירות הנוספות של הרשות.

3. DHCP Request (בקשה):

- המכשור המבקש מקבל את ההצעה ושולח הודעה DHCP Request עם הכתובת IP שהוזעה.
- ההודעה מסמנת שהמכשור מבקש לשימוש בכתובת IP המוצעת.

4. DHCP Acknowledge (אישור):

- השירות DHCP מקבל את הבקשה ושולח הודעה DHCP Acknowledge כשאישר את השימוש בכתובת IP.
 - המכשור שקיבל את האישור מתחילה לשימוש בכתובת IP ובהגדרות הרשות שנשלחו על ידי השירות DHCP.
- כדוגמה – כאשר מכשור מתחבר לנקודת הגישה של הרחפן, כל התהליך הזה מתבצע.

OSI MODEL

שכבות ISO הן דרך לתאר את המבנה התיאורטי של התקשרות ברשות מחשבים. הן מסייעות בהבנת התקשרות בין מכשירים בראשת, וספקות מודל להבנת הפעולה של כל שכבה בראשת. הנה השכבות המרכזיות במודל OSI:

שכבה יישום (Application Layer):

זו השכבה העליונה ביותר שבמודל ISO. מספקת פרוטוקולים לאפליקציות לתקשורת, כמו HTTP לדפדפניים או SMTP לאימייל. פועלת ברמה המופשטת ביותר ומטפלת בפרטיות התקשרות הספציפית לאפליקציה.

שכבת העברה (Transport Layer):

אחראית על המיקומות הסופיים בין מקור ויעד, ניהול החיבורים והשליחה וקבלת הנתונים בין המכשירים. מבצעת שימוש בפרוטוקולים כמו TCP ו-UDP.

שכבת הרשת (Network Layer):

מספקת שימוש בכתובות IP וניהול הפקודות (תעבורה) בראשת. פועלת לפי נתוני מהשכבה הקודמת כדי לקבוע את המסלולים ולהוביל את החבילות ליעדן.

שכבת העצמת הרשת (Data Link Layer):

אחראית על שליטה בתקשורת בין מכשירים ישירים בראשת פיזית, ד"א מקבלת מידע משכבת הרשת ועוטפת את הפקודות (encapsulate) על מנת שיוכלו לעבור בשכבה הפיזית. כוללת פרוטוקולים כמו Ethernet או Wi-Fi.

שכבת הפיזית (Physical Layer):

השכבה התחתונה ביותר שבמודל ISO. אחראית על השליטה באופן הפיזי בחיבורו הרשת, כולל כבלים, אוטות וזרמים חמימים.

אות אנלוגי

אות אנלוגי הוא אות המשדר מידע בצורה רציפה בזמן, בניגוד לאות דיגיטלי שהוא שידור ביטים בודדים.

כיצד ADC עובד

ADC הוא תהליך שמקבל קלט של אות אנלוגי על גבי זמן – ומיצא אותו למידע קרייא, ביטים בודדים אשר מתבצע בשלבים הבאים:

1. דוגמה :

האות האנלוגי נמדד בנקודות זמן מסוימות וקבועות מראש. תדרות הדגימה היא מספר הדגימות לשניה.

2. החזק ודוגמם :

בשלב זה, מתבצע "הקפה" של הערך האנלוגי בנקודות הדגימה כדי שהאות יישאר קבוע בזמן קצר בזמן שהממיר מבצע את תהליכי הקואנטיזציה.

3. קואנטיזציה :

כל ערך מוגדר מומר לערך הקרוב ביותר בין קבוצה מוגדרת של ערכים בדים. בעצם מתבצעת נורמליזציה של הערכים.

4. קידוד :

הערכים הבדים מוקודים לרצף של ביטים. לדוגמה, אם משתמשים בקידוד של 8 סיביות, כל דגימה מיוצגת כערך מספרי בין 0 ל-255 (2 בחזקת 8 פחות 1).

כדוגמה – אות אנלוגי שמתנדנד בין 0 וולט ל 5 וולט שנרצה ליצא כ3 ביטים ->

1. נדגום כל כמות זמן מסוימת, לצורך העניין 1ms.

2. נחזיק את הדגימה שלקחנו ונבצע את שלב 3

3. אם מתח הדגימה הוא 2.7 וולט מתוך 5, מתווך 8 ערכים אפשריים (2 בחזקת 3) שהם:
0, 0.625, 1.25, 1.875, 2.5, 3.125, 3.75, 4.375, 5 וולט. ולכן 2.7 וולט יועגל לערך 2.50.

4. ע"פ ספירה בינארית נתחיל:

0 וולט (000 בינארי)

0.625 וולט (001 בינארי)

- 1.25 וולט (010 בビינארי)**
- 1.875 וולט (011 בビינארי)**
- 2.5 וולט (100 בビינארי)**
- 3.125 וולט (101 בビינארי)**
- 3.75 וולט (110 בビינארי)**
- 4.375 וולט (111 בビינארי)**

לכן התוצאה תהיה 100 בסיס 2. ז"א 4 בסיס דצימלי.

כיצד DAC עובד

אם ADC הוא תהליך שמקבל קלט של אות אנלוגי על גבי זמן – ומיצא אותו למידע קרייא, ביטאים בודדים אז DAC הוא בדיקת ההפך. מקבל קלט של ביטאים בודדים ומיצא לאור זמןאות אנלוגי. אשר מתבצע בשלבים הבאים

1. קבלת קלט דיגיטלי המייצג מספר מסוימים.
2. מיפוי מתחים – רצף הביטאים מומר לערך מתח אנלוגי בתוך הטווח הנוכחי.
3. חילקה – ביטול רעשים באמצעות פילטרים מתמטיים.

כדוגמה – אות דיגיטלי כמו 5 ⇔ מטורגם ל101 בביטים->

1. נכניס **DAC** את האות הבא '101'
2. נכניס **DAC** את הטווח מתח הרצוי – לדוגמה בין 0 ל5 וולט
3. נחשב את מדרגת הרזרזציה שהיא המתח המקסימלי חלקו 2 בחזקת מס' הביטאים.
 0.625 מדרגת רזרזציה של 0.625 וולט.

עת נחשב את הערך האנלוגי בvolt באמצעות הכפלת מדרגת הרזרזציה בערך הדצימלי של האות הנוכחי. $0.625 \cdot 5 = 3.125$ וולט.

לאחר מכון במידת הצורך למשה(**apply to**) פילטרים מתמטיים על מנת להחליק את תוצאת המתח האנלוגי, ככל שהמתח האנלוגי יורד לדרגות נמוכות יותר(מילי וולט, מיקרו וולט וכו) כך הוא רגייש יותר להפרעות, כדוגמת גלים אלקטромגנטיים ואפלו תזוזה פיזית של הקבלים המחברים בין **DAC** לבין הרכיב הצורך את המידע.

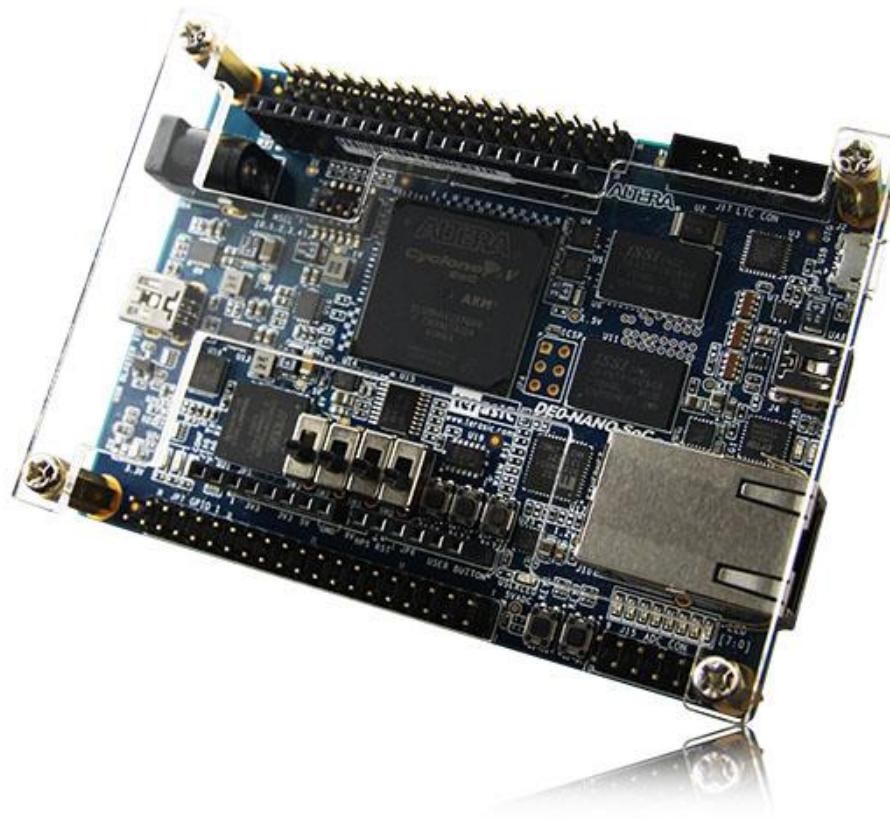
הרכיבים

FPGA (Field-Programmable Gate Array)

מעגל משולב (צ'יפ) שנitin לתוכנת ולהגדיר על ידי משתמשים לאחר הייצור. זהו מכשיר לוגיקה דיגיטלי וגייש שנitin להתאים אותו לביצוע **משימות או פונקציות** **שונות**.

שלא כמו מעגלים משלבים המיעודים ליישומים ספציפיים (ASICs), ניתן לתוכנת חדש או להגדיר מחדש רכיבי FPGA כדי לישם מעגלים דיגיטליים שונים. תוכנות FPGA כרוכ בטיור המעגל הלוגי הרצוי באמצעות שפת תיאור חומרה (HDL) כמו VHDL (בה משתמש בפרויקט).

אני משתמש בפרויקט ברכיב SOC NANO DE0 של חברת Terasic עם FPGA בעל מחולל שעון בתדרות של 50 MHz ועם כניסה מתח DC של 5V.



אוסף גם תמונות עד שלקחי מתוכן DataSheet של הכרטיס שלו. הם עזרו לי מאוד במהלך הפיתוח מבחינת החיווט והמידע על הכרטיס.

Table 3-5 Pin Assignment of Clock Inputs

Signal Name	FPGA Pin No.	Description	I/O Standard
FPGA_CLK1_50	PIN_V11	50 MHz clock input	3.3V
FPGA_CLK2_50	PIN_Y13	50 MHz clock input	3.3V
FPGA_CLK3_50	PIN_E11	50 MHz clock input (share with FPGA_CLK1_50)	3.3V
HPS_CLK1_25	PIN_E20	25 MHz clock input	3.3V
HPS_CLK2_25	PIN_D20	25 MHz clock input	3.3V

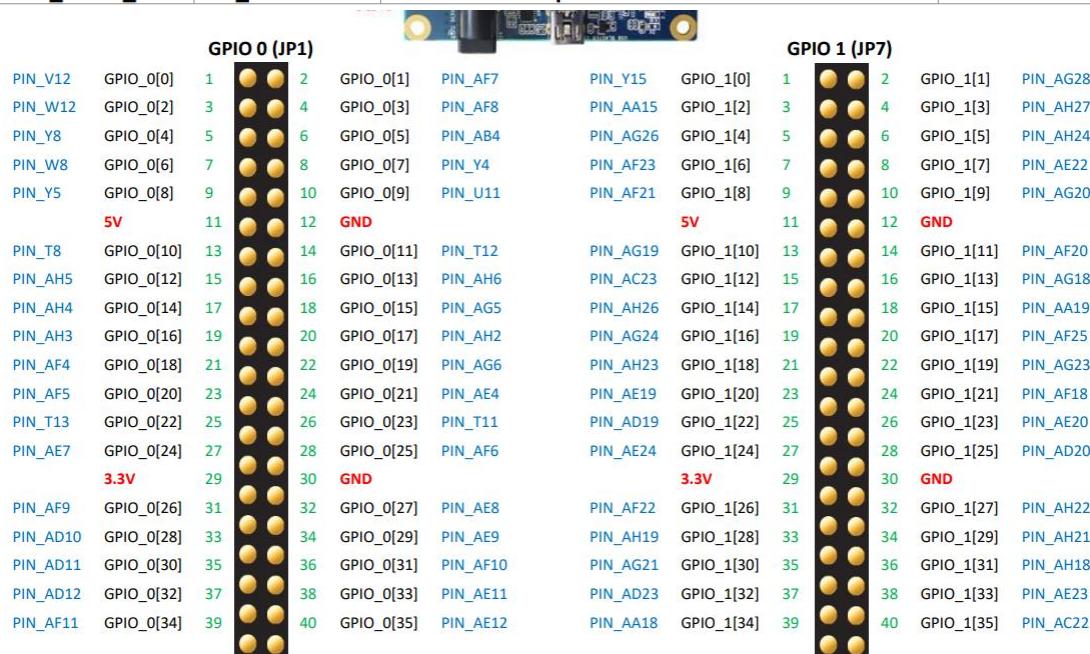


TABLE 3. Input Channel Selection

ADD2	ADD1	ADD0	Input Channel
0	0	0	IN0 (Default)
0	0	1	IN1
0	1	0	IN2
0	1	1	IN3
1	0	0	IN4
1	0	1	IN5
1	1	0	IN6
1	1	1	IN7

Table 3-6 Pin Assignment of Slide Switches

Signal Name	FPGA Pin No.	Description	I/O Standard
SW[0]	PIN_L10	Slide Switch[0]	3.3V
SW[1]	PIN_L9	Slide Switch[1]	3.3V
SW[2]	PIN_H6	Slide Switch[2]	3.3V
SW[3]	PIN_H5	Slide Switch[3]	3.3V

Table 3-7 Pin Assignment of Push-buttons

Signal Name	FPGA Pin No.	Description	I/O Standard
KEY[0]	PIN_AH17	Push-button[0]	3.3V
KEY[1]	PIN_AH16	Push-button[1]	3.3V

Table 3-8 Pin Assignment of LEDs

Signal Name	FPGA Pin No.	Description	I/O Standard
LED[0]	PIN_W15	LED [0]	3.3V
LED[1]	PIN_AA24	LED [1]	3.3V
LED[2]	PIN_V16	LED [2]	3.3V
LED[3]	PIN_V15	LED [3]	3.3V
LED[4]	PIN_AF26	LED [4]	3.3V
LED[5]	PIN_AE26	LED [5]	3.3V
LED[6]	PIN_Y16	LED [6]	3.3V
LED[7]	PIN_AA23	LED [7]	3.3V

Table 3-12 Pin Assignment of ADC

Signal Name	FPGA Pin No.	Description	I/O Standard
ADC_CONVST	PIN_U9	Conversion Start	3.3V
ADC_SCK	PIN_V10	Serial Data Clock	3.3V
ADC_SDI	PIN_AC4	Serial Data Input (FPGA to ADC)	3.3V
ADC_SDO	PIN_AD4	Serial Data Out (ADC to FPGA)	3.3V

APPLICATIONS INFORMATION

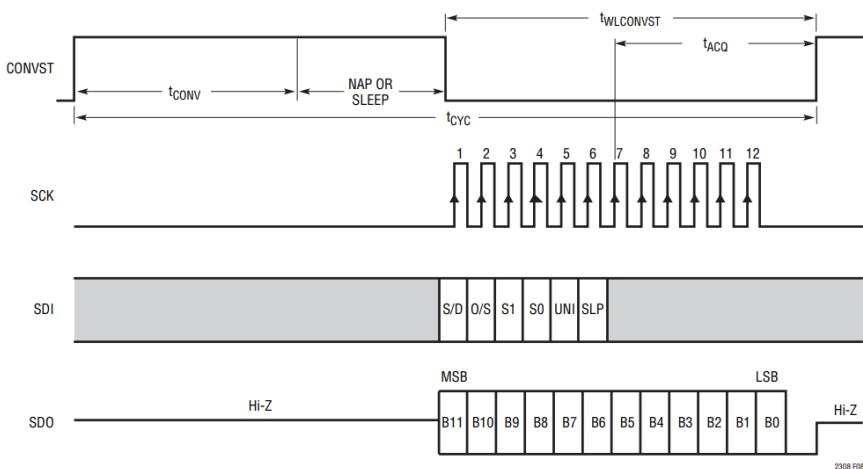
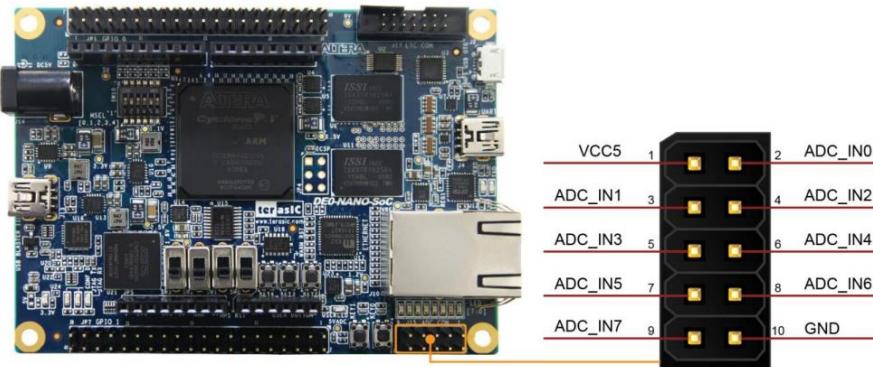


Figure 8. LTC2308 Timing with a Long CONVST Pulse

The LTC2308 is a low noise, 500ksps, 8-channel, 12-bit ADC with a SPI/MICROWIRE compatible serial interface. This ADC includes an internal reference and a fully differential sample-and-hold circuit to reduce common mode noise. The internal conversion clock allows the external serial output data clock (SCK) to operate at any frequency up to 40MHz.

It can be configured to accept eight input signals at inputs ADC_IN0 through ADC_IN7. These eight input signals are connected to a 2x5 header, as shown in [Figure 3-20](#).



כיצד השעון בכרטיס עופד?

בכרטיס אין באמת שעון אמיתי. אי אפשר להכניס שעון שלם לתוך הכרטיס FPGA, גם לא לתוך לוח האם במחשב שלכם או אפילו בטלפון... איז איר כל הקסם מתרחש?

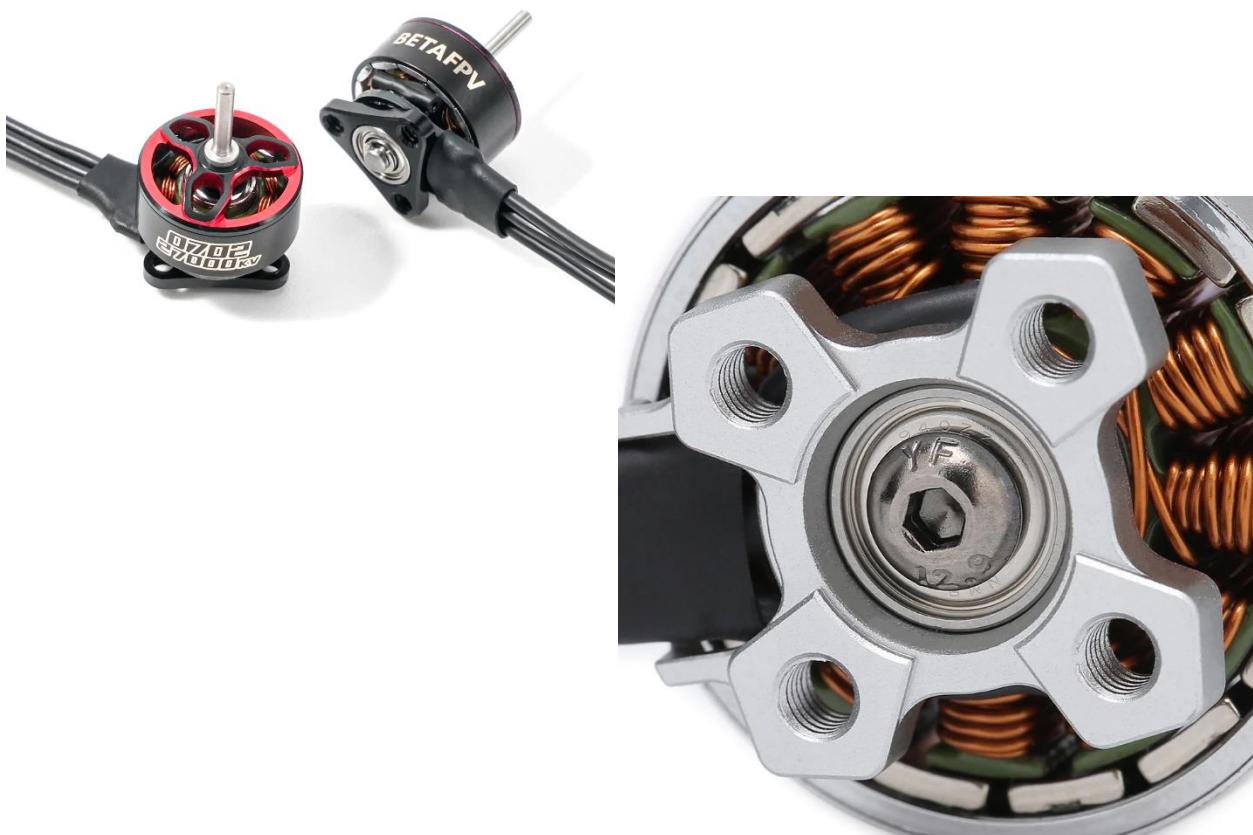
קייםת קבוצת רכיבים בשם "מתנד גבישי", תפקידם הוא להיות מתנד חשמלי – אלה רכיבים שמתבססים על גבישים עם תוכנות פיאזו-אלקטניות או פיזיו-אלקטניות (תוכנה דוא כיווניות שפועלת כך – שינוי בלחץ המכני יגרום לשינוי במתוח החשמלי על הרכיב וההפך – שינוי במתוח החשמלי יגרום לשינוי בלחץ המכני). ורכיבים נוספים כמו מגבר שרף.

היות והתוכנה הינה דוגמאות לכך שרכיבים אלה ביכולות מואז יכולים להיכנס לתנועה הרמוניית בה הם מייצרים "פעימות" נורא מדוקיקות -> תדר! ובאמצעות מתנדים אלו יכולים לעקוב אחרי זמן שעובר, להיות ואנו יודעים כמה פעימות יצאו מן המתנד בפרק זמן מסוים.

כדוגמה: בכרטיס FPGA בו אני משתמש, קיים מתנד גבישי שמיצור מקווארץ שפועל בתדר של 50MHz. ד"א 50 מיליון פעימות בשנייה אחת -> מפה אנו יכולים לבצע פעולות מתמטיות להשגת כל זמן רצוי אך ורק באמצעות מס' פעימות מתנד. חשוב לציין שיש מתנדים שפועלים בתדרים שונים ולכן יש לבדוק את סוג המתנד בכל מערכת בה נרצה לפעול.

המנועים שבhem אני משתמשים:

אני משתמש במנועים מסוג **brushless** שפועלים על זרם ישיר DC. המנוע מורכב משני רכיבים מרכזיים - מגנטים וסלילי מתכת. המתח החשמלי נשלח בכל פעם לסליל אחר במנוע ובכך נוצרALKTRONIK המושך או דוחה את המגנטיים המקיים אותו. ככל מר, כאשר מגיע המגנטי המקיים את סלילי המתכת לאזור מסוים במנוע, מופעלים שני סלילים הצמודים לו במתח מנוגד כך שאחד מושך אותו והשני דוחה אותו. לרוב, יהיה יותר מגנט אחד במנוע ולכן תהליך זה יתבצע בצורה נפרדת עבור כל מגנט. בדומה זו נוצרת תנועה סיבובים רצופה ובעל כוח קבוע. במנועי **brushed** (עם פחמים) יש חיכוך במנוע שהפחמים מייצרים, מה שעלול לגרום לנפילות מתח, לכן נבחרו מנועי **brushless**.



Driver

ברחפן יהיו 4 מסוג drivers - esc - electronic speed controller וכל אחד מהם יהיה מחובר למנוע אחד, FPGA ולוח פיזור המתח (power distribution board).

ESC או בקר מהירות אלקטרוני, זהו מכשיר אלקטרוני המשמש לשילטה במהירות ובכיוון של מנוע חשמלי. התפקיד העיקרי של ESC הוא להמיר את אותות הבקרה הנכנים אליו לאותות חשמליים מתאימים להנעת המנוע. הוא לוקח אותות כניסה מקורה בקרה (במקרה שלנו זה רכיב FPGA) ומתאים את תפקת הכוח של המנוע בהתאם. זה מאפשר שליטה מדויקת על המהירות, התאוצה והכיוון של המנוע. תפקיד נוסף של esc הוא להגביר את הזרם שמקבל מפניה מהרכיב FPGA לא מייצר זרם מספק למנועים.

Regulator

רגולטור הוא מכשיר או מעגל שומר על מתח או זרם פלט יציב למגוון שימושים אחד מהנכיסת שלו יכול להשתנות. זה מבטיח שהכוח המסופק לרכיבים האלקטרוניים נשאר בטוח מוגדר, מגן עליהם ומבטיח פעולה תקינה.

אני בחרתי לשימוש בזוסטט regulator ולא בנגד להורדת המתח מכיוון שרגולטור משתמש בمعالגים מתקדמים וטכנולוגיה לתכנון מדויק ויעיל של מתח וכך, יש לו רמת נצילות גבוהה. בנויגוד לכך, נגד משתמש בפילטרים ושיטות אחרות המביאות לאובדן אנרגיה ולהתרפשות חום במערכת וכך, יש לו רמת נצילות נמוכה. על ידי שימוש ברגולטור, אני יכולם להפחית את האובדן ולשפר את יעילות המעגל.

כמו כן נדרש שימוש בהכרח ברגולטור בשל תוכנות ייצובי הפלט שלו כר שלא יפגע בלוח FPGA בו אני משתמש, היות והשימושים בפרויקט הינם בעלי זרם גבוה (40A) נדרש רכיב שידע ליציא פלט יציב ואמין לרכיב יקר כמו הGA.

ESP32

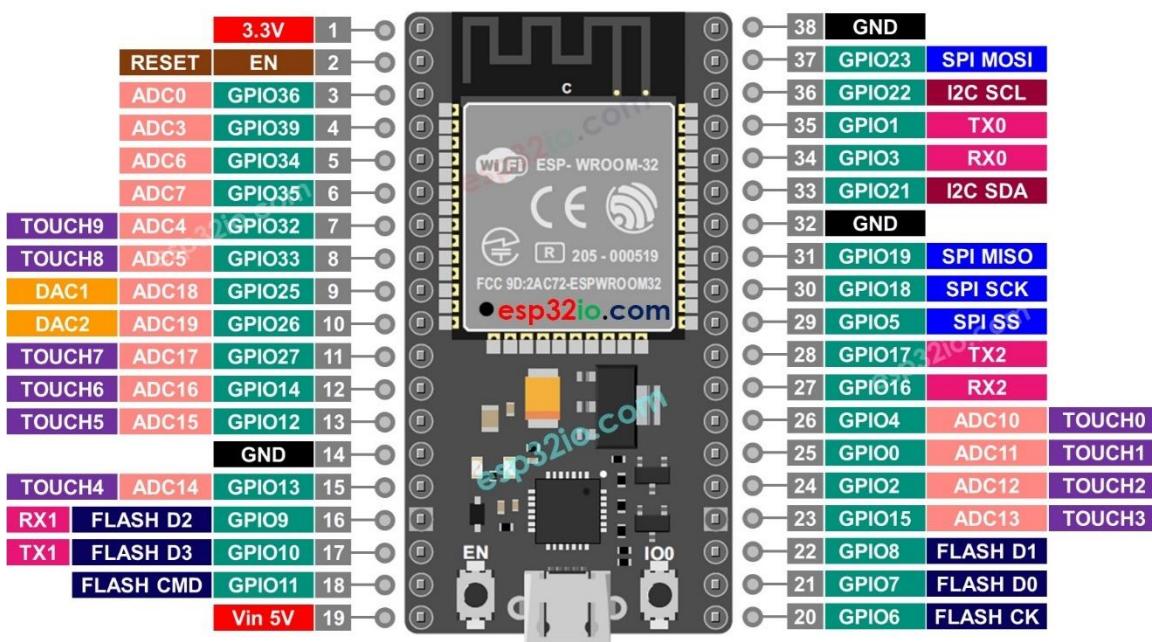
ה-ESP32 הינו מיקרו-בקר רב תכלייתי ומערכת Wi-Fi על-שבב (SoC) אשר נמצא בשימוש נפוץ ביישומי IoT (Internet Of Things).

הESP32 מציע יכולות עיבוד חזקות, קישוריות Wi-Fi מבנית, ניתן לתוכנות באמצעות שפות כמו C/C++. בפרויקט אני משתמש בעורך ARDUINO IDE לכתיבת הקוד אשר מבוסס על C++. ה-ESP32 ידוע באינטגרציה שלו עם פלטפורמת הפיתוח Arduino.

עם ה-ESP32, אתה יכול ליצור מגוון פרויקטים של>To כגון מערכות אוטומציה ביתית, רשתות חישנים, אוגרי נתונים, התקני שלט רחוק ועוד.

הESP32 משמש בפרויקט שלי כבקר תקשורת - Z'א:

קיים ESP32 אחד על הרחפן, הוא מחובר ומעביר מידע באמצעות ממשק התקשרותシリאלית א-סיליקונית (UART) לא-PGA, כאשר באמצעות יכולות WiFi שלו הוא מייצא ממשק שליטה על גבי HTTP.



PWM (pulse width modulation)

רכיב ה`driver` מקבל גל ריבועי (`pwm`) מה FPGA ומספק מתח ממוצע שונה למנוע וcrc שולט ב מהירות ועוצמת המנוע. האות `pwm` מזכיר גל ריבועי שבו הפולסים חוזרים על עצם בתדירות מסוימת, אבל אורך הפולס בכל מחזור יכולם להשתנות.

Pulse: הפולס הוא האות שמשתנה בין שני מצבים - 1 ו- 0, כך שהמפסקים נסגרים כאשר הפולס עולה ל 1 ונפתחים כאשר הפולס יורד ל 0. המצביעים האלה מייצגים שתי רמות של מתח. 1 - מתח גבוה ו 0 - מתח נמוך.

Width: הרוחב של הפולס מייצג את כמות הזמן שהאות נשאר במצב ON (ערך 1) ביחס לזמן המחזור של האות. הזמן שהפולס הוא ON מיוצג בדרך כלל על ידי אחוזים ונקרא **duty cycle**.

לפי ה-**duty cycle** שנכניס כמשמעותו לקוד VHDL של FPGA אפשר לקבוע את כמות הזמן שבו הפולס `pwm` יהיה דולק. כך, נוצר גל ריבועי `pwm` שיהיה מותאם לרצוננו וכ吐וצה מכך מהירות המנוע תהיה ניתנת לשילטה.

תדריות השעון של FPGA בו אני משתמש היא 50 MHz וזמן המחזור שלו הוא 20nSec, זמן המחזור של אות `pwm` עבר בקרי ומהירות הוא 20mSec. בשביל שהמנוע יוכל להתחל לעבוד ציריך שרוחב הפולס ב`pwm` יהיה לפחות 1mSec, וברוחב פולס של 2mSec המנוע יגיע למהירות המקסימלית. כך, כאשר ה-**duty cycle** הוא 100%, הרוחב של הפולס הוא המקסימלי - 2mSec, וכך ה-**duty cycle** הינו 0%, רוחב הפולס יהיה המינימלי - 1mSec.

הairyousim בשעון של FPGA מתקיים כל 20nSec (זמן של פולס אחד) ולכן יש 100,000 airyousim בכל :

$$100\% DC = \frac{2ms}{20ns} \left[\frac{sec}{sec} \right] = 100,000$$

ובכל 20mSec (זמן המחזור של האות `pwm`) יש 1,000,000 airyousim של השעון:

$$\frac{20 \text{ m}}{20 \text{ n}} \left[\frac{\text{sec}}{\text{sec}} \right] = 1 \cdot 10^6 = 1,000,000$$

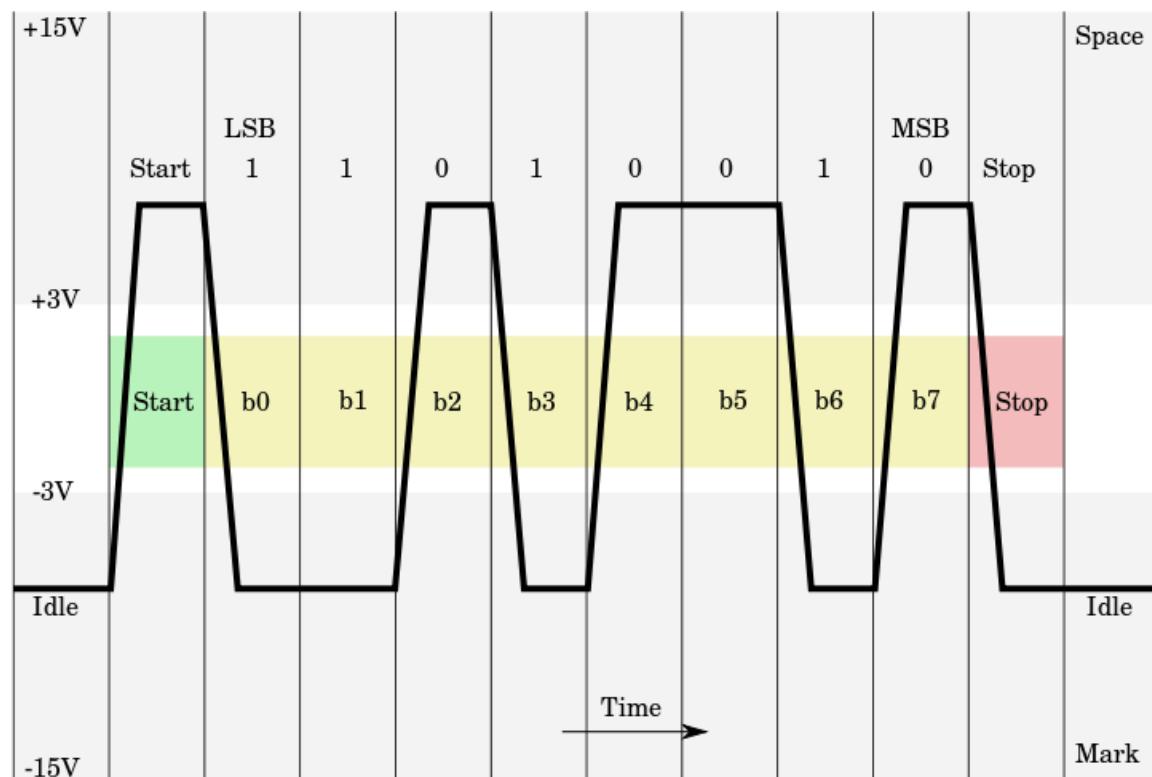
תכנית ה-HDL שאחריות על ייצור אוטות הWM תיצור 2 "גלים" – מעטפת חיצונית שאורך זמן מחרוז שלה יהיה 20mSec – ספירה עד 10^6 אירופי גביש. ומעטפת פנימית שתתחל ביחד עם תחילת הפולס החיצוני ואורכה ישנה, בין 1mSec ל- 2mSec – בין 10^4 ל- 10^5 .

תקשורת UART ופרוטוקול RS232

תקשורת טוירית אסינכרונית היא צורה של תקשורת טוירית שבה מתקשרות נקודות קצה, היחידות המוקשורות אינן מחוברות בינו לבין שער משותף, לכל אחת מהן שער משלה. במקום זה כל מקטע נתוניים מכיל מידע סינכרון בתחילתו ובסיומו, אותן הפתיחה שבראשית המקטע מכין את המקלט להגעה של הנתוניים ואות הסיום מאפס את מצב המקלט כדי לאפשר הפעלה של רצף חדש.

לפני שהתקשרות תעבור השולח וה מקבל צריכים להסכים ביניהם על כמה פרמטרים; מהירות הקו(BAUD RATE), האם השידור הוא חד או דו סיטרי, סדר הביטאים בכל בית(MSB ראשון או אחרון?)

באמצעות תקשורת UART אני ממש את פרוטוקול RS232, פרוטוקול זה אחראי על רמות המתוח השונות בתקשורת, שכן ככל שפונציואל המתוח בכל גובה יותרvr כה הוא עמיד יותר לרעשים חיצוניים.



התקנות עצמה ופירוט עליה בשלבים:

כאמור, התוכנה שלי בנויה באמצעות VHDL עבור כרטיס ה **soc nano de0**. את חיל

מהYTOP ENTITY

```

LIBRARY IEEE;
USE IEEE.STD_LOGIC_1164.ALL;
USE IEEE.NUMERIC_STD.ALL;

-- Entity Declaration
ENTITY DroneTop IS
    PORT (
        RXPin : IN STD_LOGIC; -- Serial receiver pin
        gclk : IN STD_LOGIC; -- Global clock
        STRT : IN STD_LOGIC; -- Start motors
        rst : IN STD_LOGIC; -- Reset
        PRGRM : IN STD_LOGIC; -- Program motors mode
        Forward : IN STD_LOGIC; -- Forward signal
        Backward : IN STD_LOGIC; -- Backward signal
        Rightward : IN STD_LOGIC; -- Rightward signal
        Leftward : IN STD_LOGIC; -- Leftward signal

        M1 : OUT STD_LOGIC; -- Motor 1 output
        M2 : OUT STD_LOGIC; -- Motor 2 output
        M3 : OUT STD_LOGIC; -- Motor 3 output
        M4 : OUT STD_LOGIC; -- Motor 4 output

        LED : OUT STD_LOGIC -- LED for programming mode
    );
END DroneTop;

ARCHITECTURE ARC_DroneTop OF DroneTop IS

    -- Signals for handling communication
    SIGNAL s_Data : STD_LOGIC_VECTOR(7 DOWNTO 0); -- Received data
    SIGNAL s_CounterRecieve : STD_LOGIC_VECTOR(3 DOWNTO 0); -- Receive
    counter
    SIGNAL s_Done : STD_LOGIC; -- Receive done flag
    SIGNAL ResInt : INTEGER; -- Received integer

    -- Signals for directing motor signals
    SIGNAL DC1 : INTEGER; -- Duty cycle for motor 1
    SIGNAL DC2 : INTEGER; -- Duty cycle for motor 2
    SIGNAL DC3 : INTEGER; -- Duty cycle for motor 3
    SIGNAL DC4 : INTEGER; -- Duty cycle for motor 4

    -- Signals for internal calculations and delays
    SIGNAL CalcPrec : INTEGER; -- Calculated precise duty cycle
    SIGNAL timeoutPassed1 : STD_LOGIC := '0'; -- Timeout passed flag
    SIGNAL timeoutPassed2 : STD_LOGIC := '0'; -- Timeout passed flag

    -- Component declaration for receiving logic
COMPONENT RecieveLogic IS
    PORT (
        gclk : IN STD_LOGIC;
        rst : IN STD_LOGIC;
        RXD : IN STD_LOGIC;
        REdata : OUT STD_LOGIC_VECTOR(7 DOWNTO 0);
        counterREcieve : OUT STD_LOGIC_VECTOR(3 DOWNTO 0);
        done : OUT STD_LOGIC
    );
END COMPONENT;

    -- Component declaration for PWM signal generation
COMPONENT PWM_sig IS
    PORT (
        rst : IN STD_LOGIC;
        gclk : IN STD_LOGIC;
        DC : IN INTEGER; -- Duty cycle input
        pwm : OUT STD_LOGIC -- PWM output
    );
END COMPONENT;

    -- Component declaration for TimeoutManager
COMPONENT TimeoutManager IS
    PORT (
        rst : IN STD_LOGIC;
        gclk : IN STD_LOGIC;
        TimeToWait : IN INTEGER; -- Time to wait, in ms (10^-3)
        passed : OUT STD_LOGIC -- bool/1bit, is the specified time
        passed.
    );
END COMPONENT;

```

(1)

TOP ENTITY CODE: DRONE TOP

```

BEGIN

-- Instantiation of the receiving logic component
Inst : RecieveLogic
PORT MAP(
    gclk => gclk,
    rst => rst,
    RXD => RXPin,
    REdata => s_Data,
    counterREceive => s_CounterRecieve,
    done => s_Done
);

-- Instantiation of the PWM signal generation components for each motor
u1 : PWM_sig
PORT MAP(
    gclk => gclk,
    rst => rst,
    DC => DC1,
    pwm => M1
);

u2 : PWM_sig
PORT MAP(
    gclk => gclk,
    rst => rst,
    DC => DC2,
    pwm => M2
);

u3 : PWM_sig
PORT MAP(
    gclk => gclk,
    rst => rst,
    DC => DC3,
    pwm => M3
);

u4 : PWM_sig
PORT MAP(
    gclk => gclk,
    rst => rst,
    DC => DC4,
    pwm => M4
);

-- Instantiation of the TimeoutManager component
timeout_u1 : TimeoutManager
PORT MAP(
    gclk => gclk,
    rst => rst,
    TimeToWait => 1000, -- Time to wait in ms, so 1000 ms = 1s
    passed => timeoutPassed1
);

-- Instantiation of the TimeoutManager component
timeout_u2 : TimeoutManager
PORT MAP(
    gclk => gclk,
    rst => rst,
    TimeToWait => 20000, -- Time to wait in ms, so 20000 ms = 20s
    passed => timeoutPassed2
);

```

(2)

```
-- Main process to handle duty cycle updates
PROCESS (gclk, rst)
BEGIN
    IF STRT = '0' AND PRGRM = '0' THEN
        -- Do nothing
    ELSIF STRT = '0' AND PRGRM = '1' THEN
        -- PROGRAMMING MODE
        LED <= '1';
        IF timeoutPassed2 = '0' THEN
            -- Let the user connect to WiFi, until then wait. do
            nothing
        ELSE
            -- Convert the received 8-bit data to an integer
            ResInt <= to_integer(unsigned(s_Data));
            -- Calculate the precise duty cycle
            CalcPrec <= (ResInt * 4095) / 255;
            -- Assign the calculated duty cycle to all motors
            DC1 <= CalcPrec;
            DC2 <= CalcPrec;
            DC3 <= CalcPrec;
            DC4 <= CalcPrec;
        END IF;
    ELSIF STRT = '1' THEN
        IF rst = '1' THEN
            -- If reset is active, set all duty cycles to 0
            DC1 <= 0;
            DC2 <= 0;
            DC3 <= 0;
            DC4 <= 0;
        ELSIF gclk'event AND gclk = '1' THEN
            IF timeoutPassed1 = '1' THEN
                -- Convert the received 8-bit data to an integer
                ResInt <= to_integer(unsigned(s_Data));
                -- Calculate the precise duty cycle
                CalcPrec <= (ResInt * 4095) / 255;
            ELSE
                -- First calibration for ESCs
                CalcPrec <= 0;
            END IF;
            -- Assign the calculated duty cycle to all motors
            DC1 <= CalcPrec;
            DC2 <= CalcPrec;
            DC3 <= CalcPrec;
            DC4 <= CalcPrec;
        END IF;
    END IF;
END PROCESS;

END ARCHITECTURE ARC_DroneTop;
```

(3)

קוד בעל 186 שורות והוא המרכדי – מתנו יוצאים כל ה”עופים”.

תחליה אני מגדר את הישות עצמה – פינים IO, מגדר ככינה 3 אותות/פינים;

- רgel התקשרות מהESP אל ה **fpagat RX** – RECEIVER,fpagat של TRANSMITTER (esp) ; מחווט

אל רgel PIN_AC22 שהוא ה **rgel 40 בסוג 1** הימני.

- שעון גלובלי – GCLK; מחווט אל הקרייסטל שנמצא בתוך הcrcטיש, תדר של 50MHz.
- PIN_V11.

- רgel ניתוק – RST; מחווט ל- AH6_PIN שהוא פין IO ב00IO. מחווט לESP.
- ולאחר מכן יציאה עוד 4 אותות/פינים;



ולבסוף גם מס' פינים נוספים למען ”נוחיות מפתח”: PRGRM,STRT שהם מחווטים לפינים IO על גבי crcטיש, למען שליטה מלאה דרך ESP מבלי להתעסק במתגים. כמו כן 4 פינים שהם Forward, Backward, Rightward, Leftward. התכנון הוא שאוכל להפעיל אחד מהם ורוחפן יתקדם בכיוון הרלוונטי. הסבר בהמשך

לאחר מכן אתחיל בהגדרת הארכיטקטורה של הישות – ההתנהגות של crcטיש: אגדיר מספר אותות/משתנים פנימיים, חלק לקבלת מידע מהתקשורת עם הESP, חלק לניהול התקשרות עצמה וחלק לניהול מידע פנימי ולכך יהיה מסוג integer,into, אבצע עליהם פעולות מתמטיות. כל שאר הינים ביטים, מידע הוא ביט (= 8 ביטים), CounterReceive הוא 4 ביטים, ואות DONE הינו ביט אחד.

שורות 46-76 אתחול של מודולים אחרים בפרויקטם שאפרט עליהם בהמשך ע”פ ה **portmap** שלהם.

ואחל משורה 80 עד 140 אני מייצר ישות (instance) עבור כל חלק הכרחי בפרויקט וע”פ ייצוא הפינים שלהם. ישות עברו מודול קבלת המידע, 4 יישויות עברו כל אחד מהמנועים. קר אוכל לשלוט בהמשך בכל מנוע בנפרד ו 21 יישויות עברו דילאי.

ובמקרה קוד האחרון לעט entity top אני מאתחל את ההתנהגות (ארכיטקטורה) של הרכזיס ע"פ רשימת רגישיות (Sensitive list) שכולל בתוכו את השעון ואת אות האיפוס RST.

התנהגות של ENTITY TOP מכילה בתוכה לוגיקה מעט מורכבת:

1. על מנת לשנות תוכנה כלשהי במנועים נדרש לבצע "תכנות" ע"פ בייפים/מורס לבקרים המהירות שלהם, בהמשך לדבר על כיצד הפכתי את ייונן התנוועה של חלק מהמנועים. אז מוד הPROGRAM (PRGRM) היה על מנת שאוכל לעשות זאת בעדרת FPGA.

2. במשק השיטה האינטגרטי יש כפתור START וכפתור RST שמטרתם תהיה השטילתם במקרה הצורך על הרחפן. בעבר אם הייתה מדliqu את הרחפן אז עד שהייתי מתחבר לWIFI המנועים היו עובדים חלש אבל עובדים. אז באמצעות הוספה כפתור START MOTORSocr תאפשר לך שתהיה גם בטיחותי וגם תיקנתי את הבעיה הזאת.

a. אם START כבוי וPROGRAM כבוי אל תעשה כלום

b. אם START כבוי וPROGRAM דלוק: - חכה 20 שניות ואל תעשה כלום (תען זמן לשימוש להתחבר לWIFI) ולאחר מכן תשדר את כל מה שהמשתמש משדר.

c. אם START דלוק וRST דלוק: תען 0 לכל המנועים. שיהיו כבויים.

d. אם START דлок וRST כבוי: אם התרחשה עליית שעון (אירוע שעון + הוא עכשו 1) אז חכה 1 שנייה בשבייל שבקרים המהירות יעשו קליברציה, לאחר מכן שדר את הנתונים מהמשק שליטה אל בקרים המהירות.



CODE: RECIEVELOGIC

```
library IEEE;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_ARITH.all;
use IEEE.STD_LOGIC_UNSIGNED.all;
use ieee.numeric_std.all;

entity RecieveLogic is
    port (clk           : in  std_logic;
          rst           : in  std_logic;
          RXD          : in  std_logic; -- Serial Reciever Pin AE25
          REdata        : out std_logic_vector(7 downto 0);
          counterREceive : out std_logic_vector(3 downto 0);
          done          : out std_logic
        );
end entity RecieveLogic;

architecture Behavioral of RecieveLogic is
    type state_vector is (wait_for_start_bit, start_bit_skip,
    wait_for_middle_point, counts_for_8_bit);
    signal state                      : state_vector;
    signal bit_counter                : std_logic_vector(3 downto 0);
    signal Sdata                      : std_logic_vector(7 downto 0);
    signal sample_cnt                 : std_logic_vector(7 downto 0);
    signal cnt                        : std_logic_vector(4 downto 0);
    signal q1, rxd_fall, sample_cnt_en : std_logic;
    signal shift_en                  : std_logic;
    signal eq_27                      : std_logic;
    signal done_s                     : std_logic;
begin
    begin
        rxd_fall <= (not rxd) and q1;
        eq_27   <= '1' when (cnt = 27) else '0';
        process(clk, rst)is
        begin
            if (rst = '1') then
                Sdata      <= (others => '0');
                state     <= wait_for_start_bit;
                bit_counter <= (others => '0');
                cnt       <= (others => '0');
                sample_cnt <= (others => '0');
                shift_en   <= '0';
                sample_cnt_en <= '0';
                q1         <= '1';
                done_s     <= '0';
                REdata     <= (others => '0');
            elsif clk'event and clk = '0' then
                shift_en <= '0';
                done_s   <= bit_counter(3) and shift_en;
                if (done_s='1') then
                    REdata <= Sdata;
                end if;
                if (shift_en = '1') then
                    Sdata <= RXD & Sdata(7 downto 1);
                end if;
                if (eq_27 = '1') then
                    q1   <= rxd;
                    cnt <= (others => '0');
                    if (sample_cnt_en = '1') then
                        sample_cnt <= sample_cnt+1;
                    else
                        sample_cnt <= (others => '0');
                    end if;
                end if;
            end if;
        end process;
        counterREceive <= bit_counter;
        done<=done_s;
    end architecture Behavioral;
```

(1)

```
        case state is
            when wait_for_start_bit =>
                sample_cnt_en <= '0';
                if (rxdfall = '1') then
                    state <= start_bit_skip;
                end if;
            when start_bit_skip =>
                sample_cnt_en <= '1';
                if sample_cnt(3 downto 0) = x"f" then
                    state <= wait_for_middle_point;
                end if;
            when wait_for_middle_point =>
                sample_cnt_en <= '1';
                if sample_cnt(1 downto 0) = "11" then
                    state     <= counts_for_8_bit;
                    sample_cnt_en <= '0';
                end if;
            when counts_for_8_bit =>
                sample_cnt_en <= '1';
                if (sample_cnt(3 downto 0) = x"0") then
                    if (bit_counter <= 7) then
                        shift_en   <= '1';
                        bit_counter <= bit_counter+'1';
                    else
                        bit_counter <= (others => '0');
                        sample_cnt_en <= '0';
                        state     <= wait_for_start_bit;
                    end if;
                end if;
            when others =>
                state <= wait_for_start_bit;
        end case;
    else
        cnt <= cnt+1;
    end if;
end if;
end process;
counterREceive <= bit_counter;
done<=done_s;
end architecture Behavioral;
```

(2)

את הקוד זהה קיבלתי מהמרצה שלי, רפי, על מנת למשתמש תקשורת טורית אסיכרונית בין הכרטיס FPGA לבין כרטיס ה-ESP. בנוסף כמה שאוכל על תהליכי;
בגהדרת היישות עצמה – 3 קלטים שהם השעון, אות איפוס ורגל קבלת הנתונים מה-ESP.
פלטים והם REdata שהוא בעצם מידע לאחר עיבוד התקשרות, counterREceive
שספק את הביטים שמתקבלים (4 ביט) וdone שהוא משתנה של בית אחד המיצג מתי המידע מוכן לקריאה והמשך תהליכי.

**בגלל שהתקשרות ע"פ RS2321 UART היא הולכת בסדר מסוים, התהליך הנכון לקריאת המידע יהיה מכונת מצבים (אוטומט סופי); בקצרה, מכונת מצבים היא מכונה בעלת זכרון מוגבל שמטרתה היא לנוהל FLOW של תהליכי כלשהו (כגון – תקשורת UART). בקוד בתוך ארכיטקטורת היישות (התנהגות) מוגדר וקבעו המיצג את המצב של מכונת המצבים מתוך `wait_for_start_bit, start_bit_skip, (wait_for_middle_point, counts_for_8_bit` רשימה של 4 מצבים: ()
כמו כן מוגדרים מספר סיגנלים (משתנים, חיישנים פנוימים) למען ניהול תהליכי עיבוד התקשרות והוצאה ביט.**

הקוד מנהל את תהליכי קבלת התקשרות וייצוא המידע בצורה "ז"א; הקוד בוחן את הביט המתתקבל ומבצע אותו, במקרה וממספר תנאים מתקיים, מעבור למצב הבא באופןוטם. **.top entity** זורם עד למצב הסופי שבו משחרר המידע כלו אל **cursor FLOW**.

```

case state is
    when wait_for_start_bit =>
        sample_cnt_en <= '0';
        if (rxn_fall = '1') then
            state <= start_bit_skip;
        end if;
    when start_bit_skip =>
        sample_cnt_en <= '1';
        if sample_cnt(3 downto 0) = x"f" then
            state <= wait_for_middle_point;
        end if;
    when wait_for_middle_point =>
        sample_cnt_en <= '1';
        if sample_cnt(1 downto 0) = "11" then
            state <= counts_for_8_bit;
            sample_cnt_en <= '0';
        end if;
    when counts_for_8_bit =>
        sample_cnt_en <= '1';
        if (sample_cnt(3 downto 0) = x"0") then
            if (bit_counter <= 7) then
                shift_en <= '1';
                bit_counter <= bit_counter+'1';
            else
                bit_counter <= (others => '0');
                sample_cnt_en <= '0';
                state <= wait_for_start_bit;
            end if;
        end if;
    when others =>
        state <= wait_for_start_bit;
end case;
```



CODE: PWM_SIG

```
LIBRARY ieee;
USE ieee.std_logic_1164.ALL;
USE ieee.numeric_std.ALL;
USE ieee.std_logic_unsigned.ALL;
USE ieee.std_logic_arith.ALL;

-- Entity Declaration
ENTITY PWM_sig IS
  PORT (
    rst : IN STD_LOGIC;
    gclk : IN STD_LOGIC;
    DFpga : IN INTEGER; -- DFpga input
    pwm : OUT STD_LOGIC -- PWM output
  );
END PWM_sig;

-- Architecture Body
ARCHITECTURE PWM_sig_architecture OF PWM_sig IS
  SIGNAL cou : INTEGER := 0; -- Counter
  SIGNAL flag : STD_LOGIC; -- Flag for state control
  SIGNAL FixedDC : INTEGER; -- Fixed duty cycle
BEGIN
  BEGIN
    PROCESS (gclk, rst)
    BEGIN
      IF rst = '1' THEN
        -- Reset condition
        cou <= 0;
        flag <= '1';
        pwm <= '0';
      ELSIF gclk'EVENT AND gclk = '1' THEN
        -- Calculate fixed duty cycle (1..2ms range), page 37
        FixedDC <= ((5 * (10 ** 4)) + ((5 * (10 ** 4) * DFpga) / 4095));
    END IF;
    cou <= cou + 1;
    IF cou >= FixedDC THEN
      flag <= '0';
      cou <= 0;
    END IF;
  END PROCESS;
END PWM_sig_architecture;
```

(1)

```
IF flag = '1' THEN
  -- Off-period state
  IF cou < 1000000 THEN
    cou <= cou + 1;
    pwm <= '0';
  ELSE
    flag <= '0';
    cou <= 0;
  END IF;
ELSE
  -- On-period state
  IF cou < 1000000 THEN -- counting until 10 * 10^5 =>
    Cycles*signal = time => 10^6*20*10^-9 = 20ms
    cou <= cou + 1;
    IF cou <= FixedDC THEN -- counting until throttle wave
      itself(1..2ms)
      pwm <= '1';
    ELSE
      pwm <= '0';
    END IF;
  ELSE
    cou <= 0;
    flag <= '0';
  END IF;
END IF;
END IF;
END PROCESS;
END PWM_sig_architecture;
```

(2)

קובץ בעל 60 שורות שאחראי לייצרת את WM למען שליטה במנועי הברשלס. כביכול הקוד נראה פשוט אך הושקעה מחשבה רבה מאחוריו – אתחיל מהקוד המופשט וatkdm לתחילה החשיבה מאחוריו.

בהגדרת היישות ישנו 3 כניסה ויציאה אחת אל המודול עצמו – כניסה: שעון, אות איפוס ומשתנה בשם DFpga. יציאה אחת והוא אות WM. כל הכניסות והיציאות הן בית אחד. (מלבד DFpga והוא מספר טבעי שלם שמוגדר 32 ביט למרות שבפועל עד 4095 = 12 ביט).

בהתנגנות המודול הגדרתי תחילת מספר חיוטים פנימיים למען ניהול התהלים, עיבוד האותות והשעון. ולאחר מכן מתחילה בתהlixir עצמה עם רשיימת רגישיות שהיא אות השעון ואות האיפוס.

בנייתי את הקוד הזה כמעין מכונת מצבים, לא דטרמיניסטי – אך בשל העניין שיש רק 2 מצבים לא יהיה צורך להגדיר state vector אלא השתמשתי בסיגנל שאורכו בית אחד ושמו flag. במידה ואות האיפוס דולק (high-active reset) אז יש לאפס את המונה שעון, אות החמק שווה ל0 והדגל (flag) יהיה שווה ל1 – מצב התחלתי.

כאשר במצב התחלתי – ספור 20mSec ואל תעשה דבר (תשאיר 0=>wm) על מנת לתת softstart לבكري המהירויות. לאחר מכן כאשר הדגל עובר ל'0' אז מכונות המצבים נמצאת במצב העיקרי והאיןסופי (כביכול, עד שהRTS ידלק), במצב העיקרי כאשר הדגל '0' אז מתבצע תהליך כזה: היות והשעון של הכרטיס מיצא תדר של 50 מיליון הרץ – ד"א שכל אות שעון אורכו 20 ננו שניות

$$f \cdot t = 1$$

$$\rightarrow t = \frac{1}{f} \quad [f = 50 \cdot 10^6 (Hz)]$$

$$\rightarrow t = \frac{1}{50 \cdot 10^6}$$

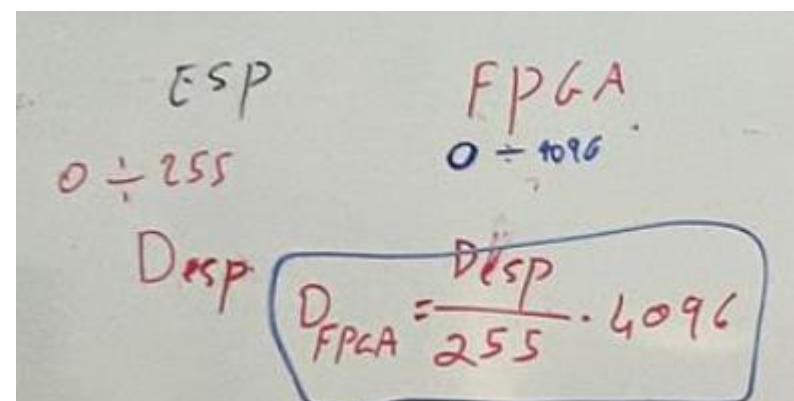
$$\rightarrow t = 2 \cdot 10^{-8} (\text{sec}) = 20 \cdot 10^{-9} (\text{sec})$$

ככל עוד מונה פעימות השעון קטן מ-1000000 הגדל את D_{FPGA} ב-1 כל עלייה שעון. אם D_{FPGA} קטן או שווה ל- $FixedDC$ אז ' $1 = D_{FPGA}$ ' אחרת '0'. זה פועל לפי עיקרונו המעטפת החיצונית- פנימית שהסבירתי עליו קודם, אם הספירה שלי נמצאת גם בתוך 10^6 שזה כמות הפעימות הנדרשות עבור 20 מילישניות וגם נמצאת בתוך $DutyCycle$ התואם לערך D_{FPGA} לפי החישוב שביצעת, תן ל- PWM 1 לוגי. אחרת 0 לוגי. כעת הדבר על $FixedDC$.

בתחילת הפיתוח של הרחפן, במקום לחבר **ESP** וلتקשר עימו על גבי **WIFI** והעברת מידע באמצעות **UART**, השתמשתי בפוטנציאומטר פשוט, כאשר הוא מחובר ל- $5V$ ואני מתקשר עימו באמצעות **ADC** (Analog Digital Converter) אשר בנוי בתוך הcrcטיס (רכיב **LTC2308**). כך שהרכיב נתן לי במינימום 0 ובמקסימום 4096 וכן השתמשתי בו. ואז במעבר שלי אל **ESP** התרחשו מספר תהליכיים, בשביל לעברו בין שניים עשויתי הדמייה מה**ESP** אל הפוטנציאומטר כך – בהמשך שאציג את הקוד של **ESP** יהיה ניתן לראות כי המקסימום שיוצא מתחם **ESP** (נקרא לו **Desp**) הוא 255 והמינימום הוא 0. ואילו המקסימום שה**FPGA** מייצא (**Dfpga**) הוא 4096 והמינימום הוא 0.

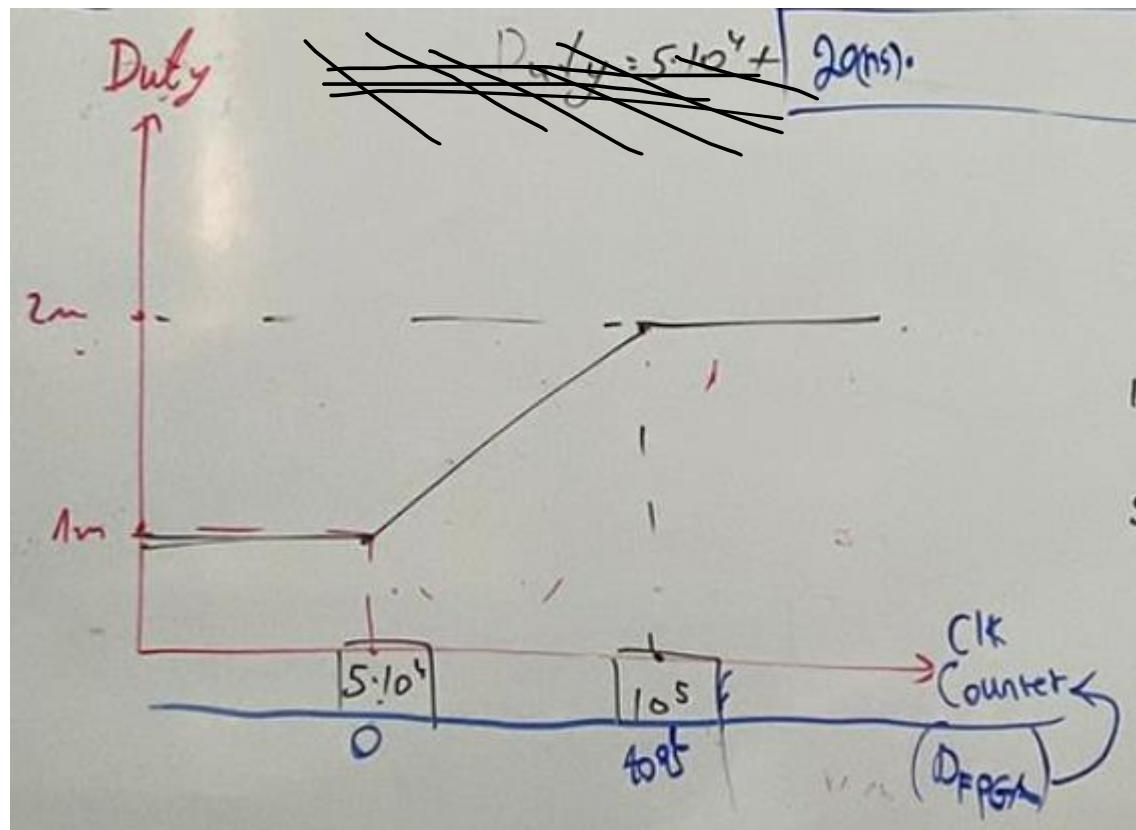
בשביל לתפעיל את המנועים, בקרי המהירות מתוכנתים כך שאורך הגל "הגדול" יהיה $20ms$ ובתוכם יהיה גל קטן שיתחיל מתחילה הגל גדול ואורכו ישנה ע"פ הכוח שנרצה להביא למנוע, האצה מהירה/אייטה יותר. אורך גל קטן מקסימלי יהיה $2ms$ ואורך גל קטן מינימלי יהיה $1ms$.

על מנת להתאים בין ערכי **ESP** לבין ערכי **DutyCycle** המתאימים נדרשנו לעשות התאמת מתמטית בין שנייהם.



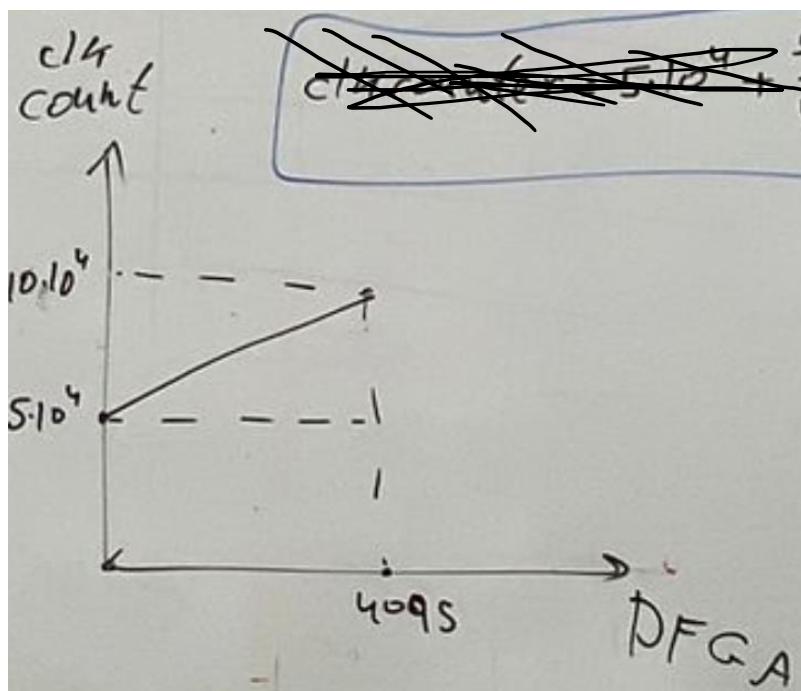
$$D_{FPGA} = \frac{Desp}{255} \cdot 4096$$

באמצעות משוואת הישר הזה ביצענו התאמת בין שנייהם, כעת ביצענו נורמליזציה ל- D_{FPGA} וניתן להתקדם למשוואות יותר מסובכות ועמוקות.



כאן ניתן לראות גраф המתאר את הקשרים בין המונחים פעימות השעון לבין ה **DutyCycle**.
הנדרש על מנת לפעול את המנוועים ב **throttle** המתאים.

ד"א מצאנו משווה ישר המתארת את התופעה ומתקימת בין רק בתחום מוגדר ע"פ הגרף.
נפеш את הגרף יותר:





וכך הגענו לקשר שמתאר כמה מונה פעימות השעון צריך להיות לפני תלות בDFPGA המנורמל. נבנה משוואת ישר ע"פ נוסחאות מתמטיות בסיסיות:

$$slope = \frac{\Delta y}{\Delta x} = \frac{10 \cdot 10^4 - 5 \cdot 10^4}{4095 - 0} = \frac{5 \cdot 10^4}{4095}$$

שימוש בנוסחת הבסיס עבור משוואת ישר אשר חותכת את ציר Y בנקודה (0,0)

$$y = slope \cdot x + n$$

נציב לבפנים את כל המשתנים הרלוונטיים ונקבל את הקשר הבא:

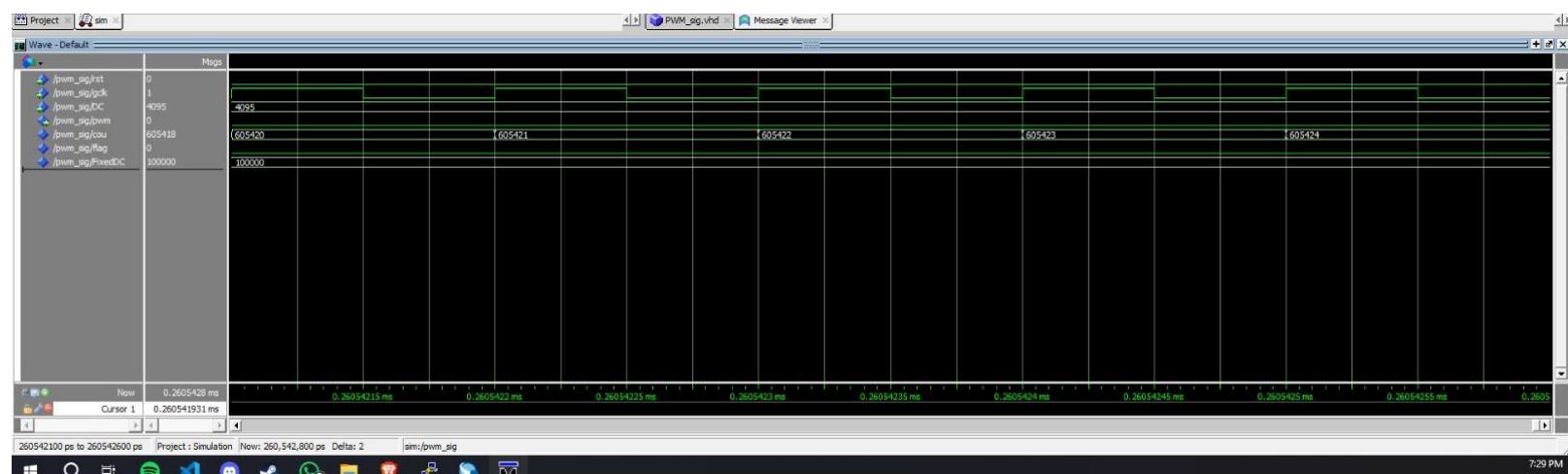
$$clkCounter = 5 \cdot 10^4 + \frac{5 \cdot 10^4}{4095} \cdot Dfpga$$

נוסחה זו מפשטת את ADFPGA וכן ניתן יהיה לקבל טווח שליטה עבור בקריה המהירות. נוסחה זו מתקיימת רק בתנאי הבא: ADFPGA בטוח בין 0 לבין 4095. כאשר 4095 הנוסחה תתן 10 בחזקת 5 שהוא 2 מילישניות דיווי סייקל (100%) וכאשר 0 הנוסחה תתן 10 כפול 5 שהוא 1 מילישניה דיווי סייקל (0%).

```
FixedDC <= ((5 * (10 ** 4)) + ((5 * (10 ** 4) * DC) / 4095));
```

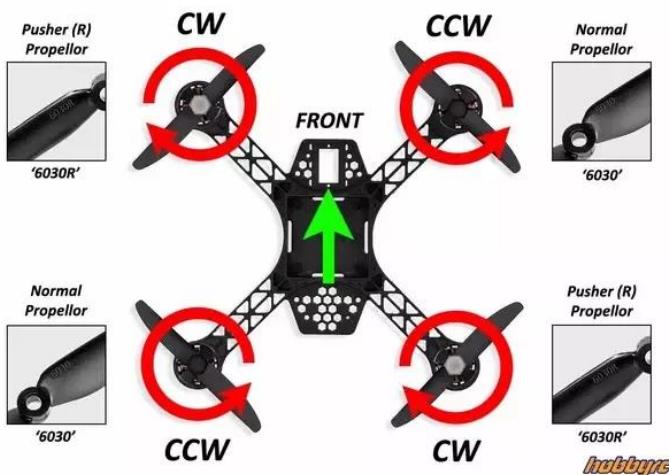
בקוד, הקולט DC הוא ה-ADFGA.

כמובן שביצועי סימולציה במודלים models בכדי לראות שהכל עובד – כי כל הפיתוח קרה בביטחון והכרטוס עצמו היה במעבדה איז לא רציתי לבזבז זמן;



CODE: Reverse Motors

Quadcopter prop diagram



על מנת שרחפן ירחף, כל מנוע צריך לדחוף את הפרופלורוים

בצורה שידחפו אוויר מטה כרך שתנועת המנועים

אמורה להיראות כרכ:

על מנת להחליף את הכיוונים של תזוזה המנוע;

1. מתוך כל מנוע אל ESC יוצאים 3 כבלים: 3 פאוזות. בשבייל להפוך את כיוון הסיבוב

יש להחליף 2 פאוזות מתוך 3, היות והמנוע פועל עם סליל לא קרייטו אילו פאוזות

נחליף היות ובכל מקרה הפאזה המתאימה תגרום למשיכת המגנטים בתוך המנוע, מה

שיגרום לתנועה חלקה של המנוע.

2. היות ובקרי המהירות שלו הם בקרים "חכמים" = **programmable**. באמצעות
שליטה על **throttle (PWM)** אני מסוגל לגשת לkonfiguracija שלהם ולשנות את

סיבובם. ע"פ המספר הבא: (פרמטר 8 הוא הפרמטר שאחראי על כיוון סיבוב המנוע)

F. Programming via Transmitter

Step 1: Enter program mode

Switch the transmitter on→Pull the throttle stick to the top position→Switch the ESC on, wait 2 seconds, you will hear two “beep” sounds, which denotes that Max. throttle has been confirmed→Hold the throttle stick at the top position, and then wait 2 seconds until you hear tune “ ↗ 1 2 3 ↗ 1 2 3 ” , that means you have entered the transmitter programming mode.

Step 2: Select program parameters

Hold the throttle stick on top position, there’re 7 parameters can be set by using your transmitter. You would hear 7 different indicating sounds which correspond to 7 different parameters. Pull the throttle stick to the bottom position (full Off throttle) within 2 seconds after you hear the correspondent sound will brings you to the correspondent parameter setting status. The indicating sounds will repeat in turn as follow.

1. “beep-” (a short sound) which indicates the Brake Type
2. “beep-beep-” (two short sounds) which indicates the Timing Mode
3. “beep-beep-beep-” (three short sounds) which indicates the Start Force
4. “beep-beep-beep-beep-” (four short sounds) which indicates the Curve Mode
5. “beep----” (a long sound) which indicates the Control Frequency
6. “beep----beep-” (a long sound and a short) which indicates the Low-voltage Protection
7. “beep----beep-beep-” (a long sound and two short) which indicates the Cutoff Mode
8. “beep----beep-beep-beep-” (a long sound and three short) which indicates the Rotation Direction

Step 3: Select program values

After entering parameter setting status, hold the throttle stick on the bottom position, you will be led to the repeat selection of that parameter setting status. Each sound likes 4 short sounds and one long sound (1 long sound=5short sounds), and by that analogy. After some sound, pull the throttle stick to the top position in 2 seconds, after you hear a tune “ ↗ 2 1 ↗ 3 2 1 ”, which means the correspondent value has been chosen and saved. Hold the stick on the top position, return to the second step and continue programming.

מתוך ידיעה כמפתח ואדם עם היגיון – נונסה את הגישה הפחות פולשנית בהתחלה! מעדיף כמה שפחות לפתח רחפן שעובד וסגור, אך ניגשתי לגישה 2 עם התכונות של ה-ESCs. סימנתו כל מנוע אצלי לפי מספר. על מנת שהרחפן אכן ירחף באוויר מבלי להסתובב במקום נדרש לייסובב את מנועים 1, 2, 3, 4 שיהיו נגד כיוון תנועת השעון – **CCW/Counter Clock Wise**. אחרי שניסיתי לgesht אל תכונות המנועים מהגישה של שימוש בPRGRM וכשלתי היות ובזמן הפיתוח לא קלטתי מה הייתה הבעה בקוד של ה-GPIOS כי גם אם PRGRM היה דלוק אך זה היה מתחל את המנועים עם throttle 0. אך ניגשתי אל כך בצוora מעט שונה שהלכה לי מצוין!. בברזי המהירות קיימים **דַּלְגָּלִים (Flipflops)** שזכרים קונפיגורציות שונות, ובניהם גם את כיוון תנועת המנועים, ז"א אני צריך לבצע את התהיליך פעם אחת וזהו.

כל ESC מתחבר למקור המתח ומצד שני מיצא 3 כבילים, אדום ושחור -> (BEC) **Battery Elimination Circuit** שזה בעצם "ניצול חכם" של בקרוי המהירות, הם מיצאים 5V לא אמנים בעליabil בחלק מהפעמים משתמשים ב- BEC בשבייל להدليل כל מיני דברים דניחסים וכן המעלג נראה יותר אסתטי או משתמשים בהם בשבייל להגדיר נק' ייחוס כאשר אותן הפעולה של המנוע מגיע למערכת חיצונית בלי קרקע משותפת. וככל אחד צחוב שהוא אותן הפעולה. חיבורתי את ה-5V של שני בקרוי המהירות של מנועים 1, 4, 2, 3 בטור ואליו חיבורתי 3V מארודואינו אוננו שהוא ברשותי. וכך עם ה-GND של בקרוי המהירות והארודואינו. את הכבילים של אותן הפעולות של כל בקרוי המהירות לפני עלי גבי הארדואינו(*יש פינים שלא יכולים לצאת גלי PWM. אני השתמשתי בפינים 5, 6, 7 שמסומנים על גבי הרכביס עם ~) וכעת באמצעות הקוד שאציג לך יכולתי לפעול ע"פ MANUAL של בקרוי המהירות ולתכנות אותם!

```
//CODE WITH NO POTENTIOMETER:
#include <Servo.h>
Servo ESC1; // create servo object to control ESC1
Servo ESC2; // create servo object to control ESC2
int potValue; // value from the analog pin
String pot;

void setup() {
  Serial.begin(2000000);
  ESC1.attach(5,1000,2000); // (pin, min pulse width, max pulse width in microseconds)
  ESC2.attach(6,1000,2000); // (pin, min pulse width, max pulse width in microseconds)
}

void loop() {
  // send data only when you receive data:
  if (Serial.available() > 0) {
    // read the incoming byte:
    pot = Serial.readString();
    potValue = pot.toInt();

    delay(10);
    // say what you got:
    Serial.print("ESC write: ");
    Serial.println(potValue, DEC);
  }
  potValue = constrain(potValue, 0, 180); // constrain the value of potValue between 0 and 180.
  ESC1.write(potValue); // Send the signal to ESC1
  ESC2.write(potValue); // Send the signal to ESC2
  delay(10);
}
```

תהליך התכנות הLR כר:

1. התחל את הארדואינו על 180% (100%)
2. לחבר את אותות הפעולה של 2 המנועים – חכה 2 שניות – שמע 2 ביפים
3. 100% – חכה 2 שניות – 3 ביפים פעמיים
4. חכה עד ההגדלה הנכונה (ביפ ארוור, 3 ביפים) (כיוון המנוע) – משור ל0%
5. חכה עד הכיוון הנכון (הפור – 2 ביפים) – משור ל100%
6. כבה PWM.

וזהו!Cutת כל פעם שהפעלתי את המנועים רגיל(תחילה 0%) המנועים נעו בכיוון הנכון וכך סודרה בעיית כיוון המנועים! :)



CODE: TimeoutManager

```
LIBRARY ieee;
USE ieee.std_logic_1164.all;
USE ieee.numeric_std.all;

-- Entity Declaration
ENTITY TimeoutManager IS
    PORT
    (
        rst : IN STD_LOGIC;
        clk : IN STD_LOGIC;
        TimeToWait : IN INTEGER;          -- Time to wait, in ms (10^-3)
        passed : OUT STD_LOGIC           -- bool/1bit, is the specified time passed.
    );
END TimeoutManager;

-- Architecture Body
ARCHITECTURE timeoutManager_architecture OF TimeoutManager IS
    SIGNAL cou : INTEGER := 0;          -- Counter
    SIGNAL FixedCount : INTEGER;        -- Fixed count based on TimeToWait
BEGIN
    PROCESS(clk, rst)
    BEGIN
        IF rst = '1' THEN
            -- Reset condition
            cou <= 0;
            passed <= '0';
        ELSIF clk'event and clk = '1' THEN
            -- Calculate fixed count based on TimeToWait (1 ms = 50,000 clock cycles at 50 MHz)
            FixedCount <= TimeToWait * 50000;

            -- Increment counter until the fixed count is reached
            IF cou < FixedCount THEN
                cou <= cou + 1;
                passed <= '0';
            ELSE
                passed <= '1';
            END IF;
        END IF;
    END PROCESS;
END timeoutManager_architecture;
```

קוד שנורא דומה לSig_PWM, זה כלי עוזר שיצרתי עבורו למען פונקציית דילאי – הוא מקבל כקלט אות איפוס, שעון גלובלי(50MHz), מס' מסוג שלם שמוצג במילישניות את כמות הזמן שיש לחכות ומיצא משתנה בוליאני/ביט אחד שבמידה והוא true => כמות הזמן הדרישה עברה. **הчисוב הוא פשוט** – כמה פעמים אות 20 נכנס ב³? 10? יצא 50k ולכון 50(**מספר הפעימות עבור 1 מili שניה**) כפול מס' מילישניות יתן את הזמן הנדרש. יצורתי את היחסות הזה על מנת להמתין X זמן על Duty Cycle 0%, כך שהייה זמן לבكري המהירות לעבור קליברציה/קונפיגורציה התחלתית(נדרשת התחלתה של גל מינימלי 5ms) לפני שמתחלים להפעיל את המנועים, על מנת שבקרי המהירות יעשו קליברציה בהתאם



CODE: ESP CONTROLLER

```
#include <WiFi.h>
#include <WebServer.h>
#include "driver/dac.h"

// WiFi Hotspot credentials
const char* ssid = "DroneWiFi"; // WiFi Hotspot SSID
const char* password = ""; // No password

WebServer server(80); // Create a WebServer on port 80

void setup() {
    Serial.begin(115200);

    // Set up WiFi hotspot
    WiFi.softAP(ssid, password);

    // Print the IP address of the WiFi hotspot
    Serial.println("WiFi Hotspot IP Address: ");
    Serial.println(WiFi.softAPIP());

    // DAC leftovers
    // Initialize DAC
    // dac_output_enable(DAC_CHANNEL_1);

    // Route for root / webpage
    server.on("/", HTTP_GET, handleRoot);
    // Route to handle slider value updates
    server.on("/update", HTTP_GET, handleUpdate);
    // Start the server
    server.begin();
}

void loop() {
    server.handleClient(); // Handle client requests
}
```



```
// Handle root / webpage
void handleRoot() {
    // HTML content for the webpage with an inverted slider and JavaScript
    String html = "<!DOCTYPE html><html><head><title>Thrust Slider</title>
</head><body>";
    html += "<h1>Thrust Slider</h1>";
    html += "<input type='range' id='slider' name='slider' min='0' max='255'
oninput='updateValue(this.value)' >";
    html += "ontouchmove='updateValue(this.value)'
onmousemove='updateValue(this.value)'><br>";
    html += "<script>function updateValue(value) {";
    html += "var xhttp = new XMLHttpRequest();";
    html += "xhttp.onreadystatechange = function() {";
    html += "if (this.readyState == 4 && this.status == 200) {";
    html += "console.log('Value updated:', value);";
    html += "};";
    html += "xhttp.open('GET', '/update?value=' + value, true);";
    html += "xhttp.send();";
    html += "}</script>";
    html += "</body></html>";

    server.send(200, "text/html", html); // Send HTML response
}

// Handle update request
void handleUpdate() {
    // Get the value of the slider from the request
    String sliderValue = server.arg("value");
    int thrustValue = sliderValue.toInt();

    // Print binary representation of the value
    Serial.print("Thrust Value (Binary): ");
    for (int i = 7; i >= 0; i--) {
        Serial.print((thrustValue >> i) & 1);
    }
    Serial.println(); // New line

    Serial.write(thrustValue); // Send thrust value via Serial

    // DAC leftovers
    // Map the value from 0-255 to 0-255*255 (0-65535) for DAC
    // dacWrite(25, thrustValue);
    // dac_output_voltage(DAC_CHANNEL_1, thrustValue * 255);

    server.send(200, "text/plain", "OK"); // Send response to the client
}
```



הקוד הזה נכתב באמצעות **Arduino ide** ושפת **cpp**. הקוד הזה מTARGET את ההתקנות של **כרטיס ESP**.

הקוד בעצם מאתחל נק' גישה ב**localhost** ועליה מאתחל שרת **HTTP** שרעץ ב**ת"א** (localhost) דמיין אך ורק למי שמחובר ל**WIFI**) בפורט 80.

בפונקציית **setup** הוא מאתחל את התקשרות הסיריאלית על באוד רייט של 115200 ביטים לשניה, כך שיתאים ל**FPGA**. לאחר מכן מאתחל את ה**WIFI** ואת שרת **HTTP** שדרכו אשלווט **throttle** (רוחב גל WMM) של המנועים.

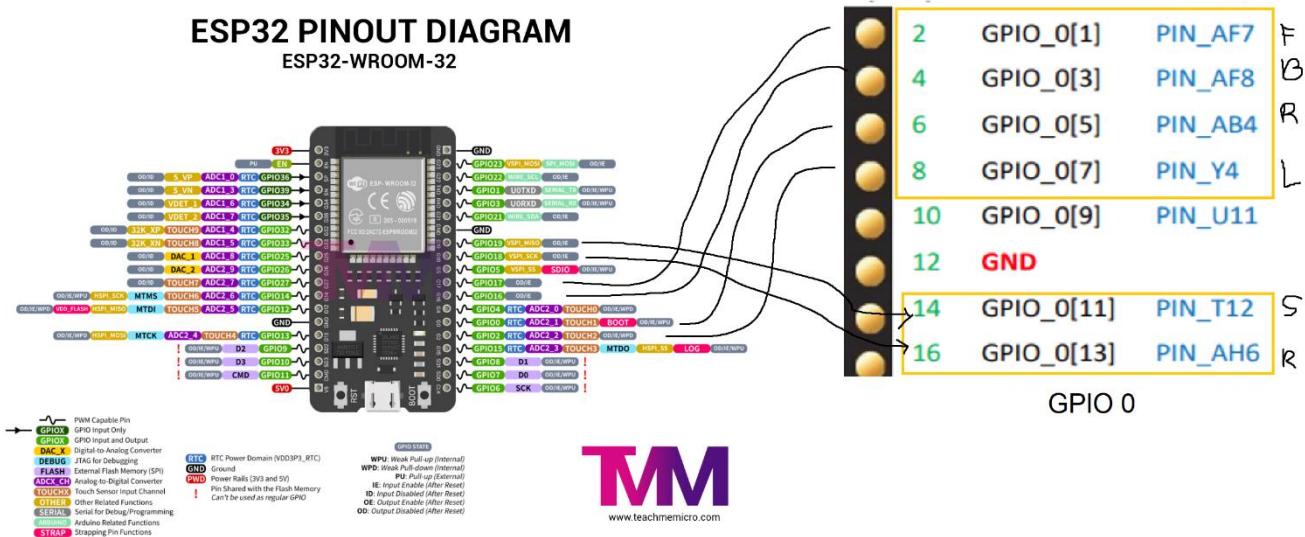
בפונקציית **loop** קראתי לפונקציה בשם **handleRoot** שמטרתה כמו שהיא – להיות אחראי על השורשים/בסיס של האתר, אני שולח לשרת את **html** שמאפיין אותו, לא השקעתה המונע זמן בעיצוב דף השליטה דוקא כי זה היה החלק הכי פחות רלוונטי עבורו. בנוסף יצרתי פונקציה בשם **handleUpdate** שמופעלת אך ורק מתי שהים בדף השליטה מתעדכנים – ניתן לראות בקוד HTML שהטוווח של הסליידר הוא בין 0 לבין 255 ולכן זה מה שישלח ל**handleUpdate**. הוסףתי בפונקציית **handleUpdate** כל מיוני דברים שולטים שמטרתם היא לעזור לי לדבג את הקוד בהתחלה כשהיו איתו בעיות.

השורה הכי רלוונטית היא

```
Serial.write(thrustValue); // Send thrust value via Serial
```

שמה שהוא הוא לשלוח באמצעות **TX** של **ESP** את התמונה הזה שמתאר את הערך של הסליידר (0 ל255). -> מכאן התחומים של **Desp** לפי המשוואות שתיארתי קודם.

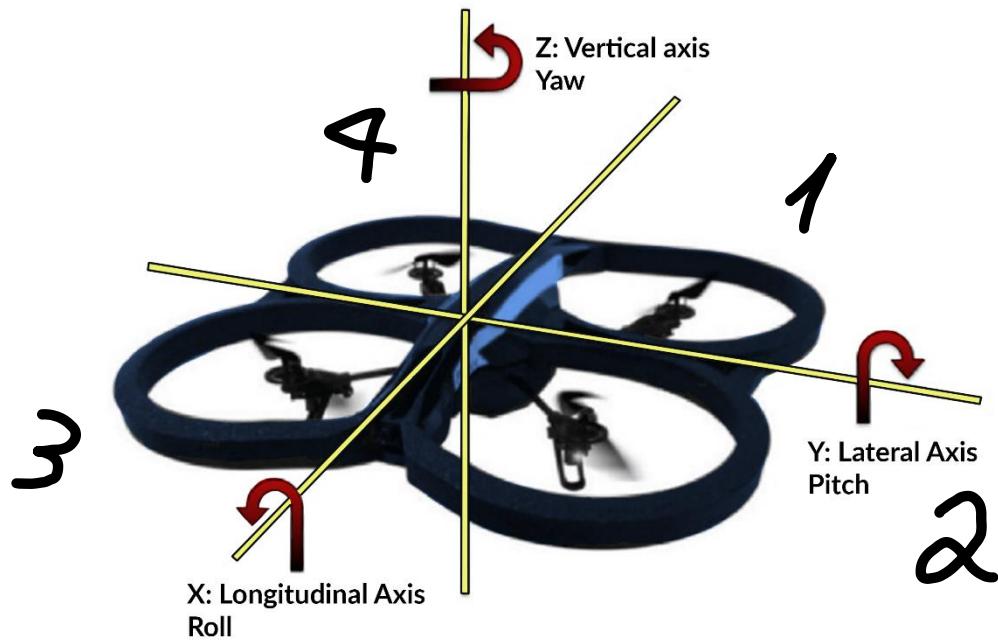
הערה: הקוד עצמו לא השתנה יותר מידי פשוט התווסף עוד כפתורים אז הוא ממש ארוך אבל אין לו פונקציונליות מיוחדות. בהמשך להסביר על הceptors ב-YITY ENTITY, נוספו כפתורים של אתחול המנוועים, אותן איפוס, קידימה אחורה ימינה שמאליה. מציף סכמת חיבור.



הכוונה היא בהמשך – לדעתי כבר אחרי בוחינת הבגרות המיעודת, אפתח פיצרים אוטונומיים כך שאוכל לעלות לאוויר, להזוז על כפתור לדוגמה FORWARD והרחפן ינוע קדימה. כך עם כל הceptors ומן הסתם עם שילוב של יחידת IMU.

תנועה ברחפן

רחפן הוא יצירת מופת הנדסית ופיזיקלית, מעבר לעובדה שהוא מרוחף והוא יכול לבצע פעולות מורכבות כמו תנועה קדימה אחורה ימינה שמאליה, סיבובים סביב עצמו בציר המישורי שמקביל לקרקע ואפיו סיבובים בציר שאנכי לקרקע של כדה"א ועוד. לתנועות אלו יש שמות:



1. סיבוב סביב ציר א (מקביל לקרקע) – **ROLL**
2. סיבוב סביב ציר ז (ניצב ל-X) – **PITCH**
3. סיבוב סביב ציר ז (ניצב לקרקע) – **YAW**
4. לא תנועה סיבובית אבל אכן תנועה: עלייה או ירידת במרחב – **THROTTLE**

על מנת לבצע כל אחד מהפעולות נדרש לבצע מניפולציות על מנועי הרחפן או ליתר דיוק על הפוטופולרים.

- **ROLL** – לבצע חלוקת סימטריה על גוף הרחפן ע"פ ציר X – צד אחד פועל יותר חזק מהשני עד להשלמת סיבוב / חזרה למישור הייחוס (באמצעות IMU)
 - **PITCH** – לבצע חלוקת סימטריה על גוף הרחפן ע"פ ציר Y – צד אחד פועל יותר חזק מהשני עד להשלמת סיבוב / חזרה למישור הייחוס (באמצעות IMU)
 - **YAW** – מנועים באילנסון יפעלו ב-RPM גבוהה יותר ובכך איזון הרחפן בציר הסיבובי יושבר. גוף הרחפן יסתובב לכיוון "אילנסון המנוועים" שפועלים ב-RPM גבוהה יותר.
- מכאן יש נגזרת פשוטה של תנועות בסיסיות עבור המנווע:

1. קדימה – מניפולציה על PITCH: נרצה לתת יותר כוח לזוג מנועים האחוריים ש”ייחפו” את המנועים הקדמיים.
2. אחורה – מניפולציה על PITCH: נרצה לתת יותר כוח לזוג מנועים הקדמיים ש”ייחפו” את המנועים האחוריים.
3. ימינה – מניפולציה על ROLL: נרצה לתת יותר כוח לזוג המנועים השמאליים ש”ייחפו” את המנועים הימניים.
4. שמאלה – מניפולציה על ROLL: נרצה לתת יותר כוח לזוג המנועים הימניים ש”ייחפו” את המנועים השמאליים.

וכמובן יש יכולת לתנוועות יותר מורכבות כמו פעולהים ופליפים באוויר שימושיים גם את yaw והTHROTTLE. כמו בתחום ה”אקסטרים” של תחום הרחפנות – רחפני FPV: FIRST PERSON VIEW. המטיס לבש על עצמו משקפיים שדומים למשקפי VR אך מחוברים בחיבור אלחוטי אל מצלמה שקיימת על הרחפן וכך רואה הכל כפיו הוא על הרחפן עצמו ולכן מתאפשר לבצע פעולות מורכבות שיכולות להגיע לרמת דיוק כירורגית.

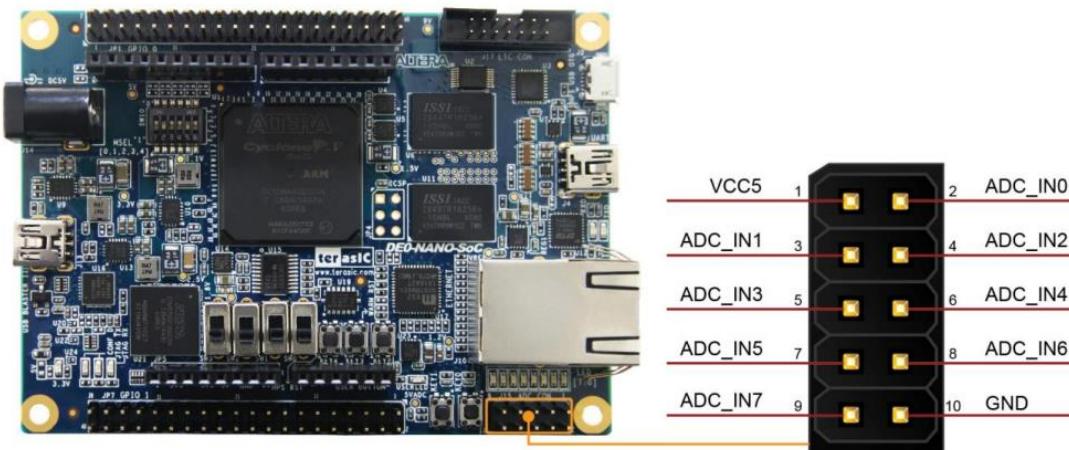


תקשות

לפרק זהה בעמוד שלו יש 2 סאגות ואפרט עליון כאן;
בעבר/תחילת הפיתוח, התרכזתי יותר בהפעלת המנועים, ייצור גל WSW תקין ולמידה על תוכנת ה`ESCs`. כך שבסביל לפטור "זמןית" את הנושא זהה עשויתי את הדבר הבא: חיבורתי פוטנציאומטר אל רכיב ADC הבנוי בתוך הכרטיס. במקרה הזה שמו של הרכיב הוא **LTC2308**, הנה כמה פרטיים עליוו:

The LTC2308 is a low noise, 500ksps, 8-channel, 12-bit ADC with a SPI/MICROWIRE compatible serial interface. This ADC includes an internal reference and a fully differential sample-and-hold circuit to reduce common mode noise. The internal conversion clock allows the external serial output data clock (SCK) to operate at any frequency up to 40MHz.

It can be configured to accept eight input signals at inputs ADC_IN0 through ADC_IN7. These eight input signals are connected to a 2x5 header, as shown in **Figure 3-20**.



אני השתמשתי בצannel 000, את הפוטנציאומטר חיבורתי בהדק אחד ל`VCC5` ובהדק השני לאדמה. היה לי רכיב בתוכנה בשם `adc_ltc2308` וכך היתי מתחל את היישות:

```

ADC_inslt: adc_ltc2308
port map (
-- Serial SDI communication with LTC2308
    ADC_CONVST => ADC_CONVST,
    ADC_SCK => ADC_SCK,
    ADC_SDI => ADC_SDI,
    ADC_SDO => ADC_SDO,

    -- Clock
    clk => outclk_1,

    -- Serial details for SPI
    measure_ch => "000", -- Channel selection
    measure_dataread => data, -- Data !!!!
    measure_done => done, -- Conversion done
    measure_start => measure_start -- Conversion start
);

```



את הקוד עצמו לא עיצבתி, תחילה ניסיתי אך מכיוון שמתעסק בעקרונות ובהליך נורא עמוק בכרטיס וגישה عمוקה לא כל כר הצלחתי למש את התהילה. קוד Verilog

```
module adc_ltc2308(
    clk, // max 40mhz

    // start measure
    measure_start, // posedge trigger
    measure_ch,
    measure_done,
    measure_dataread,

    // adc interface
    ADC_CONVST,
    ADC_SCK,
    ADC_SDI,
    ADC_SDO
);

    input                      clk;
    // start measure
    input                      measure_start;
    input      [2:0]            measure_ch;
    output reg     [11:0]        measure_done;
    output      [11:0]          measure_dataread;

    output                     ADC_CONVST;
    output                     ADC_SCK;
    output reg     ADC_SDI;
    input                      ADC_SDO;

///////////////////////////////
// Timing definition

// using 40MHz clock
// to achieve fsample = 500KHz
// ntcyc = 2us / 25ns = 80

#define DATA_BITS_NUM      12
#define CMD_BITS_NUM       6
#define CH_NUM              8

#define tWHCONV             3 // CONVST High Time, min 20 ns
#define tCONV                64 //52 // tCONV: type 1.3 us, MAX 1.6 us,
1600/25(assumed clk is 40mhz)=64 -> 1.3us/25ns = 52
// set 64 to suit for 1.6 us max
// +12 //data

#define tHCONVST           320 // 12 // here set 320 ( fsample = 100KHz)
for if ADC input impedance is high, see below
// If the source impedance of the driving
circuit is low, the ADC inputs can be driven directly.
//Otherwise, more acquisition time should be
allowed for a source with higher impedance.

#define tCONVST_HIGH_START  0
#define tCONVST_HIGH_END    (`tCONVST_HIGH_START+`tWHCONV)

#define tCONFIG_START        (`tCONVST_HIGH_END)
#define tCONFIG_END          (`tCLK_START+`CMD_BITS_NUM - 1)

#define tCLK_START           (`tCONVST_HIGH_START+`tCONV)
#define tCLK_END              (`tCLK_START+`DATA_BITS_NUM)

#define tDONE                 (`tCLK_END+`tHCONVST)
```



```
// create trigger message: reset_n
reg pre_measure_start;
always @ (posedge clk)
begin
    pre_measure_start <= measure_start;
end

wire reset_n;
assign reset_n = (~pre_measure_start & measure_start) ? 1'b0 : 1'b1;

// tick
reg [15:0] tick;
always @ (posedge clk or negedge reset_n)
begin
    if (~reset_n)
        tick <= 0;
    else if (tick < `tDONE)
        tick <= tick + 1;
end

/////////////////////////////
// ADC_CONVST
assign ADC_CONVST = (tick >= `tCONVST_HIGH_START && tick < `tCONVST_HIGH_END) ? 1'b1 : 1'b0;

/////////////////////////////
// ADC_SCK

reg clk_enable; // must sync to clk in clk low
always @ (negedge clk or negedge reset_n)
begin
    if (~reset_n)
        clk_enable <= 1'b0;
    else if ((tick >= `tCLK_START && tick < `tCLK_END))
        clk_enable <= 1'b1;
    else
        clk_enable <= 1'b0;
end

assign ADC_SCK = clk_enable ? clk : 1'b0;

/////////////////////////////
// read data
reg [(`DATA_BITS_NUM-1):0] read_data;
reg [3:0] write_pos;

assign measure_dataread = read_data;

always @ (negedge clk or negedge reset_n)
begin
    if (~reset_n)
        begin
            read_data <= 0;
            write_pos <= `DATA_BITS_NUM-1;
        end
    else if (clk_enable)
        begin
            read_data[write_pos] <= ADC_SD0;
            write_pos <= write_pos - 1;
        end
    end
end
```



```
///////////
// measure done
wire read_ch_done;

assign read_ch_done = (tick == 'd107) ? 1'b1 : 1'b0;

always @ (posedge clk or negedge reset_n)
begin
    if (~reset_n)
        measure_done <= 1'b0;
    else if (tick == 'd108)
        measure_done <= 1'b1;
    else
        measure_done <= 1'b0;
end

///////////
// adc channel config

// pre-build config command
reg [(`CMD_BITS_NUM-1):0] config_cmd;

`define UNI_MODE      1'b1 //1: Unipolar, 0:Bipolar
`define SLP_MODE       1'b0 //1: enable sleep

always @(negedge reset_n)
begin
    if (~reset_n)
        begin
            case (measure_ch)
                0 : config_cmd <= {4'h8, `UNI_MODE, `SLP_MODE};
                1 : config_cmd <= {4'hC, `UNI_MODE, `SLP_MODE};
                2 : config_cmd <= {4'h9, `UNI_MODE, `SLP_MODE};
                3 : config_cmd <= {4'hD, `UNI_MODE, `SLP_MODE};
                4 : config_cmd <= {4'hA, `UNI_MODE, `SLP_MODE};
                5 : config_cmd <= {4'hE, `UNI_MODE, `SLP_MODE};
                6 : config_cmd <= {4'hB, `UNI_MODE, `SLP_MODE};
                7 : config_cmd <= {4'hF, `UNI_MODE, `SLP_MODE};
                default : config_cmd <= {4'hF, 2'b00};
            endcase
        end
    end
end

// serial config command to adc chip
wire config_init;
wire config_enable;
wire config_done;
reg [2:0] sdi_index;

assign config_init = (tick == `tCONFIG_START) ? 1'b1 : 1'b0;
assign config_enable = (tick > `tCLK_START && tick <= `tCONFIG_END) ? 1'b1 :
1'b0; // > because this is negative edge trigger
assign config_done = (tick > `tCONFIG_END) ? 1'b1 : 1'b0;
always @ (posedge clk)
begin
    if (config_init)
        begin
            ADC_SDI <= config_cmd[`CMD_BITS_NUM-1];
            sdi_index <= `CMD_BITS_NUM-2;
        end
    else if (config_enable)
        begin
            ADC_SDI <= config_cmd[sdi_index];
            sdi_index <= sdi_index - 1;
        end
    else if (config_done)
        ADC_SDI <= 1'b0;
end
endmodule
```

בצורה מופשטת – מטרת הקוד היא לתקשר עם יחידת ה-**ADC ltc2308**, הקוד כולל בתוכו קאונטר של 16 ביטים למטרת הקפדה על זמנים ופלטים סטנדרטיים של כל תקשורת כמו **measure_ch**, **measure_start = enable** ו**dataready done**.

כעת אני משתמש במצבה חדשה יותר אשר עובדת טוב יותר למטרת רחפן – דבר שדורש שליטה מרוחק ולא יכול להיות מוגבל על ידי חוטים וכן היא WiFi.



כשידרתי שאין מכוון ל-**WiFi** בחרתי בברkr ה- **ESP**. אך מחשבתי הייתה כי מדובר עם ה-**ESP** וה**ESP** יוציאו אונלוגי לא**FPGA** כך שלא נדרש לשנות את המשק הקיים שעובד. לכן בחרתי ב**ESP32** שכולל בתוכו **DAC** ביליט-אין.

לאחר התנסות עם ה-**ESP** לפני הגישה הראשונה שלי, זה עבד. הצלחתי לתקשורת חצי דרך עם **WIFI** וחצי דרך עם אונלוגי אך זה היה מועד לפורענותם בഗל' שאוט אונלוגי יכול לקבל הפרעות כל כך בקלות, במיוחד ברוחפן שודד ורועד. הנה הקוד שראה כיצד יכולתי לשולח אונלוגיים באמצעות הרכיב על גבי ה-**ESP32**:

(מאתחל את DAC, לאחר מכן כותב לו והזע)

שלוח רמות מתח מתאימות

```
#include <WiFi.h>
#include <WebServer.h>
#include "driver/dac.h"

// WiFi Hotspot credentials
const char* ssid = "DroneWiFi"; // WiFi Hotspot SSID
const char* password = ""; // No password

WebServer server(80); // Create a WebServer on port 80

void setup() {
    Serial.begin(115200);

    // Set up WiFi hotspot
    WiFi.softAP(ssid, password);

    // Print the IP address of the WiFi hotspot
    Serial.println("WiFi Hotspot IP Address: ");
    Serial.println(WiFi.softAPIP());

    // DAC leftovers
    // Initialize DAC
    dac_output_enable(DAC_CHANNEL_1);

    // Route for root / webpage
    server.on("/", HTTP_GET, handleRoot);
    // Route to handle slider value updates
    server.on("/update", HTTP_GET, handleUpdate);
    // Start the server
    server.begin();
}

void loop() {
    server.handleClient(); // Handle client requests
}
```

```
// Handle root / webpage
void handleRoot() {
    // HTML content for the webpage with an inverted slider and JavaScript
    String html = "<!DOCTYPE html><html><head><title>Thrust Slider</title></head><body>";
    html += "<h1>Thrust Slider</h1>";
    html += "<input type='range' id='slider' name='slider' min='0' max='255' ";
    html += "oninput='updateValue(this.value)'>";
    html += "onmousemove='updateValue(this.value)'><br>";
    html += "<script>function updateValue(value) {";
    html += "var xhttp = new XMLHttpRequest();";
    html += "xhttp.onreadystatechange = function() {";
    html += "if (this.readyState == 4 && this.status == 200) {";
    html += "console.log('Value updated:', value);";
    html += "}}";
    html += "xhttp.open('GET', '/update?value=' + value, true);";
    html += "xhttp.send();";
    html += "};</script>";
    html += "</body></html>";

    server.send(200, "text/html", html); // Send HTML response
}

// Handle update request
void handleUpdate() {
    // Get the value of the slider from the request
    String sliderValue = server.arg("value");
    int thrustValue = sliderValue.toInt();

    // Print binary representation of the value
    Serial.print("Thrust Value (Binary): ");
    for (int i = 7; i >= 0; i--) {
        Serial.print((thrustValue >> i) & 1);
    }
    Serial.println(); // New line

    Serial.write(thrustValue); // Send thrust value via Serial

    // DAC leftovers
    // Map the value from 0-255 to 0-255*255 (0-65535) for DAC
    // dacWrite(25, thrustValue);
    // dac_output_voltage(DAC_CHANNEL_1, thrustValue * 255);

    server.send(200, "text/plain", "OK"); // Send response to the client
}
```



ולבסוף – הגרסת האחרונה וההכי טובה;

התהlixir הוא כך: אני מייצר תקשורת WiFi עם הESP32 באמצעות נק' הגישה שהוא מייצר מתוכו. אני מתחבר אל שרת HTTP שלו ובוחר ערכיהם ל throttle. הוא דואג לאתחל תקשורת סיריאלית ב115200 ביטים לשנייה(באוד ריבט) כך שיהיה תואם לFPGA. כל ערך שאבחר בסליידר בדף הנחיתה הוא ישלח as – כמספר פשוט. כך נחשבת המונע עבודה ביןאריות ותסכולים עמוקים שיכולים לנבוע מتوقفות טبع פשוטות כמו רוחות שמציגות כבלים או גלים אלקטرومגנטיים שישבשו.

```
#include <WiFi.h>
#include <WebServer.h>
#include "driver/dac.h"

// WiFi Hotspot credentials
const char* ssid = "DroneWiFi"; // WiFi Hotspot SSID
const char* password = ""; // No password

WebServer server(80); // Create a WebServer on port 80

void setup() {
    Serial.begin(115200);

    // Set up WiFi hotspot
    WiFi.softAP(ssid, password);

    // Print the IP address of the WiFi hotspot
    Serial.println("WiFi Hotspot IP Address: ");
    Serial.println(WiFi.softAPIP());

    // DAC leftovers
    // Initialize DAC
    // dac_output_enable(DAC_CHANNEL_1);

    // Route for root / webpage
    server.on("/", HTTP_GET, handleRoot);
    // Route to handle slider value updates
    server.on("/update", HTTP_GET, handleUpdate);
    // Start the server
    server.begin();
}

void loop() {
    server.handleClient(); // Handle client requests
}
```

הדפסות תלת ממד

הדפסה תלת-ממדית היא טכנולוגיית ייצור, המאפשרת ליצור דגמים תלת-ממדיים הקיימים מתוך המחשב. לרוב, מכונות הדפסה מייצרות את הדגמים (אבטיפוס) מפולימרים שונים, שכבה אחרי שכבה, ולבסוף מתקבל דגם סופי, כפי שתוכנן בתוכנות התב"ם (תוכנן בעזרת מחשב - CAD).

תוכנת סלייסר היא תוכנה לייצור מסלולי הדפסה המשמשת בהדפסת תלת-ממד. תוכנה זו ממירה מודל תלת-ממדי שנוצר בפורמט (Stereolithography) STL להוראות ספצייפיות עבור הדפסת בפורמט G-code. התהליך כולל חיתוך המודל לשכבות דקות והגדלת מסלול הדפסה לכל שכבה, כך שהמדפסת תוכל לבנות את המודל שכבה אחר שכבה.

במהלך התהליך, תוכנת הסלייסר מגדרה את הפרמטרים הבאים:

- עובי כל שכבה.
- מהירות הדפסה.
- טמפרטורת הדפסה.
- דפוס המילוי הפנימי של המודל.
- תומכים (Supports) וمبرנים נוספים הנדרשים להדפסת חלקים מורכבים.

הסלייסר הוא כלי חיוני בתהליך הדפסה בתלת-ממד מכיוון שהוא מתרגם את המודל התלת-ממדי לקובץ הוראות מפורטות שהמדפסת יכולה להבין ולבצע.

בפרויקט שלי אני משתמש בתוכנה אשר משלבת בתוכה **CAD SLICER1** – **Bambu Studio**.
היות והדפסת של ביה"ס היא מדפסת P1S של חברת Bambu, החברה מספקת תוכנת "בית" שמשתיכת לכל המדפסות ובulant פיצרים של **SLICER1 CAD**.

הלייר הבנין

כמובן שגם חלק זהו נדרש להגיע.. אין אפשר לעשות ספר שלם על רחפן בלי לדבר על איך בניתו בכלל?

כמובן שגם כל חלק בפרויקט גם הוא עבר אבולוציה רבה...

אתחיל מכך שתודות לكريית החינוך ע"ש גינסבורג ולמרצה שלנו רפי אמלסם יש לרשوت ביה"ס 3 מדפסות תלת מימד מצוינות שאפשרו את קידום הפרויקט ואספקט נוסף בפרויקט זהה מלבד החומרה והקוד אלא גם עיצוב והנדוס חומרים ומבנה.

תחילה התעסקתי רק בקילibrציות(Calibrations) פשוטות על מנת לסדר את המדפסת ולהוות בסיס להדפסה. יש לי ידע בסיסי ביוטר בבלנדר וSolidworks כך שלא היה קשה במיוחד אבל כן היה מאוד מתאים..

בorschיה הראשונה חיפשתי בסיס לשילדה מוכנה – ניתקלתי בקובץ ראשון שהיה נראה מבטיח במיוחד וכך היה נראה:



והיה נראה טוב במיוחד – אך לא חשוב על המקום הנדרש כי הרוי כרטיס FPGA ובקרי מהירות אישיות לכל מנווע (לעומת המוצרים הקיימים בשוק כתע.. – ESC 4in1 ... כל אלה דורותים המון מקום. הדפסתי:



בקצורה; יצא מזעע. בגלל שזו הייתה ההדפסה הראשונה ה"אמיתית" שלי, כל הגדרות Slicer שלי היו היפות. fill חלקי, בחלק מהמקומות לא היה סופורטים טובים, קירות פנימיים בהדפסה ואפלו קירות חיצוניים (לא היו גבולות חיצוניים – שום דבר לא היה חלק באמת והתפרק בקלות). עברתי להלא עם המסקנות האלה: הגדרות של סייר והדפסה, שטח פנימי גדול בשבייל כל הרכיבים

והמשכתי בחיפוש אחר שלדה שיאפשר תתאים לצרכים שלי ונטקלתי בדבר הבא:

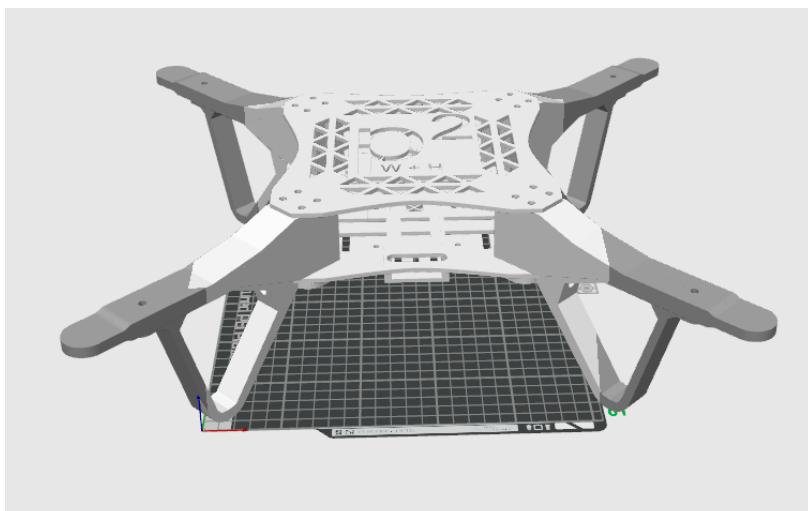


<https://grabcad.com/library/drone-frame-2>

ועל כן נאמר: **perfect!** 😊 וצר התחלתי בפיתוח פרוטווטיפ ראשון;

העברתי את קובץ STEP העיקרי (—קובץ המכיל אובייקטים תלת ממדים) אל תוכנת הסליינר שלי (Bambu Studio) ומכאן התחלתי לפרק את הרחפן לחלקים נפרדים, שאוכל לעבוד על כל חלק בנפרד ולבצע שינויים, וכן כMOVIN גם להדפיס אותם. המדפסת לא גדולה מספיק 😊.

בתמונה: המודל בתוך תוכנת הסליינר שלי.





כמו כן בתמונה רשימה האובייקטים שמרכיבים את step: (שמאל)

drone frame(onder...)

drone arm

drone arm

drone arm

drone arm

voet

voet

voet

voet

batterijhouder

drone frame (bove...

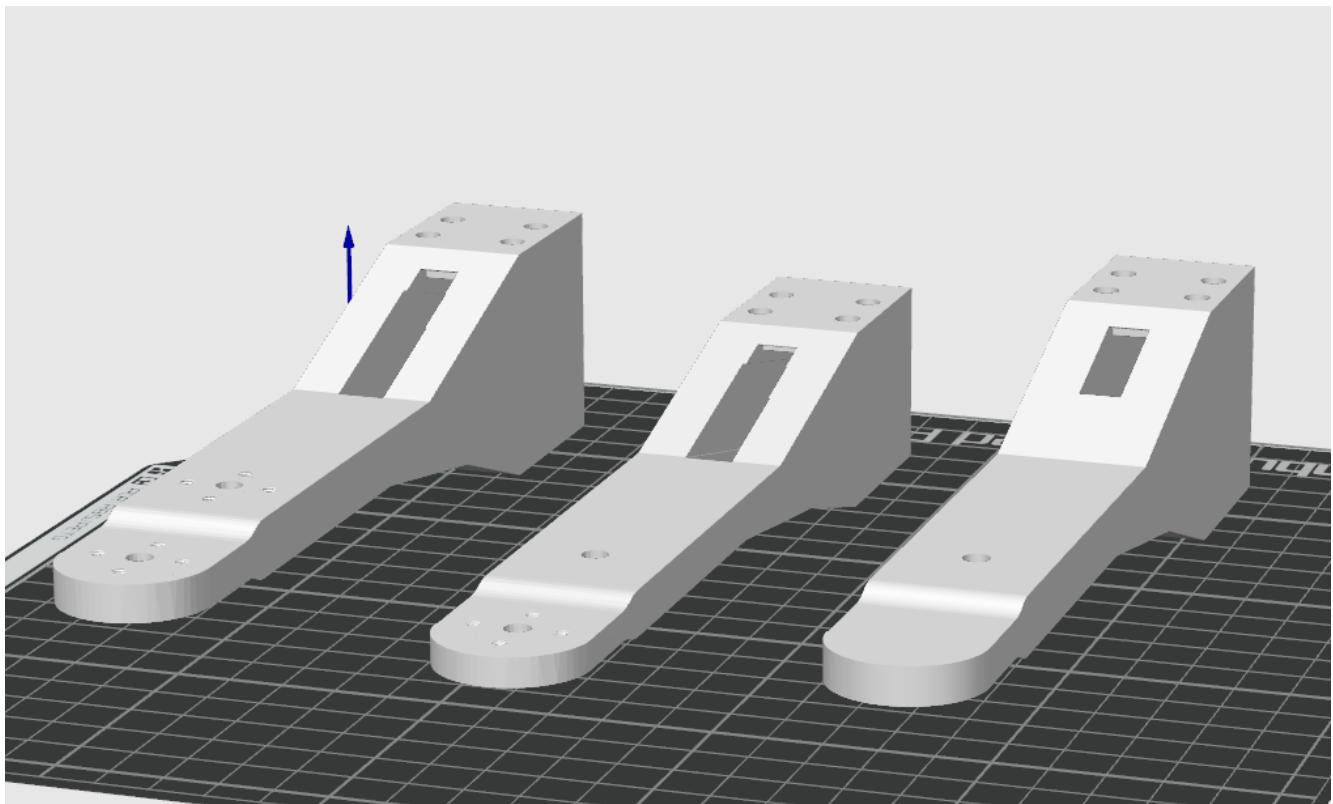
אלה הולנדיתים...

	droneARM1.stl
	droneARM2.stl
	droneARM3.stl
	droneARM4.stl
	droneBatteryHolder.stl
	droneBUTTON.stl
	droneLEG1.stl
	droneLEG2.stl
	droneLEG3.stl
	droneLEG4.stl
	droneTOP.stl

לאחר שפרקתי את קובץ האב לקבצים קטנים שמכילים כל אחד חלק מהקובץ: (ימין) יכולתי להתחיל לעבוד! בעבודתי הבנתי כי כל הרגליים זהות לחלוטין אך בשיבוב שונה (כרגע עם הזרועות) לכן מכל הקבצים ובדקתי על קובץ אחד מכל סוג והדפסתי אותו 4 פעמים.

אבולוציה של הזרוע:

הזרוע של הרחפן עברה ורسيות רבות על מנת להתאים לצרכי: האפשרות לחבר את המנועים היבט באמצעות ברגים M3, חור מספיק רחב ואורך על מנת להעביר את הקבלים של המנועים (3 כבלים – 3 פאוזות שונות) ואפילו לאחסן את בקר המהירות בפנים.



בתמונה: 3 הגרסאות, ימין הכיו יישן ושמאל הכיו חדש. ניתן לראות שבייצועי שינויים בגובה הזרוע, חורים בקצוות אשר מדדי במיוחד שיתאימו לstand של המנועים עם קוטר 3 מ"מ וגם חור רחב יותר וגובה יותר כך שאוכל להעביר את הקבלים ואת בקר המהירות למרכז הרחפן – היכן שהCPU FPGA נמצא.

מעבר לזרוע בייצורי מס' שניינים קענים, בעיקר אסתטיים:

החליטתי שלא להשתמש ברגליים, שכן הוא רק מוסף משקל לרחפן, ככל שאקבע את כל האלקטרוניקה הפנימית של הרחפן כרך גם לא יהיה קרייטי אם יוחזק מכחה פה ושמה. מעבר לכך הופתעתי לפתוח בפלטות התחתונות והעליונות. וכך גם התעלמתי מהמחזיק לסוללה, במקום פשוט לחבר את הסוללה בפלטה העליונה מעל הרחפן, לחבר אותה עם סקוצים מיוחדים לסוללות על רחפני FPV.

קצת ניסויים

כחול מההכנות שלי לאופטימיזציה לרחפן הכרתי חומר בשם **W-L-ePLA** והוא חומר דמו^ר PLA אך **LightWeight**! החומר מיועד לדגמי מל"י טים ורחפנים בשל צפיפותו הנמוכה. בתרור החומר יש שימוש בטכנולוגיית הקזפה שמאפשרת יחס נפח קצף של כ-220%. ד"א פי 2.2 קל יותר מאשר אותו דגם באותו מידות בחומר PLA גנייל. כאן ניתן לראות תמונה ראשונה מהתהליכי התנסות עם החומר:



תוצאה סופית: (חלול וקל!)

אר לבסוף בחרתי שלא להשתמש בחומר זהה מכיוון שאינו עמיד כל!

נשבר בקלות

הרכבת הרחפן עצמו

לאחר שאספתי את כל החלקים הנדרשים – בניהם גם בריגים באורךים שונים, אוממים **keylock** שלא ישתחרר מהרעד של הרחפן, כל הלחמות הרלוונטיות לשלב הראשוני. ניתן להתחיל להרכיב!



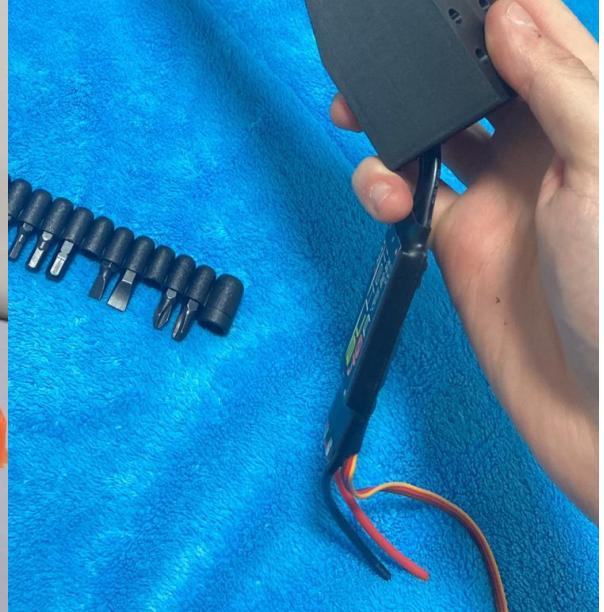
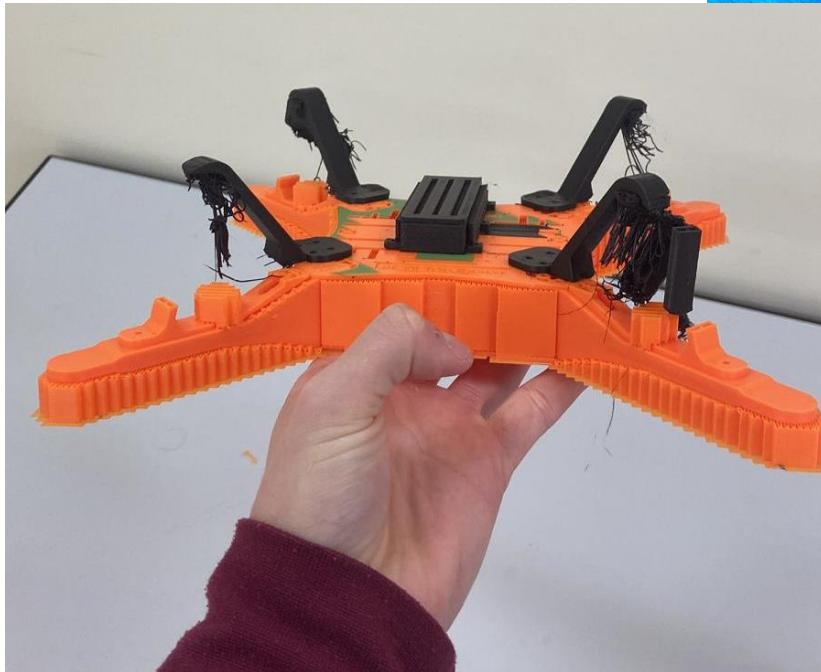
לאחר 4 שעות עבודה שככלו התנסות במבנה וגם קצר קוד כי התחשק...



עוד קצת תמונות:



כמובן לפעמים שיש גם פאשלות בדרכ...



וכעת לייצור המופת הסופית :



מערכת על שבב – System on a chip

מערכת על שבב (באנגלית: **System on a Chip**, או בקיצור **SOC**) היא כינוי למעגל משולב אשר מכיל בתוכו את רוב תפקידיו של מחשב שלם: מעבד, זיכרון, בקרים שונים לאמצעי קלט ופלט ולפעמים גם אמצעי אחסון. מערכות אלו נפוצות במיוחד בתחום המערכות המשובצות, טאבלטים וטלפוןים חכמים.

איך זה מתקשר אליו? אסביר – הרכיב שאני משתמש בו **SOC FPGA DE0 NANO** כפי שהוא, זה קרטיס שמכיל בתוכו 2 חלקים מרכזים –

- **FPGA**: חלק לעיבוד לוגי, סביבה שבוססת על רכיב **VH** **Cyclone** של אלטרה. סביבת **FPGA** כוללת בתוכה מערכיים "גמיישום" של רכיבים דיגיטליים (לוגיים), צמתים גמיישום ובלוקים של קלט/פלט שיוכולים להיות מתוכנתים בהתאם לצורכי המפתח.
- **ARM**: חלק לעיבוד מתמטי, סביבה מבוססת מעבד. סביבת **SOC** מתאימה יותר ותואמת יותר לשביבה "מסורתית" לפיתוח ותוכנות, היא בנואה על מעבד **CPU** שעליו מרכיבים סביבה הפעלה, לרובה נגררת של לינוקס, מאפשר הרצה של שפות פיתוח **Low level**.

הקרטיס מבצע אינגרציה של 2 המערכות באמצעות רביים; חלק מהם הם גישה משותפת של **ARM** ושל **FPGA** אל הדיזרון ה"נדיל" של הקרטיס ופריפריאליים נוספים כגון רגלי **GPIOS**. וכן גם מכיל בתוכו מערכת **HPS** (**Hard Processor System**) שמטרתה היא להתmeshק עם **FPGA** ועם **ARM** ולהעביר בינהם מידע.

בשלב תכנון ארכיטקטורת הפרויקט יש חשיבות רבה להיכן כל דבר קורה בקרטיס, אלו דברים שיוכולים להשפיע רבות על זמן הרצה התוכנה והם חלק מרכזי באופטימיזציה של המערכת.

כלי עזר ל-SoC

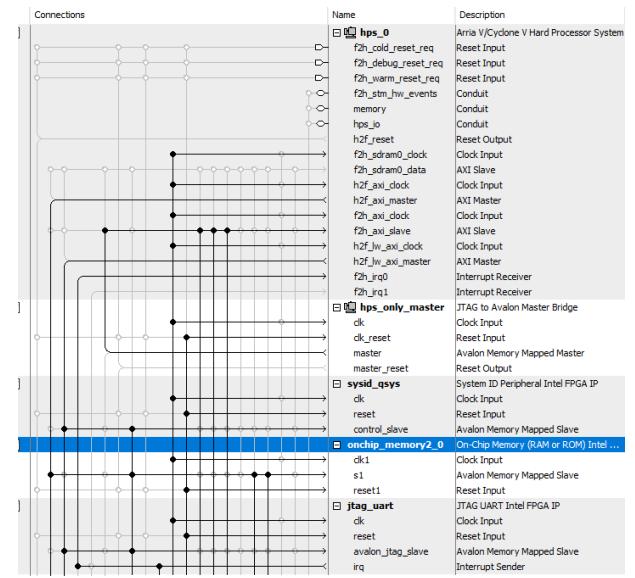
פלטפורמת עיצוב SoC: SoC הוא כלי בתוך חבילת התוכנה של Intel Quartus Prime המשמש לתכנון וシילוב מערכות על גבי התקני FPGA ו-SoC. הכלי מספק משק גרפי לחיבור בלוקי IP שונים, אשר יכולים לכלול מעבדים, משAKER זיכרון ולוגיקה.

באמצעות פלטפורמה זו ניתן לעצב דברים רבים בארכיטקטורה של הפרויקט כגון:

- **חיווט אוגרים (Registers .. לא החיה) מהARM.** ד"א זכרון מושותף כפי שהסבירתי קודם.
- **חיווט בלוקי לוגיקה (IP) אחד לשני (וגם ל-HPS)** לפי הפורטים המתאים, כדוגמה **תמונה 1**.
- **ייצוא (Export) של אוגרים/פורטים לבлокי IP אל HPS,** כדוגמה **תמונה 2**.
- **הקצתה של כתובות זכרון לאוגרים,** כדוגמה **תמונה 3**.

Name	Description	Export
unreset	reset output	
pio_led	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> pio_led_external_connection
pio_reg1	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> pio_reg1_external_connection
pio_reg2	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> pio_reg2_external_connection
pio_reg3	PIO (Parallel I/O) Intel FPGA IP Clock Input Reset Input s1 external_connection	<i>Double-click to export</i> <i>Double-click to export</i> <i>Double-click to export</i> pio_reg3_external_connection

תמונה 2, הכונה לעמודת EXPORT



תמונה 1

System: soc_system Path: onchip_memory2_0	hps_0.h2f_axi_master	hps_0.h2f_axi_master	hps_only_master.master	mm_clock_crossing_bridge_0.m0
MemoryDMA.csr				
dmi_0.control_port_slave				
hps_0.f2h_sdram0_data		0x0000_0040 - 0x0000_007f		
hps_0.f2h_axi_slave		0x0000_0080 - 0x0000_009f		
intr_capturer_0_avalon_slave_0	??		0x0000_0000 - 0xffffffff	
jtag_uart.avilon_jtag Slave	??	0x0002_0000 - 0x0002_0007		
mm_clock_crossing_bridge_0.s0		0x0003_0000 - 0x0003_000f		
mifbus.s1				0x0000 - 0x000f
onchip_memory2_0.s1	ff	0x0000_0000 - 0x0000_ffff		
onchip_memory2_1.s1		0x0002_0000 - 0x0002_ffff		
pio_led.s1		0x0001_0000 - 0x0001_000f		
pio_reg1.s1		0x0000_0010 - 0x0000_001f		
pio_reg2.s1		0x0000_0020 - 0x0000_002f		
pio_reg3.s1		0x0000_0030 - 0x0000_003f		
sysid_qsys.control_slave	??	0x0001_0000 - 0x0001_0007		
mybus.s1.via mm_clock_crossing_bridge_0.m0		0x0003_0000 - 0x0003_000f		

תמונה 3

כיצד זה רלוונטי לפרויקט שלי:

מימוש של סביבת ARM בפרויקט שלי

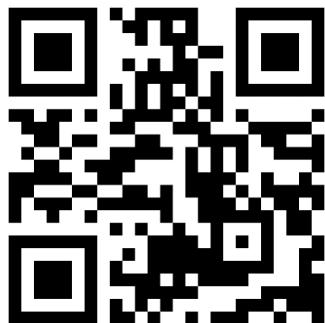
הຕכנון הראשוני שלי היה שאממש את התקשרות הסיריאלית ב C במקום ב FPGA וכן "אשפר" את מהירות הריצה וأكل על ה FPGA.

הגדרתי את מטרותי בניסויים הראשוניים בתור שני מטרות מופשטות מאוד –

- דרך ARM לשולח 2 ביטים בנפרד (2 מספרים עד 8 ביט) אל הזיכרון ולמשוך אותם מה FPGA, לבצע במספרים פעולות אРИטמטיות ולהחזיר לARM תוצאה דרך הזיכרון הפריפריאלי.
- לשולח אותן כלשהו (לצורך העניין במקורה הזה בית בודד) דרך הזיכרון הפריפריאלי אל ה FPGA, לבצע עליו שער לוגי כלשהו וליוציא אותו אל רגלי GPIO (מהלינוקס עד לפעולת אקטיבית –яд, מנוע וכו')

התחלתי ניסויים ראשוניים עם SOC כך:

1. השתמשתי בעיצוב GHARD (Golden Hardware Reference Design) – עיצוב שמסופק ע"י יצרני SoC & FPGA שמשרת כרפנס למפתחים. ה GHRD כולל בתוכו מס' קונפיגורציות עבור ה FPGA וסביבת ARM, כמו כן הוא בעצם הבסיס להPS והוא מקשר בין ה FPGA לARM.



לצערי הוא ארוך מדי ☺☺☺ אוד אצוף QR CODE לקוד אליו :

2. בהמשך ל GHARD, השתמשתי בפלטפורמת עיצוב SoC על מנת לחווט את בЛОקי ה IP אל ממשק HPS, כמו כן הקצתתי כתובות זיכרון לARM להיות ובנישון שלי להשתמש בARM רציתי למשם כמה שיותר עקרונות ופתרונות. תמונה 2 בעמוד הקודם היא דוגמה של הבלוקים שיצרהתי, אליהם התחברתי מצד אחד עם ה EXPORT ומצד שני עם ה GHRD FPGA. צ"א GHARD המתווור. בתמונה 3 ניתן לראות את ההקצאות זיכרון (יהיו רלוונטיות תכף)

3. בנוסף בפרויקט שלי בQUARTUS הוסיף קובץ VHDL בשם **SimpleAdd** שמתממש את 2 הדברים שרציתי לבדוק בניסויים, מצ"ב תמונה של הקובץ. וניתן לראות בסוף קובץ הGRHD קרואתי לישות של **SimpleAdd**.

```
library ieee;
use ieee.std_logic_1164.all;
use IEEE.numeric_std.all;

entity SimpleAdd is
  port
  (
    reg1      : in std_logic_vector(7 downto 0);
    reg2      : in std_logic_vector(7 downto 0);
    reg3      : out std_logic_vector(7 downto 0);
    te        : in std_logic;
    teSS      : out std_logic
  );
end entity;

architecture rtl of SimpleAdd is
begin
  teSS <= not te;
  reg3 <= std_logic_vector(unsigned(reg1) + unsigned(reg2));
end rtl;
```

וכעת אגש אל ARM עצמו, סביבת האימים. (סתם, היה כיף)

swinfo2header.

השתמשתי בכללי שיש לפלטפורמת העיצוב בשם: **"swinfo2header"**.
כללי זה מייצר עבורו קבצי **header** לקוד C עם כל הקצאות

הזכרון במערכת, כתובות של אוגרים ועוד.

השתמשתי בבסיס שפת C שמשכתי מהיזכרן, **Altera**.

בסיס זה מהו יסוד לשימוש בARM והתקשרות

לפרופראליים בcrcvis. הבסיס מכיל בתוכו מקטע קוד

ראשוני אשר מתחבר לכתובת במעבד **"/dev/mem"**

שמאפשרת גישה רחבה לכל הפרופראליים המשותפים אשר

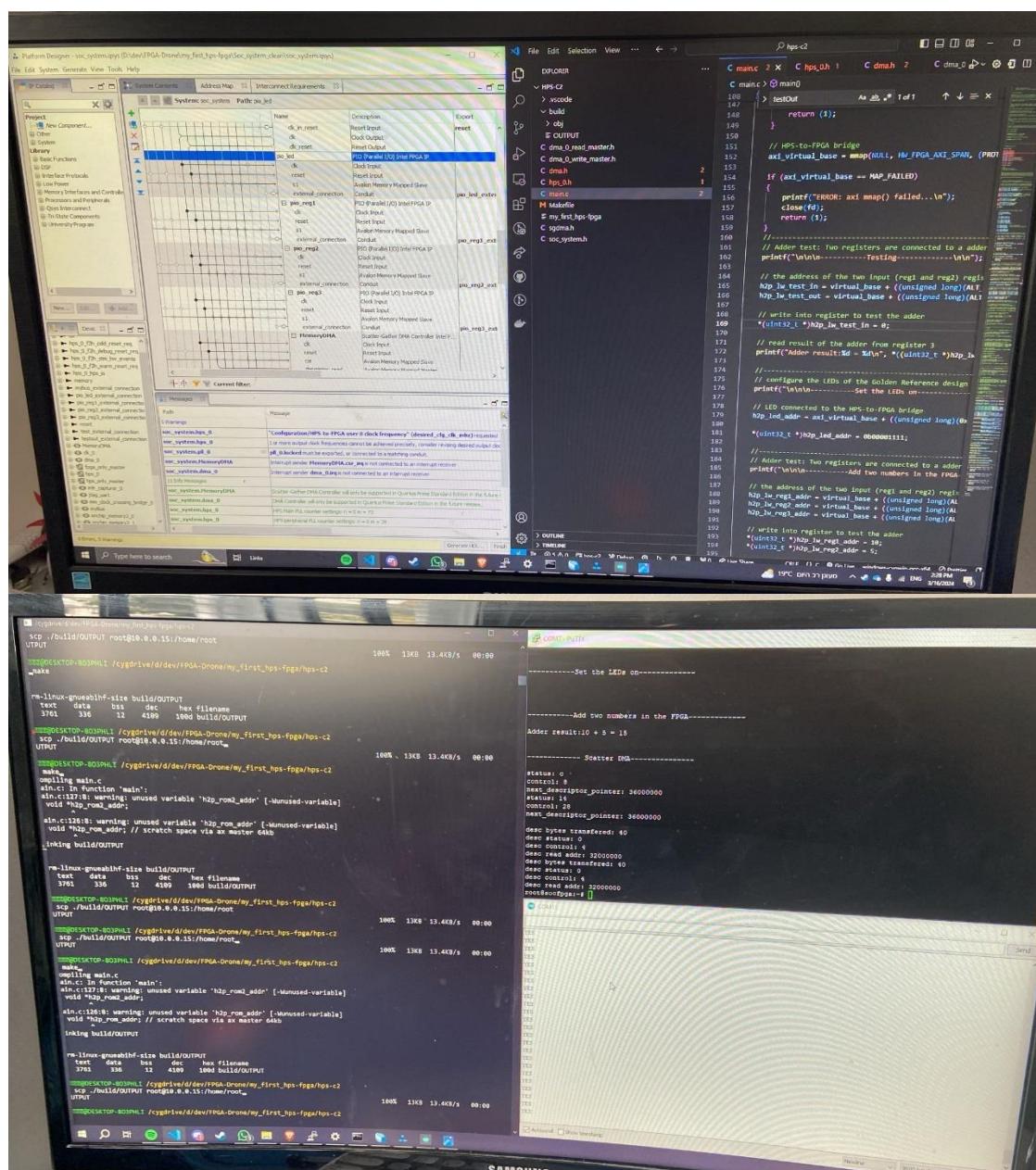
תיארתי קודם. כמו כן הקוד מייצר את ה"גשר" בין המתוור(HPS) לבין FPGA, וכן נותן גישה לכל הכתובות של האוגרים לפי עיצוב הארכיטקטורה **QSYS**.

כמו מקודם עם הGRHD, הקוד אורך מידיו לתמונות בספר וזה עולה לי כסף על כל דף... 😞 מצורף
makefile שמקשור לתיקיה בrepository **headers** בגייטהב עם כל הרכונטיים, QR CODE ו- **main.c**.



מצטער שאנו קצת מזוכיסט.... החלק הבא מדבר על כיצד שילבתי בין השביבות: איר קימפלטי קובץ C, בצורה שמיועדת בכלל לשביבה לינוקסית על גבי שביתת וינדוס, כיצד התקשרתי לARM, מילאתי אותו בבילד המוגמר והרצתי פקודות על המעבד דרך המחשב שלי.

1. תחילת הרעיון היה לייצר `sandbox` לינוקסי ועליו אקמפל את הקוד באמצעות קובץ `makefile` אך לאחר ניסיון בו יצרתי `sandbox` מבוססת `ubuntu` או `linux`, העברת הקבצים בין `host` (windows) (`child`) לבין `host` (Linux), הורדת קומפイルר C וקימפל, העברת הקובץ המקומפל אל `host` ואותו תחילה לARM. התהליך לקח לי המון זמן, היה מסורבל וקצת CAB לי הראש מכל התוכנות הפעילות בתהילן אז מצאתי דרך קלה יותר;
2. חלק מהבאנדל של תוכנות Altera היא תוכנת CMD בשם EDS. בקצתה: SoC EDS הוא חיבורו של utilities שמטרתם לתמוך בפיתוח SoC ובכל היוצא מכך, חלק מסוימת היא חיבורו של קובץ EDS ניתן לקרוא לפועלה `make` שפועלת לפי ה`makefile` עם שנמצא באותה התקינה, מקמפל את הקבצים לפי `makefile` ומיציא תיקית `build` עם הקובץ הסופי.
3. בשביל להעביר את הקובץ הסופי אל השביבה הלינוקסית בARM, ביצעתני מניפולציה באמצעות רשותות. חיבורתי את FPGA אל כבל רשת, וכן גם את המחשב שלי ולאחר מכן לינוקסית כלל. באמצעות EDS ניתן לקרוא לפועלה `make` שפועלת לפי ה`makefile` עם וCCR היינו ל-2 מכשירים באותה רשת ENTHERNET ויכולתי להעביר בניהם קבצים.
4. בTerminal Command Line יש פקודה בשם `scp` שמטרתה היא להעביר קבצים על גבי פרוטוקול SSH, כך יוכל להעביר תיקיה/קובץ מסוימים על גבי האלחות אל ה-ARM.
5. בשビル להתחבר אל ה-ARM השתמשתי בPuTTY Client. בקצתה: תוכנה חינמית ופתוחה שמאפשרת למשתמשים להתחבר למערכות מרוחקות על ידי פרוטוקולים שונים כמו SSH, Telnet וחיבורים קווים. PuTTY משמשת כTERMinal להרצה פקודות ולהעברת קבצים באופן מאובטח על רשותות שונות/קשרים חוטיים. באמצעות COM של המחשב (MASK USB) התחברתי אל הפורט של ARM על גבי הכרטיס, והגדרתי את הPUTTY על המחשב שלו שיווה על 115200 baudrate כמו ה-ARM עצמו. ובכך הייתה לי גישה לשביבה הלינוקסית, בה השם משתמש הוא `root` והסיסמה `root` והרצתי את הקובץ המקומפל שהעbertei באמצעות `scp`.



**בתמונה: חיבור של הכרטיס
למחשב(שchor)
ולאינטראנט(כבל רשות צהוב)**



מחשובות עתידיות

בשעה טובה סיימנו עם הعبر והוועה.. אני חושב שיש מקום ראוי לעתיד!
קצת הקדמה ונתخيل: (

מטריצות

הקדמה

מטריצות הן כתוב מתמטי למטרה סיידור מערכת משווהות בצורה שונה.

$$\begin{bmatrix} 4 & 3 & 4 \\ 2 & 5 & 1 \end{bmatrix}$$

לדוגמה מטריצה יכולה להיראות כך:

מטריצה מחולקת לשורות ועמודות, ומוגדרת כך (עמודות x שורות) : (rows x columns)

בדוגמה, גודל המטריצה הוא (3×2) מפני שיש שני שורות ושלוש עמודות.

מטריצות עמודה הן מטריצות שבהן מספר העמודות או מספר השורות הוא 1, כלומר נשארה רק עמודה או שורה אחת, למטריצות אלו קוראים וקטורים.

לא כמו הווקטורים הפיזיקליים הכלולים כוח וכיוון, וקטור מטריצionario פועל בדומה למטריצה, רישימת נתוניים אך רק בעמודה או בשורה.

לכן מנק' זו והלאה, נתיחש למילה וקטורים כאלו וקטור מטריצionario ולא פיזיקאלי

חיבור מטריצות

על מנת לחבר בין שני מטריצות או יותר, הן חייבות להיות באותו הגודל בדיק, אותו מספר שורות ואותו מספר עמודות. נחבר כל איבר במטריצה עם האיבר המקביל לו במטריצה

$$\begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix}_{[3 \times 1]} + \begin{bmatrix} 6 \\ 2 \\ 1 \end{bmatrix}_{[3 \times 1]} = \begin{bmatrix} 7 \\ 6 \\ 4 \end{bmatrix}_{[3 \times 1]}$$

השנייה.
לדוגמה:

מכפלת מטריצות

חשיבות לציין שוקטוריים אינם מטריצות, כלומר כפל וקטוריים וכפל מטריצות נעשים בדרךים שונות. כאשר מדובר בוקטוריים, המכפל מתבצע כמו בפעולת החיבור, אלמנט באלמנט.

$$\begin{bmatrix} 1 \\ 4 \\ 3 \end{bmatrix} \odot \begin{bmatrix} 2 \\ 4 \\ 5 \end{bmatrix} = \begin{bmatrix} 2 \\ 16 \\ 15 \end{bmatrix}$$

לדוגמה:

סימון המכפלה הזה נקרא "Hadamard Product", מכפלה של אלמנט באלמנט.

בשביל להכפיל מטריצות נדרש להשתמש במכפלה סקלרית (Dot Product).

על מנת לבצע מכפלה סקלרית, נדרש לנקח בחשבון את גודל המטריצות, מספר העמודות של המטריצה הראשונה (השמאלית) חייב להיות זהה למספר השורות של המטריצה השנייה (הימנית), והמטריצה שתצא מהמכפלה תהיה בגודל השורות של המטריצה הראשונה והעמודות של המטריצה השנייה.

כשנכפיל את המטריצות, אנו נכפיל שורה בעמודה.

לדוגמה:

$$\begin{bmatrix} a & b & c \\ d & e & f \end{bmatrix}_{[2 \times 3]} * \begin{bmatrix} g & h \\ i & j \\ k & l \end{bmatrix}_{[3 \times 2]} = \begin{bmatrix} a * g + b * i + c * k & a * h + b * j + c * l \\ d * g + e * i + f * k & d * h + e * j + f * l \end{bmatrix}_{[2 \times 2]}$$

$$\begin{bmatrix} 0 & 3 & 5 \\ 5 & 5 & 2 \end{bmatrix}_{[2 \times 3]} * \begin{bmatrix} 3 & 4 \\ 3 & -2 \\ 4 & -2 \end{bmatrix}_{[3 \times 2]} = \begin{bmatrix} 0 * 3 + 3 * 3 + 5 * 4 & 0 * 4 + 3 * (-2) + 5 * (-2) \\ 5 * 3 + 5 * 3 + 2 * 4 & 5 * 4 + 5 * (-2) + 2 * (-2) \end{bmatrix}_{[2 \times 2]}$$

Transpose

במתמטיקה, כאשר מדובר על מטריצות, טרנספורמציה של מטריצה מתאפיינת בפעולת של החלפת השורות והעמודות שלה. כלומר, הטרנספורמציה של מטריצה מתקבלת על ידי הפיכת כל שורה לעמודה ולהיפך.

[2x3] $A = \begin{bmatrix} 2 & 5 & 3 \\ 4 & 7 & 0 \end{bmatrix}$ לדוגמה:

ננפרק את 2 השורות (התחתונה תהיה עליונה וההפך) ונסובב הכל ב90 מעלות ימינה:

[3x2] $A^t = \begin{bmatrix} 2 & 4 \\ 5 & 7 \\ 3 & 0 \end{bmatrix}$

1. ייצוב עצמי באמצעות IMU

- (Inertial Measurement Unit) IMU זו יחידה שמודדת נתוניים עצמאיים כפי שהיא;
בד"כ היא כוללת בתוכה **accelerometer**, **gyroscope** ו-**magnetometer**. מטרת
היחידה היא הספקת מידע רציף לגבי תנועה ואוריאנטציה של גוף מסוים.

אפרק את תהליך הייצוב לכמה שלבים:

1. אסוף מידע רציף באמצעות תקשורת C²
2. שימוש בфиילטר מתמטי שיישלב את המידע משלוש הסנסורים – כדוגמת **Kalman filter**
3. אגדיר מישור מתמטי באמצעות המידע המודע שאספתי
4. אשווה למישור גלובלי (reference) ומבצע המרת מטריצות מתמטיות לפי זווית תטא וכור
מצאת השינוי הנדרש בכל ציר.

I²C

פרוטוקול תקשורת דיגיטלי שמשמש לחבר מכשירים במערכות מסובכות. באמצעות פרוטוקול זה מכשירים יכולים לשלוח מידע ולקבל מידע. ב프וטוקול זה יש 2 סוגים מכשירים: מאסטר ועבד (Master and slave). המאסטר שולט על התקשרות ומתקשר עם העבד (אבל לא כמו פעם...).

בשביל למש את הפרוטוקול הזה נדרשים 2 פינים:

- SDA (Serial Data) for bidirectional data transfer
- SCL (Serial Clock) for synchronization

וכן גם תוכנה נוספת היא אפשרות לסקל – לכל רכיב כתובות ייחודית וכן ניתן לנגן מערכ מכשירים עם תקשורת זו. שכן זו הסיבה שאתקשר באמצעותו לכל החישונים בIMU.

Kalman Filter

фильтр Калмана – это фильтр алгоритмический, который осуществляет оценку состояния системы на основе измерений и предыдущих оценок. Алгоритм фильтра Калмана включает в себя следующие основные этапы:

- 1. Инициализация: определение начальных значений состояния и ковариантной матрицы ошибки.
- 2. Поступление измерений: получение измерений от датчиков.
- 3. Оценка состояния: вычисление текущей оценки состояния на основе измерений и предыдущих оценок.
- 4. Обновление ковариантной матрицы ошибки: обновление ковариантной матрицы ошибки на основе измерений.

Фильтр Калмана является эффективным методом для обработки нелинейных и нестационарных сигналов, так как он учитывает неопределенность измерений и предыдущих оценок. Алгоритм фильтра Калмана может быть реализован с помощью различных языков программирования, таких как Python, C, C++, Java и др.