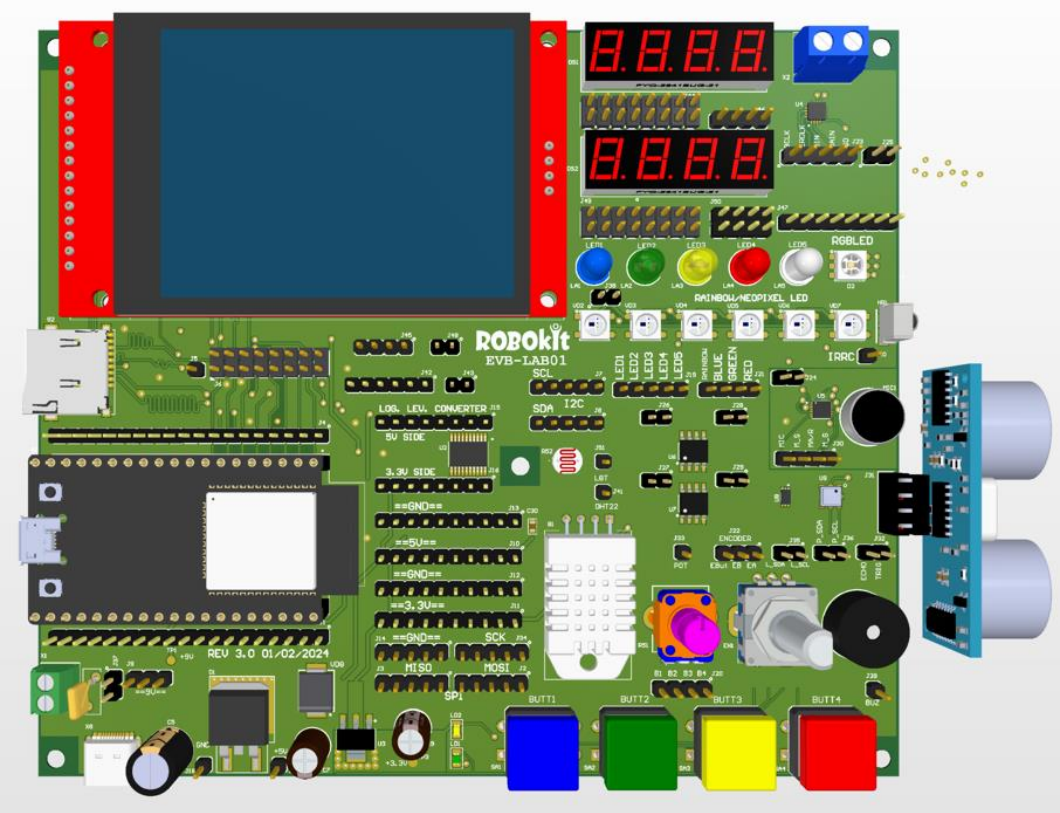
חוברת ניסויים ותיאוריה לערכת לימוד

.EDU-ESP32-STD



גרסה 1 ינואר 2024

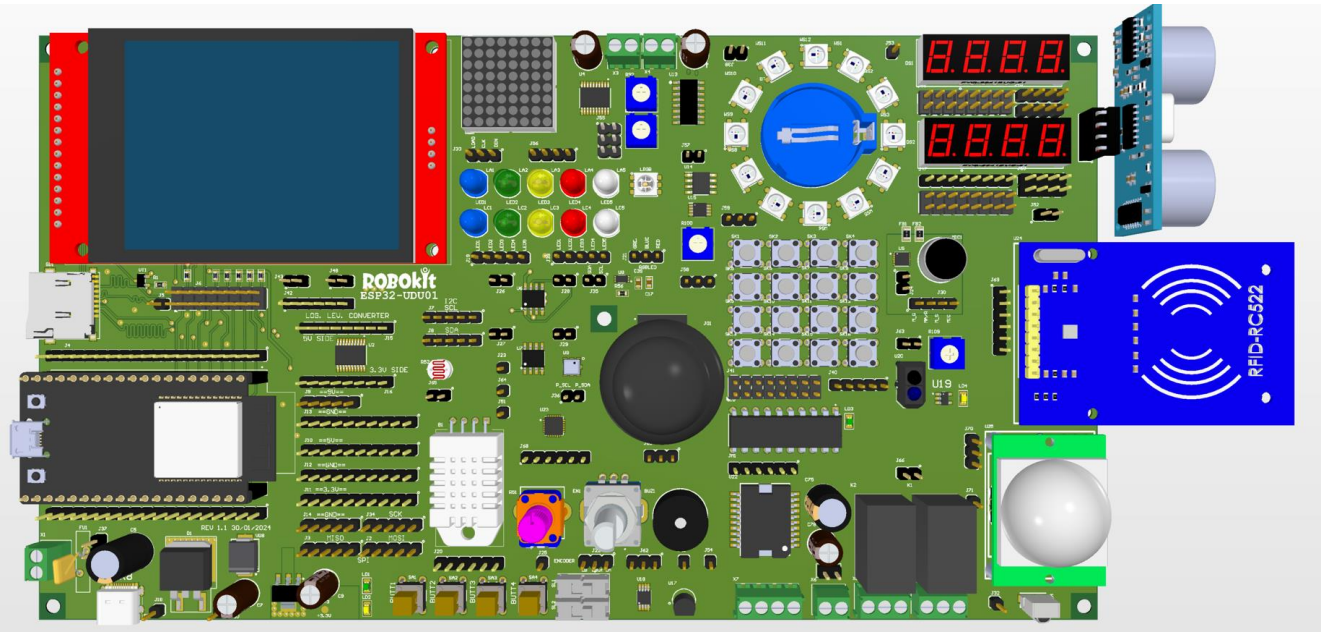
תוכן עניינים:

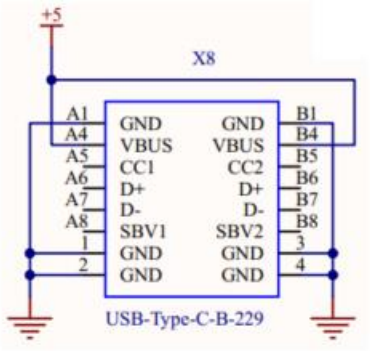
Contents

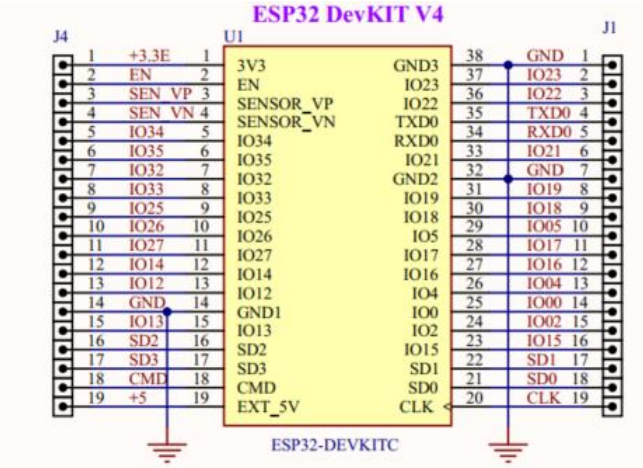
מבוא למיקרו בקר ESP32.

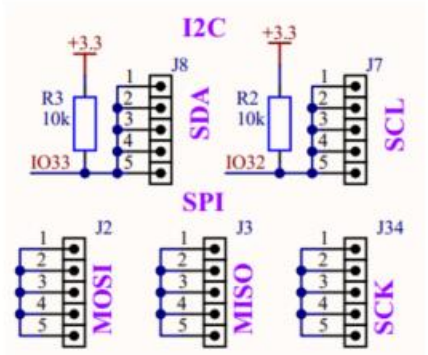
מה מכילה ערכה.

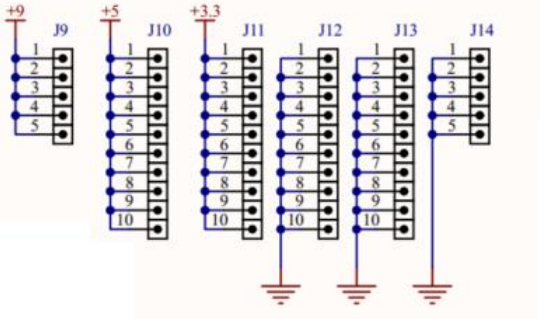
N		
1	ESP32-DevKitC V4 Development Board with ESP-WROOM-32	
	Data storage	
2	Micro SD card holder	
	Graphical and character indication	
3	WH1602B Character LCD 16x2, 5x8 dots includes cursor	
4	Dual Color OLED Display COG+PCB 128x64, 0.96" SSD1306	
5	MSP2807 ILI9341 2.8" 240x320 TFT LCD SPI module with touch XPT2046	
	7-seg indication	
6	E40281-L-O-8-W 0,28 inch 4x Common **CATHODE** drive by directly and by HEF4511(BCD to 7-seg decoder)	
7	E40281-I-O-0-W 0,28 inch 4x Common **ANODE** drive by directly and by TM1637	
	LEDS	
8	5x 5mm color leds with common cathode connection. Drive directly from CPU	
9	5x 5mm color leds with common anode connection. Drive directly from CPU	
10	1x 5050 RGB FYLS-5050RGB led drive by CPU and open drain. Common Anode connection.	
11	12x WS2812B rainbow(NeoPixel) leds arranged in a circle.	
	Buttons and switch	
12	2x pull UP tactile switch	
13	2x pull DOWN tactile switch	
14	2x SPDT slide switch EG1218	
15	4x4 matrix keyboard consist from 6mm tactile switch drive and read directly from CPU or MM74C922 IC	
16	Incremental rotary encoder PEC11R-4120F-S0018	
	Dot matrix display	
17	TC08-81SRWA 20mm (0.8 INCH) RED COLOR DOT MATRIX DISPLAY drive from max7219.	
	Analog and digital sensors	
18	PGM5516 5k PGM5 CDS Photoresistor	
19	10k compact type potentiometer RK09K1130081	
20	Microphone amplifier MAX9814ETD+ with analog output	
21	I2C light sensor BH1750FVI-TR 0-65535 lux	
22	I2C pressure and temperature sensor BMP180	
23	Digital DHT22 humidity and temperature sensor	
24	Digital HC-SR04 ultrasonic distance sensor	
25	IR receiver VS1838B for RC5 and other protocol	
28	PIR sensor HC-SR501 or HC-SR505	
29	LM35 in TO92 case analog temperature sensor	
30	DS1307 I2C 64 X 8 Serial Real Time Clock with battery holder	
31	MPU6050 3-axis gyroscope and 3-axis accelerometer (must be external board)	
32	Mifare RC522 card reader module (Mifare: 1K, 4K, Ultralight, DESFire, Pro)	
33	2x analog input with buffered Operational Amplifier MCP6002	
34	2x analog output with buffered Operational Amplifier MCP6002	
35	Joystick 2x analog and 1x button	
36	I2C LM75 temperature sensor	
37	TCRT5000 reflective optical sensor with analog and digital output	
38	FSR (force sensitive resistor)with divider	
	DC and step motor	
39	L298 2x DC or 1x bipolar step motor driver	
	Audio	
40	Active buzzer HCM1205X	
41	Two-channel stereo amplifier D-class 2x3 W PAM8403 (analog input)	
42	I2S DAC PCM5102 32bit/384kHz	



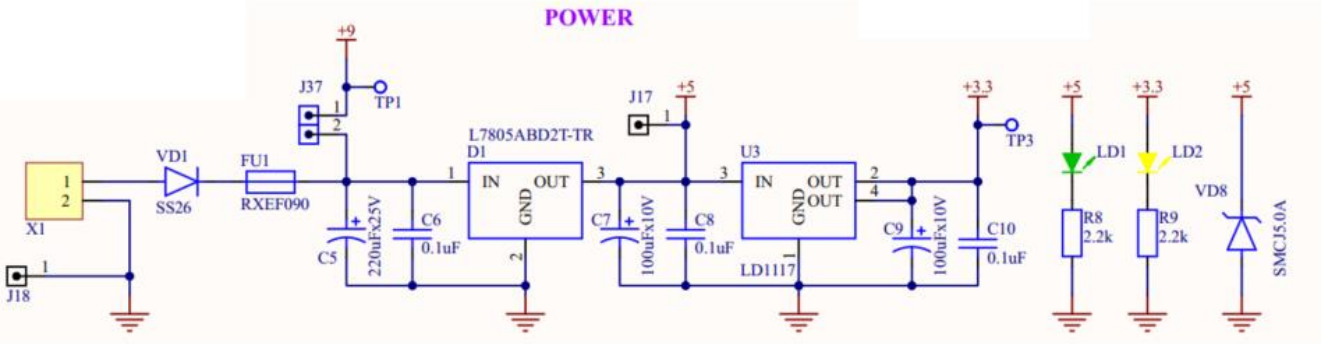




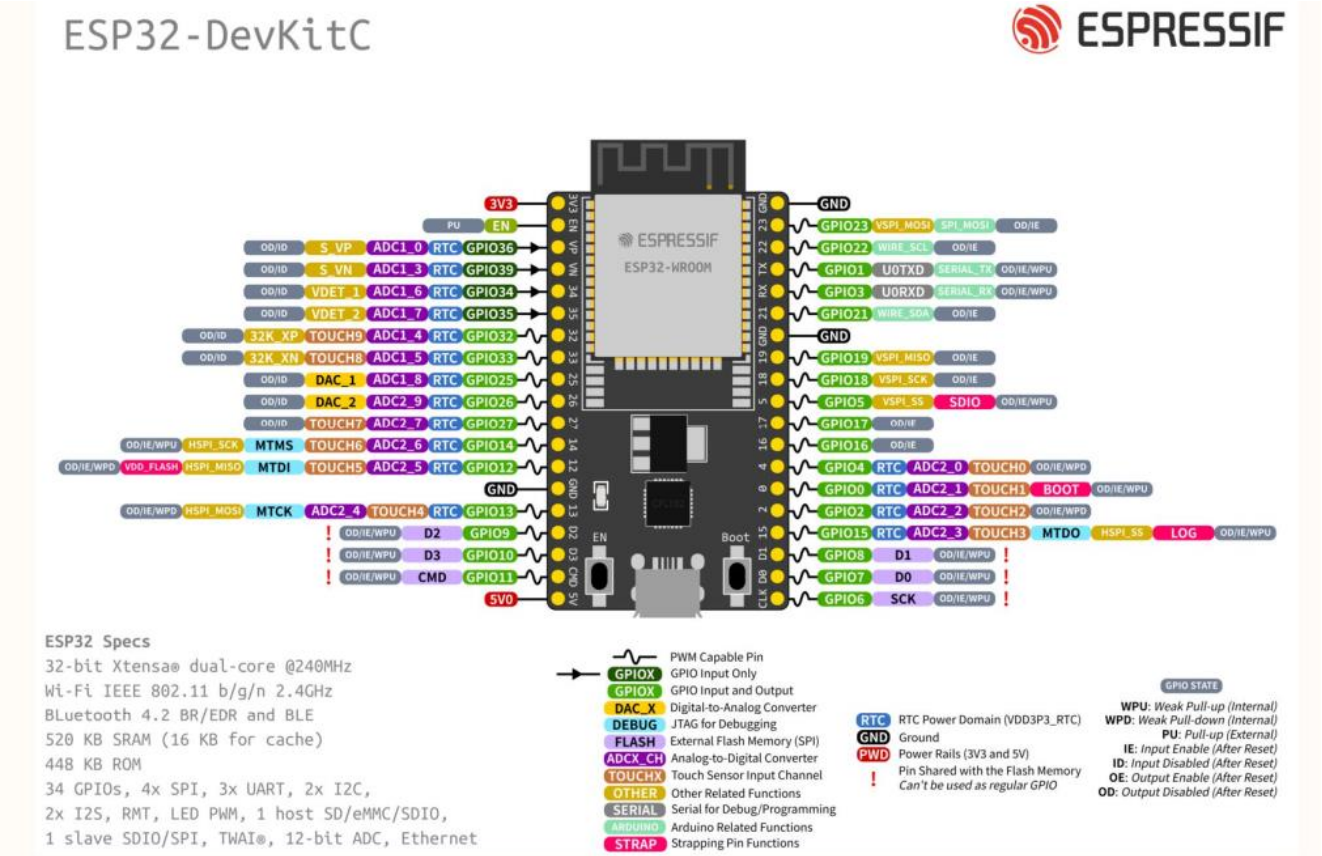




הספקת מתח של ערכה.



מבנה ותכונות של מיקרו בקר.



המיקרו-בקר ESP32 התגלה כפלטפורמה רב-תכליתית וחזקה, שחולל מהפכה בעולם המערכות המשובצות וייישומי האינטרנט של הדברים (IoT). פותח על ידי Espressif Systems, ה-ESP32 מתבסס על הצלחתו של קודמו, ESP8266, ומביא ליכולות ותכונות משופרות.

ה-ESP32 הוא מיקרו-בקר כפול ליבה הכולל ארכיטקטורה ניתנת להרחבה והתאמה. הוא משלב מעבד Xtensa LX6 בעל ביצועים גבוהים, מעבד משותף בהספק נמוך ושפע של ציוד היקפי שהופכים אותו למתאים למגוון רחב של יישומים.

המאפיינים העיקריים של ה-ESP32 כוללים:

1. עיבוד ליבה כפולה:

ה-ESP32 מתהדר בליבות Tensilica Xtensa LX6 כפולות, המאפשרות עיבוד מקביל ויכולות ריבוי משימות משופרות. זה הופך אותו למתאים היטב עבור יישומים הדורשים עיבוד והיענות בזמן אמת.

2. קישוריות אלחוטית:

אחד המאפיינים הבולטים של ה-ESP32 הוא יכולות ה-Wi-Fi וה-Bluetooth המובנות שלו. זה מאפשר קישוריות חלקה לאינטרנט ולהתקנים אחרים, מה שהופך אותו לבחירה אידיאלית עבור פרויקטי IoT, יישומי בית חכם ועוד.

3. סט עשיר של ציוד היקפי:

ה-ESP32 מצויד במגוון רחב של ציוד היקפי, כולל פיני GPIO, UART, SPI, I2C, ADC, DAC ועוד. הרבגוניות הזו מאפשרת למפתחים להתממשק בקלות עם חיישנים, מפעילים והתקנים חיצוניים שונים.

4. צריכת חשמל נמוכה במיוחד:

בנוסף ליכולות הביצועים הגבוהות שלו, ה-ESP32 מצטיין ביעילות הספק. הוא כולל מצבי צריכת חשמל נמוכה, מה שהופך אותו למתאים למכשירים ולאפליקציות המופעלות על סוללות שבהן חיסכון באנרגיה הוא חיוני.

5. תכונות אבטחה:

אבטחה היא דאגה גבוהה ביישומי IoT. ה-ESP32 משלב תכונות אבטחה חזקות, כולל מאיצי חומרה עבור אלגוריתמים קריפטוגרפיים, אתחול מאובטח ומחולל מספרים אקראיים.

פיתוח מערכת אקולוגית

ה-ESP32 נהנה מפופולריות נרחבת, הודות לקהילה התוססת והפעילה שלו ומערכת אקולוגית פיתוח עשירה. להלן כמה

מרכיבים מרכזיים:

1. תמיכת Arduino IDE:

ה-ESP32 נתמך היטב על ידי Arduino IDE, מה שמפשט את תהליך הפיתוח למי שמכיר את תכנות Arduino. ליבת Arduino עבור ESP32 מספקת סביבה מוכרת למשתמשים להתחיל במהירות.

2. מסגרת ESP-IDF:

עבור משתמשים מתקדמים יותר, ה-ESP-IDF (Espressif IoT Development Framework) מציע סט מקיף של ספריות וכלים לבניית יישומים מורכבים. זה מספק שליטה רבה יותר על החומרה וגישה לתכונות מתקדמות.

3. תמיכת MicroPython ו-Lua:

ה-ESP32 תומך ב-MicroPython וב-Lua, ומציע למפתחים את הגמישות לבחור את שפת הסקריפט המועדפת עליהם לפיתוח. זה מרחיב את טווח ההגעה של ה-ESP32 לקהל רחב יותר.

יישומים של ESP32

הרבגוניות של ה-ESP32 פותחת את הדלת למספר עצום של יישומים בתחומים שונים:

1. התקני IoT:

ה-ESP32 הוא בחירה טבעית עבור יישומי IoT, המאפשר למפתחים ליצור מכשירים מחוברים שיכולים לתקשר באמצעות Wi-Fi או Bluetooth. זה כולל מכשירי בית חכם, מערכות ניטור סביבתיות ופתרונות IoT תעשייתיים.

2. תקשורת אלחוטית:

עם יכולות ה-Wi-Fi וה-Bluetooth המובנות שלו, ה-ESP32 משמש ליישומי תקשורת אלחוטית. זה נע בין העברת נתונים פשוטה בין מכשירים לתרחישים מורכבים יותר כגון רשת רשת.

3. מערכות אוטומציה ובקרה:

יכולות ה-ESP32 בזמן אמת הופכות אותו למתאים למערכות אוטומציה ובקרה. ניתן להפעיל אותו בפרויקטים החל מאוטומציה ביתית ועד יישומי בקרה תעשייתיים.

4. מכשירים לבגדים:

בשל גודלו הקומפקטי, צריכת החשמל הנמוכה והקישוריות האלחוטית שלו, ה-ESP32 הוא בחירה מתאימה לטכנולוגיה לבישה. מפתחים יכולים ליצור שעונים חכמים, מעקבי כושר וגאדג'טים לבישים אחרים.

5. חינוך ויצירת אב טיפוס:

הנגישות של ה-ESP32, יחד עם סבירותו, הופכת אותו לכלי מצוין למטרות חינוכיות ויצירת אב טיפוס מהיר. זה מאפשר לתלמידים וליוצרים להביא את הרעיונות שלהם לחיים במהירות.

לסיכום:

המיקרו-בקר ESP32 הפך לאבן יסוד בעולם המערכות המשובצות וה-IoT. השילוב של עיבוד כפול ליבה, קישוריות אלחוטית, ציוד היקפי עשיר ויעילות אנרגטית הופכים אותו לבחירה מומלצת עבור מגוון רחב של יישומים. בין אם אתה חובב, סטודנט או מפתח מקצועי, ה-ESP32 מספק פלטפורמה חזקה וגמישה להפוך את הרעיונות שלך למציאות.
כשהטכנולוגיה ממשיכה להתקדם, ה-ESP32 נשאר בחזית, ומניע חדשנות בעולם המכשירים המחוברים והמערכות החכמות.

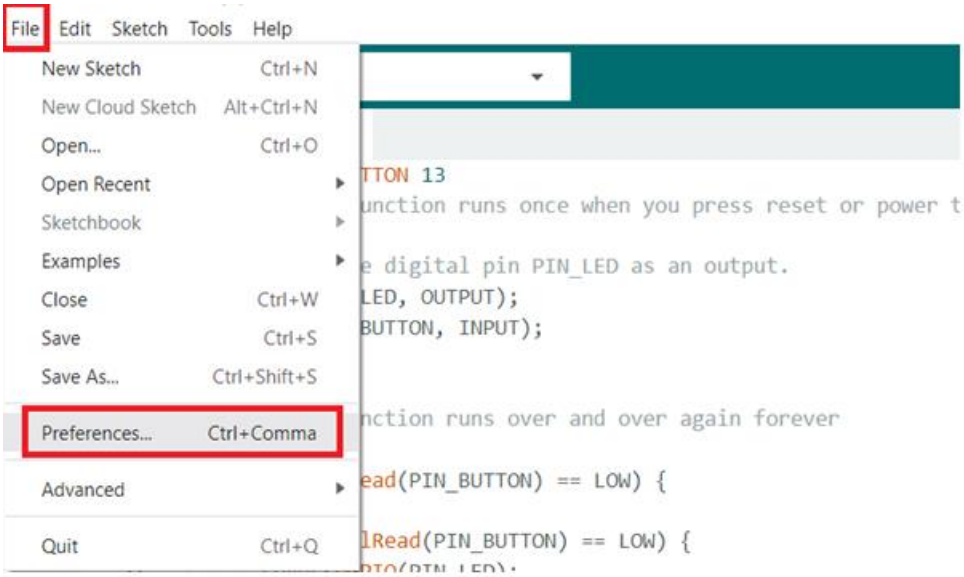
התקנת סביבת עבודה.

צעד הראשון בתהליך התקנת ערכת הפיתוח ESP32 הוא להוריד ולהתקין את תוכנת ה Arduino IDE-מהאתר הרשמי של Arduino.לאחר התקנת התוכנה, יש לפתוח אותה ולהתקין את התוסף הנדרש עבור ESP32. תוסף זה נקרא "ESP32 boards manager"וניתן למצוא אותו בתפריט "קובץ" ב- Arduino IDE.לאחר התקנת התוסף, יש להגדיר את הגדרות התוכנה כך שתתמוך בפלטפורמה של ESP32 .

כנסו לאתר רשמי של ארדואינו: https://www.arduino.cc/en/Main/Software

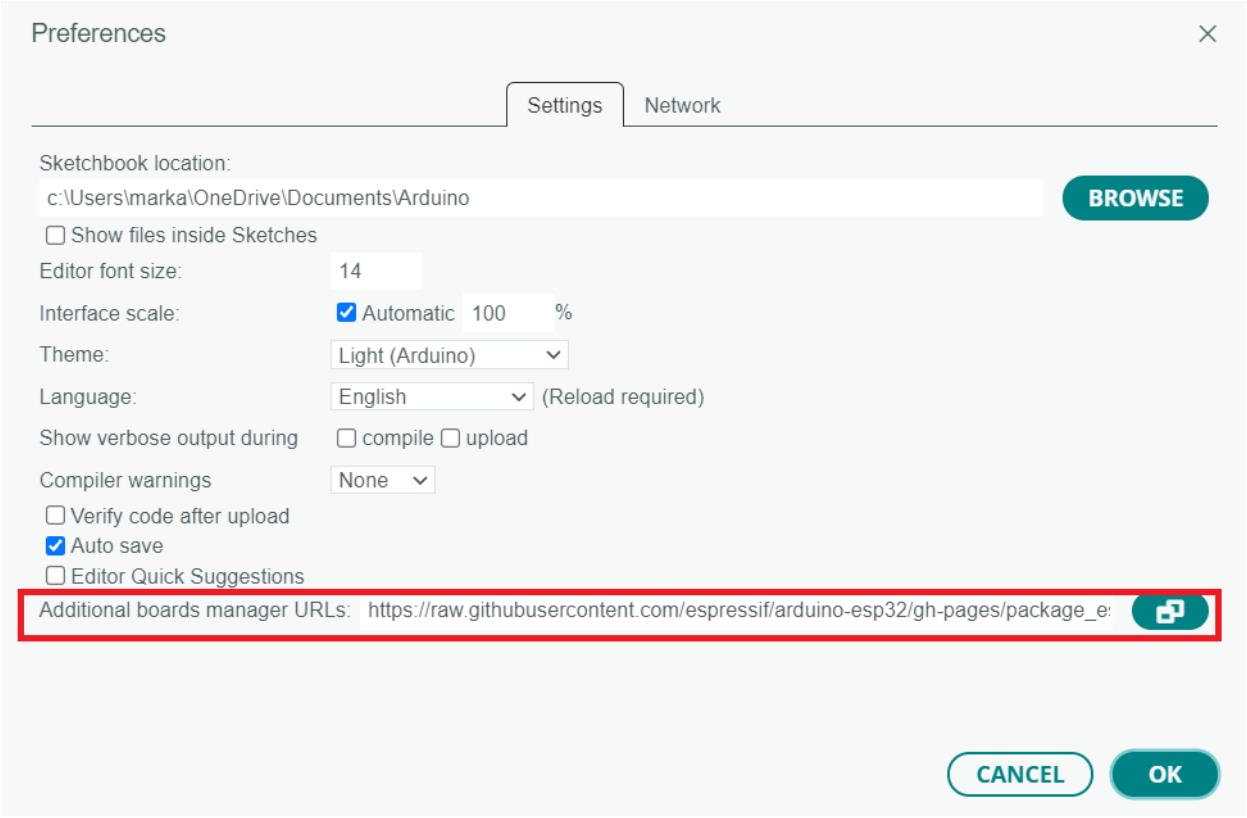
תורידו גרסה אחרונה שמופיעה באתר:

לאחר התקנת הסביבה יש להגדיר ערכות נוספות ממשפחת ESP32.

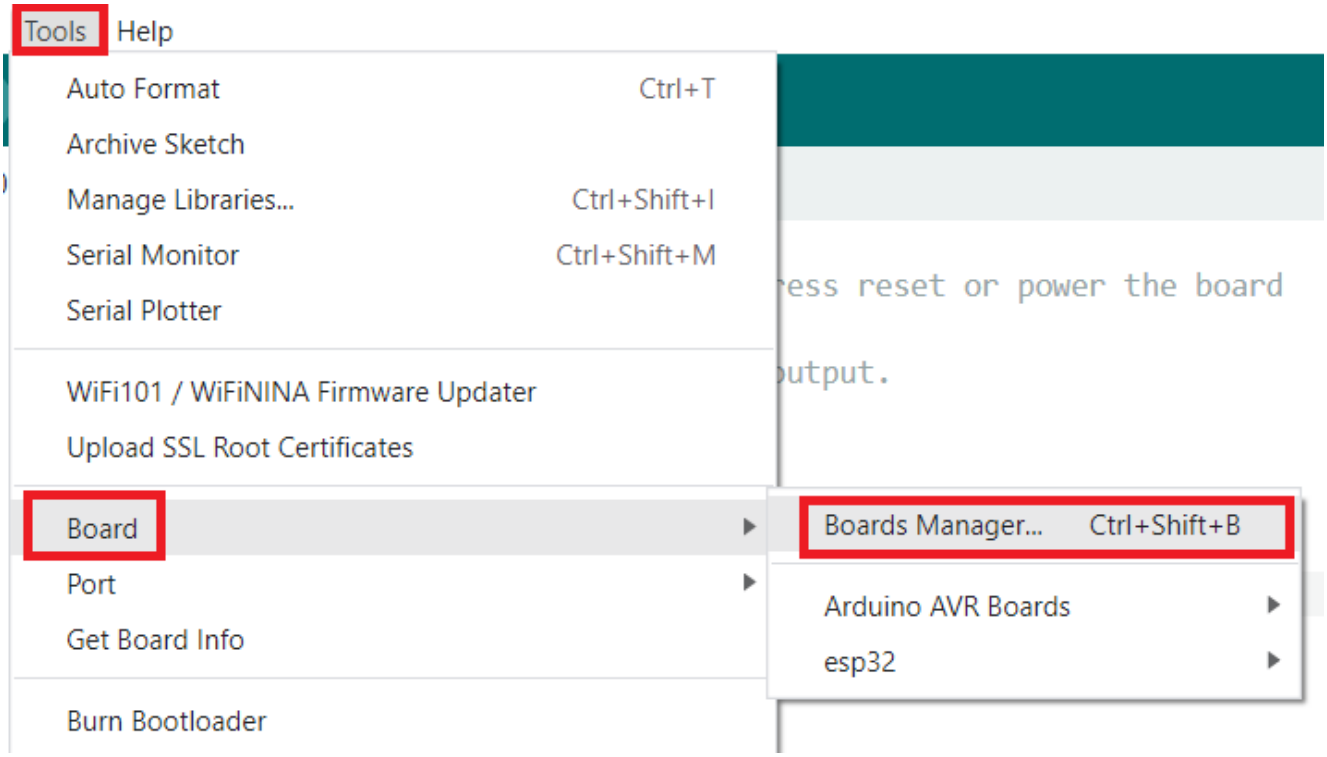


אחרי זה נוסיף קישור לערכות:

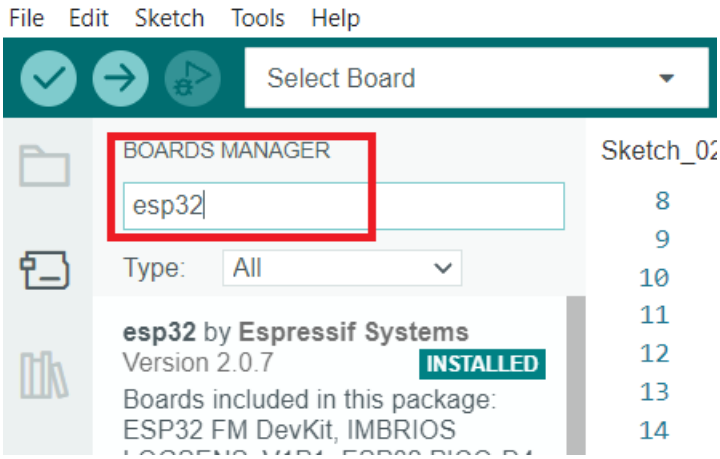
https://dl.espressif.com/dl/package_esp32_index.json



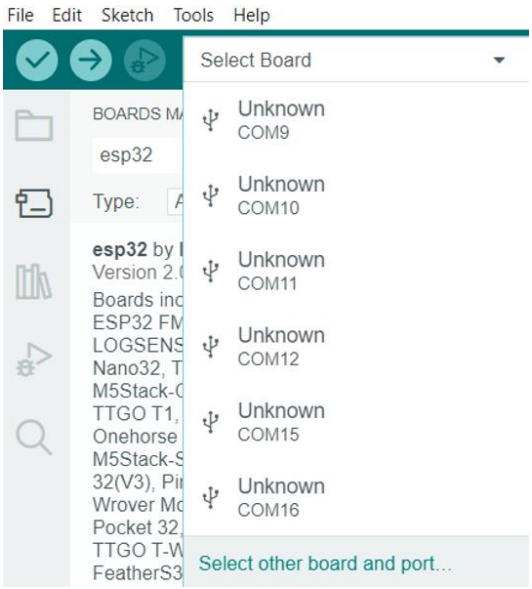
לאחר התקנה יש לעבור לשלב הבא:



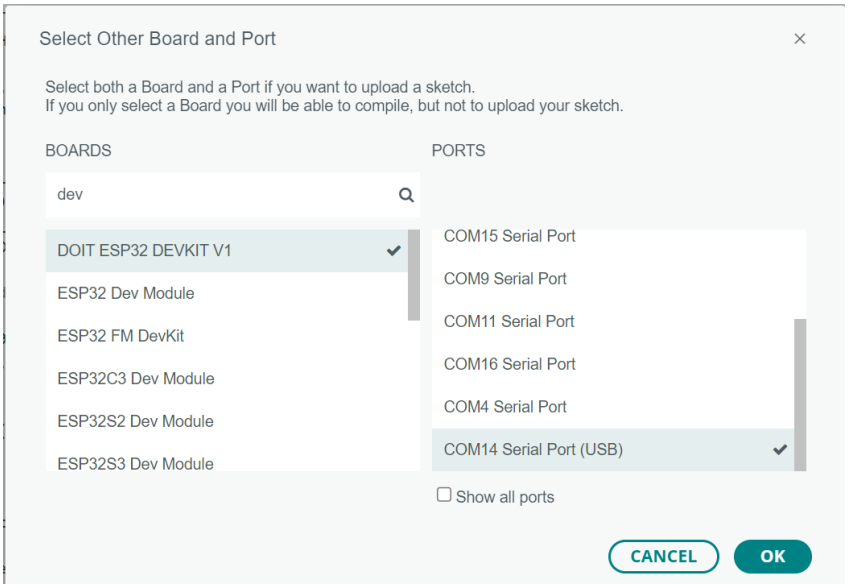
לאחר פתיחת חלון – בוחרים סט ערכות esp32 by Espressif Systems .



לאחר התקנה נחבר ערכת ESP32 למחשב ונבחר סוג הערכה ופורט שדרכו



אנו נבחר ערכה הבאה:



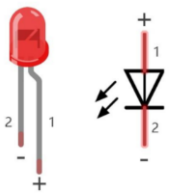
הצגת הודעות וקלט טורי.

פרק 1. פלט וקלט ספרתי.

הפעלת לדים רגילים.

LED הוא סוג של דיודה. כל הדיודות פועלות רק אם הזרם זורם בכיוון הנכון ויש להן שני קטבים.

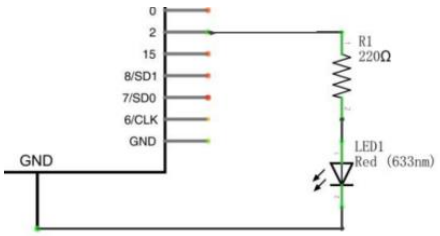
LED תפעל (תידלק) רק אם אנודה A (הפין הארוך יותר) (+) של LED מחובר לפלט החיובי ממקור מתח והפין קתודה C (הקצר יותר) מחובר לשלילי (-). פלט שלילי מכונה גם Ground (GND). סוג זה של רכיב ידוע בשם "דיודות" (חשבו על רחוב חד-סטרי).



כמו דיודה רגילה LED בנוי מ2 סוגים של מוליכים למחצה (N – שלילי, P – חיובי). בצומת PN נופל מתח מסוים שמעל מתח זה דיודה מתחילה להוליך. ישנם הגבלות זרם לכל סוג דיודה (LED). ישנו זרם מינימלי, זרם מומלץ וזרם מקסימלי. לכל צבע של LED יש שוני מפני מתח שנופל בצומת PN וזרם שזורם דרכו:

LED	Voltage	Maximum current	Recommended current
Red	1.9-2.2V	20mA	10mA
Green	2.9-3.4V	10mA	5mA
Blue	2.9-3.4V	10mA	5mA

בהתאם לצבעים ודרישות מתח/זרם אנו יכולים לחשב מה היא התנגדות הנדרשת, כדי שלד יעבוד במקסימום בהירות ללא חשש לשרפה.



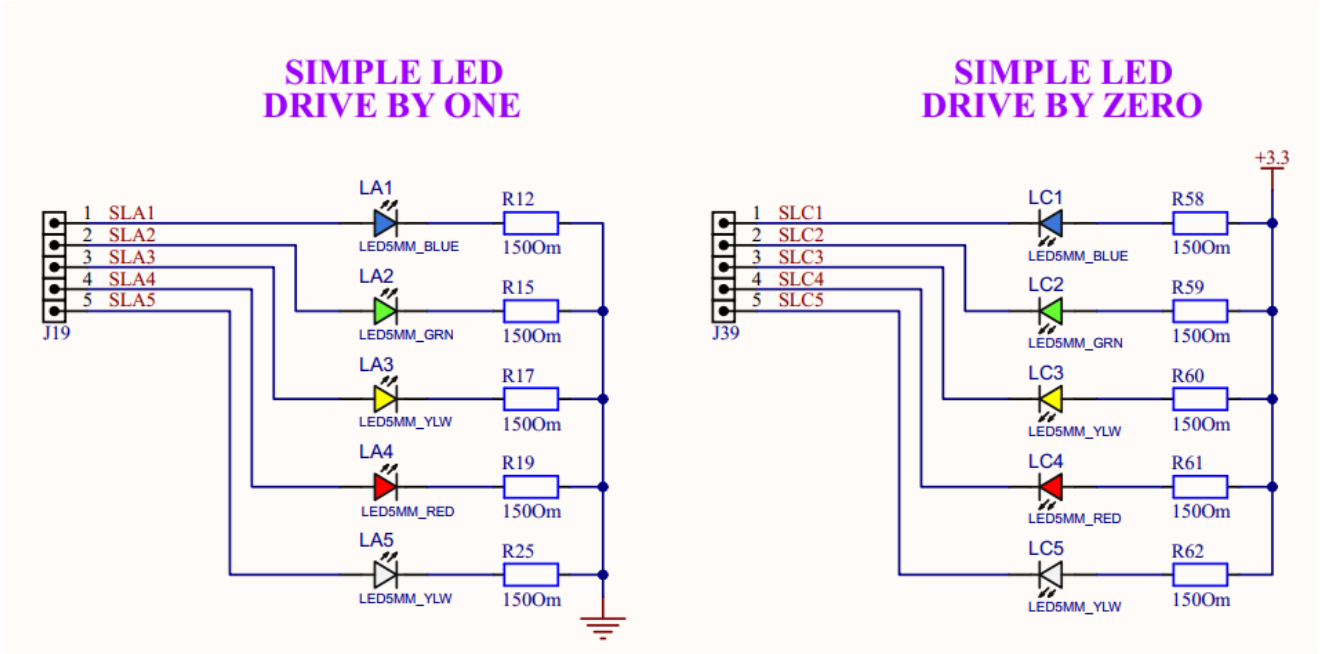
נעשה חישוב עבור לד בצבע אדום.

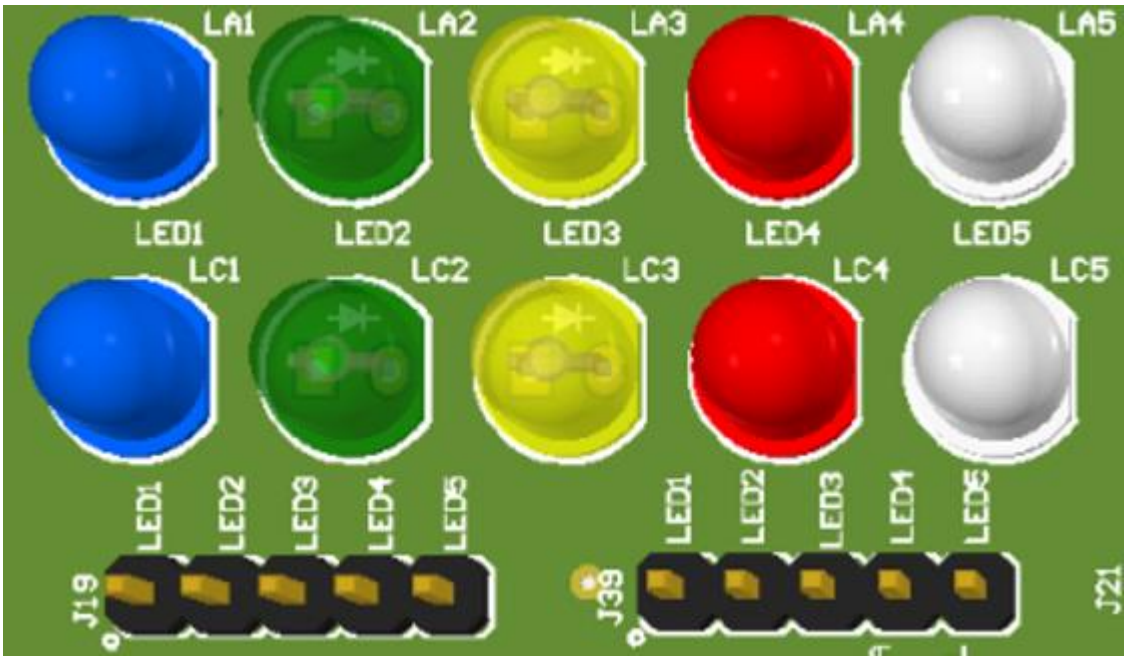
$Vcc=V_{LED}+V_R$

$Vcc=V_{LED}+I_{LED}*R$

$R=(Vcc-V_{LED})/I_{LED}=(3.3V-2.1V)/10mA=120\ Ohm$

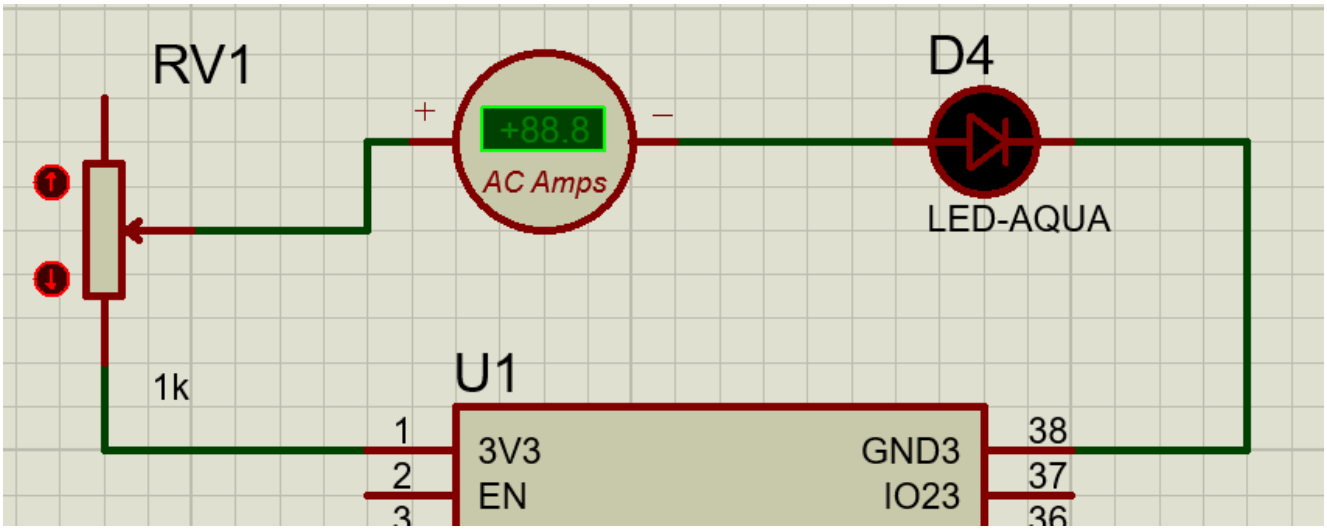
אם נתכנן מעגל לזרם של 5 mA, נקבל נגד של 240 Ohm.


SIMPLE LED
DRIVE BY ONE
1 SLA1
2 SLA2
3 SLA3
4 SLA4
5 SLA5
J19
LA1
LED5MM_BLUE
R12
150Om
LA2
LED5MM_GRN
R15
150Om
LA3
LED5MM_YLW
R17
150Om
LA4
LED5MM_RED
R19
150Om
LA5
LED5MM_YLW
R25
150Om
SIMPLE LED
DRIVE BY ZERO
+3.3
1 SLC1
2 SLC2
3 SLC3
4 SLC4
5 SLC5
J39
LC1
LED5MM_BLUE
R58
150Om
LC2
LED5MM_GRN
R59
150Om
LC3
LED5MM_YLW
R60
150Om
LC4
LED5MM_RED
R61
150Om
LC5
LED5MM_YLW
R62
150Om

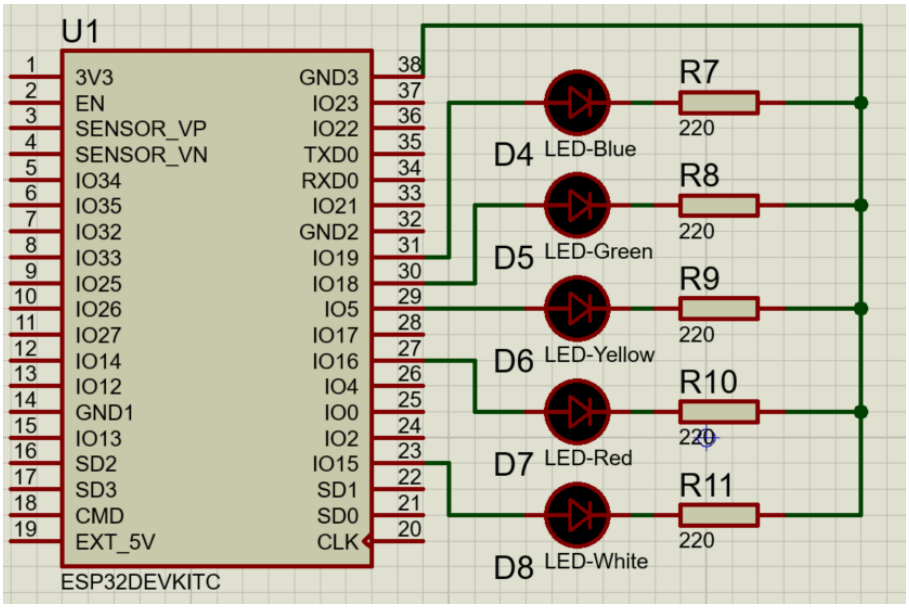

LA1
LA2
LA3
LA4
LA5
LED1
LED2
LED3
LED4
LED5
LC1
LC2
LC3
LC4
LC5
J19
J39
J21

ניסוי מס' 1. מדידת זרם דרך LED וצפיה בשינוי בהירות בהתאם לשינוי פוטנציומטר.

תסובבו פוטנציומטר מערך מקסימלי (1 קילו אוהם) עד לערך מינימלי. תבדוק כל פעם איזה זרם עובר במעגל.



חיבור לד בצורה קתודה משותפת CC – הפעלה ב"1".



ניסוי מס' 2. הפעלת לד מהבהב.

```
#define LED 15
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED as an output.
  pinMode(LED, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
```

```
}
```

תרגיל מס' 1. תכנון השהייה משתנה.

תכנן ניסוי מס' 2 כך שהשהייה בין הדלקה לכיבוי תשתנה מ-100 עד 2000 מילי שניות בקפיצות של 100.

תרגיל מס' 2. תכנון השהייה משתנה.

שנה תרגיל מס' 1 כך שהשהייה בין הדלקה לכיבוי תשתנה מ-100 עד 2000 מילי שניות בקפיצות של 100 ואז תרד שוב לכיוון 100 מילישניות.

ניסוי מס' 3. הפעלת לד רץ.

חבר לדים לפורטים 15,16,17,18,19 (כך שלדים ירוצו מלד לבן עד ללד כחול). אין צורך לחבר נגדים – הם מחוברים בלוח מראש.

```
int leds[]={15,16,17,18,19};
int i;
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pins LEDs as an output.

  for(i=0;i<5;i++)
pinMode(leds[i], OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
   for(i=0;i<5;i++)
   {
  digitalWrite(leds[i], HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(leds[i], LOW);    // turn the LED off by making the voltage LOW
  delay(1000);    // wait for a second
  }
}
```

תרגיל מס' 3. לד רץ בצורה הפוכה.

שנה רק את הלולאה:

```
    for(i=0;i<5;i++)
  {
  digitalWrite(leds[i], HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(leds[i], LOW);    // turn the LED off by making the voltage LOW
  delay(1000);    // wait for a second
  }
```

כך שלדים ירוצו מלד כחול עד ללד לבן.

תרגיל מס' 4. לד רץ קדימה ואחורה.

תשלב ניסוי 3 ותרגיל 3 כך שלד ירוץ שמאלה וימינה.

תרגיל מס' 5. רמזור.

חבר 3 לדים – אדום, צהוב וירוק.

תכנן לדים לפי סדר הבא:

נדלק לד אדום במשך 3 שניות.

לד צהוב נדלק יחד עם לד אדום למשך חצי שניה.

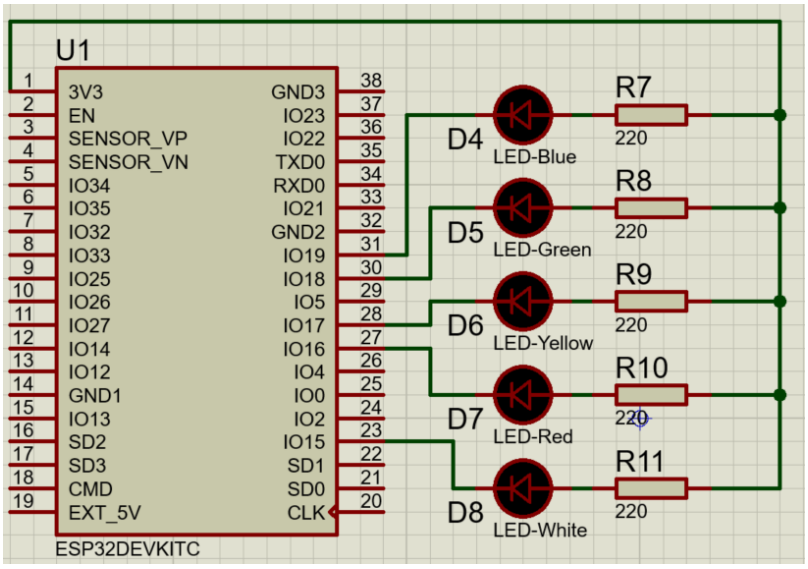
מכבים לד אדום וצהוב ומדליקים לד ירוק למשך 3 שניות.

לד ירוק מהבהב 5 פעמים (0.3 שניות ON 0.3 שניות OFF).

לאחר כיבוי של לד ירוק מדליקים לד צהוב למשך שניה וחצי.

מכבים לד צהוב וחוזרים לתחילת התוכנית.

חיבור לד בצורה אנודה משותפת CA – הפעל ב"0".
(רק בערכה מתקדמת).



ניסוי מס' 4. הפעלת לד מהבהב.

```
#define LED 15
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED as an output.
  pinMode(LED, OUTPUT);
digitalWrite(LED, HIGH);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED , LOW);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                       // wait for a second
  digitalWrite(LED, HIGH);    // turn the LED off by making the voltage LOW
  delay(1000);                       // wait for a second
}
```

תרגיל מס' 6. תכנון השהייה משתנה.

תכנן ניסוי מס' 2 כך שהשהייה בין הדלקה לכיבוי תשתנה מ-100 עד 2000 מילי שניות בקפיצות של 100.

תרגיל מס' 7. תכנון השהייה משתנה.

שנה תרגיל מס' 6 כך שהשהייה בין הדלקה לכיבוי תשתנה מ-100 עד 2000 מילי שניות בקפיצות של 100 ואז תרד שוב לכיוון 100 מילישניות.

ניסוי מס' 5. הפעלת לד רץ.

חבר לדים לפורטים 15,16,17,18,19 (כך שלדים ירוצו מלד לבן עד ללד כחול). אין צורך לחבר נגדים – הם מחוברים בלוח מראש.

```
int leds[]={15,16,17,18,19};
int i;
// the setup function runs once when you press reset or power the board
void setup() {
```

```
  // initialize digital pins LEDs as an output.

  for(i=0;i<5;i++)
pinMode(leds[i], OUTPUT);
  for(i=0;i<5;i++)
  digitalWrite(leds[i], HIGH);
}

// the loop function runs over and over again forever
void loop() {
   for(i=0;i<5;i++)
   {
  digitalWrite(leds[i], LOW);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                  // wait for a second
  digitalWrite(leds[i], HIGH);    // turn the LED off by making the voltage LOW
  delay(1000);    // wait for a second
  }
}
```

תרגיל מס' 8. לד רץ בצורה הפוכה.

שנה רק את הלולאה:

```
    for(i=0;i<5;i++)
  {
  digitalWrite(leds[i], LOW);   // turn the LED on (HIGH is the voltage level)
  delay(1000);                  // wait for a second
  digitalWrite(leds[i], HIGH);    // turn the LED off by making the voltage LOW
  delay(1000);    // wait for a second
  }
```

כך שלדים ירוצו מלד כחול עד ללד לבן.

תרגיל מס' 9. לד רץ קדימה ואחורה.

תשלב ניסוי 5 ותרגיל 8 כך שלד ירוץ שמאלה וימינה.

תרגיל מס' 10. רמזור.

חבר 3 לדים – אדום, צהוב וירוק.

תכנן לדים לפי סדר הבא:

נדלק לד אדום במשך 3 שניות.

לד צהוב נדלק יחד עם לד אדום למשך חצי שניה.

מכבים לד אדום וצהוב ומדליקים לד ירוק למשך 3 שניות.

לד ירוק מהבהב 5 פעמים (0.3 שניות ON 0.3 שניות OFF).

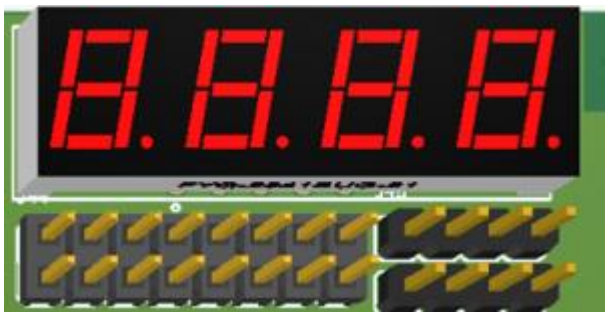
לאחר כיבוי של לד ירוק מדליקים לד צהוב למשך שניה וחצי.

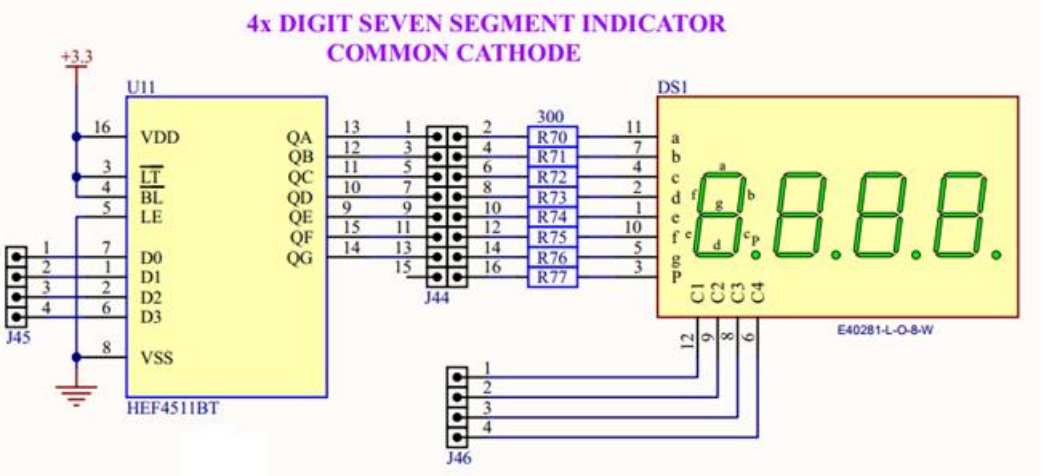
מכבים לד צהוב וחוזרים לתחילת התוכנית.

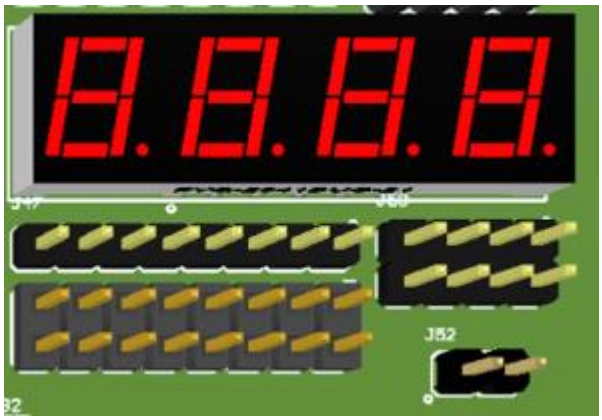
הפעלת צג 7 סגמנט.

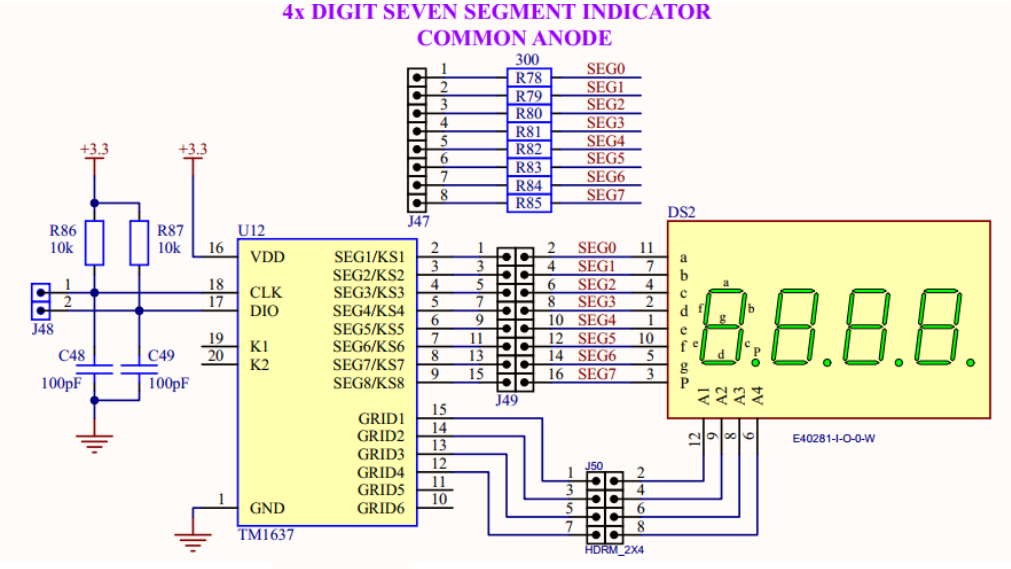
7 סגמנט בחיבור מקבילי.

הבנה ושימוש בתצוגות 7 סגמנט עם ESP32 ב-Arduino IDE: מצבי קתודה ואנודה משותף.









תצוגות 7 סגמנט הן שיטות פופולריות ופשוטות להצגת נתונים מספריים בפרויקטים אלקטרוניים. בממשק עם ESP32 באמצעות Arduino IDE, התקנים האלה יכולים להיות שימושיים במיוחד במגוון יישומים. סעיף זה מספק סקירה כללית של צגים בעלי 7 מקטעים, תוך התמקדות בפעולתם בתצורות קתודה משותפת והן בתצורות אנודה משותפת, כאשר משתמשים ב-ESP32.

תצוגה בת 7 מקטעים מורכבת משבע נוריות LED המסודרות בתבנית ספציפית (בדרך כלל תצורת איור-8), המאפשרת הצגת ספרות עשרוניות וכמה תווים אלפביתיים. כל אחת משבעת הנוריות (מקטעים) מסומנת מ-'a' עד 'g', וישנה נקודה נוספת (dp) להצגת נקודות עשרוניות.

ישנם שני סוגים של צגים בעלי 7 מקטעים המבוססים על החיבור הפנימי שלהם: קתודה משותפת ואנודה משותפת.

צגי קתודה משותפת.

בתצוגות קתודה משותפת, כל הקתודות (מסופים שליליים) של מקטעי LED מחוברים יחד. כדי להאיר סגמנט, ה-ESP32 חייב לשלוח אות HIGH לפין המתאים.

תצוגות אנודה משותפת.

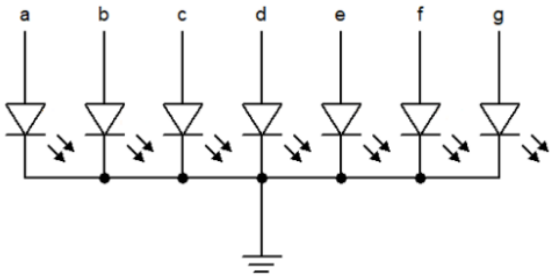
לעומת זאת, בתצוגות האנודה משותפת, כל האנודות (הטרמינלים החיוביים) מחוברים יחד. סגמנט נדלק כאשר ה-ESP32 שולח אות LOW לפין המתאים.

החיווט של הצג ל-ESP32 משתנה מעט בהתאם אם מדובר בקטודה או מסוג אנודה משותפת.

קתודה משותפת.

חבר את פין הקתודה המשותפת לאדמה (GND) של ה-ESP32.

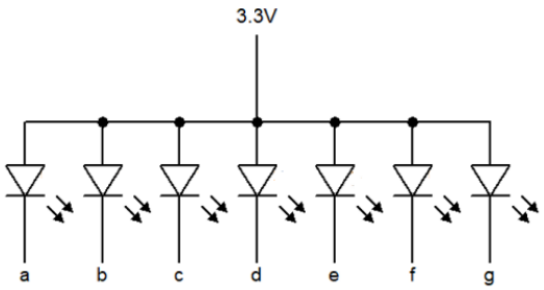
חבר כל אחד מפיני המקטע (a-g ו-dp) לפיני ה-GPIO של ה-ESP32 (אין צורך בנגד, כי נגדים נמצאים על הלוח).



אנודה משותפת

חבר את פין האנודה המשותפת ל-V 3.3 של ה-ESP32.

בדומה לגרסת הקתודה, חבר כל פין מקטע לפיני GPIO של ESP32 (אין צורך בנגד, כי נגדים נמצאים על הלוח).

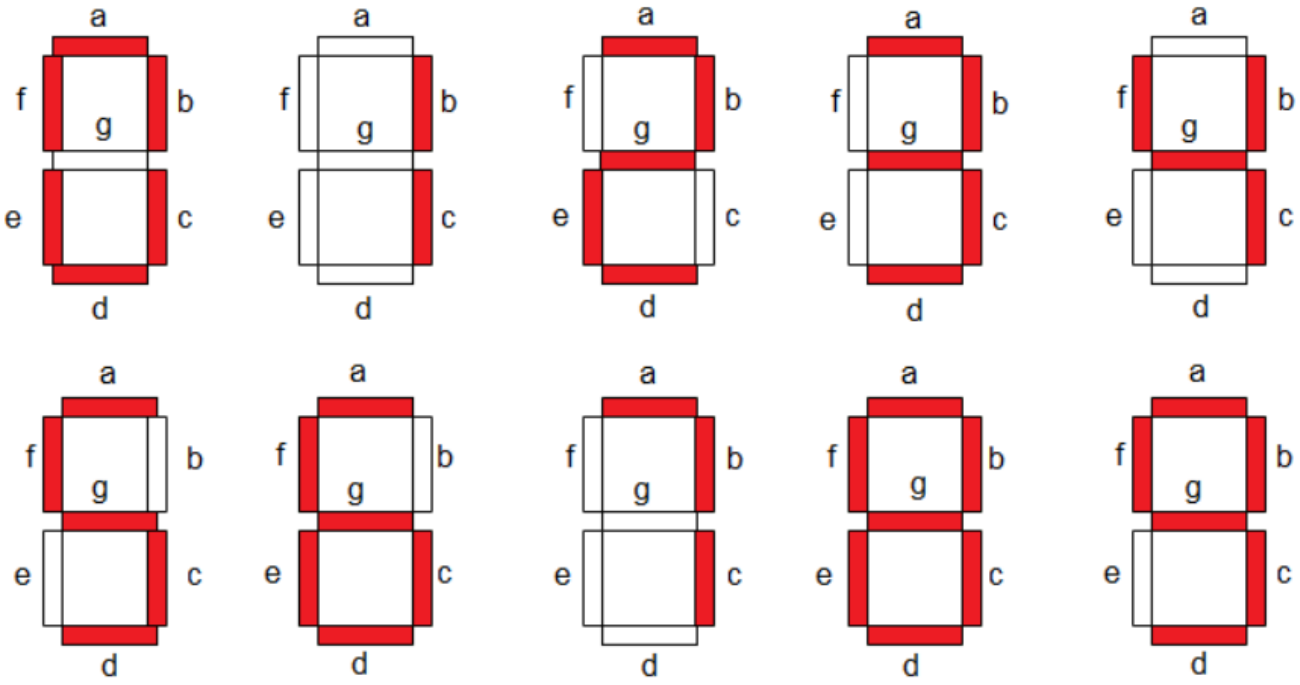


קידוד ב- Arduino IDE

הקוד לשליטה בתצוגה בת 7 מקטעים עם ESP32 ב-Arduino IDE כולל הגדרת פיני GPIO כיציאות ולאחר מכן שליחת האותות HIGH או LOW המתאימים לכל מקטע כדי להציג את המספר או התו הרצויים.

הצגת מספר

כדי להציג מספר, עליך ליצור תבנית שבה המקטעים המתאימים מוארים. לדוגמה, כדי להציג '3', קטעים a, b, c, d ו-g צריכים להיות מוארים.



טבלת חיבורים.

Segments							
a	b	c	d	e	f	g	
1	1	1	1	1	1	0	0
0	1	1	0	0	0	0	1
1	1	0	1	1	0	1	2
1	1	1	1	0	0	1	3
0	1	1	0	0	1	1	4
1	0	1	1	0	1	1	5
1	0	1	1	1	1	1	6
1	1	1	0	0	0	0	7
1	1	1	1	1	1	1	8
1	1	1	1	0	1	1	9
1	1	1	0	1	1	1	A
0	0	1	1	1	1	1	b
1	0	0	1	1	1	0	C
0	1	1	1	1	0	1	d
1	0	0	1	1	1	1	E
1	0	0	0	1	1	1	F

ניסוי מס' 1. הצגת מספר "3" על הצג בחיבור CC.

חבר 7 סגמנט לפי מה שכתוב בתוכנה והפעל אותה.

```
int segments[] = {2, 3, 4, 5, 6, 7, 8}; // Connect to segments a-g
void setup() {
  for (int i = 0; i < 7; i++) {
    pinMode(segments[i], OUTPUT);
  }
  // Display number '3'
  digitalWrite(segments[0], HIGH); // a
  digitalWrite(segments[1], HIGH); // b
  digitalWrite(segments[2], HIGH); // c
  digitalWrite(segments[3], HIGH); // d
  digitalWrite(segments[4], LOW);   //e
  digitalWrite(segments[5], LOW);   //f
  digitalWrite(segments[6], HIGH); // g
}
void loop() {
  // Rest of your code
}
```

תרגיל מס' 1. תכנון ספרות בחיבור CC.

תכנן כל ספרות מ-0 עד 9 ותריץ בדיקה ספרה אחרי ספרה בחיבור CC.

ניסוי מס' 2. הצגת מספר "3" על הצג בחיבור CA.

חבר 7 סגמנט לפי מה שכתוב בתוכנה והפעל אותה.

```
int segments[] = {2, 3, 4, 5, 6, 7, 8}; // GPIO pins connected to segments a-g
void setup() {
  // Initialize all the segment pins as output
  for (int i = 0; i < 7; i++) {
    pinMode(segments[i], OUTPUT);
  }
  // Display the number '3'
  digitalWrite(segments[0], LOW); // Segment a
  digitalWrite(segments[1], LOW); // Segment b
  digitalWrite(segments[2], LOW); // Segment c
  digitalWrite(segments[3], LOW); // Segment d
  digitalWrite(segments[6], LOW); // Segment g
  // Turn off the segments e and f
  digitalWrite(segments[4], HIGH); // Segment e
  digitalWrite(segments[5], HIGH); // Segment f
}
void loop() {
  // Your code here (if needed)
}
```

תרגיל מס' 2. תכנון ספרות בחיבור CA.

תכנן כל ספרות מ-0 עד 9 ותריץ בדיקה ספרה אחרי ספרה בחיבור CA.

שימוש במערך כדי להריץ ספרות על הצג.

ניסוי מס' 3. הרצת ספרות לפי צורת חיבור CC.

חבר 7 סגמנט לפי פינים שמוגדרים בתוכנה, תריץ את הקוד ותראה האם ספרות מתחלפות כמו שצריך.

```
// Segment connection pins: a, b, c, d, e, f, g
int segments[] = {2, 3, 4, 5, 6, 7, 8};
// Digit patterns for 0-9
byte digits[10][7] = {
  {1, 1, 1, 1, 1, 1, 0},  // 0
  {0, 1, 1, 0, 0, 0, 0},  // 1
  {1, 1, 0, 1, 1, 0, 1},  // 2
  {1, 1, 1, 1, 0, 0, 1},  // 3
  {0, 1, 1, 0, 0, 1, 1},  // 4
  {1, 0, 1, 1, 0, 1, 1},  // 5
  {1, 0, 1, 1, 1, 1, 1},  // 6
  {1, 1, 1, 0, 0, 0, 0},  // 7
  {1, 1, 1, 1, 1, 1, 1},  // 8
  {1, 1, 1, 1, 0, 1, 1}   // 9
};
void setup() {
  for (int i = 0; i < 7; i++) {
    pinMode(segments[i], OUTPUT);
  }
}
void loop() {
  for (int digit = 0; digit < 10; digit++) {
    for (int segment = 0; segment < 7; segment++) {
      digitalWrite(segments[segment], digits[digit][segment]);
    }
    delay(1000); // Display each number for 1 second
  }
}
```

תרגיל מס' 3. הרצת ספרות בחיבור CC.

כתוב תוכנית שתריץ ספרות מ-0 עד 9 ובחזרה בחיבור CC ותבדוק תקינות התוכנית לאחר צריבת הקוד.

ניסוי מס' 4. הרצת ספרות לפי צורת חיבור CA.

חבר 7 סגמנט לפי פינים שמוגדרים בתוכנה, תריץ את הקוד ותראה האם ספרות מתחלפות כמו שצריך.

```
// Segment connection pins: a, b, c, d, e, f, g
int segments[] = {2, 3, 4, 5, 6, 7, 8};
// Digit patterns for 0-9
byte digits[10][7] = {
  {0, 0, 0, 0, 0, 0, 1},  // 0
  {1, 0, 0, 1, 1, 1, 1},  // 1
  {0, 0, 1, 0, 0, 1, 0},  // 2
  {0, 0, 0, 0, 1, 1, 0},  // 3
  {1, 0, 0, 1, 1, 0, 0},  // 4
  {0, 1, 0, 0, 1, 0, 0},  // 5
  {0, 1, 0, 0, 0, 0, 0},  // 6
  {0, 0, 0, 1, 1, 1, 1},  // 7
  {0, 0, 0, 0, 0, 0, 0},  // 8
  {0, 0, 0, 0, 1, 0, 0}   // 9
```

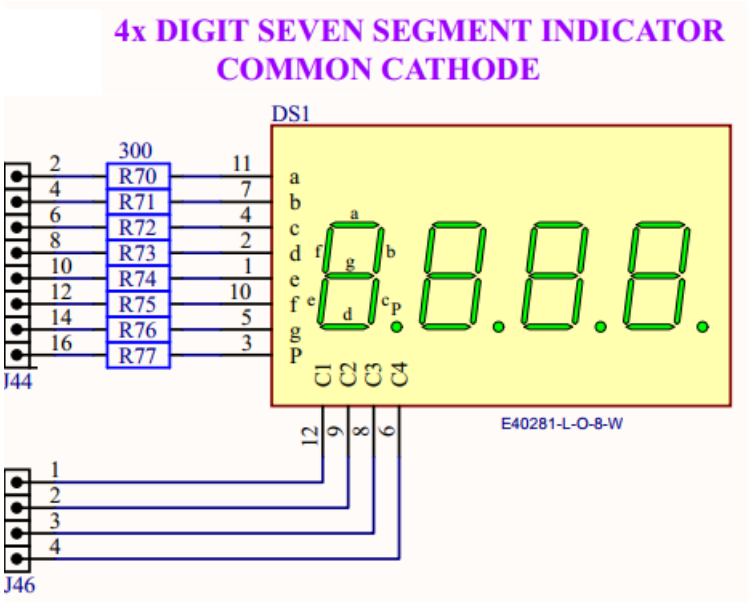
```
};
void setup() {
  for (int i = 0; i < 7; i++) {
    pinMode(segments[i], OUTPUT);
  }
}
void loop() {
  for (int digit = 0; digit < 10; digit++) {
    for (int segment = 0; segment < 7; segment++) {
      digitalWrite(segments[segment], digits[digit][segment]);
    }
    delay(1000); // Display each number for 1 second
  }
}
```

תרגיל מס' 4. הרצת ספרות בחיבור CA.
כתוב תוכנית שתריץ ספרות מ-0 עד 9 ובחזרה בחיבור CA ותבדוק תקינות התוכנית לאחר צריבת הקוד.

צג 4 ספרות 7 סגמנט מקבילי.

ניסוי מס' 1. הפעלת צג 4 ספרות 7 סגמנט CC.

בפרק זה, נחקור כיצד ליצור תצוגה בת 4 ספרות בת 7 מקטעים באמצעות תצורת קתודה משותפת מבלי להסתמך על ספריות חיצוניות. נשתמש במיקרו-בקר ESP32 ב-Arduino IDE כדי לשלוט בתצוגה ולהציג ערכים מספריים מ-0 עד 9 בכל ספרה.



מחברים 8 רגליים של ESP לבחירת סגמנטים במחבר J44 ו-4 רגליים נוספות של ESP לבחירת ספרה במחבר J46.

```
// Define GPIO pins for segments (a-g) and common anodes for each digit
int segmentPins[] = {2, 3, 4, 5, 6, 7, 8};
int digitPins[] = {9, 10, 11, 12};
void setup() {
   // Initialize all pins as outputs
   for (int i = 0; i < 7; i++) {
      pinMode(segmentPins[i], OUTPUT);
   }
   for (int i = 0; i < 4; i++) {
```

```
        pinMode(digitPins[i], OUTPUT);
    }
}
void displayDigit(int digit) {
    // Define segment patterns for numbers 0 to 9
    int digitPatterns[][7] = {
        {1, 1, 1, 1, 1, 1, 0},  // 0
        {0, 1, 1, 0, 0, 0, 0},  // 1
        {1, 1, 0, 1, 1, 0, 1},  // 2
        {1, 1, 1, 1, 0, 0, 1},  // 3
        {0, 1, 1, 0, 0, 1, 1},  // 4
        {1, 0, 1, 1, 0, 1, 1},  // 5
        {1, 0, 1, 1, 1, 1, 1},  // 6
        {1, 1, 1, 0, 0, 0, 0},  // 7
        {1, 1, 1, 1, 1, 1, 1},  // 8
        {1, 1, 1, 1, 0, 1, 1}   // 9
    };
    // Loop through each segment
    for (int i = 0; i < 7; i++) {
        digitalWrite(segmentPins[i], digitPatterns[digit][i]);
    }
    // Activate current digit
    digitalWrite(digitPins[digit], LOW);
    // Short delay for multiplexing
    delay(5);
    // Deactivate current digit
    digitalWrite(digitPins[digit], HIGH);
}
void loop() {
    // Loop through digits 0 to 9
    for (int digit = 0; digit < 4; digit++) {
        // Display the current digit
        displayDigit(digit);
    }
}
```

ניסוי מס' 2. הפעלת צג 4 ספרות 7 סגמנט CA.

ההבדל בין ניסוי מס' 1 וניסוי מס' 2, זה רק צירופים להפעלה:

```
// Define GPIO pins for segments (a-g) and common cathodes for each digit
int segmentPins[] = {2, 3, 4, 5, 6, 7, 8};
int digitPins[] = {9, 10, 11, 12};
void setup() {
    // Initialize all pins as outputs
    for (int i = 0; i < 7; i++) {
        pinMode(segmentPins[i], OUTPUT);
    }
    for (int i = 0; i < 4; i++) {
        pinMode(digitPins[i], OUTPUT);
    }
}
void displayDigit(int digit) {
    // Define segment patterns for numbers 0 to 9
    int digitPatterns[][7] = {
        {0, 0, 0, 0, 0, 0, 1},  // 0
        {1, 0, 0, 1, 1, 1, 1},  // 1
        {0, 0, 1, 0, 0, 1, 0},  // 2
        {0, 0, 0, 0, 1, 1, 0},  // 3
```

```
    {1, 0, 0, 1, 1, 0, 0},  // 4
    {0, 1, 0, 0, 1, 0, 0},  // 5
    {0, 1, 0, 0, 0, 0, 0},  // 6
    {0, 0, 0, 1, 1, 1, 1},  // 7
    {0, 0, 0, 0, 0, 0, 0},  // 8
    {0, 0, 0, 0, 1, 0, 0}   // 9
  };
  // Loop through each segment
  for (int i = 0; i < 7; i++) {
    digitalWrite(segmentPins[i], digitPatterns[digit][i]);
  }
  // Activate current digit
  digitalWrite(digitPins[digit], HIGH);
  // Short delay for multiplexing
  delay(5);
  // Deactivate current digit
  digitalWrite(digitPins[digit], LOW);
}
void loop() {
  // Loop through digits 0 to 9
  for (int digit = 0; digit < 4; digit++) {
    // Display the current digit
    displayDigit(digit);
  }
}
```

תרגול בהפעלת צג 4 ספרות 7 סגמנט.

תרגיל מס' 1.

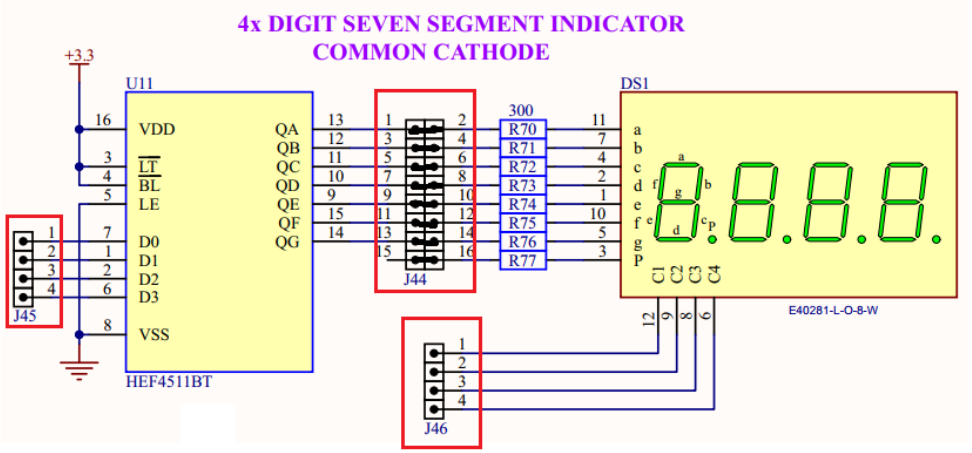
תבנה שעון 4 ספרות (דקות ושעות) כך ששעון סופר מ-00:00 עד 23-59. דקות מתחלפות פעם ברבע שניה. פעם חיבור CC ופעם שניה חיבור CA.

תרגיל מס' 2.

תבנה סטופר 4 ספרות (דקות ושניות) שסופר לכיוון מטה ממספר מסוים שמוגדר בזיכרון עד המצב 00:00. כאשר הוא מגיע לסוף תוכנית מדליקה לד. פעם חיבור CC ופעם שניה חיבור CA.

צג 4 ספרות 7 סגמנט BCD בחיבור CC.

כדי להפעיל מעגל יש לבצע פעולות הבאות:



בעזרת חוטי נקבה-נקבה או בעזרת ג'מפרים מקצרים פינים סמוכים של J44, מחברים 4 רגליים של ESP ל-J45 ו-4 רגליים נוספות ל-J46 כדי לבחור ספרה מתאימה.

ניסוי 1. הרצת מספרים בעזרת צג 4 ספרות 7 סגמנט BCD בחיבור CC לספרה בודדת.

```
int segmentBCD[] = {2, 3, 4, 5};
int Pin = 9;
int bcdPatterns[10][4] = {
      {0, 0, 0, 0},  // 0
      {0, 0, 0, 1},  // 1
      {0, 0, 1, 0},  // 2
      {0, 0, 1, 1},  // 3
      {0, 1, 0, 0},  // 4
      {0, 1, 0, 1},  // 5
      {0, 1, 1, 0},  // 6
      {0, 1, 1, 1},  // 7
      {1, 0, 0, 0},  // 8
      {1, 0, 0, 1} }; // 9
int digit, i, j;
void setup() {    // Initialize all pins as outputs
   for ( i = 0; i < 4; i++) {
      pinMode( segmentBCD [i], OUTPUT);
   }
     pinMode(Pin, OUTPUT);
 }
void loop() {
      for(i=0;i<9;i++)
{
for(j=0;j<4;j++)
{
segmentBCD[j]=bcdPatterns[i][j];
}
delay(1000);
}
   }
}
```

ניסוי 2. הרצת מספרים בעזרת צג 4 ספרות 7 סגמנט BCD בחיבור CC ל-4 ספרות.

```
// Define GPIO pins for segments (a-g) and common anodes for each digit
int segmentPins[] = {2, 3, 4, 5, 6, 7, 8};
int digitPins[] = {9, 10, 11, 12};
void setup() {
   // Initialize all pins as outputs
   for (int i = 0; i < 7; i++) {
      pinMode(segmentPins[i], OUTPUT);
   }
   for (int i = 0; i < 4; i++) {
      pinMode(digitPins[i], OUTPUT);
   }
}
void displayDigit(int digit) {
   // Define segment patterns for numbers 0 to 9
int bcdPatterns[10][4] = {
      {0, 0, 0, 0},  // 0
      {0, 0, 0, 1},  // 1
      {0, 0, 1, 0},  // 2
      {0, 0, 1, 1},  // 3
      {0, 1, 0, 0},  // 4
      {0, 1, 0, 1},  // 5
```

```
      {0, 1, 1, 0},  // 6
      {0, 1, 1, 1},  // 7
      {1, 0, 0, 0},  // 8
      {1, 0, 0, 1} } ; // 9
  // Loop through each segment
  for (int i = 0; i < 7; i++) {
    digitalWrite(segmentPins[i], bcdPatterns[digit][i]);
  }
  // Activate current digit
  digitalWrite(digitPins[digit], LOW);
  // Short delay for multiplexing
  delay(5);
  // Deactivate current digit
  digitalWrite(digitPins[digit], HIGH);
}
void loop() {
  // Loop through digits 0 to 9
  for (int digit = 0; digit < 4; digit++) {
    // Display the current digit
    displayDigit(digit);
  }
}
```

תרגול בהפעלת צג BCD 4 ספרות 7 סגמנט.

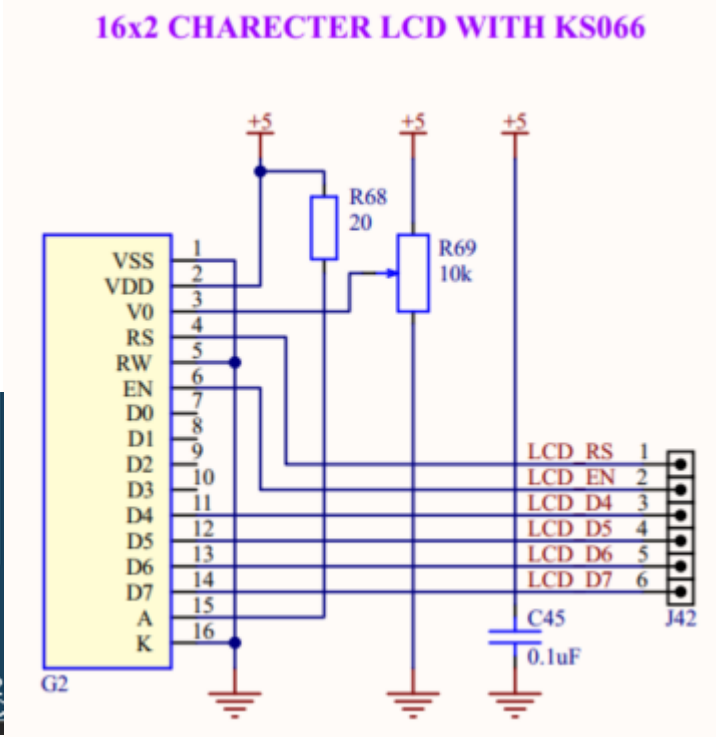
תרגיל מס' 1.

תבנה שעון 4 ספרות (דקות ושעות) כך ששעון סופר מ-00:00 עד 23-59. דקות מתחלפות פעם ברבע שניה.

תרגיל מס' 2.

תבנה סטופר 4 ספרות (דקות ושניות) שסופר לכיוון מטה ממספר מסוים שמוגדר בזיכרון עד המצב 00:00. כאשר הוא מגיע לסוף תוכנית מדליקה לד.

מסך LCD מקבילי.



LCD (צג גביש נוזלי) (16x2) הוא מודול תצוגה אלפאנומרי נפוץ שנמצא בשימוש נרחב בפרויקטים אלקטרוניים שונים. הוא מורכב משתי שורות, כל אחת מסוגלת להציג עד 16 תווים, מה שהופך אותו למתאים להצגת מידע טקסט. בסעיף זה, נחקור כיצד לממשק צג LCD 2x16 עם מיקרו-בקר ESP32 באמצעות Arduino IDE. אנו נספק הסבר על פונקציות בסיסיות ונספק דוגמאות לקוד כדי להדגים את השימוש בו.

סקירה כללית של תצוגת LCD 2x16:

מאפיינים:

2 שורות x 16 תווים: מודול התצוגה מורכב משתי שורות, כל אחת מסוגלת להציג עד 16 תווים, מה שמאפשר הצגת מידע טקסטואלי.

ממשק מקביל: הוא משתמש בדרך כלל בממשק מקביל כדי לתקשר עם מיקרו-בקרים, הדורש מספר נתונים ופיני בקרה.

תאורה אחורית: צגי LCD רבים בגודל 2 x16 מגיעים עם תאורה אחורית מובנית, הניתנת לשליטה כדי לשפר את הנראות בתנאי תאורה שונים.

בקר HD44780: שבב הבקר הנפוץ ביותר בשימוש בתצוגות LCD 2x16 הוא ה-HD44780, המפשט את ההתממשקות עם מיקרו-בקרים.

ממשק עם ESP32:

חיבור חומרה:

כדי לממשק צג LCD 2x16 עם מיקרו-בקר ESP32, יהיה עליך לבצע את החיבורים הבאים:

חבר את פין ה-RS (Register Select) של ה-LCD לפין GPIO ב-ESP32 (למשל, GPIO 2).

חבר את פין ה-Enable (EN) של ה-LCD לפין GPIO אחר ב-ESP32 (לדוגמה, GPIO 3).

חבר את הפינים D4, D5, D6 ו-D7 של ה-LCD לפיני GPIO ב-ESP32 (למשל, GPIO 4, 5, 6 ו-7).

לחלופין, חבר את סיכת התאורה האחורית (אם זמין) לפין GPIO לשליטה בתאורה אחורית.

שילוב תוכנה:

כדי לשלוט בתצוגת ה-LCD 2x16 מהמיקרו-בקר ESP32 ב-Arduino IDE, בדרך כלל תשתמש בספריית LiquidCrystal. ספרייה זו מספקת פונקציות לאתחול התצוגה, כתיבת טקסט לתצוגה, שליטה במיקום הסמן וניהול תאורה אחורית (אם רלוונטי).

```
#include <LiquidCrystal.h>

// Define LCD pins
#define LCD_RS 2
#define LCD_EN 3
#define LCD_D4 4
#define LCD_D5 5
#define LCD_D6 6
#define LCD_D7 7

// Define LCD dimensions
#define LCD_COLUMNS 16
#define LCD_ROWS 2

// Create an instance of the LiquidCrystal library
LiquidCrystal lcd(LCD_RS, LCD_EN, LCD_D4, LCD_D5, LCD_D6, LCD_D7);

void setup() {
  // Initialize the LCD with the specified dimensions
  lcd.begin(LCD_COLUMNS, LCD_ROWS);

  // Print a welcome message
  lcd.print("Hello, ESP32!");
  lcd.setCursor(0, 1); // Move cursor to the second line
  lcd.print("LCD 2x16 Example");
}

void loop() {
  // Main program loop (empty for demonstration purposes)
}
```

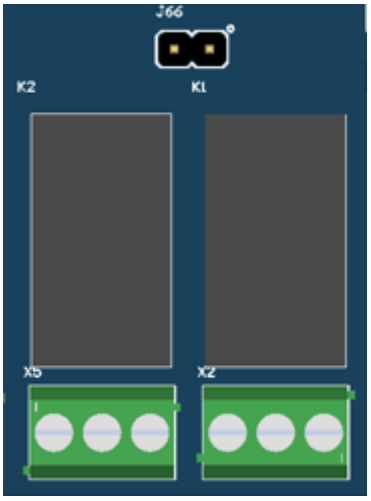
רשימת פונקציות בסיסיות:

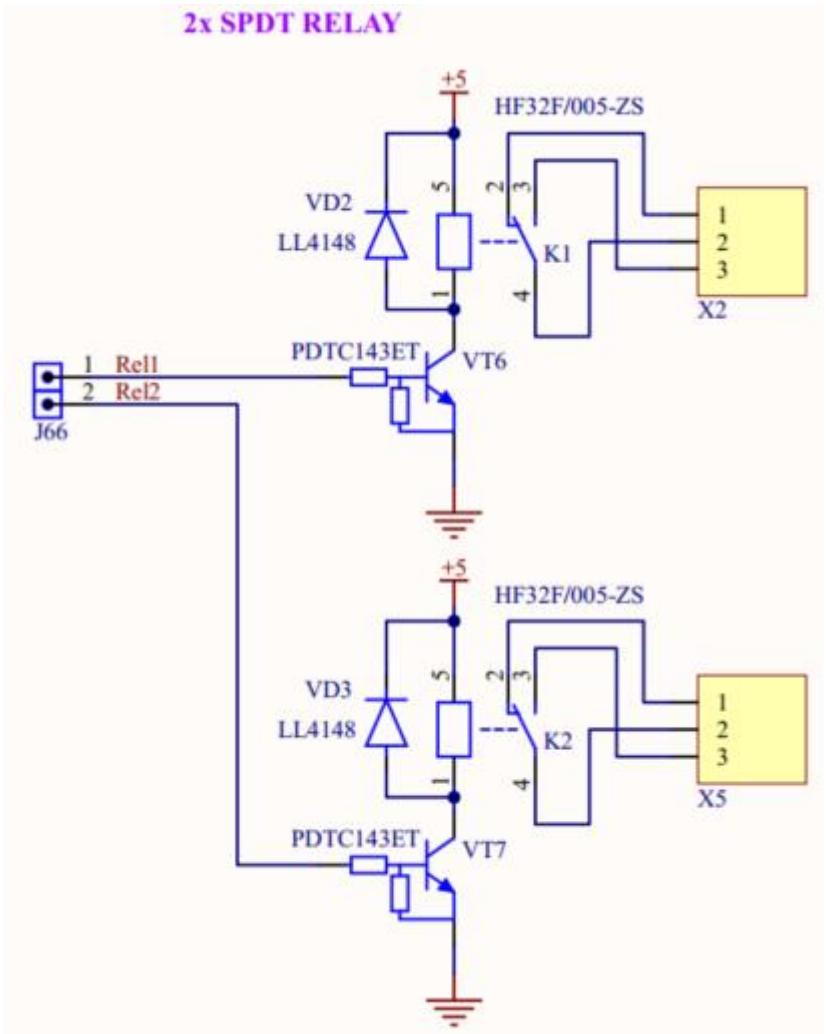
פונקציה	פרוש
begin(16, 2)	איתחול
clear()	ניקוי מסך
print()	הדפסת טקסט על המסך
println()	הדפסת טקסט על המסך עם העברת שורה
write()	כתיבה ישירה למסך
`home()`	חזרה לתחילת המסך
`setCursor()`	להעביר סמן לשורה ועמודה מסוימת
`cursor()`	הופעת הסמן
`noCursor()`	הסתרת הסמן
`blink()`	הסמן עם הבהוב
`noBlink()`	הסמן ללא הבהוב
`createChar()`	ייצור אות חדש ספציפי שלא סטנדרטי
`scrollDisplayLeft()`	הזזה מסך מיקום שמאלה
`scrollDisplayRight()`	הזזה מסך מיקום ימינה

דוגמאות שחמוש במסך.

```
LiquidCrystal My_LCD(13, 12, 14, 27, 26, 25);

byte Speaker[] = {
  B00001,
  B00011,
  B00111,
  B11111,
  B11111,
  B00111,
  B00011,
  B00001
};

void setup()
{
  My_LCD.begin(16, 2);
  My_LCD.createChar(0, Speaker);
  My_LCD.clear();
  My_LCD.print("Speaker");
  My_LCD.write(byte(0));
}

void loop()
{

}
```

הפעלת ממסר כפול.





ממסרים הם מתגים אלקטרו-מכאניים המאפשרים למיקרו-בקרים במתח נמוך כמו ESP32 לשלוט בהתקני מתח גבוה וזרם גבוה כגון מנועים, אורות ומכשירים. בסעיף זה, נחקור כיצד לממשק מודול ממסר עם מיקרו-בקר ESP32, נסביר את המעגלים המעורבים ונספק דוגמאות למנגנוני ממסר.

סקירה כללית של מודול ממסר:

ממסר: הלב של מודול הממסר הוא סליל אלקטרומגנטי שכאשר הוא מופעל, מעביר את מגעי הממסר בין מצב פתוח לסגור.

מגעי ממסר: לממסרים יש בדרך כלל זוג מגעים אחד או יותר (Normally Open and Normally Closed) המשנים מצב כאשר הסליל מופעל.

מעגל דרייבר: מודול הממסר כולל רכיבים כגון טרנזיסטורים, דיודות ונגדים כדי להניע את סליל הממסר ולהגן על המיקרו-בקר מפני קוצים במתח.

סוגי ממסרים:

מתג חד-קוטבי (SPST): מכיל זוג מגעים אחד פתוחים או סגורים.

מתג כפול חד-קוטבי (SPDT): מכיל זוג אחד של מגעים פתוחים בדרך כלל (NO) וזוג אחד של מגעים סגורים בדרך כלל (NC).

מתג כפול דו-קוטבי (DPDT): מכיל שני זוגות של מגעי NO ו-NC.

ממשק עם ESP32:

חיבור מעגל:

כדי להפעיל מודול ממסר עם בקר מיקרו ESP32, בצע את השלבים הבאים:

חבר את פיני VCC ו-GND של מודול הממסר לפיני 5 V ו-GND של ESP32, בהתאמה.

חבר את פין האות (IN) של מודול הממסר לפין GPIO ב-ESP32 (לדוגמה, GPIO 2).

מנגנון ממסר:

מודול הממסר נשלט על ידי המיקרו-בקר ESP32 דרך פין האות (IN). כאשר פין GPIO מוגדר ל-HIGH, זרם זורם דרך סליל הממסר, ממריץ אותו וגורם למגעי הממסר להחליף מצב. זה מאפשר לזרם לזרום דרך המגעים הנפוצים (COM) והפתוחים בדרך כלל (NO), מפעיל את העומס המחובר (למשל, נורה). בערכה שלנו ממסרים פועלים בגבוה. (ישנם ממסרים שעובדים בנמוך, אם נחבר זוג אופטי בכניסה).

דוגמה לקוד הפעלה:

```
#define RELAY_PIN 2  // GPIO pin connected to relay module

void setup() {
  pinMode(RELAY_PIN, OUTPUT);  // Set relay pin as output
}

void loop() {
  // Turn on the relay for 2 seconds
  digitalWrite(RELAY_PIN, HIGH);
  delay(2000);  // 2-second delay
  // Turn off the relay for 2 seconds
  digitalWrite(RELAY_PIN, LOW);
  delay(2000);  // 2-second delay
}
```

מנגנון ריטוטים – Bouncing.

ריטוטים היא בעיה שכיחה שנתקלת בה בעת חיבור מתגים מכאניים, לחצנים או התקני קלט אחרים עם מיקרו-בקרים כמו ESP32. זה מתרחש עקב המאפיינים הפיזיים של מגעי המתג, וכתוצאה מכך מעברים מהירים (קפיצות) מרובים בין מצב פתוח וסגור בעת לחיצה או שחרור של המתג. בסעיף זה, נתעמק במושג הריטוטים, נסביר את טכניקות ביטול (טיפול ב-) ריטוטים (Debouncing) בתוכנה והחומרה, ונספק דוגמאות ליישום שלהן עם המיקרו-בקר ESP32.

הבנת ריטוטים:

גורם ל:

ריטוטים נגרמים בעיקר על ידי תנועה מכנית וקפיצת מגע של מגעי מתג בעת לחיצה או שחרור. זה מוביל למעברי מצב מהירים מרובים תוך פרק זמן קצר, מה שעלול לגרום להתנהגות לא מכוונת או לשגיאות במערכות מבוססות מיקרו-בקר.

אפקטים:

ההשפעות של ריטוטים יכולות לכלול:

הפעלה כוזבת של שגרות מבוססות פסיקות.

קריאה לא מדויקת של מצבי מתג.

התנהגות לא סדירה של המערכת.

טיפול בתוכנה:

ביטול בעזרת תוכנה כולל הטמעת אלגוריתמים או טכניקות כדי לסנן את הריטוטים ולספק מצב מתג יציב ואמין. טכניקות נפוצות לטיפול בתוכנה כוללות:

Delay-based Debouncing: הצגת השהייה קצרה (זמן דה-bounce) לאחר זיהוי מעבר מתג כדי לאפשר קביעת זמן לפני קריאת מצב המתג.

זיהוי מצב שינוי: רישום השעה של שינוי מצב המתג האחרון והתעלמות משינויים עוקבים בתוך חלון זמן יציאה מוגדר.

מכונת מצב סופי (FSM): הטמעת מכונת מצב סופי כדי לעקוב אחר מעברי מצב המתג ולסנן הקפצות.

הוצאת חומרה:

ניתוק חומרה כולל שילוב של רכיבים או מעגלים נוספים כדי להפחית הקפצה ברמת החומרה. טכניקות ניתוק חומרה נפוצות כוללות:

מסנן RC: שימוש במסנן נגד-קבלים (RC) כדי להחליק את אות המתג ולבטל שינויי מתח מהירים הנגרמים מהקפצה.

Schmitt Trigger: שימוש בכניסת הדק של Schmitt כדי לספק היסטרזה ולייצב את אות המתג, תוך הבטחת מעברים נקיים בין רמות לוגיות.

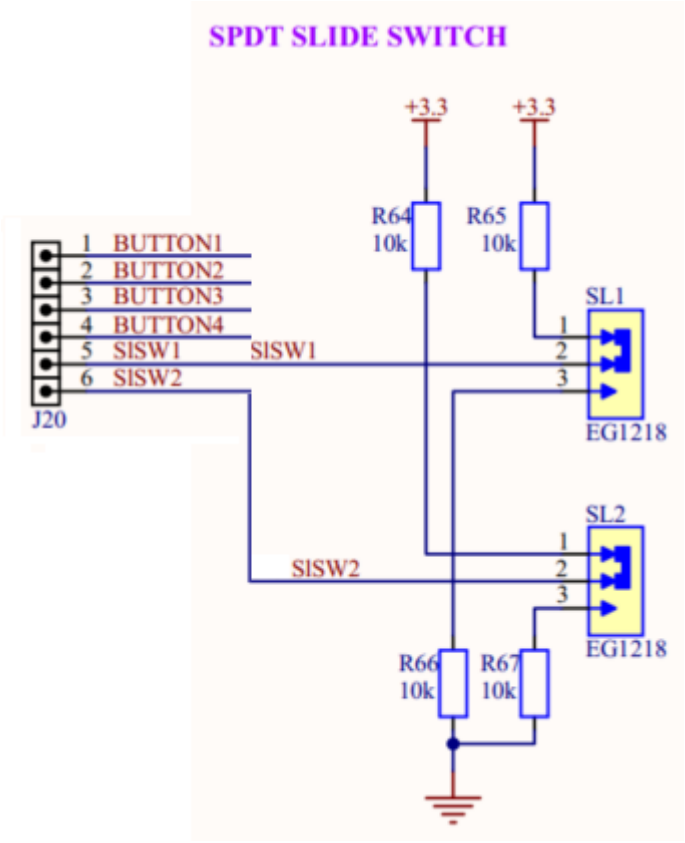
IC ייעודי ל-Debouncing: שימוש במעגלים משולבים (ICs) ייעודיים ל-debouncing או ICs ל-Debounce כדי לספק יציאה מהימנה של רכיבים חיצוניים מינימליים.

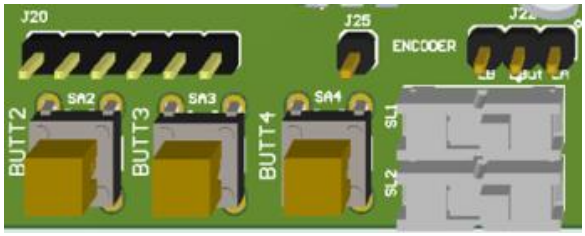
```
#define SWITCH_PIN 2
#define DEBOUNCE_DELAY 50 // Debounce delay in milliseconds

void setup() {
  pinMode(SWITCH_PIN, INPUT_PULLUP);
  Serial.begin(9600);
}

void loop() {
  static unsigned long lastDebounceTime = 0;
  static int lastSwitchState = HIGH;
  int switchState = digitalRead(SWITCH_PIN);

  if (switchState != lastSwitchState) {
    lastDebounceTime = millis();
  }

  if (millis() - lastDebounceTime > DEBOUNCE_DELAY) {
    if (switchState != lastSwitchState) {
      lastSwitchState = switchState;
      Serial.print("Switch state: ");
      Serial.println(switchState);
    }
  }
}
```

הבנה וטיפול בבעיות הקפצות הן חיוניות להבטחת הפעולה האמינה של מערכות מבוססות מתגים עם מיקרו-בקרים כמו ESP32. על ידי הטמעת טכניקות של שחרור תוכנה וחומרה, אתה יכול לסנן ביעילות הקפצות חולפות ולהשיג זיהוי מצב מתג יציב ומדויק בפרויקטים מבוססי ESP32 שלך. בין אם אתה משתמש בהרחקה מבוססת עיכוב בתוכנה או שימוש

במסנני RC או בטריגרים של Schmitt בחומרה, בחירת שיטת ההקפצה המתאימה תלויה בדרישות ובאילוצים הספציפיים של היישום שלך.

קלט ממפסקי הזזה.





מתגי החלקה הם רכיבים פשוטים אך יעילים המשמשים בדרך כלל במעגלים אלקטרוניים כדי לשלוט בזרימת הזרם. בסעיף זה, נחקור כיצד להפעיל מתג Slide Switch עם מיקרו-בקר ESP32, נסביר את המעגל המעורב ונספק דוגמאות לשימוש בו.

סקירה כללית של מתג שקף:

רכיבים:

Slide Switch: מתג מכני עם ידית הזזה הנעה בין שני מצבים או יותר כדי ליצור או לשבור חיבורים חשמליים.

מגעים: מתג ההחלקה מכיל מגעי מתכת מוליכים המתחברים או מתנתקים בעת הזזת הידית.

מסופים: למתג יש מסופים אליהם ניתן לחבר חוטים או מובילים כדי ליצור חיבורים חשמליים.

סוגי מתגי החלקה:

זריקה חד-קוטבית (SPST): מכיל סט אחד של מגעים פתוחים או סגורים.

זריקה כפולה חד-קוטבית (SPDT): מכיל סט אחד של מגעים שניתן לחבר לאחד משני מסופים.

ממשק עם ESP32:

חיבור מעגל:

כדי לקלוט מתג שקף עם מיקרו-בקר ESP32, בצע את השלבים הבאים:

חבר מסוף אחד של מתג ההחלקה לפין GPIO ב-ESP32 (לדוגמה, GPIO 2).

חבר את המסוף השני של מתג ההחלקה לנגד Pull Up (בדרך כלל 10 kΩ).

חבר את הקצה השני של הנגד המשיכה לפין 3.3 V של ה-ESP32.

לחלופין, חבר קבל בין פין ה-GPIO להארקה כדי לשחרר את המתג (להפחית רעש).

מנגנון מתג החלקה:

מתג ההחלקה פועל על ידי החלקה פיזית של מנוף בין שני מצבים או יותר. כאשר הידית נמצאת במצב אחד, המגעים סגורים, מה שמאפשר לזרום לזרם בין המסופים. כאשר מעבירים את הידית למצב השני, המגעים נפתחים ומנתקים את החיבור החשמלי.

```
#define SWITCH_PIN 2  // GPIO pin connected to slide switch
void setup() {
  pinMode(SWITCH_PIN, INPUT_PULLUP);  // Set switch pin as input with internal pull-up resistor
  Serial.begin(9600);  // Initialize serial communication
}
void loop() {
  // Read the state of the slide switch
  int switchState = digitalRead(SWITCH_PIN);
  // Print the state of the switch
  if (switchState == HIGH) {
    Serial.println("Switch is OFF");
  } else {
    Serial.println("Switch is ON");
  }
  delay(1000);  // Delay for readability
}
```

ניסוי מס' 1. בקרת מתג בסיסית עם LED

הסבר מעגל:

חבר מסוף אחד של מתג ההחלקה לפין GPIO ב-ESP32 (למשל, GPIO 2), וחבר את המסוף השני לנגד מגביל זרם ולאחר מכן ל-LED. חבר את המסוף השני של הנורית לאדמה. הגדרה זו מאפשרת למתג ההחלקה לשלוט במצב LED.

```
#define SWITCH_PIN 2  // GPIO pin connected to slide switch
#define LED_PIN 13     // GPIO pin connected to LED

void setup() {
  pinMode(SWITCH_PIN, INPUT_PULLUP);  // Set switch pin as input with internal pull-up resistor
  pinMode(LED_PIN, OUTPUT);           // Set LED pin as output
}

void loop() {
  // Read the state of the slide switch
  int switchState = digitalRead(SWITCH_PIN);

  // Control the LED based on the switch state
  digitalWrite(LED_PIN, switchState);
}
```

ניסוי מס' 2. תצוגת מצב מתג החלקה באמצעות צג טורי

הסבר מעגל:

חבר מסוף אחד של מתג ההחלקה לפין GPIO ב-ESP32 (למשל, GPIO 2), וחבר את המסוף השני לאדמה. הגדרה זו מאפשרת לקרוא את מתג ההחלקה על ידי ה-ESP32.

```
#define SWITCH_PIN 2  // GPIO pin connected to slide switch

void setup() {
```

```
  pinMode(SWITCH_PIN, INPUT_PULLUP);  // Set switch pin as input with internal pull-up resistor
  Serial.begin(9600);                 // Initialize serial communication
}

void loop() {
  // Read the state of the slide switch
  int switchState = digitalRead(SWITCH_PIN);

  // Print the switch state to the serial monitor
  Serial.print("Switch state: ");
  Serial.println(switchState);

  delay(1000);  // Delay for readability
}
```

ניסוי מס' 3. החלפת מצב LED עם מתג.

הסבר מעגל:

חבר מסוף אחד של מתג ההחלקה לפין GPIO ב-ESP32 (למשל, GPIO 2), וחבר את המסוף השני לנגד מגביל זרם ולאחר מכן ל-LED. חבר את המסוף השני של הנורית לאדמה. הגדרה זו מאפשרת למתג ההחלקה להחליף את מצב LED.

```
#define SWITCH_PIN 2  // GPIO pin connected to slide switch
#define LED_PIN 13    // GPIO pin connected to LED

void setup() {
  pinMode(SWITCH_PIN, INPUT_PULLUP);  // Set switch pin as input with internal pull-up resistor
  pinMode(LED_PIN, OUTPUT);           // Set LED pin as output
}

void loop() {
  // Read the state of the slide switch
  int switchState = digitalRead(SWITCH_PIN);

  // Toggle the LED state based on the switch state
  if (switchState == LOW) {
    digitalWrite(LED_PIN, !digitalRead(LED_PIN));
    delay(250);  // Debounce delay
  }
}
```

ניסוי מס' 4. שליטה במספר נוריות LED עם מתג הזזה.

הסבר מעגל:

חבר מסוף אחד של מתג ההחלקה לפין GPIO ב-ESP32 (למשל, GPIO 2), וחבר את המסוף השני לאדמה. חבר מספר נוריות LED לפיני GPIO שונים ב-ESP32. הגדרה זו מאפשרת למתג ההחלקה לשלוט במצב של מספר נוריות LED בו זמנית.

```
#define SWITCH_PIN 2   // GPIO pin connected to slide switch
#define LED_PIN_1 13   // GPIO pin connected to LED 1
#define LED_PIN_2 14   // GPIO pin connected to LED 2
#define LED_PIN_3 15   // GPIO pin connected to LED 3

void setup() {
```

```
  pinMode(SWITCH_PIN, INPUT_PULLUP);  // Set switch pin as input with internal pull-up resistor
  pinMode(LED_PIN_1, OUTPUT);         // Set LED pins as output
  pinMode(LED_PIN_2, OUTPUT);
  pinMode(LED_PIN_3, OUTPUT);
}

void loop() {
  // Read the state of the slide switch
  int switchState = digitalRead(SWITCH_PIN);

  // Set the state of multiple LEDs based on the switch state
  digitalWrite(LED_PIN_1, switchState);
  digitalWrite(LED_PIN_2, switchState);
  digitalWrite(LED_PIN_3, switchState);
}
```

ניסוי מס' 5. בקרת מתג שקוף מתקדמת עם צג טורי

הסבר מעגל:

חבר מסוף אחד של מתג ההחלקה לפין GPIO ב-ESP32 (למשל, GPIO 2), וחבר את המסוף השני לאדמה. הגדרה זו מאפשרת לקרוא את מתג ההחלקה על ידי ה-ESP32. בנוסף, חבר את ה-ESP32 למחשב באמצעות USB לתקשורת טורית.

```
#define SWITCH_PIN 2  // GPIO pin connected to slide switch

void setup() {
  pinMode(SWITCH_PIN, INPUT_PULLUP);  // Set switch pin as input with internal pull-up resistor
  Serial.begin(9600);                 // Initialize serial communication
}

void loop() {
  // Read the state of the slide switch
  int switchState = digitalRead(SWITCH_PIN);

  // Print the switch state to the serial monitor
  Serial.print("Switch state: ");
  Serial.println(switchState);

  // Wait for serial input to change LED state
  if (Serial.available()) {
    char command = Serial.read();
    if (command == '1') {
      digitalWrite(LED_PIN, HIGH);
    } else if (command == '0') {
      digitalWrite(LED_PIN, LOW);
    }
  }
}
```

תרגול בנושא מתגי הזזה:

תרגיל 1: זיהוי מצב מתגים

מטרה: לקרוא את מצב של 2 מתגי החלקה המחוברים ל-ESP32 ולהדפיס את מצבו לצג הטורי.

הסבר: תרגיל זה מציג את הרעיון של קריאת קלט דיגיטלי ממתג שקף ומדגים כיצד לזהות את מצבו (ON או OFF) באמצעות פיני ה-GPIO של ה-ESP32.

תרגיל 2: ספירת זמן בהחלפת מצב של מתג.

מטרה: להציג על מסך הטורי כמה זמן עבר בין החלפת מצב 0 למצב 1 והפוך.

הסבר: תרגיל זה מתבסס על תרגיל 1 על ידי חישוב זמנים בהתבסס על מצב מתג ההחלקה, ומספק חוויה מעשית עם בקרת פלט על מסך טורי.

תרגיל 3: שליטה על נוריות LED מרובות עם מתג Slide

מטרה: לשלוט במצב של מספר נוריות LED בו-זמנית באמצעות מתג החלקה בודד.

הסבר: תרגיל זה מציג את הקונספט של שליטה במספר יציאות עם קלט יחיד, ומדגים את הרבגוניות של מתגי החלקה בניהול התקנים מרובים.

תרגיל 4: תצוגת מצב מתג Slide Advanced

מטרה: להציג את המצב של מתג החלקה על צג LCD המחובר ל-ESP32.

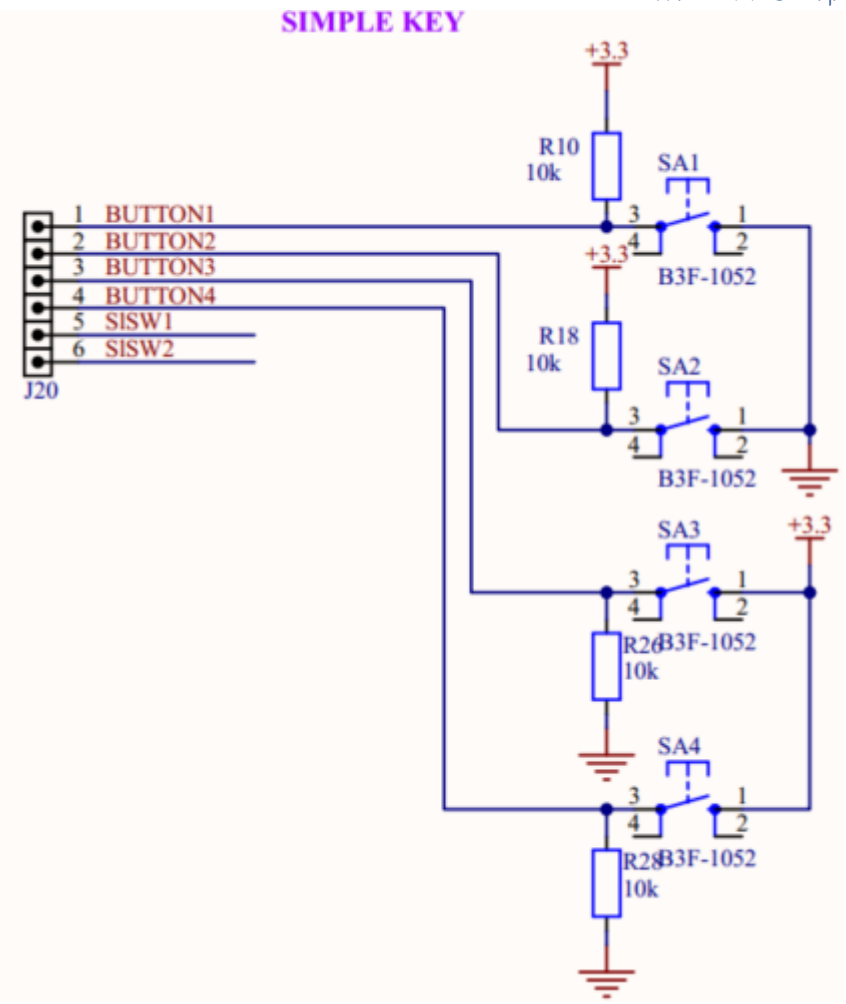
הסבר: תרגיל זה מרחיב את היישום של מתגי החלקה על ידי שילובם עם התקני פלט נוספים, שיפור המשוב והאינטראקציה של המשתמשים.

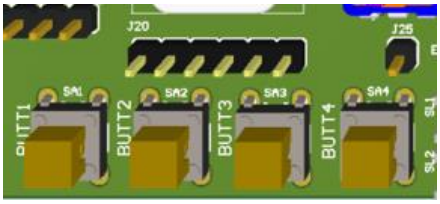
תרגיל 5: ספירת זמן בהחלפת מצב של מתג בעזרת LCD.

מטרה: להציג על מסך LCD כמה זמן עבר בין החלפת מצב 0 למצב 1 והפוך.

הסבר: תרגיל זה מתבסס על תרגיל 2 על ידי חישוב זמנים בהתבסס על מצב מתג ההחלקה, ומספק חוויה מעשית עם בקרת פלט על מסך LCD.

קלט ספרתי מלחצנים.





בערכה שלנו יש 2 לחצנים פתוחים בדרך כלל ו-2 לחצנים סגורים בדרך כלל. מה אפשר לעשות כדי להחליט מי מהם (.N.O) ומי (.N.C).

הנה קוד לדוגמה שמדגים כיצד לקבוע אילו לחצנים פתוחים בדרך כלל (.N.O) ואילו סגורים בדרך כלל (.N.C) באמצעות מיקרו-בקר ESP32:

```
#define BUTTON_NO_PIN_1 2   // GPIO pin connected to normally open button 1
#define BUTTON_NO_PIN_2 3   // GPIO pin connected to normally open button 2
#define BUTTON_NC_PIN_1 4   // GPIO pin connected to normally closed button 1
#define BUTTON_NC_PIN_2 5   // GPIO pin connected to normally closed button 2

void setup() {
  Serial.begin(9600);

  pinMode(BUTTON_NO_PIN_1, INPUT_PULLUP);
// Set normally open button pins as input with internal pull-up resistors
  pinMode(BUTTON_NO_PIN_2, INPUT_PULLUP);
  pinMode(BUTTON_NC_PIN_1, INPUT_PULLUP);
// Set normally closed button pins as input with internal pull-up resistors
  pinMode(BUTTON_NC_PIN_2, INPUT_PULLUP);
}

void loop() {
  // Read the state of normally open buttons
  int stateNO1 = digitalRead(BUTTON_NO_PIN_1);
```

```
  int stateNO2 = digitalRead(BUTTON_NO_PIN_2);

  // Read the state of normally closed buttons
  int stateNC1 = digitalRead(BUTTON_NC_PIN_1);
  int stateNC2 = digitalRead(BUTTON_NC_PIN_2);

  // Print the states of the buttons
  Serial.println("States of the buttons:");
  Serial.print("Normally Open Button 1: ");
  Serial.println(stateNO1 == HIGH ? "Pressed" : "Released");
  Serial.print("Normally Open Button 2: ");
  Serial.println(stateNO2 == HIGH ? "Pressed" : "Released");
  Serial.print("Normally Closed Button 1: ");
  Serial.println(stateNC1 == HIGH ? "Released" : "Pressed");
  Serial.print("Normally Closed Button 2: ");
  Serial.println(stateNC2 == HIGH ? "Released" : "Pressed");

  delay(1000);  // Delay for readability
}
```

בקוד זה, אנו מגדירים ארבעה פיני GPIO עבור שני לחצנים פתוחים בדרך כלל (N.O.) ושני לחצנים סגורים בדרך כלל (N.C.).

אנו מגדירים את הפינים הללו ככניסות עם נגדי משיכה פנימיים באמצעות פונקציית pinMode() כדי להבטיח שהקלט נמשך גבוה כאשר הכפתורים אינם נלחצים.

בתוך הפונקציה loop() אנו קוראים את המצב של כל כפתור באמצעות פונקציית digitalRead(), המחזירה HIGH אם הכפתור לא נלחץ ו-LOW אם הכפתור נלחץ.

לאחר מכן אנו מדפיסים את מצבי הכפתורים לצג הטורי, ומציינים אם כל כפתור נלחץ או משוחרר.

בהתבסס על המצבים המודפסים, אנו יכולים לקבוע אילו לחצנים פתוחים בדרך כלל (N.O.) ואילו סגורים בדרך כלל (N.C.). כפתורים פתוחים בדרך כלל יציגו "לחוץ" כאשר הם נלחצים, וכפתורים סגורים בדרך כלל יציגו "לחוץ" כאשר לא ילחצו עליהם.

על ידי התבוננות במצבי הכפתורים המודפסים לצג הטורי, תוכל לזהות אילו לחצנים פתוחים בדרך כלל ואילו סגורים בדרך כלל. התאם את החיווט בהתאם אם ההתנהגות הנצפית אינה תואמת את ההתנהגות הצפויה.

אם אינך בטוח אילו לחצנים פתוחים בדרך כלל (N.O.) ואילו סגורים בדרך כלל (N.C.), תוכל לקבוע את המצבים שלהם באופן תוכנתי על ידי בדיקת ההתנהגות שלהם כאשר המעגל סגור ומתי הוא פתוח. להלן קטע קוד שעוזר לזהות את סוג הכפתורים:

```
#define BUTTON_PIN 2  // GPIO pin connected to the button

void setup() {
  Serial.begin(9600);

  pinMode(BUTTON_PIN, INPUT_PULLUP);  // Set button pin as input with internal pull-up resistor
}

void loop() {
  // Read the initial state of the button
  int initialState = digitalRead(BUTTON_PIN);

  // Wait for the user to close the circuit (press the button)
  Serial.println("Please press the button...");
  while (digitalRead(BUTTON_PIN) == initialState) {
    // Wait until the button state changes
    delay(100);
```

```
  }

  // Record the state of the button when pressed
  int pressedState = digitalRead(BUTTON_PIN);

  // Wait for the user to release the button
  Serial.println("Please release the button...");
  while (digitalRead(BUTTON_PIN) != initialState) {
    // Wait until the button state returns to its initial state
    delay(100);
  }

  // Record the state of the button when released
  int releasedState = digitalRead(BUTTON_PIN);

  // Determine the button type based on its behavior
  if (initialState == HIGH && pressedState == LOW && releasedState == HIGH) {
    Serial.println("The button is normally open (N.O.)");
  } else if (initialState == LOW && pressedState == HIGH && releasedState == LOW) {
    Serial.println("The button is normally closed (N.C.)");
  } else {
    Serial.println("Unable to determine button type. Please try again.");
  }

  // Wait for a brief moment before repeating the process
  delay(1000);
}
```

הסבר:

בקוד זה, אנו קוראים תחילה את המצב ההתחלתי של הכפתור כדי לקבוע קו בסיס.

לאחר מכן, אנו מבקשים מהמשתמש ללחוץ על הכפתור ולתעד את מצבו בעת הלחיצה.

לאחר מכן, אנו מבקשים מהמשתמש לשחרר את הכפתור ולתעד את מצבו בעת שחרורו.

בהתבסס על ההתנהגות שנצפתה במהלך שלבים אלה, אנו יכולים לקבוע אם הכפתור פתוח בדרך כלל (N.O.) או סגור בדרך כלל (N.C.).

אם הכפתור פתוח בדרך כלל, המצב ההתחלתי שלו יהיה HIGH, והוא יקרא LOW בלחיצה ויחזור ל-HIGH כשישחררו אותו.

אם הכפתור סגור בדרך כלל, המצב ההתחלתי שלו יהיה LOW, והוא יקרא HIGH בלחיצה ויחזור ל-LOW כשישחררו אותו.

אם ההתנהגות אינה תואמת את הדפוסים הללו, הקוד יציין שאין באפשרותו לקבוע את סוג הכפתור, וניתן לחזור על התהליך.

על ידי ביצוע הליך זה, תוכל לקבוע את סוג הכפתורים (פתוחים בדרך כלל או סגורים בדרך כלל) ללא ידע מוקדם על המאפיינים שלהם.

ניסוי מס' 1. ספירת לחיצות בעזרת כפתורי N.O. ללא הגבלות ללא Debouncing.

מטרת הניסוי: חבר 2 כפתורי NO לכנסות ספרתיות. כתוב תוכנית שמציגה על מסך הטורי COUNTER עם מצב התחלתי 0. כל לחיצה על כפתור ראשון מגדילה COUNTER ב-1, כל לחיצה על כפתור שני מקטינה COUNTER ב-1. שימו לב שללא טיפול בריטוטים מונה משתגע וקופץ לא ב-1, אלא במספרים אחרים אקראיים.

```
#define BUTTON_INC_PIN 2   // GPIO pin connected to the increment button
#define BUTTON_DEC_PIN 3   // GPIO pin connected to the decrement button
#define INITIAL_STATE 0    // Initial state of the counter
```

```
int counter = INITIAL_STATE;

void setup() {
  pinMode(BUTTON_INC_PIN, INPUT_PULLUP);
  pinMode(BUTTON_DEC_PIN, INPUT_PULLUP);
  Serial.begin(9600);
  // Print initial state of the counter
  Serial.print("Counter initialized to: ");
  Serial.println(counter);
}
void loop() {
  // Check if the increment button is pressed
  if (digitalRead(BUTTON_INC_PIN) == LOW) {
    counter++;
    Serial.print("Counter incremented to: ");
    Serial.println(counter);
  }
  // Check if the decrement button is pressed
  if (digitalRead(BUTTON_DEC_PIN) == LOW) {
    counter--;
    Serial.print("Counter decremented to: ");
    Serial.println(counter);
  }
}
```

ניסוי מס' 2. ספירת לחיצות בעזרת כפתורי N.C. ללא הגבלות עם Debouncing.

מטרת הניסוי: חבר 2 כפתורי N.C. לכנסות ספרתיות. כתוב תוכנית שמציגה על מסך הטורי COUNTER עם מצב התחלתי 0. כל לחיצה על כפתור ראשון מגדילה COUNTER ב-1, כל לחיצה על כפתור שני מקטינה COUNTER ב-1. שימו לב שללא טיפול בריטוטים מונה משתגע וקופץ לא ב-1, אלא במספרים אחרים אקראיים.

```
#define BUTTON_INC_PIN 2   // GPIO pin connected to the increment button
#define BUTTON_DEC_PIN 3   // GPIO pin connected to the decrement button
#define INITIAL_STATE 0    // Initial state of the counter
int counter = INITIAL_STATE;
void setup() {
  pinMode(BUTTON_INC_PIN, INPUT_PULLUP);
  pinMode(BUTTON_DEC_PIN, INPUT_PULLUP);
  Serial.begin(9600);
  // Print initial state of the counter
  Serial.print("Counter initialized to: ");
  Serial.println(counter);
}
void loop() {
  // Check if the increment button is pressed
  if (digitalRead(BUTTON_INC_PIN) == HIGH) {
    counter++;
    Serial.print("Counter incremented to: ");
    Serial.println(counter);
    // Wait for the button to be released (debouncing)
    while (digitalRead(BUTTON_INC_PIN) == HIGH) {
      delay(10);
    }
  }
  // Check if the decrement button is pressed
  if (digitalRead(BUTTON_DEC_PIN) == HIGH) {
    counter--;
```

```
    Serial.print("Counter decremented to: ");
    Serial.println(counter);
    // Wait for the button to be released (debouncing)
    while (digitalRead(BUTTON_DEC_PIN) == HIGH) {
      delay(10);
    }
  }
  // Add any additional functionality here
}
```

ניסוי מס' 3. ספירת לחיצות בעזרת כפתורי N.C. עם הגבלת גבולות ועם Debouncing.

מטרת הניסוי: חבר 2 כפתורי N.C. לכנסות ספרתיות. כתוב תוכנית שמציגה על מסך הטורי COUNTER עם מצב התחלתי 0. כל לחיצה על כפתור ראשון מגדילה COUNTER ב-1, כל לחיצה על כפתור שני מקטינה COUNTER ב-1. שימו לב שללא טיפול בריטוטים מונה משתגע וקופץ לא ב-1, אלא במספרים אחרים אקראיים. שים לב ש- COUNTER לא יכול לרדת מתחת ל-0 או לעלות מעל 255.

```
#define BUTTON_INC_PIN 2   // GPIO pin connected to the increment button
#define BUTTON_DEC_PIN 3   // GPIO pin connected to the decrement button
#define INITIAL_STATE 0    // Initial state of the counter
int counter = INITIAL_STATE;
void setup() {
  pinMode(BUTTON_INC_PIN, INPUT_PULLUP);
  pinMode(BUTTON_DEC_PIN, INPUT_PULLUP);
  Serial.begin(9600);
  // Print initial state of the counter
  Serial.print("Counter initialized to: ");
  Serial.println(counter);
}
void loop() {
  // Check if the increment button is pressed
  if (digitalRead(BUTTON_INC_PIN) == HIGH && counter<255) {
    counter++;
    Serial.print("Counter incremented to: ");
    Serial.println(counter);
    // Wait for the button to be released (debouncing)
    while (digitalRead(BUTTON_INC_PIN) == HIGH) {
      delay(10);
    }
  }
  // Check if the decrement button is pressed
  if (digitalRead(BUTTON_DEC_PIN) == HIGH && counter>0) {
    counter--;
    Serial.print("Counter decremented to: ");
    Serial.println(counter);
    // Wait for the button to be released (debouncing)
    while (digitalRead(BUTTON_DEC_PIN) == HIGH) {
      delay(10);
    }
  }
  // Add any additional functionality here
}
```

תרגול בנושא מפסקים (לחצנים):

תרגיל 1: זיהוי מצב מפסק.

מטרה: לקרוא את מצב של 4 מפסקים המחוברים ל-ESP32 ולהדפיס את מצבו לצג הטורי.

הסבר: תרגיל זה מציג את הרעיון של קריאת קלט דיגיטלי ממתג שקף ומדגים כיצד לזהות את מצבו (ON או OFF) באמצעות פיני ה-GPIO של ה-ESP32.

תרגיל 2: ספירת זמן בהחלפת מצב של מפסק.

מטרה: להציג על מסך הטורי כמה זמן עבר בין החלפת מצב 0 למצב 1 והפוך (לחיצה ועזיבה).

הסבר: תרגיל זה מתבסס על תרגיל 1 על ידי חישוב זמנים בהתבסס על מצב מתג ההחלקה, ומספק חוויה מעשית עם בקרת פלט על מסך טורי.

תרגיל 3: שליטה על נוריות LED מרובות עם מפסק.

מטרה: לשלוט במצב של מספר נוריות LED בו-זמנית באמצעות מפסק החלקה בודד.

הסבר: תרגיל זה מציג את הקונספט של שליטה במספר יציאות עם קלט יחיד, ומדגים את הרבגוניות של מפסקים בניהול התקנים מרובים.

תרגיל 4: תצוגת מצב מפסק.

מטרה: להציג את המצב של מפסק החלקה על צג LCD המחובר ל-ESP32.

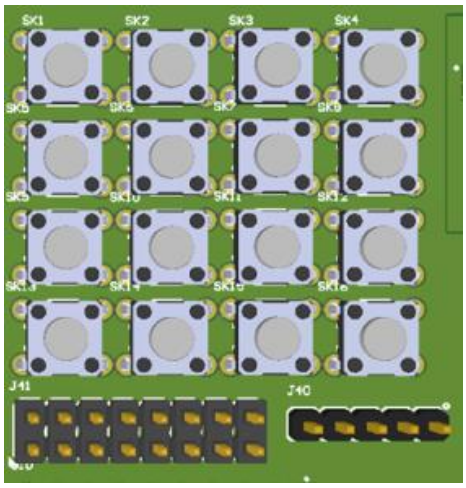
הסבר: תרגיל זה מרחיב את היישום של מפסקים על ידי שילובם עם התקני פלט נוספים, שיפור המשוב והאינטראקציה של המשתמשים.

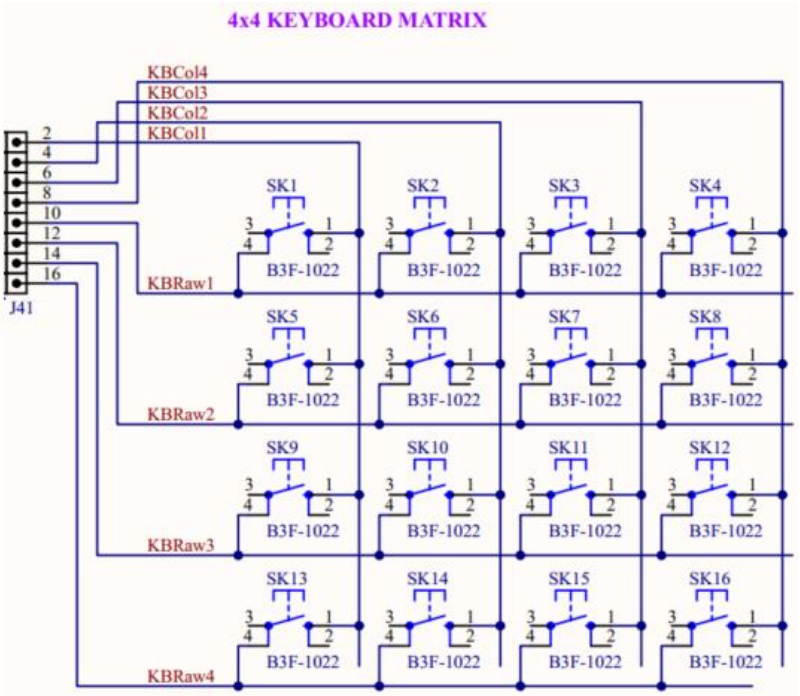
תרגיל 5: ספירת זמן בהחלפת מצב של מפסק בעזרת LCD.

מטרה: להציג על מסך LCD כמה זמן עבר בין החלפת מצב 0 למצב 1 והפוך.

הסבר: תרגיל זה מתבסס על תרגיל 2 על ידי חישוב זמנים בהתבסס על מצב מפסק, ומספק חוויה מעשית עם בקרת פלט על מסך LCD.

הפעלת לוח מקשים מקבילי 4X4.





מטריצת מקלדת 4x4 היא התקן קלט נפוץ בפרויקטים אלקטרוניים, המציע דרך קומפקטית ויעילה להזנת נתונים. בסעיף זה, נדון כיצד לממשק מקלדת 4x4 עם מיקרו-בקר ESP32. נסקור הן את הגדרת החומרה והן את יישום התוכנה, יחד עם דוגמאות קוד המדגימות כיצד לקרוא קלט מהמקלדת.

הגדרת חומרה:

כדי לממשק מטריצת מקלדת 4x4 עם ESP32, בצע את השלבים הבאים:

חיבורי חוטים: חבר את השורות והעמודות של מטריצת המקלדת 4x4 לפיני GPIO של ה-ESP32. כל שורה ועמודה צריכות להיות מחוברות לפין GPIO נפרד.

ספק כוח: ודא שמטריצת המקלדת מופעלת כראוי. רוב מקלדות ה-4x4 פועלות ב-5 V, אז או להפעיל אותה באמצעות 5 V או להשתמש בטכניקות של שינוי רמה כדי להתממשק עם ESP32 3.3 V.

חיבור הארקה: חבר את פין הארקה (GND) של המקלדת לפין הארקה (GND) של ה-ESP32.

יישום תוכנה:

כדי לקרוא קלט ממטריצת המקלדת 4x4 בתוכנה, בצע את השלבים הבאים:

הגדר פינים: הגדר את פיני GPIO המחוברים לשורות ולעמודות של מטריצת המקלדת בקוד שלך.

סריקה מטריצת: יישם קוד כדי לסרוק את השורות והעמודות של מטריצת המקלדת כדי לזהות לחיצות מקשים. זה כרוך בדרך כלל בהגדרת השורות כפלטים ועמודות כקלט, ולאחר מכן הנעה ברצף של כל שורה LOW תוך קריאת מצב העמודות.

פענוח לחיצות מקשים: מפה את לחיצות המקשים שזוהו לתווים או לפעולות התואמות בתוכנית שלך.

דוגמה לקוד - קריאת תו אחד:

להלן דוגמה פשוטה של קוד המדגימה כיצד לקרוא תו אחד ממטריצת המקלדת 4x4:

בקוד זה, אנו מגדירים את הפריסה של מטריצת המקלדת 4x4 (מקשים) ומציינים את פיני ה-GPIO המחוברים לשורות ולעמודות.

הפונקציה getKey() סורקת את המטריצה עבור לחיצות מקשים. זה מניע כל שורה LOW ברצף ובודק את מצב העמודות כדי לזהות לחיצות מקשים.

כאשר מזוהה לחיצה על מקש, התו המתאים מוחזר.

הפונקציה loop() קוראת ברציפות לחיצות מקשים ומדפיסה את התווים שזוהו לצג הטורי.

```
#define ROWS 4
#define COLS 4

char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};

byte rowPins[ROWS] = {2, 3, 4, 5};
byte colPins[COLS] = {6, 7, 8, 9};

void setup() {
  Serial.begin(9600);
}

void loop() {
  char key = getKey();
  if (key != '\0') {
    Serial.println(key);
  }
}

char getKey() {
  for (byte row = 0; row < ROWS; row++) {
    pinMode(rowPins[row], OUTPUT);
    digitalWrite(rowPins[row], LOW);
    for (byte col = 0; col < COLS; col++) {
      pinMode(colPins[col], INPUT_PULLUP);
      if (digitalRead(colPins[col]) == LOW) {
        delay(50); // Debounce delay
        return keys[row][col];
      }
    }
    pinMode(rowPins[row], INPUT);
  }
  return '\0';
}
```

ניסוי מס' 1. בדיקת סיסמה.

קוד זה משתמש בספריית לוח המקשים כדי להתממשק עם מקלדת 4x4. הוא קורא לחיצות מקשים ובודק אם המפתח שהוזן הוא חלק מהסיסמה המוגדרת מראש.

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
```

```
byte rowPins[ROWS] = {2, 3, 4, 5};
byte colPins[COLS] = {6, 7, 8, 9};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

void setup() {
  Serial.begin(9600);
}

void loop() {
  char key = keypad.getKey();
  if (key != NO_KEY) {
    Serial.print("Pressed: ");
    Serial.println(key);
    // Check if the entered key is part of the password
    if (key == '1' || key == '2' || key == '3') {
      // Add logic to handle correct password input
    } else {
      // Add logic to handle incorrect password input
    }
  }
}
```

ניסוי מס' 2. כתיבת ממשק ניווט בתפריט.

קוד זה מיישם מערכת ניווט פשוטה בתפריט באמצעות מקלדת 4x4. משתמשים יכולים לגלול בין אפשרויות התפריט באמצעות המקשים 'A' ו-'C'.

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {2, 3, 4, 5};
byte colPins[COLS] = {6, 7, 8, 9};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

int menuIndex = 0;
String menuItems[] = {"Option 1", "Option 2", "Option 3", "Option 4"};

void setup() {
  Serial.begin(9600);
}

void loop() {
  char key = keypad.getKey();
  if (key != NO_KEY) {
    if (key == 'A' && menuIndex > 0) {
      menuIndex--;
    } else if (key == 'C' && menuIndex < 3) {
      menuIndex++;
    }
    Serial.print("Selected: ");
```

```
    Serial.println(menuItems[menuIndex]);
  }
}
```

ניסוי מס' 3. שעון מעורר עם זמן ניתן להגדרה.

קוד זה מאפשר למשתמשים להגדיר את זמן ההתראה באמצעות מקלדת 4x4. מקש 'A' מגדיל שעות, מקש 'B' מגדיל דקות ומקשי מספרים מכניסים ספרות להגדרת שעות ודקות.

```
#include <Keypad.h>
const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {2, 3, 4, 5};
byte colPins[COLS] = {6, 7, 8, 9};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
int hours = 0;
int minutes = 0;
void setup() {
  Serial.begin(9600);
}
void loop() {
  char key = keypad.getKey();
  if (key != NO_KEY) {
    if (key >= '0' && key <= '9') {
      // Enter digits for setting hours and minutes
    } else if (key == 'A') {
      hours++;
    } else if (key == 'B') {
      minutes++;
    }
    Serial.print("Time: ");
    Serial.print(hours);
    Serial.print(":");
    Serial.println(minutes);
  }
}
```

ניסוי מס' 4. יישום מחשבון לפעולות אריתמטיות בסיסיות.

קוד זה מיישם יישום מחשבון בסיסי באמצעות מקלדת 4x4. משתמשים מכניסים מספרים ואופרטורים אריתמטיים לביצוע חישובים, והתוצאה מוצגת בצג או בצג טורי.

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
```

```
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {2, 3, 4, 5};
byte colPins[COLS] = {6, 7, 8, 9};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

int num1 = 0;
int num2 = 0;
char operator;

void setup() {
  Serial.begin(9600);
}

void loop() {
  char key = keypad.getKey();
  if (key != NO_KEY) {
    if (key >= '0' && key <= '9') {
      if (operator == '\0') {
        num1 = num1 * 10 + (key - '0');
      } else {
        num2 = num2 * 10 + (key - '0');
      }
    } else if (key == '+' || key == '-' || key == '*' || key == '/') {
      operator = key;
    } else if (key == '=') {
      // Perform calculation based on operator and display result
    }
  }
}
```

ניסוי מס' 5. בקרת גישה למערכת אבטחה:

קוד זה מיישם מערכת בקרת גישה אבטחה פשוטה באמצעות מקלדת x44. משתמשים מזינים סיסמה ולוחצים על מקש 'A' כדי לבדוק אם הסיסמה שהוזנה תואמת לסיסמה שהוגדרה מראש.

```
#include <Keypad.h>

const byte ROWS = 4;
const byte COLS = 4;
char keys[ROWS][COLS] = {
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {2, 3, 4, 5};
byte colPins[COLS] = {6, 7, 8, 9};
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);

String password = "1234";
String inputPassword = "";

void setup() {
  Serial.begin(9600);
}
```

```
void loop() {
  char key = keypad.getKey();
  if (key != NO_KEY) {
    if (key == 'A') {
      // Check if entered password is correct
      if (inputPassword == password) {
        // Grant access
      } else {
        // Deny access
      }
      inputPassword = ""; // Clear input password
    } else if (key == 'B') {
      // Clear input password
      inputPassword = "";
    } else {
      inputPassword += key;
    }
  }
}
```

תרגול בנושא לוח מקשים.

תרגיל מס' 1. בניית ממשק הזנת טקסט עבור יישומי הודעות.

כתוב קוד מאפשר למשתמשים להזין הודעות טקסט באמצעות מקלדת 4x4. משתמשים מזינים תווים ולוחצים על מקש '#' כדי לשלוח את ההודעה.

תרגיל מס' 2. תכנות לוח בקרה לפרויקטים של אוטומציה ביתית.

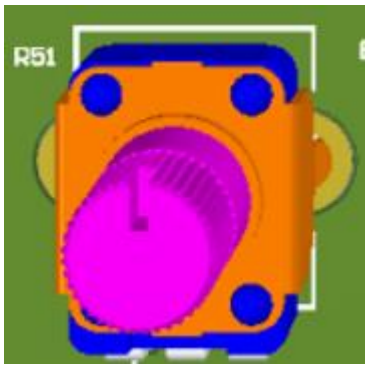
כתוב קוד משמש כלוח בקרה לפרויקטים של אוטומציה ביתית באמצעות מקלדת 4x4. משתמשים יכולים לשלוט במכשירים ומערכות ביתיות שונות על ידי לחיצה על מקשים במקלדת.

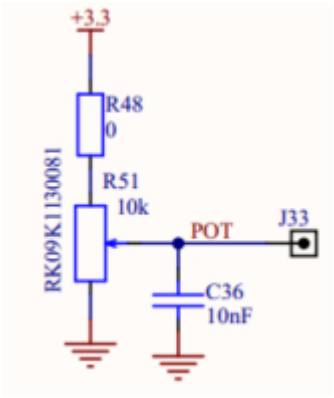
תרגיל מס' 3. בקר משחק למשחקי "רטרו".

כתוב קוד שהופך את המקלדת 4x4 לבקר משחק למשחקי "רטרו". משתמשים יכולים לשלוט בתנועות המשחק על ידי לחיצה על מקשים במקלדת.

פרק 2. קלט ופלט אנלוגי.

קלט אנלוגי מנגד משתנה.





הפונקציה analogRead() במיקרו-בקר ESP32 מאפשרת לך לקרוא ערכי מתח אנלוגיים מחיישנים כגון פוטנציומטרים. בסעיף זה, נחקור כיצד לממשק פוטנציומטר עם ESP32 באמצעות הפונקציה analogRead() . נסקור הן את הגדרת החומרה והן את יישום התוכנה, ולאחר מכן חמש דוגמאות שונות המדגימות את השימוש בה בפרויקטים קטנים.

הגדרת חומרה:

כדי לממשק פוטנציומטר עם ESP32, בצע את השלבים הבאים:

חבר את הפוטנציומטר: חבר את הפין האמצעי (מגב) של הפוטנציומטר לפין אנלוגי (למשל, 32) ב-ESP32. חבר את אחד מהפינים החיצוניים לפין 3.3 V ואת הפין החיצוני השני לפין האדמה (GND).

ספק כוח: ודא שה-ESP32 מופעל כראוי.

חיבור הארקה: חבר את פין האדמה (GND) של הפוטנציומטר לפין האדמה (GND) של ה-ESP32.

יישום תוכנה:

כדי לקרוא ערכי מתח אנלוגי מהפוטנציומטר באמצעות הפונקציה analogRead(), בצע את השלבים הבאים:

אתחול פין קלט אנלוגי: הגדר את הפין המחובר לפוטנציומטר כסין כניסה אנלוגי בקוד שלך.

קרא ערך אנלוגי: השתמש בפונקציה analogRead() כדי לקרוא את ערך המתח האנלוגי מהפוטנציומטר. פונקציה זו מחזירה ערך בין 0 ל-4095, המייצג את רמת המתח ביחס למתח הייחוס (בדרך כלל 3.3 V).

ערך מפה: לחלופין, מפה את הערך האנלוגי לטווח רצוי באמצעות הפונקציה map() במידת הצורך.

דוגמאות לקוד:

להלן חמש דוגמאות המדגימות את השימוש ב-analogRead() עם פוטנציומטר:

ניסוי מס' 1. קליטת מצב של פוטנציומטר והצגתו על מסך טורי.

```
const int potPin = 32;
void setup() {
  Serial.begin(9600);
}
void loop() {
  int potValue = analogRead(potPin);
```

```
  Serial.print("Potentiometer Value: ");
  Serial.println(potValue);
  delay(500);
}
```

ניסוי מס' 2. קליטת מצב של פוטנציומטר והפעלת לד במקרה שערך הנקלט מעבר לחצי.

```
1. const int potPin = 32;
2. const int led=13;
3. void setup() {
4.   Serial.begin(9600);
5. pinMode(led,OUTPUT);
6. }
7. void loop() {
8.   int potValue = analogRead(potPin);
9.   Serial.print("Potentiometer Value: ");
10.   Serial.println(potValue);
11. if(potValue>=2048)
12. digitalWrite(led,HIGH);
13. else
14. digitalWrite(led,LOW);
15.   delay(500);
16. }
17.
```

גרסה שנייה של פיטרון:

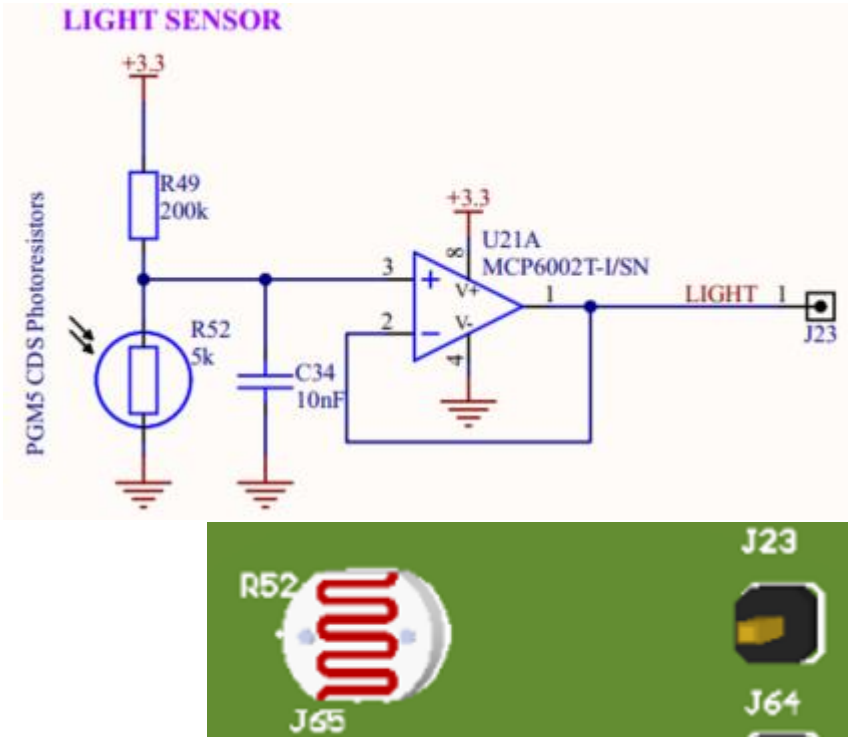
```
const int potPin = 32;
const int ledPin = 13;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  int potValue = analogRead(potPin);
  digitalWrite(ledPin, potValue > 2048 ? HIGH : LOW); // Turn LED ON if potValue is greater than half
}
```

ניסוי מס' 3. קליטת מצב של פוטנציומטר והפעלת לדים RGB לערכים שונים של קליטה.

```
const int potPin = 32;
const int Red=13;
const int Green=14;
const int Blue=15;
void clear();
void setup() {
Serial.begin(9600);
pinMode(Red,OUTPUT);
pinMode(Green,OUTPUT);
pinMode(Blue,OUTPUT);
}
void loop() {
  int potValue = analogRead(potPin);
  Serial.print("Potentiometer Value: ");
```

```
  Serial.println(potValue);
clear();
if(potValue>=3072)
digitalWrite(Blue,HIGH);
else if(potValue>=2048)
digitalWrite(Green,HIGH);
else if(potValue>=1024)
digitalWrite(Red,HIGH);
delay(500);
}
void clear()
{
digitalWrite(Red,LOW);
digitalWrite(Green,LOW);
digitalWrite(Blue,LOW);
}
```

קלט אנלוגי מחיישן אור.



נגדי תלוי אור (LDRs), הידועים גם בתור פוטו-נגדים, הם רכיבים אלקטרוניים פסיביים שהתנגדותם משתנה בהתאם לעוצמת האור הנופל עליהם. הם מוצאים יישום רחב במעגלי חישת אור, זיהוי אור הסביבה ומערכות תאורה אוטומטיות. בסעיף זה, נחקור כיצד לממשק LDR עם מיקרו-בקר ESP32 באמצעות Arduino IDE, יחד עם דוגמאות קוד והסברים.

הגדרת חומרה:

כדי לבדוק LDR עם ESP32, בצע את השלבים הבאים:

חבר LDR: חבר רגל אחת של ה-LDR לפין 3.3 V ב-ESP32.

חבר נגד: חבר נגד (לדוגמה, 10 k אוהם) בין הרגל השנייה של ה-LDR לבין פין האדמה (GND) ב-ESP32.

חבר פין אנלוגי: חבר את החיבור בין ה-LDR והנגד לכל אחד מהפינים האנלוגיים (למשל, 32) ב-ESP32.

ניסוי מס' 1. מדידת רמת האור הסביבה והצגה על מסך טורי.

```
const int ldrPin = 32;
void setup() {
  Serial.begin(9600);
}
void loop() {
  int ldrValue = analogRead(ldrPin);
  Serial.print("LDR Value: ");
  Serial.println(ldrValue);
  delay(1000); // Delay for stability
}
```

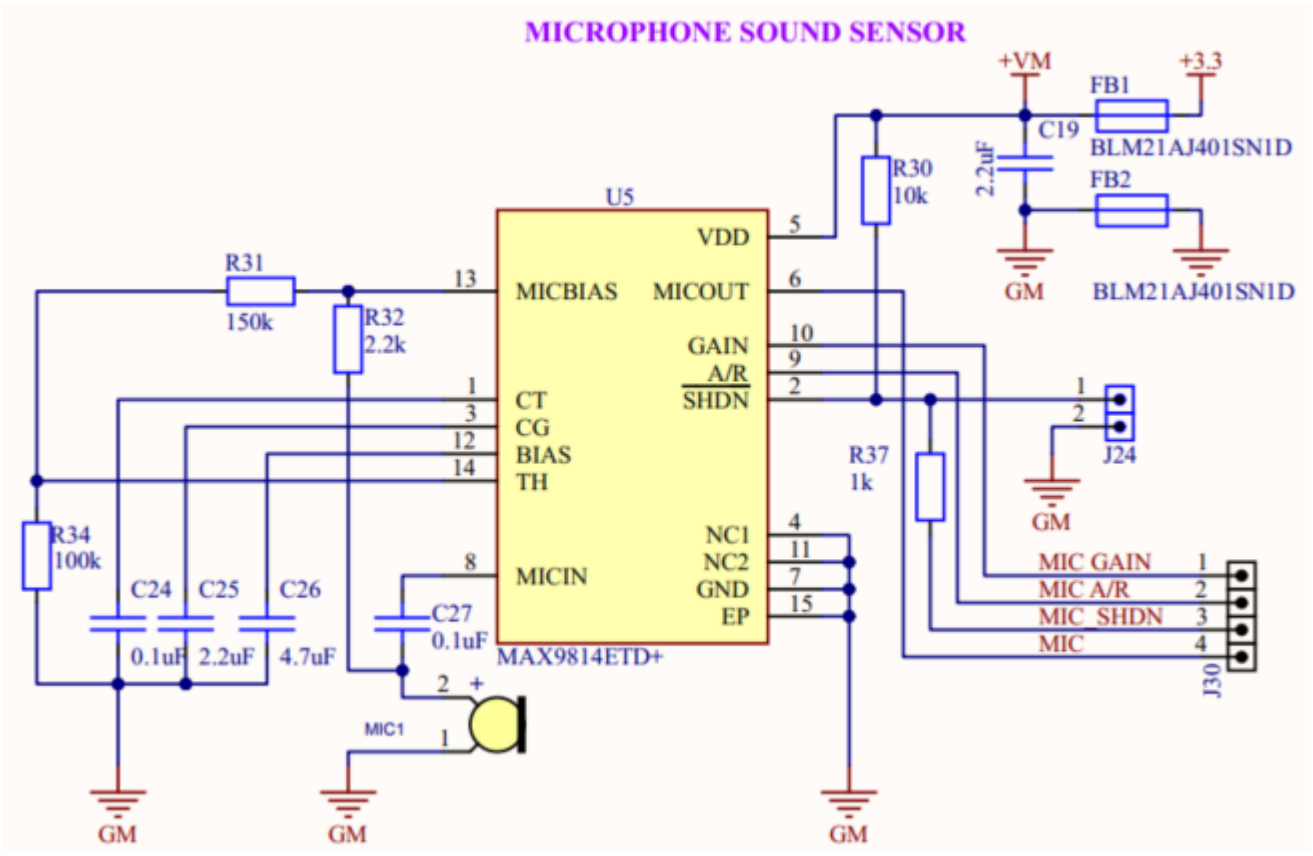
ניסוי מס' 2. הפעלת לד בהתאם לרמת האור הסביבה.

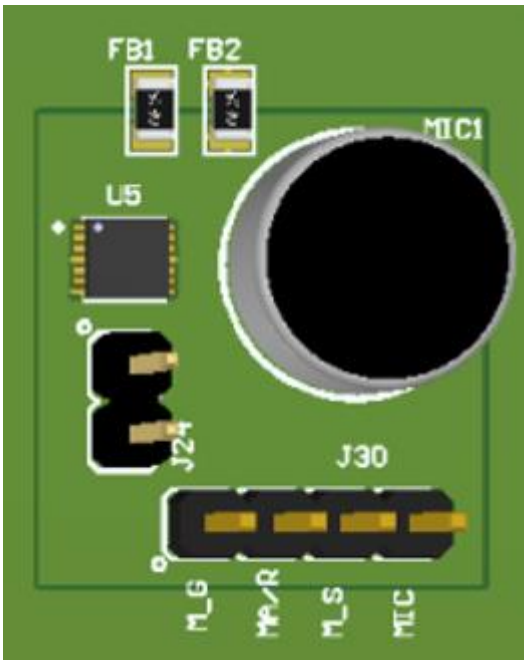
```
const int ldrPin = 32;
const int ledPin = 13;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  int ldrValue = analogRead(ldrPin);
  if(ldrValue< 1024)
  digitalWrite(ledPin,HIGH);
else digitalWrite(ledPin,LOW);
  delay(100); // Delay for stability
}
```

ניסוי מס' 3. הפעלת לד בהתאם לרמת האור הסביבה, כאשר משתמש לוחץ על הכפתור.

```
const int ldrPin = 32;
const int btnPin = 31;
const int ledPin = 13;
void setup() {
  pinMode(ledPin, OUTPUT);
  pinMode(btnPin, INPUT);
}
void loop() {
while(digitalRead(btnPin==LOW);
  int ldrValue = analogRead(ldrPin);
  if(ldrValue< 1024)
  digitalWrite(ledPin,HIGH);
else digitalWrite(ledPin,LOW);
  delay(100); // Delay for stability
}
```

קלט אנלוגי מחיישן קול.





חיישני קול, הידועים גם בשם חיישני זיהוי קול או מיקרופונים, הם מכשירים אלקטרוניים המזהים גלי קול וממירים אותם לאותות חשמליים. הם מוצאים יישומים בתחומים שונים, כולל אוטומציה ביתית, מערכות אבטחה וניטור רמת הרעש. בסעיף זה, נחקור כיצד אפשר לדגום חיישן קול עם מיקרו-בקר ESP32 באמצעות Arduino IDE.

מעגל חיישן קול:

מעגל חיישן קול טיפוסי מורכב ממודול חיישן קול, הכולל מיקרופון, מעגלי הגברה וממשק פלט. ממשק הפלט עשוי לספק אותות אנלוגיים או דיגיטליים, בהתאם לדגם החיישן.

תכונות של חיישני קול:

התאמת רגישות: חיישני קול מסוימים מאפשרים לך להתאים את הרגישות כדי לזהות צלילים ברמות שונות.

פלט אנלוגי ודיגיטלי: חיישני קול עשויים לספק פלט אנלוגי, המייצג את עוצמת הקול, או פלט דיגיטלי, המצביע על נוכחות או היעדר צליל מעל סף מסוים.

טווח תדרים רחב: חיישני קול מסוגלים לזהות גלי קול בטווח תדרים רחב, מה שהופך אותם למתאימים ליישומים שונים.

שילוב קל: קל לשלב חיישני קול בפרויקטים מבוססי מיקרו-בקר, כולל אלו המשתמשים בפלטפורמת ESP32.

ממשק עם ESP32:

כדי לדגום חיישן קול עם מיקרו-בקר ESP32, בצע את השלבים הבאים:

חבר את חיישן הקול: חבר את פין המוצא של מודול חיישן הקול לכל אחד מפיני ה-GPIO ב-ESP32.

ספק כוח: ספק כוח למודול חיישן הקול באמצעות המתח המתאים (בדרך כלל 3.3 V או 5 V) מה-ESP32.

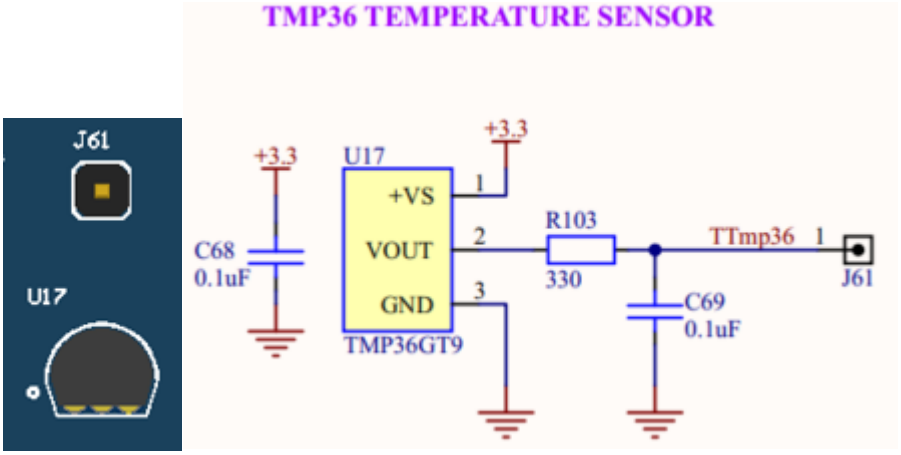
חיבור הארקה: חבר את פין האדמה (GND) של מודול חיישן הקול לפין האדמה (GND) ב-ESP32.

יישום תוכנה:

קרא ערך אנלוגי או דיגיטלי: השתמש בפונקציות digitalRead() או analogRead() כדי לקרוא את ערך הפלט מחיישן הקול, תלוי אם הוא מספק פלט דיגיטלי או אנלוגי.

עבד את הערך: בהתאם לערך הפלט הנקרא מהחיישן, בצע פעולות כגון הפעלת אזעקות, הקלטת רמות קול או שליטה במכשירים אחרים.

קלט אנלוגי מחיישן טמפרטורה TMP36.



ה-TMP36 הוא חיישן טמפרטורה אנלוגי במתח נמוך ומדויק המספק פלט ליניארי פרופורציונלי לטמפרטורה. הוא נפוץ בשימוש ביישומי ניטור ובקרה של טמפרטורה בשל קלות השימוש, הדיוק וטווח הטמפרטורות הרחב. בסעיף זה, נחקור את חיישן TMP36, המעגל שלו, תכונותיו, ההבדלים לחיישן LM35, וכיצד לדגום אותו עם מיקרו-בקר ESP32 באמצעות Arduino IDE.

מעגל חיישן TMP36:

לחיישן TMP36 יש שלושה פינים: VCC, OUT ו-GND.

חבר את פין VCC למתח האספקה החיובי (3.3 V).

חבר את פין ה-OUT לכל פין כניסה אנלוגי של ה-ESP32.

חבר את פין GND לאדמה (GND) של ה-ESP32.

תכונות של חיישן TMP36:

טווח פעולה רחב: חיישן TMP36 יכול למדוד טמפרטורות מ-C 40° - עד +C 125°.

פעולת מתח נמוך: הוא פועל בטווח מתח נמוך של 2.7 V עד 5.5 V.

פלט ליניארי: מתח המוצא הוא פרופורציונלי ליניארי לטמפרטורה, מה שמקל על התממשק עם מיקרו-בקרים.

מכויל: החיישן מכויל במפעל, ומבטל את הצורך בכיול נוסף.

צריכת חשמל נמוכה: הוא צורך חשמל נמוך מאוד, מה שהופך אותו למתאים ליישומים המופעלים על ידי סוללה.

ההבדל בין TMP36 ל-LM35:

גם TMP36 וגם LM35 הם חיישני טמפרטורה אנלוגיים עם מאפיינים דומים, אבל יש כמה הבדלים:

טווח טמפרטורה: ל-TMP36 טווח מדידת טמפרטורה רחב יותר (40° C - עד +C 125°) בהשוואה ל-LM35 (0° C עד + 100° C).

מתח מוצא: ה-TMP36 מייצר מתח מוצא שהוא פרופורציונלי לינארי לטמפרטורה עם מקדם קנה מידה של 10mV/°C, בעוד שה-LM35 מייצר מתח מוצא עם מקדם קנה מידה של 10 mV/°C ללא צורך לתיקון של 0.5.

קוד לדגימת TMP36.

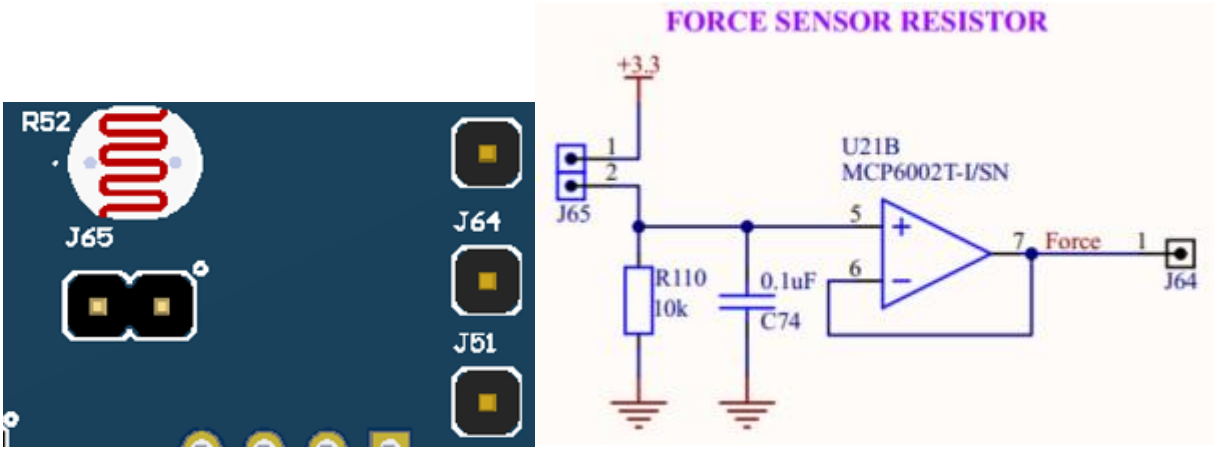
```
const int tempPin = 32; // Analog pin connected to TMP36 sensor
void setup() {
  Serial.begin(9600);
}
void loop() {
  int sensorValue = analogRead(tempPin);
  float voltage = sensorValue * (3.3 / 4095.0); // Convert sensor value to voltage (3.3V reference)
  float temperature = (voltage - 0.5) * 100; // Convert voltage to temperature in Celsius
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");
  delay(1000); // Delay for stability
}
```

קוד להשוואה של LM35.

```
const int tempPin = 32; // Analog pin connected to LM35 sensor
void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(tempPin);
  float voltage = sensorValue * (3.3 / 4095.0); // Convert sensor value to voltage (3.3V reference)
```

```
  float temperature = voltage * 100; // LM35 outputs 10mV per degree Celsius
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.println(" °C");
  delay(1000); // Delay for stability
}
```

קלט אנלוגי מחיישן משקל.



חיישני Force-Sensing Resistor (FSR) הם רכיבים חזקים ומגוונים בשימוש נרחב ביישומים שונים, מרובוטיקה ועד מכשירים רפואיים. הם מציעים דרך פשוטה אך יעילה למדידת כוח או לחץ המופעלים על משטח. בסעיף זה, אנו מתעמקים בפעולה של חיישני FSR, חוקרים את המעגלים, התכונות, מאפייני הפלט שלהם, ומספקים מדריך מפורט על העבודה עם המיקרו-בקר ESP32 באמצעות Arduino IDE.

הבנת חיישני FSR:

חיישני FSR עשויים מחומר פולימרי מוליך המשנה התנגדות בעת הפעלת לחץ. ההתנגדות יורדת ככל שהכוח על החיישן גדל. שינוי זה בהתנגדות משמש למדידת הכוח או הלחץ המופעלים. חיישני FSR מגיעים בצורות, גדלים וטווחי כוח שונים כדי להתאים ליישומים שונים.

מעגלים:

המעגל לחיבור חיישן FSR עם ESP32 הוא פשוט יחסית. הוא מורכב בדרך כלל מחיישן FSR המחובר בתצורת מחלק מתח עם נגד נשלף. המתח על פני חיישן FSR נמדד לאחר מכן באמצעות אחד מפיני הכניסה האנלוגיים של ה-ESP32.

תכונות של חיישני FSR:

מגוון רחב של אפשרויות רגישות לכוח

שילוב קל במערכות שונות

צריכת חשמל נמוכה

זמן תגובה מהיר

עמיד ואמין

מאפייני פלט:

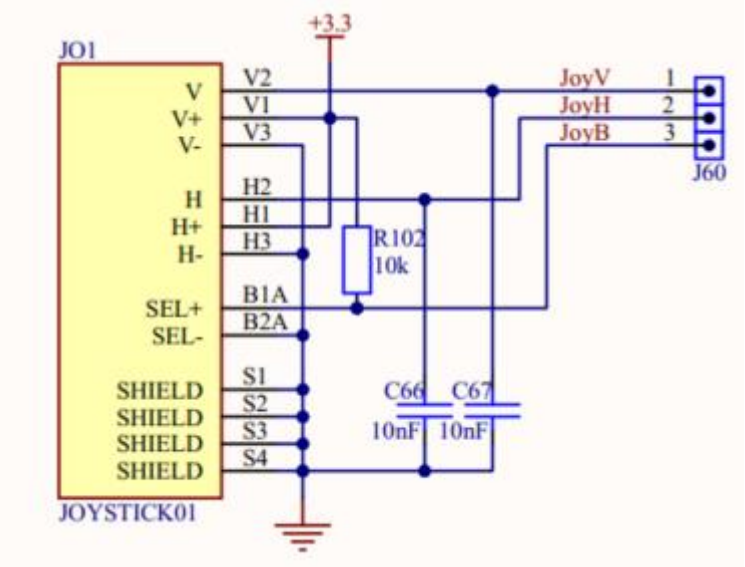
הפלט של חיישן FSR הוא אנלוגי ומשתנה בהתאם לכוח המופעל. ככל שהכוח עולה, ההתנגדות של ה-FSR יורדת, וכתוצאה מכך פלט מתח אנלוגי גבוה יותר. ניתן לכייל פלט זה כך שיתאים לכוח המופעל על החיישן.

יישום קוד עבור ESP32:

להלן קוד לדוגמה המדגים כיצד לממשק חיישן FSR עם ESP32 באמצעות Arduino IDE:

```
const int fsrPin = 32; // Analog pin connected to FSR
int fsrValue; // Variable to store FSR reading
void setup() {
  Serial.begin(9600); // Initialize serial communication
}
void loop() {
  fsrValue = analogRead(fsrPin); // Read analog value from FSR
  Serial.print("FSR Value: ");
  Serial.println(fsrValue); // Print FSR value
  delay(1000); // Delay for stability
}
```

קלט אנלוגי וספרתי מג'ויסטיק.



ג'ויסטיקים אנלוגיים הם התקני קלט הנמצאים בכל מקום המשמשים בשפע של יישומים אלקטרוניים, החל מקונסולות משחקים ועד רכבים הנשלטים מרחוק. רכיבים מגוונים אלה מספקים שליטה אינטואיטיבית על שני צירים, ומאפשרים מניפולציה מדויקת בשני הכיוונים. בסעיף זה, נתעמק בפעולת הג'ויסטיקים האנלוגיים, נבחן את המעגלים, התכונות, מאפייני הפלט שלהם, ונספק מדריך מקיף על עבודה עם המיקרו-בקר ESP32 באמצעות Arduino IDE.

הבנת ג'ויסטיקים אנלוגיים:

ג'ויסטיקים אנלוגיים מורכבים משני פוטנציומטרים המחוברים מכנית לציר מרכזי. פוטנציומטרים אלו מייצרים אותות מתח אנלוגיים פרופורציונליים למיקום הג'ויסטיק לאורך צירי ה-X וה-Y שלו. על ידי מדידת מתחים אלה, המיקרו-בקר

יכול לקבוע את המיקום המדויק של הג'ויסטיק בשני הכיוונים. בדרך כלל לג'ויסטיק ישנו כפתור ספרתי יחד עם 2 פוטנציומטרים של המיקום.

מעגלים:

המעגל לחיבור ג'ויסטיק אנלוגי עם ESP32 הוא פשוט. כל פוטנציומטר של הג'ויסטיק מחובר לאחד מפיני הכניסה האנלוגיים של ה-ESP32. בנוסף, לג'ויסטיק יש בדרך כלל כפתור, שניתן לחבר לפין קלט דיגיטלי לזיהוי לחיצות על כפתורים.

תכונות של ג'ויסטיקים אנלוגיים:

שליטה בציר כפול למניפולציה מדויקת בשני כיוונים

עיצוב קומפקטי וארגונומי לשימוש נוח

פלט אנלוגי למיקום חלק ורציף

כפתור משולב לפונקציונליות נוספת

תואם למגוון רחב של מיקרו-בקרים ופלטפורמות פיתוח

מאפייני פלט:

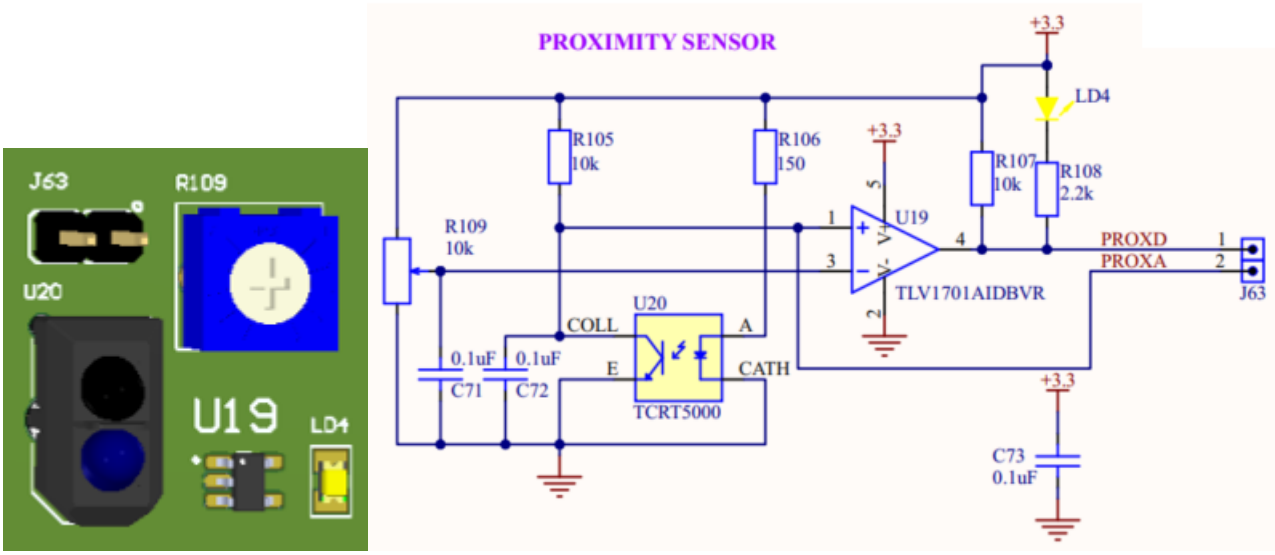
הפלט של ג'ויסטיק אנלוגי הוא אותות מתח אנלוגיים המתאימים למיקום הג'ויסטיק לאורך צירי ה-X וה-Y שלו. המתח משתנה באופן ליניארי עם המיקום, מה שמאפשר שליטה חלקה ורציפה. בנוסף, פלט הכפתור מספק אות דיגיטלי המציין את מצב הכפתור (לחוץ או משוחרר).

יישום קוד עבור ESP32:

להלן קוד לדוגמה המדגים כיצד לממשק ג'ויסטיק אנלוגי עם ESP32 באמצעות Arduino IDE:

```
const int xPin = 34; // Analog pin connected to X-axis of joystick
const int yPin = 35; // Analog pin connected to Y-axis of joystick
const int buttonPin = 2; // Digital pin connected to button of joystick
int xValue, yValue; // Variables to store joystick readings
int buttonState; // Variable to store button state
void setup() {
  Serial.begin(9600); // Initialize serial communication
  pinMode(buttonPin, INPUT_PULLUP); // Set button pin as input with pull-up resistor
}
void loop() {
  xValue = analogRead(xPin); // Read analog value from X-axis
  yValue = analogRead(yPin); // Read analog value from Y-axis
  buttonState = digitalRead(buttonPin); // Read button state
  Serial.print("X-Axis: ");
  Serial.print(xValue);
  Serial.print(" | Y-Axis: ");
  Serial.print(yValue);
  Serial.print(" | Button State: ");
  Serial.println(buttonState);
  delay(100); // Delay for stability
}
```

קלט אנלוגי וספרתי מצמד (זוג) אופטי.



חיישני מכשולים ממלאים תפקיד מכריע ביישומים שונים, מרובוטיקה ועד אוטומציה תעשייתית, על ידי זיהוי נוכחות של עצמים בסביבתם. סוג אחד נפוץ של חיישן מכשולים משתמש במצמדים אופטיים כדי לזהות מכשולים באופן אופטי. בסעיף זה, נתעמק בפעולה של חיישני מכשולים עם צמדים אופטיים, נבחן את המעגלים, התכונות, מאפייני הפלט האנלוגיים והדיגיטליים שלהם.

הבנת חיישני מכשולים עם צמד אופטי:

חיישני מכשולים עם צמד אופטי מורכבים בדרך כלל מנורית אינפרא אדום (IR) ופוטוטרנזיסטור סגורים בחבילה אחת. כאשר חפץ חוסם את הנתיב בין LED ה-IR לפוטו-טרנזיסטור, עוצמת האור האינפרא-אדום המתקבל משתנה, מה שגורם לשינוי מקביל במוליכות הפוטו-טרנזיסטור. שינוי זה במוליכות מנוצל כדי לזהות נוכחות או היעדר מכשולים.

מעגלים:

ניתן להגדיר את המעגל לחיבור חיישן מכשולים עם מצמד אופטו עם ESP32 ליציאה אנלוגית ודיגיטלית כאחד. עבור פלט אנלוגי, מתח הקולטור-מפיץ של הפוטו-טרנזיסטור נמדד באמצעות אחד מפיני הכניסה האנלוגיים של ה-ESP32. עבור פלט דיגיטלי, האספן של הפוטו-טרנזיסטור מחובר לפין כניסה דיגיטלי של ה-ESP32, ונגד משוך משמש כדי להבטיח רמות מתח מתאימות.

תכונות של חיישני מכשולים עם צמד אופטי:

עיצוב קומפקטי ורב-תכליתי

זיהוי ללא מגע

רגישות גבוהה למכשולים

זמן תגובה מהיר

יכול לפעול בתנאים סביבתיים שונים

פלט אנלוגי:

הפלט האנלוגי של חיישן מכשולים עם צמד אופטי מתאים לעוצמת האור האינפרא אדום המתקבל, המושפע מהמרחק והמאפיינים של האובייקט החוסם. ניתן למדוד ולכייל מתח אנלוגי זה כדי לקבוע את הקרבה או ההשתקפות של המכשול.

יציאה דיגיטלית:

הפלט הדיגיטלי של חיישן מכשולים עם צמד אופטי מספק אינדיקציה בינארית לנוכחות מכשול. כאשר מזוהה מכשול, מוליכות הפוטו-טרנזיסטור עולה, מה שמושך את מתח הקולט נמוך. לעומת זאת, כאשר אין מכשול, מתח הקולט נשאר גבוה. האות הדיגיטלי הזה יכול להיות ממשק ישירות עם פיני הכניסה הדיגיטליים של מיקרו-בקרים כמו ESP32.

יישום קוד עבור ESP32:

להלן קוד לדוגמה המדגים כיצד לממשק חיישן מכשולים עם צמד אופטי עם ESP32 באמצעות Arduino IDE:

```
const int analogPin = 34; // Analog pin connected to obstacle sensor for analog output
const int digitalPin = 2; // Digital pin connected to obstacle sensor for digital output
void setup() {
  Serial.begin(9600); // Initialize serial communication
  pinMode(digitalPin, INPUT_PULLUP); // Set digital pin as input with pull-up resistor
}
void loop() {
  int analogValue = analogRead(analogPin); // Read analog value from obstacle sensor
  int digitalValue = digitalRead(digitalPin); // Read digital value from obstacle sensor
  Serial.print("Analog Output: ");
  Serial.print(analogValue);
  Serial.print("Digital Output: ");
  Serial.println(digitalValue);
  delay(100); // Delay for stability
}
```

פלט אנלוגי (PWM ו-DAC).

אפנון רוחב דופק (PWM) וממיר דיגיטלי לאנלוגי (DAC) הן שתי שיטות נפוצות המשמשות להפקת אותות אנלוגיים ממיקרו-בקרים דיגיטליים כמו ESP32. בסעיף זה, נחקור את המושגים של PWM ו-DAC, נבחן את המעגלים, התכונות, מאפייני הפלט שלהם, ונדגיש את ההבדלים ביניהם. בנוסף, נספק דוגמאות קוד להפקת פלט PWM ו-DAC במיקרו-בקר ESP32 באמצעות Arduino IDE.

פלט PWM ב-ESP32:

אפנון רוחב דופק (PWM) היא טכניקה המשמשת ליצירת אותות דמויי אנלוגי על ידי מעבר מהיר של פלט דיגיטלי בין מצב גבוה לנמוך. פלט המתח הממוצע נקבע על ידי מחזור העבודה, המייצג את היחס בין הזמן שבו האות נמצא במצב גבוה בהשוואה לתקופה הכוללת. אותות PWM משמשים בדרך כלל לשליטה בבהירות של נוריות, מהירות מנוע ויצירת צלילי שמע.

מעגל עבור פלט PWM:

המעגל עבור פלט PWM ב-ESP32 כולל בדרך כלל חיבור פין פלט PWM לרכיב הרצוי, כגון LED או דרייבר מנוע. המיקרו-בקר ESP32 מספק פיני PWM מרובים, המאפשרים שליטה בו זמנית במספר התקנים.

תכונות של פלט PWM:

פשטות ורבגוניות

מגוון רחב של התאמת מחזור עבודה

זמן תגובה מהיר

שימוש יעיל במשאבי מיקרו-בקר

מאפייני פלט של PWM:

הפלט של PWM הוא גל מרובע עם מחזור עבודה משתנה (יחס גבוה לנמוך). על ידי התאמת מחזור העבודה, ניתן לשלוט בפלט המתח האפקטיבי. עם זאת, פלט PWM אינו אות אנלוגי אמיתי ועלול להכניס רעש או אדווה, במיוחד במחזורי עבודה נמוכים יותר.

פלט DAC ב-ESP32:

ממיר דיגיטלי לאנלוגי (DAC) הוא מודול חומרה ייעודי הממיר אותות דיגיטליים למתחים אנלוגיים מדויקים. שלא כמו PWM, פלט DAC מספק אותות אנלוגיים אמיתיים עם רמות מתח רציפות. זה הופך אותו למתאים ליישומים הדורשים פלט אנלוגי מדויק, כגון השמעת אודיו וממשק חיישנים.

מעגל עבור פלט DAC:

המיקרו-בקר ESP32 מצויד בערוצי DAC מובנים, אותם ניתן לחבר ישירות לרכיבים חיצוניים הדורשים אותות אנלוגיים. פלט DAC כרוך בדרך כלל בחיבור פין DAC לכניסה של רכיב היעד, כגון מגבר שמע או ממשק חיישן.

תכונות של פלט DAC:

פלט אנלוגי מדויק ורציף

רעש ועיוות נמוכים

מתאים ליישומי אודיו וחיישנים באיכות גבוהה

מספק רזולוציה ודיוק טובים יותר בהשוואה ל-PWM

מאפייני פלט של DAC:

הפלט של DAC הוא אות מתח אנלוגי אמיתי, שיכול להשתנות ברציפות בטווח מתח מוגדר. פלט DAC מספק אותות אנלוגיים חלקים ומדויקים יותר בהשוואה ל-PWM, מה שהופך אותו לאידיאלי עבור יישומים שבהם הדיוק הוא קריטי.

יישום קוד עבור PWM ו-DAC ב-ESP32:

להלן קטעי קוד לדוגמה המדגימים כיצד ליצור פלט PWM ו-DAC ב-ESP32 באמצעות Arduino IDE:

```
const int pwmPin = 2; // PWM output pin
void setup() {
  pinMode(pwmPin, OUTPUT); // Set PWM pin as output
}
void loop() {
  // Generate PWM signal with varying duty cycle
  for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
    analogWrite(pwmPin, dutyCycle); // Set PWM duty cycle
    delay(10); // Delay for smooth transition
  }
```

```
}
```

```
const int dacPin = 25; // DAC output pin
void setup() {
  dacWrite(dacPin, 0); // Initialize DAC output
}
void loop() {
  // Generate analog voltage output with varying levels
  for (int voltage = 0; voltage <= 255; voltage++) {
    dacWrite(dacPin, voltage); // Set DAC output voltage
    delay(10); // Delay for smooth transition
  }
}
```

ניסוי מס' 1. בניית גל משולש בעזרת PWM ובעזרת DAC.

חבר סקופ ליציאה שהגדרת בקוד ותשווה תוצאות של PWM ותוצאות של DAC.

```
const int pwmPin = 2; // PWM output pin
const int frequency = 1000; // PWM frequency (Hz)
void setup() {
  ledcSetup(0, frequency, 8); // Configure PWM channel 0
  ledcAttachPin(pwmPin, 0); // Attach PWM channel 0 to the specified pin
}
void loop() {
  for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
    ledcWrite(0, dutyCycle); // Set PWM duty cycle
    delay(10); // Adjust delay for desired frequency
  }
  for (int dutyCycle = 255; dutyCycle >= 0; dutyCycle--) {
    ledcWrite(0, dutyCycle); // Set PWM duty cycle
    delay(10); // Adjust delay for desired frequency
  }
}
```

```
const int dacPin = 25; // DAC output pin
const int maxVoltage = 255; // Maximum DAC output voltage
void setup() {
  dacWrite(dacPin, 0); // Initialize DAC output
}
void loop() {
  for (int voltage = 0; voltage <= maxVoltage; voltage++) {
    dacWrite(dacPin, voltage); // Set DAC output voltage
    delay(10); // Adjust delay for desired frequency
  }
  for (int voltage = maxVoltage; voltage >= 0; voltage--) {
    dacWrite(dacPin, voltage); // Set DAC output voltage
    delay(10); // Adjust delay for desired frequency
  }
}
```

ניסוי מס' 2. בניית גל שן בעזרת PWM ובעזרת DAC.

חבר סקופ ליציאה שהגדרת בקוד ותשווה תוצאות של PWM ותוצאות של DAC.

```
const int pwmPin = 2; // PWM output pin
const int frequency = 1000; // PWM frequency (Hz)
void setup() {
  ledcSetup(0, frequency, 8); // Configure PWM channel 0
  ledcAttachPin(pwmPin, 0); // Attach PWM channel 0 to the specified pin
}
void loop() {
  for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
    ledcWrite(0, dutyCycle); // Set PWM duty cycle
    delay(10); // Adjust delay for desired frequency
  }
  for (int dutyCycle = 255; dutyCycle >= 0; dutyCycle--) {
    ledcWrite(0, dutyCycle); // Set PWM duty cycle
    delay(10); // Adjust delay for desired frequency
  }
}
```

```
const int dacPin = 25; // DAC output pin
const int maxVoltage = 255; // Maximum DAC output voltage
void setup() {
  dacWrite(dacPin, 0); // Initialize DAC output
}
void loop() {
  for (int voltage = 0; voltage <= maxVoltage; voltage++) {
    dacWrite(dacPin, voltage); // Set DAC output voltage
    delay(10); // Adjust delay for desired frequency
  }
}
```

ניסוי מס' 3. בניית גל סינוס בעזרת PWM ובעזרת DAC.

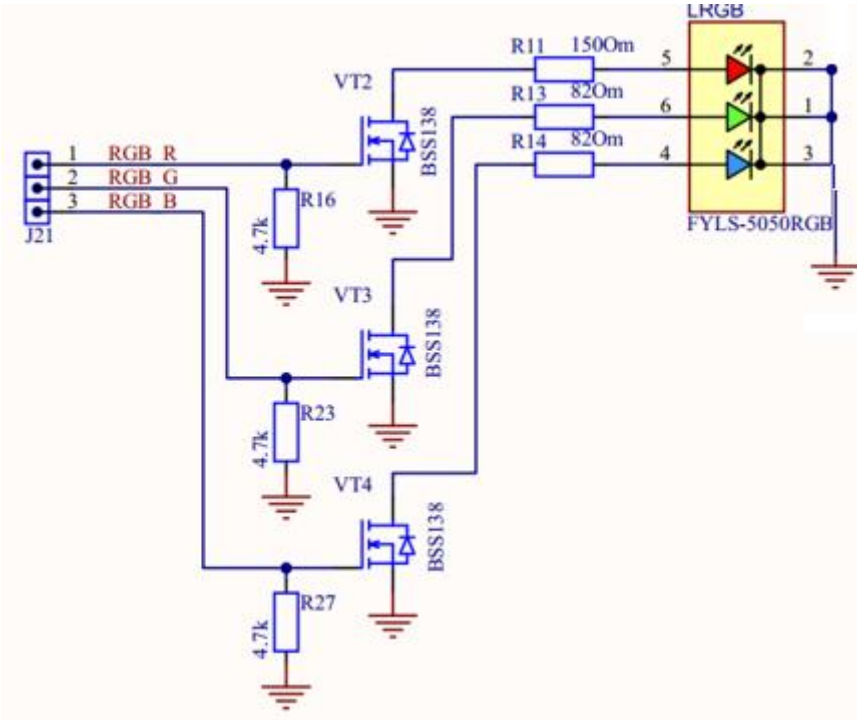
חבר סקופ ליציאה שהגדרת בקוד ותשווה תוצאות של PWM ותוצאות של DAC.

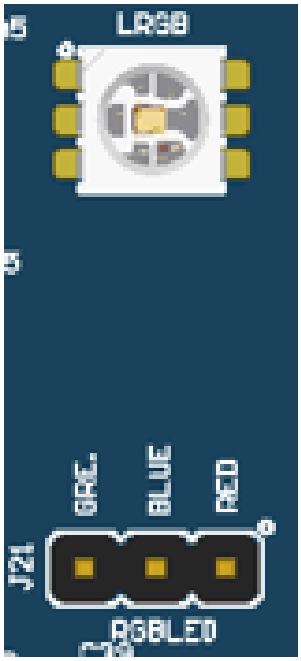
```
const int pwmPin = 2; // PWM output pin
const int frequency = 1000; // PWM frequency (Hz)
void setup() {
  ledcSetup(0, frequency, 8); // Configure PWM channel 0
  ledcAttachPin(pwmPin, 0); // Attach PWM channel 0 to the specified pin
}
void loop() {
  for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
    ledcWrite(0, dutyCycle); // Set PWM duty cycle
    delay(10); // Adjust delay for desired frequency
  }
  for (int dutyCycle = 255; dutyCycle >= 0; dutyCycle--) {
    ledcWrite(0, dutyCycle); // Set PWM duty cycle
    delay(10); // Adjust delay for desired frequency
  }
}
```

```
const int dacPin = 25; // DAC output pin
const float frequency = 1000.0; // Frequency of the sine wave (Hz)
const int amplitude = 255; // Amplitude of the sine wave
void setup() {
  // No setup is needed for DAC initialization
}
```

```
void loop() {
  float period = 1.0 / frequency;
  float increment = 2 * PI / (period * 1000); // Frequency normalized to ms
  for (float t = 0; t < period; t += increment) {
    int outputValue = amplitude * sin(t);
    dacWrite(dacPin, outputValue); // Set DAC output voltage
    delayMicroseconds(10); // Adjust delay for desired frequency
  }
}
```

הפעלת לד RGB.





נוריות RGB הן רכיבים מגוונים המסוגלים להפיק מגוון רחב של צבעים על ידי שילוב של עוצמות שונות של אור אדום, ירוק וכחול. בסעיף זה, נתעמק במעגלים, בתכונות ובמאפייני הפלט של נוריות RGB כאשר הן נשלטות באמצעות המיקרו-בקר ESP32. בנוסף, נדון בהבדלים בין שימוש ב-PWM ו-DAC כדי לשלוט על נוריות RGB ונספק דוגמאות קוד לשתי השיטות באמצעות ESP32 IDE.

הבנת נוריות RGB:

נוריות RGB מורכבות משלוש נוריות לד בודדות (אדום, ירוק וכחול) המשולבות בחבילה אחת. על ידי שינוי העוצמה של כל רכיב צבע, נוריות RGB יכולות לייצר מגוון רחב של צבעים. נוריות LED אלו משמשות בדרך כלל ביישומי תאורה דקורטיביים, תצוגות ואפקטים חזותיים.

מעגל לבקרת LED RGB:

המעגל לשליטה ב-RGB LED כולל בדרך כלל חיבור כל פין צבע (R,G,B) לפין GPIO בעל יכולת PWM ב-ESP32. זה מאפשר שליטה עצמאית בעוצמת כל צבע באמצעות אותות PWM.

תכונות של בקרת LED RGB:

אפשרויות צבע מגוונות

שליטה עדינה על עוצמת הצבע

קומפקטי וקל לשילוב

מתאים לאפקט תאורה דינמיים

ההבדל בין PWM ל-DAC עבור בקרת LED RGB:

PWM (Pulse Width Modulation): PWM היא טכניקה להדמיית פלט אנלוגי על ידי הפעלה וכיבוי מהיר של פלט דיגיטלי. כל סיכה צבעונית של LED RGB מחוברת לפין GPIO בעל יכולת PWM ב-ESP32. על ידי שינוי מחזור העבודה של אות PWM, ניתן להתאים את העוצמה של כל רכיב צבע. עם זאת, פלט PWM אינו אנלוגי אמיתי, מה שגורם לאי דיוקים פוטנציאליים בצבע ולהבהוב בעוצמות נמוכות.

DAC (ממיר דיגיטלי לאנלוגי): DAC הוא מודול חומרה ייעודי הממיר אותות דיגיטליים למתחים אנלוגיים מדויקים. בעוד שלמיקרו-בקרים ESP32 יש ערוצי DAC מובנים, הם מוגבלים במספרם וייתכן שלא יתאימו לשליטה בכל רכיב צבע של LED RGB באופן עצמאי. עם זאת, השימוש ב-DAC מספק פלט אנלוגי אמיתי, וכתוצאה מכך מעברי צבע חלקים יותר וייצוג צבע מדויק.

```
const int redPin = 2; // PWM pin for controlling red color
const int greenPin = 4; // PWM pin for controlling green color
const int bluePin = 5; // PWM pin for controlling blue color

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  // Set RGB LED to different colors using PWM
  analogWrite(redPin, 255); // Full intensity red
  delay(1000);
  analogWrite(greenPin, 255); // Full intensity green
  delay(1000);
  analogWrite(bluePin, 255); // Full intensity blue
  delay(1000);
  analogWrite(redPin, 0); // Turn off red
  analogWrite(greenPin, 0); // Turn off green
  analogWrite(bluePin, 0); // Turn off blue
  delay(1000);
}
```

```
const int redPin = 25; // DAC pin for controlling red color
const int greenPin = 26; // DAC pin for controlling green color
const int bluePin = 27; // DAC pin for controlling blue color

void setup() {
  // No setup is needed for DAC initialization
}

void loop() {
  // Set RGB LED to different colors using DAC
  dacWrite(redPin, 255); // Full intensity red
  delay(1000);
```

```
  dacWrite(greenPin, 255); // Full intensity green
  delay(1000);
  dacWrite(bluePin, 255); // Full intensity blue
  delay(1000);
  dacWrite(redPin, 0); // Turn off red
  dacWrite(greenPin, 0); // Turn off green
  dacWrite(bluePin, 0); // Turn off blue
  delay(1000);
}
```

ניסוי מס' 1. הרצת כל צבע בנפרד.

```
const int redPin = 2; // PWM pin for controlling red color
const int greenPin = 4; // PWM pin for controlling green color
const int bluePin = 5; // PWM pin for controlling blue color

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  // Run all colors of red
  for (int intensity = 0; intensity <= 255; intensity++) {
    analogWrite(redPin, intensity);
    delay(10);
  }
  delay(1000); // Wait for 1 second

  // Run all colors of green
  for (int intensity = 0; intensity <= 255; intensity++) {
    analogWrite(greenPin, intensity);
    delay(10);
  }
  delay(1000); // Wait for 1 second

  // Run all colors of blue
  for (int intensity = 0; intensity <= 255; intensity++) {
    analogWrite(bluePin, intensity);
    delay(10);
  }
  delay(1000); // Wait for 1 second

  // Turn off all colors
  analogWrite(redPin, 0);
  analogWrite(greenPin, 0);
  analogWrite(bluePin, 0);
  delay(1000); // Wait for 1 second
}
```

```
const int redPin = 25; // DAC pin for controlling red color
```

```
const int greenPin = 26; // DAC pin for controlling green color
const int bluePin = 27; // DAC pin for controlling blue color

void setup() {
  // No setup is needed for DAC initialization
}

void loop() {
  // Run all colors of red
  for (int voltage = 0; voltage <= 255; voltage++) {
    dacWrite(redPin, voltage);
    delay(10);
  }
  delay(1000); // Wait for 1 second

  // Run all colors of green
  for (int voltage = 0; voltage <= 255; voltage++) {
    dacWrite(greenPin, voltage);
    delay(10);
  }
  delay(1000); // Wait for 1 second

  // Run all colors of blue
  for (int voltage = 0; voltage <= 255; voltage++) {
    dacWrite(bluePin, voltage);
    delay(10);
  }
  delay(1000); // Wait for 1 second

  // Turn off all colors
  dacWrite(redPin, 0);
  dacWrite(greenPin, 0);
  dacWrite(bluePin, 0);
  delay(1000); // Wait for 1 second
}
```

ניסוי מס' 2. הרצת כל הצבעים ביחד.

```
const int redPin = 2; // PWM pin for controlling red color
const int greenPin = 4; // PWM pin for controlling green color
const int bluePin = 5; // PWM pin for controlling blue color

void setup() {
  pinMode(redPin, OUTPUT);
  pinMode(greenPin, OUTPUT);
  pinMode(bluePin, OUTPUT);
}

void loop() {
  // Loop through all shades of red
  for (int r = 0; r <= 255; r++) {
    analogWrite(redPin, r);
    // Loop through all shades of green
    for (int g = 0; g <= 255; g++) {
      analogWrite(greenPin, g);
      // Loop through all shades of blue
      for (int b = 0; b <= 255; b++) {
```

```
      analogWrite(bluePin, b);
      delay(10);
    }
   }
  }
}
```

```
const int redPin = 25; // DAC pin for controlling red color
const int greenPin = 26; // DAC pin for controlling green color
const int bluePin = 27; // DAC pin for controlling blue color

void setup() {
  // No setup is needed for DAC initialization
}

void loop() {
  // Loop through all shades of red
  for (int r = 0; r <= 255; r++) {
    dacWrite(redPin, r);
    // Loop through all shades of green
    for (int g = 0; g <= 255; g++) {
      dacWrite(greenPin, g);
      // Loop through all shades of blue
      for (int b = 0; b <= 255; b++) {
        dacWrite(bluePin, b);
        delay(10);
      }
    }
  }
}
```

הפעלת בקר מנוע סרוו (חיבור חיצוני).

הבנת מנועי סרוו: מבנה, תכונות ומאפיינים

מבוא:

מנועי סרוו הם מרכיבים חיוניים ברובוטיקה, אוטומציה ומערכות אלקטרומכניות שונות. השליטה המדויקת שלהם והיכולת לשמור על מיקום הופכים אותם לבעלי ערך רב ביישומים החל מזרועות רובוטיקה ועד רכבי RC. בסעיף זה נתעמק במבנה, בתכונות, במאפיינים ובהבדלים בין מנועי סרוו לסוגים אחרים של מנועים.

מבנה של מנועי סרוו:

מנוע סרוו מורכב ממספר מרכיבים מרכזיים:

מנוע DC: הליבה של מנוע סרוו היא מנוע DC, בדרך כלל עם מומנט גבוה ומאפייני מהירות נמוכה.

תיבת הילוכים: מנועי סרוו כוללים תיבת הילוכים להפחתת מהירות המנוע תוך הגדלת תפוקת המומנט שלו.

מעגלי בקרה: מנועי סרוו מצוידים במעגלי בקרה, כולל מנגנון משוב (פוטנציומטר או מקודד) כדי לקבוע במדויק את מיקום המנוע.

מנגנון מיקום: מנגנון זה מאפשר לציר המנוע להסתובב בטווח מוגדר, מוגבל בדרך כלל לסביבות 180 או 360 מעלות.

תכונות של מנועי סרוו:

בקרת מיקום מדויקת: מנועי סרוו יכולים לשלוט במדויק על מיקומם, מה שהופך אותם לאידיאליים עבור יישומים הדורשים תנועה מדויקת.

מומנט גבוה: למרות גודלם הקומפקטי, מנועי סרוו מציעים תפוקת מומנט גבוהה, מה שהופך אותם למתאימים למשימות הדורשות כוח משמעותי.

מערכת משוב: מנועי סרוו משלבים מערכת משוב המנטרת כל הזמן את מיקום המנוע, ומאפשרת מיקום מדויק והתאמות בזמן אמת.

בקרת לולאה סגורה: עם מערכת המשוב שלהם, מנועי סרוו פועלים במערכת בקרה במעגל סגור, המבטיח תנועה מדויקת ומגיבה.

צריכת חשמל נמוכה: מנועי סרוו צורכים הספק נמוך יחסית, מה שהופך אותם לחסכוניים באנרגיה ומתאימים ליישומים המופעלים על ידי סוללה.

מאפיינים של מנועי סרוו:

מהירות: מנועי סרוו פועלים בדרך כלל במהירויות נמוכות יותר בהשוואה לסוגי מנועים אחרים, מה שמאפשר שליטה מדויקת ותנועה חלקה.

מומנט: מנועי סרוו מציעים תפוקת מומנט גבוהה, המאפשרת להם להפעיל כוח משמעותי, במיוחד במהירויות נמוכות.

דיוק: מנועי סרוו מצטיינים בדיוק, הודות למערכת הבקרה בלולאה סגורה ומנגנון המשוב, המבטיחים מיקום מדויק.

טווח מיקום: טווח המיקום של מנועי סרוו מוגבל, בדרך כלל משתרע על 180 או 360 מעלות, בהתאם לעיצוב המנוע.

זמן תגובה: מנועי סרוו מציגים זמני תגובה מהירים, ומכוונים במהירות את מיקומם על סמך אותות בקרה ומידע משוב.

ההבדל בין מנועי סרוו לסוגי מנועים אחרים:

מנועי DC: בניגוד למנועי DC, המסתובבים ברציפות בכיוון אחד, מנועי סרוו יכולים להסתובב למיקומים ספציפיים בטווח המוגדר שלהם.

מנועי צעד: בעוד שמנועי צעד מציעים גם מיקום מדויק, מנועי סרוו מצטיינים בהיענות ובמהירות בזכות מערכת הבקרה שלהם בלולאה סגורה.

מנועי DC ללא מברשות (BLDC): למנועי BLDC אין בקרת מיקום מדויקת של מנועי סרוו, מה שהופך אותם לפחות מתאימים ליישומים הדורשים מיקום מדויק.

לסיכום, מנועי סרוו ממלאים תפקיד מכריע באינספור יישומים הדורשים בקרת תנועה מדויקת. עם גודלם הקומפקטי, תפוקת המומנט הגבוהה והמיקום המדויק שלהם, הם הכרחיים ברובוטיקה, אוטומציה ומערכות אלקטרומכניות שונות. על ידי הבנת המבנה, התכונות והמאפיינים של מנועי סרוו, מהנדסים וחובבים יכולים למנף את היכולות שלהם ליצירת פתרונות בקרת תנועה יעילים ומדויקים.

ניסוי מס' 1. הפעלת מנוע סרוו לזוויות מיוחדות:

```
#include <ESPServo.h>
Servo servoMotor;  // Create a servo object
const int servoPin = 2;  // GPIO pin connected to the servo
void setup() {
  servoMotor.attach(servoPin);  // Attach the servo object to the GPIO pin
}
void loop() {
  // Move the servo to the 0-degree position (minimum angle)
  servoMotor.write(0);
  delay(1000);  // Wait for 1 second
  // Move the servo to the 90-degree position (middle angle)
  servoMotor.write(90);
  delay(1000);  // Wait for 1 second
  // Move the servo to the 180-degree position (maximum angle)
  servoMotor.write(180);
  delay(1000);  // Wait for 1 second
```

```
}
```

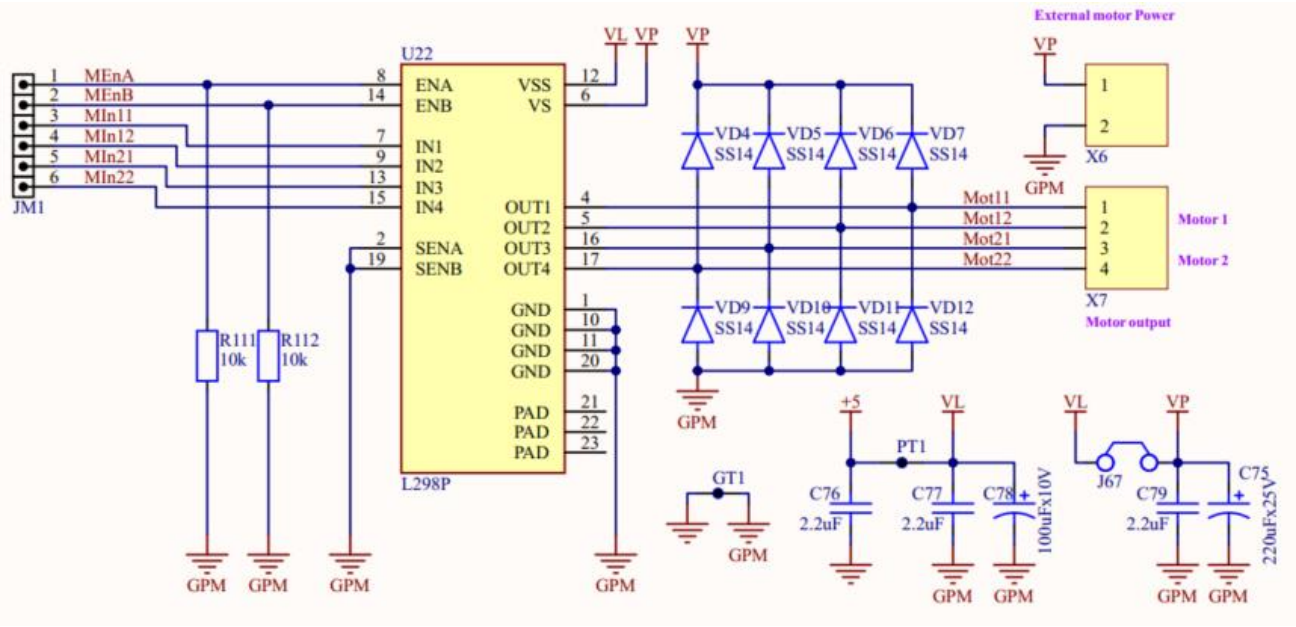
ניסוי מס' 2. הזזת מנוע סרוו בצורה סדרתית.

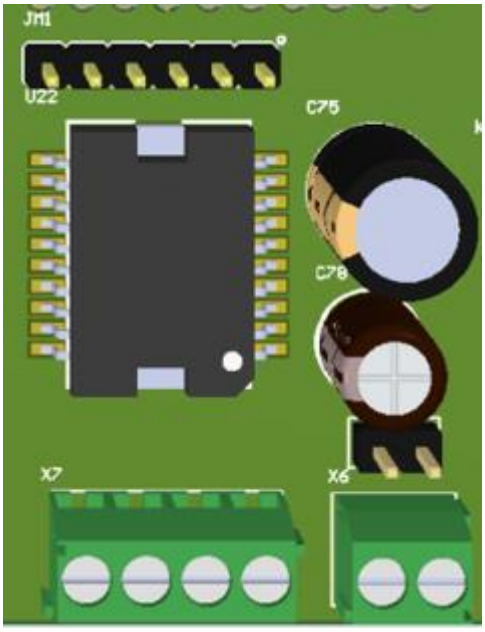
```
#include <ESPServo.h>
Servo servoMotor;  // Create a servo object
const int servoPin = 2;  // GPIO pin connected to the servo
void setup() {
  servoMotor.attach(servoPin);  // Attach the servo object to the GPIO pin
}
void loop() {
  // Sweep the servo motor from 0 to 180 degrees
  for (int angle = 0; angle <= 180; angle++) {
    servoMotor.write(angle);
    delay(10);  // Adjust delay for smoother motion
  }
  // Sweep the servo motor from 180 to 0 degrees
  for (int angle = 180; angle >= 0; angle--) {
    servoMotor.write(angle);
    delay(10);  // Adjust delay for smoother motion
  }
}
```

ניסוי מס' 3. הזזת מנוע סרוו בעזרת פוטנציומטר.

```
#include <ESPServo.h>
Servo servoMotor;  // Create a servo object
const int servoPin = 2;       // GPIO pin connected to the servo
const int potentiometerPin = 34;  // Analog pin connected to the potentiometer
int potValue;                 // Variable to store potentiometer value
void setup() {
  servoMotor.attach(servoPin);  // Attach the servo object to the GPIO pin
}
void loop() {
  // Read the value of the potentiometer
  potValue = analogRead(potentiometerPin);
  // Map the potentiometer value (0-4095) to servo angle (0-180)
  int angle = map(potValue, 0, 4095, 0, 180);
  // Set the servo angle based on the potentiometer value
  servoMotor.write(angle);
  delay(20);  // Adjust delay for smoother motion
}
```

הפעלת בקר מנוע DC.





מנועי DC נמצאים בשימוש נרחב ביישומים שונים, החל מאוטומציה תעשייתית ועד פרויקטים תחביבים, בשל הפשטות, האמינות והרבגוניות שלהם. בסעיף זה, נחקור את המבנה, התכונות והמאפיינים של מנועי DC, ולאחר מכן נעמיק כיצד לשלוט במנוע DC באמצעות דרייבר המנוע L298 עם אפנון PWM.

מבנה של מנועי DC:

מנועי DC מורכבים ממספר מרכיבי מפתח:

סטטור: החלק הנייח של המנוע, שבו נמצא המגנט או האלקטרומגנט האחראי על יצירת השדה המגנטי.

רוטור: החלק המסתובב של המנוע, בדרך כלל עם סלילי תיל מלופפים סביב ליבה מרכזית, המקיים אינטראקציה עם השדה המגנטי כדי לייצר תנועה סיבובית.

קומוטטור: התקן טבעת מפוצלת שהופך את כיוון זרימת הזרם דרך פיתולי הרוטור, מה שמבטיח סיבוב רציף.

מברשות: מגעים מבוססי פחמן השומרים על מגע חשמלי עם הקומוטטור, ומאפשרים לזרם לזרום דרך פיתולי הרוטור.

תכונות של מנועי DC:

בנייה פשוטה: למנועי DC יש עיצוב פשוט המורכב ממעט רכיבים, מה שהופך אותם קלים לייצור ולתחזוקה.

סיבוב דו כיווני: מנועי DC יכולים להסתובב בשני הכיוונים על ידי היפוך כיוון זרימת הזרם דרך פיתולי הרוטור.

מהירות משתנה: ניתן לשלוט במהירות של מנועי DC על ידי התאמת המתח או הזרם המסופקים למנוע, המאפשר ויסות מהירות מדויק.

מומנט גבוה: מנועי DC מציעים תפוקת מומנט גבוהה, מה שהופך אותם למתאימים ליישומים הדורשים כוח משמעותי.

מאפיינים של מנועי DC:

מהירות: מנועי DC פועלים בדרך כלל במהירויות משתנות, בהתאם לתנאי המתח והעומס.

מומנט: מנועי DC מספקים תפוקת מומנט גבוהה במהירויות נמוכות, כשהמומנט פוחת ככל שהמהירות עולה.

יעילות: מנועי DC יעילים יחסית, וממירים אנרגיה חשמלית לאנרגיה מכנית עם הפסדים מינימליים.

מאפייני התנעה: מנועי DC מציגים מומנט התנעה טוב, המאפשר להם להתגבר על האינרציה ולהאיץ במהירות.

ההבדל בין מנועי DC לסוגי מנועים אחרים:

מנועי AC: בניגוד למנועי AC, הפועלים על זרם חילופין ודורשים מנגנוני בקרה מורכבים, מנועי DC פועלים על זרם ישר ומציעים אפשרויות בקרת מהירות פשוטות יותר.

מנועי DC ללא מברשות (BLDC): מנועי BLDC מבטלים את הצורך במברשות ובקומוטטורים, וכתוצאה מכך דרישות תחזוקה נמוכות יותר ובלאי מופחת בהשוואה למנועי DC מסורתיים.

מנועי צעד: בעוד שמנועי צעד מציעים בקרת מיקום מדויקת, מנועי DC מצטיינים ביישומים הדורשים מהירות משתנה וסיבוב דו-כיווני.

שליטה במנוע DC עם L298 ו-PWM:

דרייבר המנוע L298 הוא בחירה פופולרית לשליטה במנועי DC עם מיקרו-בקרים כמו ESP32. הוא מאפשר שליטה דו-כיוונית של שני מנועי DC ותומך באפנון PWM לבקרת מהירות.

ESP32 GPIO1 --> IN1----------> OUT1 Motor A

ESP32 GPIO2 --> IN2----------> OUT2 Motor A

ESP32 GPIO3 --> ENA_A-----> Enable(Speed) Motor A

ESP32 GPIO4 --> IN3----------> OUT3 Motor B

ESP32 GPIO5 --> IN4----------> OUT4 Motor B

ESP32 GPIO6 --> ENA_B-----> Enable(Speed) Motor B

ניסוי מס' 1. הפעלה בסיסית של מנוע DC.

```
#define IN1 1
#define IN2 2
#define ENA 3  // PWM pin for Motor A speed control

void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);
}

void loop() {
```

```
  // Set direction and speed for Motor A
  digitalWrite(IN1, HIGH);  // Set direction forward
  digitalWrite(IN2, LOW);
  analogWrite(ENA, 200);  // Set PWM duty cycle (0-255) for speed control
  delay(2000);  // Run for 2 seconds

  // Stop Motor A
  analogWrite(ENA, 0);  // Set PWM duty cycle to 0 for stop
  delay(1000);  // Wait for 1 second

  // Set direction and speed for Motor A in reverse
  digitalWrite(IN1, LOW);  // Set direction reverse
  digitalWrite(IN2, HIGH);
  analogWrite(ENA, 150);  // Set PWM duty cycle for reverse speed control
  delay(2000);  // Run for 2 seconds

  // Stop Motor A
  analogWrite(ENA, 0);  // Stop motor
  delay(1000);  // Wait for 1 second
}
```

ניסוי מס' 2. הפעלה בסיסית של מנוע DC עם שינוי מהירות.

```
#define ENA 5 // PWM pin for Motor A speed control
#define IN1 4
#define IN2 3
void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);
}
void loop() {
  // Increase motor speed
  for (int speed = 0; speed <= 255; speed++) {
    analogWrite(ENA, speed); // Adjust PWM value for speed control
    digitalWrite(IN1, HIGH); // Set direction forward
    digitalWrite(IN2, LOW);
    delay(10); // Adjust delay for smooth speed increase
  }
  delay(1000); // Wait for 1 second
  // Decrease motor speed
  for (int speed = 255; speed >= 0; speed--) {
    analogWrite(ENA, speed); // Adjust PWM value for speed control
    digitalWrite(IN1, HIGH); // Set direction forward
    digitalWrite(IN2, LOW);
    delay(10); // Adjust delay for smooth speed decrease
  }
  delay(1000); // Wait for 1 second
}
```

ניסוי מס' 3. הפעלה בסיסית של 2 מנועים DC.

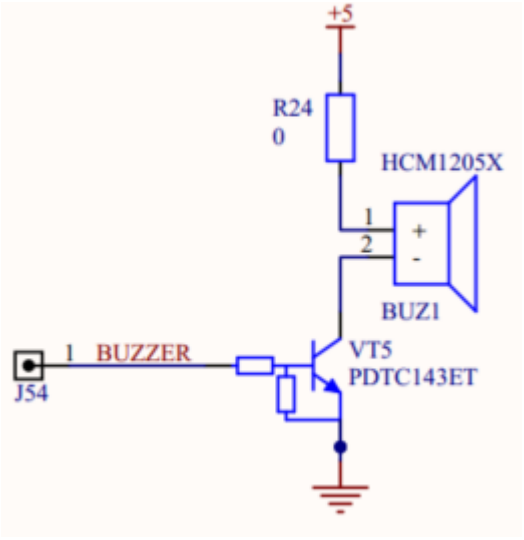
```
#define ENA 5 // PWM pin for Motor A speed control
#define IN1 4
#define IN2 3
#define ENB 6 // PWM pin for Motor B speed control
#define IN3 7
#define IN4 8
```

```
void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENB, OUTPUT);
}
void loop() {
  // Move Motor A forward and Motor B backward
  analogWrite(ENA, 200); // Adjust PWM value for Motor A speed control
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  analogWrite(ENB, 150); // Adjust PWM value for Motor B speed control
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  delay(2000); // Run for 2 seconds
  // Stop both motors
  analogWrite(ENA, 0);
  analogWrite(ENB, 0);
  delay(1000); // Wait for 1 second
}
```

ניסוי מס' 4. הפעלה בסיסית של 2 מנועים DC עם מהירות משתנה.

```
#define ENA 5 // PWM pin for Motor A speed control
#define IN1 4
#define IN2 3
#define ENB 6 // PWM pin for Motor B speed control
#define IN3 7
#define IN4 8
void setup() {
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(ENA, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  pinMode(ENB, OUTPUT);
}
void loop() {
  // Move Motor A forward and Motor B backward
  for (int speedA = 0, speedB = 255; speedA <= 255 && speedB >= 0; speedA++, speedB--) {
    analogWrite(ENA, speedA); // Adjust PWM value for Motor A speed control
    digitalWrite(IN1, HIGH);
    digitalWrite(IN2, LOW);
    analogWrite(ENB, speedB); // Adjust PWM value for Motor B speed control
    digitalWrite(IN3, LOW);
    digitalWrite(IN4, HIGH);
    delay(10); // Adjust delay for smooth speed variation
  }
  // Stop both motors
  analogWrite(ENA, 0);
  analogWrite(ENB, 0);
  delay(1000); // Wait for 1 second
}
```

הפעלת צפצפה.



זמזמים הם מכשירים אלקטרו-אקוסטיים המשמשים בדרך כלל במעגלים אלקטרוניים ליצירת התראות קוליות, התראות או צלילי אזעקה. הם מוצאים יישומים בתחומים שונים כגון מוצרי אלקטרוניקה, מערכות רכב, מכונות תעשייתיות ומערכות אבטחה. בסעיף זה, נתעמק במבנה, בתכונות, במעגלים האלקטרוניים ובמאפיינים של זמזמים, וכן נחקור את ההבדלים בין זמזמים פעילים לפסיביים.

מבנה של זמזמים:

זמזמים מורכבים בדרך כלל מהרכיבים הבאים:

אלמנט פיזואלקטרי: רוב הזמזמים משתמשים באלמנט פיזואלקטרי כמרכיב הליבה. אלמנט זה יוצר צליל כאשר הוא נתון לזרם חשמלי. הוא מתעוות בצורה מכנית בתגובה למתח מופעל, ומייצר רעידות שיוצרות גלי קול.

דיור: האלמנט הפיאזואלקטרי נמצא במארז פלסטיק או מתכת שנועד להגביר ולכוון את הצליל המופק.

תכונות של זמזמים:

גודל קומפקטי: זמזמים הם בדרך כלל קומפקטיים בגודלם, מה שהופך אותם מתאימים לשילוב במכשירים אלקטרוניים שונים מבלי לתפוס מקום רב.

צריכת חשמל נמוכה: בדרך כלל הם צורכים חשמל נמוך, מה שהופך אותם לחסכוניים באנרגיה ומתאימים למכשירים המופעלים על ידי סוללה.

טווח תדרים רחב: זמזמים יכולים להפיק צליל על פני טווח תדרים רחב, המאפשר גמישות ביצירת סוגים שונים של צלילים וצלילים.

קלות שילוב: הם קלים לשילוב במעגלים אלקטרוניים הודות לעיצובם הפשוט וחיבורי החשמל הסטנדרטיים.

מעגל אלקטרוני של זמזמים:

המעגל האלקטרוני של זמזם תלוי אם הוא זמזם אקטיבי או פסיבי:

זמזם פעיל: זמזם פעיל מכיל מעגל מתנד מובנה שיוצר צליל בתדר מסוים כשהוא מופעל. זה דורש אות זרם חילופין מתמשך (AC) כדי להפיק צליל. מעגל המתנד מתוכנן בדרך כלל להדהד בתדר מסוים, הקובע את גובה הצליל המופק.

זמזם פסיבי: זמזם פסיבי אינו מכיל מעגל מתנד. זה דורש מחולל אותות חיצוני (כגון מיקרו-בקר או מעגל אלקטרוני אחר) כדי להפיק קול. מחולל האותות החיצוני מניע את הזמזם על ידי אספקת אותות זרם חילופין (AC) בתדירות ובמשך הרצוי.

מאפיינים של זמזמים:

פלט קול: זמזמים מפיקים גלי קול בטווח התדרים הנשמע, בדרך כלל נע בין כמה מאות הרץ לכמה קילו-הרץ.

רמת לחץ הקול (SPL): רמת לחץ הקול של זמזם קובעת את עוצמתו ונמדדת בדציבלים (dB). לזמזמים שונים עשויים להיות דירוגי SPL משתנים בהתאם לעיצובם וליישומים המיועדים.

מתח הפעלה: לזמזמים יש טווח מתח הפעלה מוגדר שבתוכו הם יכולים לתפקד ביעילות. הפעלתם מחוץ לטווח זה עלולה להשפיע על הביצועים שלהם או לפגוע בהם.

תדר תהודה: תדר התהודה של זמזם מתייחס לתדר שבו הוא מפיק את פלט הצליל החזק ביותר. זה נקבע על ידי המאפיינים הפיזיים של האלמנט הפיאזואלקטרי ועיצוב הזמזם.

ההבדלים בין זמזמים פעילים לפסיביים:

מעגל מתנד: לזמזמים אקטיביים יש מעגל מתנד מובנה, בעוד לזמזמים פסיביים אין.

בקרה: זמזמים אקטיביים מפיקים צליל באופן עצמאי כאשר הם מופעלים, בעוד שזמזמים פסיביים דורשים מחולל אותות חיצוני כדי להפיק קול.

פשטות: זמזמים אקטיביים פשוטים יותר לשימוש מכיוון שהם דורשים רק אספקת חשמל כדי לפעול, בעוד שזמזמים פסיביים דורשים מעגלים נוספים ליצירת צליל.

בקרת סאונד: זמזמים אקטיביים מפיקים צליל בתדר קבוע שנקבע על ידי מעגל המתנד שלהם, בעוד שזמזמים פסיביים מאפשרים גמישות רבה יותר בשליטה בתדירות ומשך יציאת הקול באמצעות הפקת אותות חיצוני.

סיכום:

זמזמים הם מרכיבים חיוניים במעגלים אלקטרוניים להפקת התראות והודעות קוליות. הבנת המבנה, התכונות, המעגלים האלקטרוניים והמאפיינים שלהם חיונית לבחירת הזמזם המתאים ליישום מסוים. אם בחירת זמזם אקטיבי או פסיבי תלויה בדרישות הספציפיות של האפליקציה, לרבות פלט הצליל הרצוי, גמישות השליטה ומורכבות האינטגרציה.

ניסוי מס' 1. הפעלה בסיסית של צפצפה (זמזם) בתדר 50 הרץ.

```
#define BUZZER_PIN 2
void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
}
void loop() {
  // Make the buzzer beep for 100 milliseconds
  digitalWrite(BUZZER_PIN, HIGH);
  delay(10);
  digitalWrite(BUZZER_PIN, LOW);
  delay(10);
}
```

ניסוי מס' 2. הפעלת צפצפה עם פונקציה Tone.

```
#define BUZZER_PIN 2

void setup() {
}

void loop() {
  // Play a simple melody
  tone(BUZZER_PIN, 262, 200); // C4
  delay(200);
  tone(BUZZER_PIN, 294, 200); // D4
  delay(200);
  tone(BUZZER_PIN, 330, 200); // E4
  delay(200);
  tone(BUZZER_PIN, 349, 200); // F4
  delay(200);
  tone(BUZZER_PIN, 392, 200); // G4
```

```
  delay(200);
  tone(BUZZER_PIN, 440, 200); // A4
  delay(200);
  tone(BUZZER_PIN, 494, 200); // B4
  delay(200);
}
```

ניסוי מס' 3. הפעלת צפצפה ל- SOS בקוד מורס.

```
#define BUZZER_PIN 2
void setup() {
}
void loop() {
  // Play "SOS" in Morse code
  dot(); dot(); dot();
  delay(200);
  dash(); dash(); dash();
  delay(200);
  dot(); dot(); dot();
  delay(600);
}
void dot() {
  tone(BUZZER_PIN, 1000, 200);
  delay(200);
  noTone(BUZZER_PIN);
}
void dash() {
  tone(BUZZER_PIN, 1000, 600);
  delay(600);
  noTone(BUZZER_PIN);
}
```

ניסוי מס' 4. הפעלת צפצפה עם תדר משתנה.

```
#define BUZZER_PIN 2
#define POTENTIOMETER_PIN 34
void setup() {
  pinMode(BUZZER_PIN, OUTPUT);
}
void loop() {
  // Read the value from the potentiometer
  int potValue = analogRead(POTENTIOMETER_PIN);
  // Map the potentiometer value to a frequency range (100 Hz to 1000 Hz)
  int frequency = map(potValue, 0, 4095, 100, 1000);
  // Play the tone with the mapped frequency
  tone(BUZZER_PIN, frequency);
  delay(10); // Adjust for smoother response
}
```

ניסוי מס' 5. הפעלת צפצפה עם PWM.

```
#define BUZZER_PIN 2
void setup() {
}
void loop() {
  // Ramp up the buzzer sound with PWM
  for (int dutyCycle = 0; dutyCycle <= 255; dutyCycle++) {
```

```
    analogWrite(BUZZER_PIN, dutyCycle);
    delay(10);
  }
  delay(1000); // Wait for 1 second
  // Ramp down the buzzer sound with PWM
  for (int dutyCycle = 255; dutyCycle >= 0; dutyCycle--) {
    analogWrite(BUZZER_PIN, dutyCycle);
    delay(10);
  }
  delay(1000); // Wait for 1 second
}
```

התנסות מסכמת על נושא קלט ופלט אנלוגי.

ניסוי מס' 1. בקרת בהירות הלד בהתאם למצב של פוטנציומטר.

קוד זה קורא את הערך האנלוגי מהפוטנציומטר וממפה אותו לטווח שבין 0 ל-255 כדי לשלוט בבהירות של נורית LED המחוברת לפין 13.

```
const int potPin = 32;
const int ledPin = 13;
void setup() {
  pinMode(ledPin, OUTPUT);
}
void loop() {
  int potValue = analogRead(potPin);
  int brightness = map(potValue, 0, 4095, 0, 255);
  analogWrite(ledPin, brightness);
}
```

ניסוי מס' 2. בקרת מיקום מנוע סרוו

קוד זה קורא את הערך האנלוגי מהפוטנציומטר וממפה אותו לטווח הזווית של 0 עד 180 מעלות כדי לשלוט במיקום של מנוע סרוו.

```
#include <ESP32Servo.h>
const int potPin = 32;
const int servoPin = 9;
Servo myServo;
void setup() {
  myServo.attach(servoPin);
}
void loop() {
  int potValue = analogRead(potPin);
  int angle = map(potValue, 0, 4095, 0, 180);
  myServo.write(angle);
  delay(15); // Optional delay for smoother movement
}
```

ניסוי מס' 3. כיול חיישן טמפרטורה

קוד זה קורא את הערך האנלוגי מהפוטנציומטר ומשתמש בו כדי לכייל חיישן טמפרטורה על ידי הפעלת מקדם כיול.

```
const int potPin = 32;
const float calibrationFactor = 0.1;
void setup() {
  Serial.begin(9600);
}
void loop() {
  int potValue = analogRead(potPin);
  float temperature = potValue * calibrationFactor;
  Serial.print("Temperature: ");
  Serial.println(temperature);
  delay(1000);
}
```

ניסוי מס' 4. בקרת עוצמת קול.

קוד זה קורא את הערך האנלוגי מהפוטנציומטר וממפה אותו כדי לשלוט בעוצמת הקול של רמקול המחובר לפין 5.

```
const int potPin = 32;
const int speakerPin = 5;
void setup() {
  pinMode(speakerPin, OUTPUT);
}
void loop() {
  int potValue = analogRead(potPin);
  int volume = map(potValue, 0, 4095, 0, 255);
  analogWrite(speakerPin, volume);
}
```

ניסוי מס' 5. בקרת בהירות LCD

קוד זה קורא את הערך האנלוגי מהפוטנציומטר וממפה אותו כדי לשלוט בבהירות של תאורת LCD אחורית.

```
#include <LiquidCrystal.h>
const int potPin = 32;
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);
void setup() {
  lcd.begin(16, 2);
}
void loop() {
  int potValue = analogRead(potPin);
  int brightness = map(potValue, 0, 4095, 0, 255);
  lcd.setBacklight(brightness);
}
```

תרגול עצמי בנושא קלט ופלט אנלוגי.

תרגיל מס' 1.

בדיקת טמפרטורה ובקרת PWM:

חבר חיישן טמפרטורה (לדוגמה, TMP36) לפין כניסה אנלוגי של ה-ESP32.

קרא את המתח האנלוגי מהחיישן והמיר אותו לטמפרטורה באמצעות גיליון הנתונים של החיישן.

השתמש ב-PWM כדי לשלוט בבהירות של נורית LED בהתבסס על קריאת הטמפרטורה. טמפרטורה גבוהה יותר תואמת לד בהיר יותר.

תרגיל מס' 2.

בדיקת אור ובקרת מנוע סרוו:

חבר נגד תלוי אור (LDR) לפין כניסה אנלוגי של ה-ESP32.

קרא את המתח האנלוגי מה-LDR ומפה אותו לזווית מנוע סרוו. סביבה כהה יותר מתאימה לזווית קטנה יותר וסביבה בהירה יותר לזווית גדולה יותר.

תרגיל מס' 3.

בדיקת קול ובקרת זמזם:

חבר מודול חיישן קול (לדוגמה, KY-038) לפין כניסה אנלוגי של ה-ESP32.

קרא את המתח האנלוגי מהחיישן ומפה אותו לתדר של זמזם. רמות צליל גבוהות יותר מתאימות לתדרים גבוהים יותר.

תרגיל מס' 4.

בדיקת חיישן אור ובקרת מנוע DC:

חבר חיישן אור לפין כניסה אנלוגי של ה-ESP32.

קרא את המתח האנלוגי מהחיישן והשתמש בו כדי לשלוט במהירות של מנוע DC. התנגדות גבוהה יותר של חיישן גמיש תואמת למהירות מנוע מהירה יותר.

תרגיל מס' 5.

בדיקת זוג אופטי בכניסה אנלוגית ואנימציית LED:

חבר זוג אופטי לפין כניסה אנלוגי של ה-ESP32.

קרא את המתח האנלוגי מהחיישן והשתמש בו כדי להפעיל אנימציית LED. קרבה יותר מפעילה אנימציה מהירה יותר.

פרק 3. פסיקות.

פסיקות ממלאות תפקיד מכריע בתכנות מיקרו-בקרים בכך שהם מאפשרים למעבד להגיב מיידית לאירועים חיצוניים מבלי לפקח עליהם באופן רציף. בפרק זה, נחקור פסיקות בהקשר של המיקרו-בקר ESP32, נבחן את ההבדלים העיקריים בין פסיקות לעקרון הסקרים (Polling), ונספק דוגמאות להמחשת היישומים שלהם.

פסיקות ב-ESP32:

פסיקות במיקרו-בקר ESP32 הם מנגנונים מבוססי חומרה המאפשרים למעבד להפריע לזרימת הביצוע הנוכחית שלו ולטפל מיד באירועים או משימות ספציפיות. ה-ESP32 תומך בסוגים שונים של פסיקות, כולל פסיקות חיצוניות, פסיקות טיימר ופסיקות היקפיות. כאשר מתרחשת פסיקה, המעבד שומר את מצבו הנוכחי, מבצע את שגרת שירות ההפסקה (ISR), ולאחר מכן ממשיך את הביצוע הרגיל.

ההבדל בין פסיקות ל-Polling :

מונחי אירועים לעומת Polling:

פסיקות: בגישה מונעת פסיקה, המעבד מגיב לאירועים חיצוניים (כגון אותות מחיישנים או כניסות משתמש) מיד כשהם מתרחשים. הוא אינו עוקב באופן רציף אחר האירועים, אלא נשאר פעיל עד להופעת הפרעה.

Polling: לעומת זאת, הסקרים כוללים בדיקה מתמדת של מצב אירועים או תנאים ספציפיים באמצעות שאילתות חוזרות או "סקרים". המעבד מקדיש חלק מזמן הביצוע שלו לסקר, מה שעלול להוביל לניצול משאבים לא יעיל ולצריכת חשמל גבוהה יותר.

ניצול משאבים:

פסיקות: פסיקות ממזערות בזבוז משאבים על ידי מתן אפשרות למעבד להישאר לא פעיל עד להתרחשות אירוע, תוך חיסכון בכוח ובמשאבי חישוב.

Polling: סקר צורך מחזורי מעבד, גם כאשר אין אירועים, וכתוצאה מכך ניצול משאבים לא יעיל והיענות מופחתת למשימות רגישות לזמן.

היענות:

פסיקות: תכנות מונע פסיקות מציע זמני תגובה נמוכים, שכן המעבד מגיב מיד לאירועים חיצוניים ללא דיחוי.

Polling: סקר מציג חביון, שכן המעבד חייב להמתין למחזור הסקר הבא כדי לזהות אירועים ולהגיב אליהם, מה שמוביל לעיכובים אפשריים בטיפול במשימות קריטיות בזמן.

השוואה בין קוד להפעלת פסיקה לבין קוד לשיטת Polling.

```
const int buttonPin = 2;
volatile bool buttonPressed = false;
void IRAM_ATTR handleInterrupt() {
  buttonPressed = true;
}
void setup() {
  pinMode(buttonPin, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(buttonPin), handleInterrupt, RISING);
}
void loop() {
  if (buttonPressed) {
    // Handle button press event
    // Perform necessary actions
    buttonPressed = false; // Reset flag
  }
}
```

```
const int temperaturePin = A0;
const int threshold = 25;
void setup() {
  pinMode(temperaturePin, INPUT);
}
void loop() {
  int temperature = analogRead(temperaturePin);
  if (temperature > threshold) {
    // Perform actions based on high temperature
    // Wait for a period before polling again
    delay(1000);
  }
}
```

תכנות לטיפול בפסיקות הוא היבט בסיסי של פיתוח מערכות משובצות, המאפשר למיקרו-בקרים כמו ESP32 להגיב מיידית לאירועים חיצוניים תוך ניהול יעיל של משאבי המערכת. בסעיף זה, נתעמק במורכבויות של תכנות לטיפול בפסיקות בפלטפורמת ESP32, ונחקור את המנגנונים, היישומים ושיטות העבודה המומלצות שלה.

הבנת פסיקות ב-ESP32:

פסיקות במיקרו-בקר ESP32 מספקות מנגנון לתגובה לאירועים או אותות חיצוניים מבלי לבצע סקר רציף להתרחשותם. ה-ESP32 תומך בסוגים שונים של פסיקות, כולל פסיקות GPIO חיצוניות, פסיקות טיימר ופסיקות היקפיות. כאשר מופעלת פסיקה, המעבד קוטע את זרימת הביצוע הנוכחית שלו, מבצע שגרת שירות פסיקה מוגדרת מראש (ISR), ולאחר מכן ממשיך את הביצוע הרגיל.

סוגי פסיקות הנתמכות על ידי ESP32:

פסיקות GPIO חיצוניות: מופעלות על ידי שינויים במצב פיני GPIO, כגון קצוות עולים או יורדים.

פסיקות טיימר: נוצרות על ידי הטיימרים הפנימיים של ה-ESP32, המאפשרות תזמון מדויק וביצוע משימות תקופתיות.

פסיקות היקפיות: משויכות לציוד היקפי ספציפי, כגון UART, SPI או I2C, כדי לטפל באירועים כמו קליטת נתונים או שידור.

תכנות טיפול בפסיקות ESP32:

תכנות טיפול בפסיקות ב-ESP32 כולל בדרך כלל את השלבים הבאים:

הגדרת שגרות שירות פסיקה (ISRs): הטמעת פונקציות ISR לטיפול באירועי פסיקה ספציפיים. ISRs מוכרזים עם התכונה IRAM_ATTR כדי להבטיח שהם מאוחסנים ב-RAM לצורך ביצוע מהיר יותר.

קביעת תצורה של הפסקות: קביעת תצורת הגדרות ההפסקה, כגון סוג טריגר (למשל, קצה עולה, קצה יורד), פין GPIO ופונקציית ISR.

חיבור פסיקות: חיבור הפסיקות המוגדרות לפיני ה-GPIO המתאימים או למשאבי החומרה באמצעות הפונקציה attachInterrupt() .

יישום שגרתי של שירות הפסקה: כתיבת קוד ISR לביצוע פעולות נחוצות בתגובה לאירוע ההפרעה. ה-ISRs צריכים להיות תמציתיים, לא חוסמים, ולהתבצע במהירות האפשרית כדי למזער את זמן השהיית ההפסקה.

מיסוך פסיקות וסדר עדיפות: ניהול סדרי עדיפויות פסיקות והפעלה או השבתה סלקטיבית של פסיקות בהתבסס על דרישות היישום כדי למנוע התנגשויות ולהבטיח תגובה בזמן לאירועים קריטיים.

שיטות עבודה מומלצות לטיפול בפסיקות ב-ESP32:

שמור על ISR קצרים ופשוטים: צמצם למינימום את הקוד המופעל בתוך ISRs כדי לצמצם את זמן השהיית הפסיקות ולהימנע מחסימת זרימת התוכנית הראשית.

הימנע מפעולות חסימה: הימנע משימוש בפונקציות או פעולות חסימה שעלולות לגרום לעיכובים בתוך ISR, כגון עיכובים, תקשורת טורית או חישובים מורכבים.

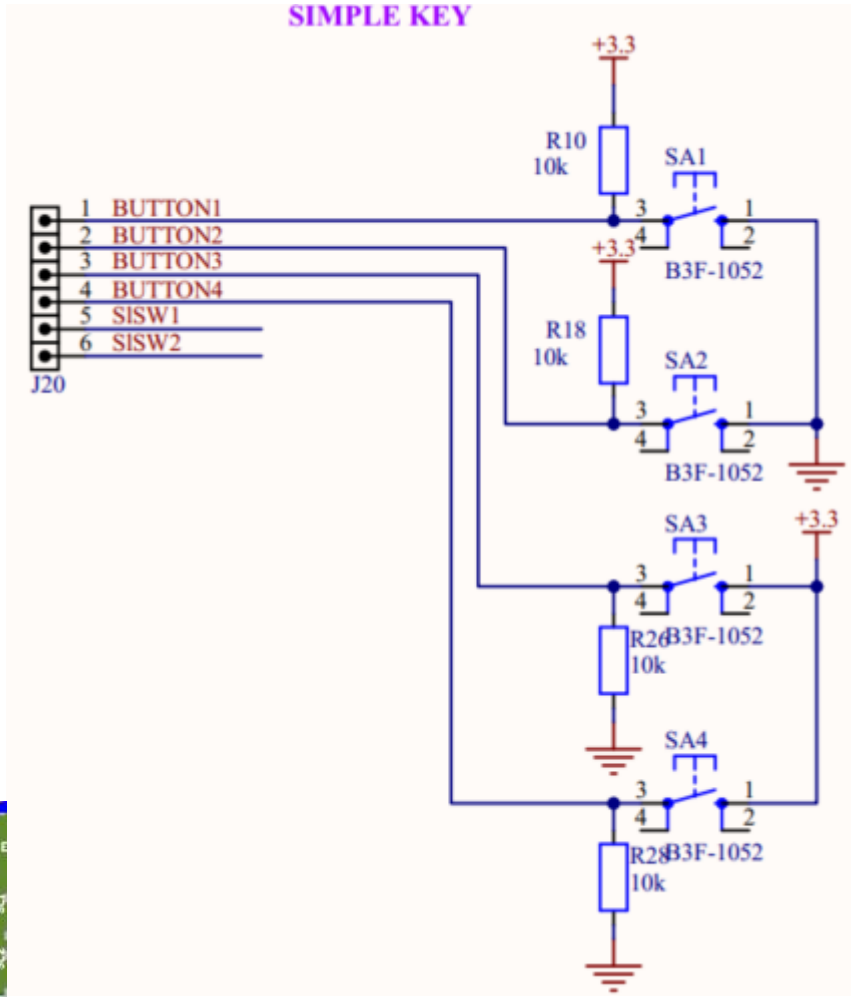
השתמש בדגלים לתקשורת: השתמש בדגלים כדי לתקשר בין ISRs ללולאת התוכנית הראשית, מה שמאפשר סנכרון וחילופי נתונים ללא חסימה.

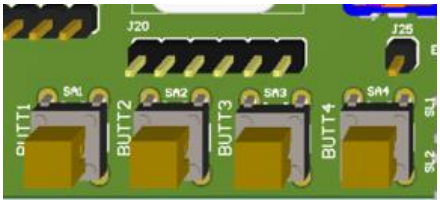
אפשר הגנה על קטע קריטי: השתמש במנגנוני השבתה/הפעלה של הפסקה (portENTER_CRITICAL() ו-portEXIT_CRITICAL()) כדי להגן על קטעי קוד קריטיים מפני פסיקות ולהבטיח שלמות הנתונים.

סדר עדיפות של פסיקות: הקצאת עדיפויות לפסיקות על סמך חשיבות וקריטיות לתפעול המערכת, תוך הבטחת טיפול בזמן באירועים בעלי עדיפות גבוהה.

```
#define BUTTON_PIN 2
void IRAM_ATTR buttonISR() {
  // ISR function to handle button press
  // Perform necessary actions here
}
void setup() {
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), buttonISR, FALLING);
}
void loop() {
  // Main program loop
  // Continuously executes other tasks
}
```

הפעלת פסיקות בעזרת לחצנים.





כפתורי לחיצה הם התקני קלט בכל מקום המשמשים במגוון רחב של מערכות אלקטרוניות, מפרויקטים משובצים פשוטים ועד למכשירי IoT מתוחכמים. בעוד ששיטות סקרים מסורתיות משמשות בדרך כלל לאיתור לחיצות על כפתורים, השימוש בפסיקות מציע גישה יעילה ומגיבה יותר. בסעיף זה, נתעמק בשימוש בלחצני לחיצה עם פסיקות במיקרו-בקר ESP32, ונחקור את המנגנונים, היתרונות וטכניקות היישום שלהם.

הבנת כפתורי לחיצה עם פסיקות:

כפתורי לחיצה, הידועים גם כמתגי מישוש, הם מתגים רגעיים שסוגרים או פותחים מעגל חשמלי בעת לחיצה. שילוב כפתורי לחיצה עם פסיקות מאפשר למיקרו-בקרים כמו ה-ESP32 לזהות לחיצות כפתורים באופן מיידי ולהגיב מיידית מבלי לבדוק באופן רציף את מצב הכפתור. פסיקות מספקות מנגנון מבוסס חומרה להפעלת תגובה מונעת אירועים כאשר מצב הכפתור משתנה, שיפור היעילות והפחתת צריכת החשמל.

יתרונות השימוש בפסיקות עם לחצני לחיצה:

תגובתיות משופרת: הפסקות מאפשרות למיקרו-בקר להגיב מיד ללחיצות על כפתורים, להפחית את זמן ההשהיה ולשפר את חווית המשתמש.

ניצול יעיל של משאבים: תכנות מונע פסיקה ממזער את השימוש במעבד על ידי כך שהוא מאפשר למיקרו-בקר להישאר במצב של צריכת חשמל נמוכה עד שמתרחשת לחיצת כפתור, חוסך אנרגיה ומאריך את חיי הסוללה.

מבנה קוד פשוט: הטמעת טיפול בכפתורים עם פסיקות מפשט את מבנה הקוד על ידי ביטול הצורך בלולאות סקר רציפות, וכתוצאה מכך קוד נקי ויעיל יותר.

ESP32 טיפול בפסיקה עבור לחצני לחיצה:

הטמעת טיפול בכפתורי לחיצה עם פסיקות ב-ESP32 כרוך בדרך כלל בשלבים הבאים:

הגדרת שגרת שירות פסיקה (ISR): הטמע פונקציית ISR לטיפול בלחיצות כפתורים. ה-ISR הוא פונקציה מיוחדת שמתבצעת בתגובה לאירוע ההפרעה.

קביעת תצורה של הפסקות: הגדר את הגדרות ההפסקה, כגון פין GPIO המחובר ללחצן הלחיצה, סוג ההדק (למשל, קצה עולה, קצה יורד) ופונקציית ISR.

חיבור פסיקות: חבר את ההפרעה המוגדרת לפין GPIO באמצעות הפונקציה attachInterrupt() המסופקת על ידי מסגרת ESP32 Arduino.

יישום שגרתי של שירות הפסקת שירות: כתוב את קוד ISR לביצוע פעולות ספציפיות כאשר הכפתור נלחץ או שחרור. שמור על קוד ISR תמציתי ולא חוסם כדי למזער את זמן השהיית ההפרעות.

```
#define BUTTON_PIN 2
volatile bool buttonPressed = false;
void IRAM_ATTR buttonISR() {
  buttonPressed = true;
}
void setup() {
  pinMode(BUTTON_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN), buttonISR, FALLING);
}
void loop() {
  if (buttonPressed) {
    // Handle button press event
    // Perform necessary actions
    buttonPressed = false; // Reset flag
  }
}
```

הפונקציה buttonISR() מוגדרת כ-ISR לטיפול בלחיצות כפתורים.

הפונקציה setup() מגדירה את פין ה-GPIO המחובר ללחצן הלחיצה ככניסה עם נגד משוך מובנה ומחברת את פונקציית buttonISR() להדק הקצה הנופל של הכפתור.

בפונקציית loop(), הדגל buttonPressed מסומן, ואם מוגדר, מטופל באירוע הלחיצה על הכפתור והדגל מאופס.

ניסוי 1. הפעלת לחצן בעליה.

```
#define BUTTON_PIN_1 2
void IRAM_ATTR button1ISR() {
  // ISR for Button 1 press
}
void setup() {
  pinMode(BUTTON_PIN_1, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_1), button1ISR, RISING);
}
void loop() {
  // Main program loop
}
```

ניסוי 2. הפעלת לחצן בירידה.

```
#define BUTTON_PIN_1 2
void IRAM_ATTR button1ISR() {
  // ISR for Button 1 release
}
```

```
void setup() {
  pinMode(BUTTON_PIN_1, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_1), button1ISR, FALLING);
}
void loop() {
  // Main program loop
}
```

ניסוי 3. הפעלת לחצן בשינוי.

```
#define BUTTON_PIN_1 2
void IRAM_ATTR button1ISR() {
  // ISR for Button 1 state change
}
void setup() {
  pinMode(BUTTON_PIN_1, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_1), button1ISR, CHANGE);
}
void loop() {
  // Main program loop
}
```

ניסוי 4. הפעלת 2 לחצנים בעליה.

```
#define BUTTON_PIN_1 2
#define BUTTON_PIN_2 3
void IRAM_ATTR button1ISR() {
  // ISR for Button 1 press
}
void IRAM_ATTR button2ISR() {
  // ISR for Button 2 press
}
void setup() {
  pinMode(BUTTON_PIN_1, INPUT_PULLUP);
  pinMode(BUTTON_PIN_2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_1), button1ISR, RISING);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_2), button2ISR, RISING);
}
void loop() {
  // Main program loop
}
```

ניסוי 5. הפעלת 2 לחצנים בירידה.

```
#define BUTTON_PIN_1 2
#define BUTTON_PIN_2 3
void IRAM_ATTR button1ISR() {
  // ISR for Button 1 release
}
void IRAM_ATTR button2ISR() {
  // ISR for Button 2 release
}
void setup() {
  pinMode(BUTTON_PIN_1, INPUT_PULLUP);
  pinMode(BUTTON_PIN_2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_1), button1ISR, FALLING);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_2), button2ISR, FALLING);
```

```
}
void loop() {
  // Main program loop
}
```

ניסוי 6. הפעלת אחד משני לחצנים בעליה.

```
#define BUTTON_PIN_1 2
#define BUTTON_PIN_2 3
void IRAM_ATTR buttonISR() {
  // ISR for Button press
}
void setup() {
  pinMode(BUTTON_PIN_1, INPUT_PULLUP);
  pinMode(BUTTON_PIN_2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_1), buttonISR, RISING);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_2), buttonISR, RISING);
}
void loop() {
  // Main program loop
}
```

ניסוי 7. הפעלת אחד משני לחצנים בירידה.

```
#define BUTTON_PIN_1 2
#define BUTTON_PIN_2 3
void IRAM_ATTR buttonISR() {
  // ISR for Button release
}
void setup() {
  pinMode(BUTTON_PIN_1, INPUT_PULLUP);
  pinMode(BUTTON_PIN_2, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_1), buttonISR, FALLING);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_2), buttonISR, FALLING);
}
void loop() {
  // Main program loop
}
```

ניסוי 8. כפתור 1 מעלה מונה ב-1 ושני מוריד ב-1.

```
#define BUTTON_PIN_INC 2
#define BUTTON_PIN_DEC 3
volatile int counter = 0;
void IRAM_ATTR buttonIncISR() {
  counter++;
}
void IRAM_ATTR buttonDecISR() {
  counter--;
}
void setup() {
  pinMode(BUTTON_PIN_INC, INPUT_PULLUP);
  pinMode(BUTTON_PIN_DEC, INPUT_PULLUP);
```

```
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_INC), buttonIncISR, FALLING);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_DEC), buttonDecISR, FALLING);
  Serial.begin(115200);
}
void loop() {
  Serial.println(counter);
  delay(1000); // Delay for better readability, adjust as needed
}
```

ניסוי 9. כפתור 1 מעלה מונה ב-1 ושני מוריד ב-1 בהתאם לערכי קיצון.

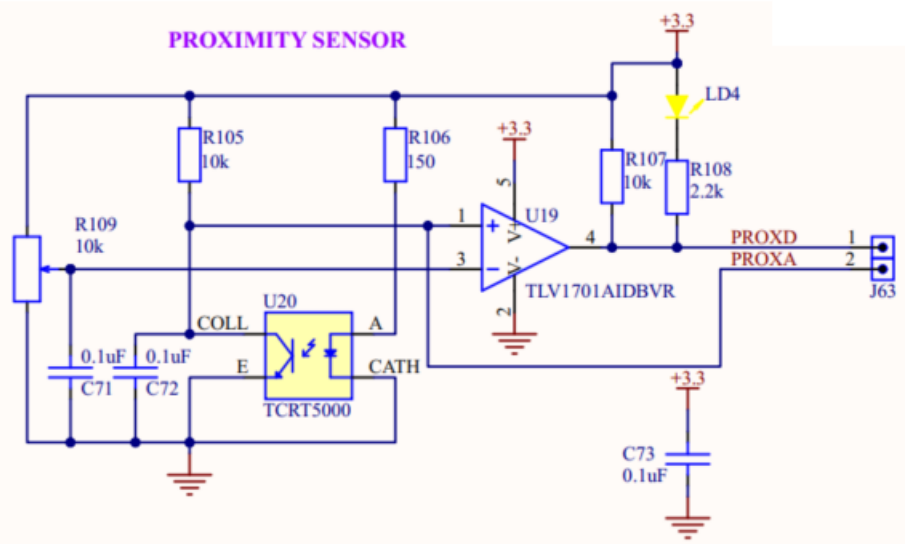
```
#define BUTTON_PIN_INC 2
#define BUTTON_PIN_DEC 3
volatile int counter = 0;
const int MAX_COUNT = 10;
const int MIN_COUNT = 0;
void IRAM_ATTR buttonIncISR() {
  if (counter < MAX_COUNT) {
    counter++;
  }
}
void IRAM_ATTR buttonDecISR() {
  if (counter > MIN_COUNT) {
    counter--;
  }
}
void setup() {
  pinMode(BUTTON_PIN_INC, INPUT_PULLUP);
  pinMode(BUTTON_PIN_DEC, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_INC), buttonIncISR, FALLING);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_DEC), buttonDecISR, FALLING);
  Serial.begin(115200);
}
void loop() {
  Serial.println(counter);
  delay(1000); // Delay for better readability, adjust as needed
}
```

ניסוי 10. הדלקת לד מונה חיובי, מונה מופעל ע"י פסיקות.

```
#define BUTTON_PIN_INC 2
#define BUTTON_PIN_DEC 3
#define LED_PIN 13
volatile int counter = 0;
void IRAM_ATTR buttonIncISR() {
  counter++;
}
void IRAM_ATTR buttonDecISR() {
  counter--;
}
void setup() {
  pinMode(BUTTON_PIN_INC, INPUT_PULLUP);
  pinMode(BUTTON_PIN_DEC, INPUT_PULLUP);
  pinMode(LED_PIN, OUTPUT);
```

```
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_INC), buttonIncISR, FALLING);
  attachInterrupt(digitalPinToInterrupt(BUTTON_PIN_DEC), buttonDecISR, FALLING);
  Serial.begin(115200);
}
void loop() {
  // Toggle LED based on counter value
  digitalWrite(LED_PIN, counter > 0 ? HIGH : LOW);
  Serial.println(counter);
  delay(1000); // Delay for better readability, adjust as needed
}
```

קליטת מצב של זוג אופטי ע"י פסיקות.



צמדים אופטיים, הידועים גם בתור זוג אופטי, הם רכיבים אלקטרוניים המשמשים להעברת אותות חשמליים בין שני מעגלים מבודדים באמצעות אור. הם מורכבים מדיודה פולטת אור (LED) בצד אחד ופוטו-גלאי (כגון פוטוטרנזיסטור או פוטודיודה) בצד השני, מופרדים על ידי מחסום אופטי שקוף. כאשר מופעל אות חשמלי על צד ה-LED, הוא פולט אור, אשר לאחר מכן מזוהה על ידי הפוטו-גלאי בצד השני, וכך מעביר את האות ללא כל חיבור חשמלי ישיר.

זוג אופטי עם פסיקות:

זוגות אופטיים משמשים בדרך כלל ביישומים שבהם נדרש בידוד חשמלי כדי להגן על רכיבים או מעגלים רגישים מפני מתחים גבוהים או רעשים. יישום אחד כזה הוא שימוש במצמדים אופטיים עם פסיקות במערכות מבוססות מיקרו-בקר. בהגדרה זו, המצמד האופטי עוזר לבודד את המיקרו-בקר מאותות או התקנים חיצוניים, ומאפשר לו להגיב בבטחה להפרעות מבלי להסתכן בפגיעה במעגלים הפנימיים שלו.

יישום קוד:

הגדר את פין ה-GPIO של המיקרו-בקר המחובר ליציאת זוג אופטי ככניסה עם נגד משיכה פנימי או חיצוני, בהתאם לקוטביות המוצא של זוג אופטי.

רשום שגרת שירות פסיקה (ISR) כדי לטפל בפסיקה המופעלת על ידי אות המוצא של זוג אופטי.

ב-ISR, בצע את הפעולות הדרושות על סמך אירוע הפסיקה, כגון עדכון משתנים, הגדרת דגלים או הפעלת פונקציות אחרות.

טיפול בפסיקות:

עם קבלת פסיקה זוג אופטי, המיקרו-בקר קופץ מיד ל-ISR כדי לטפל באירוע ההפסקה.

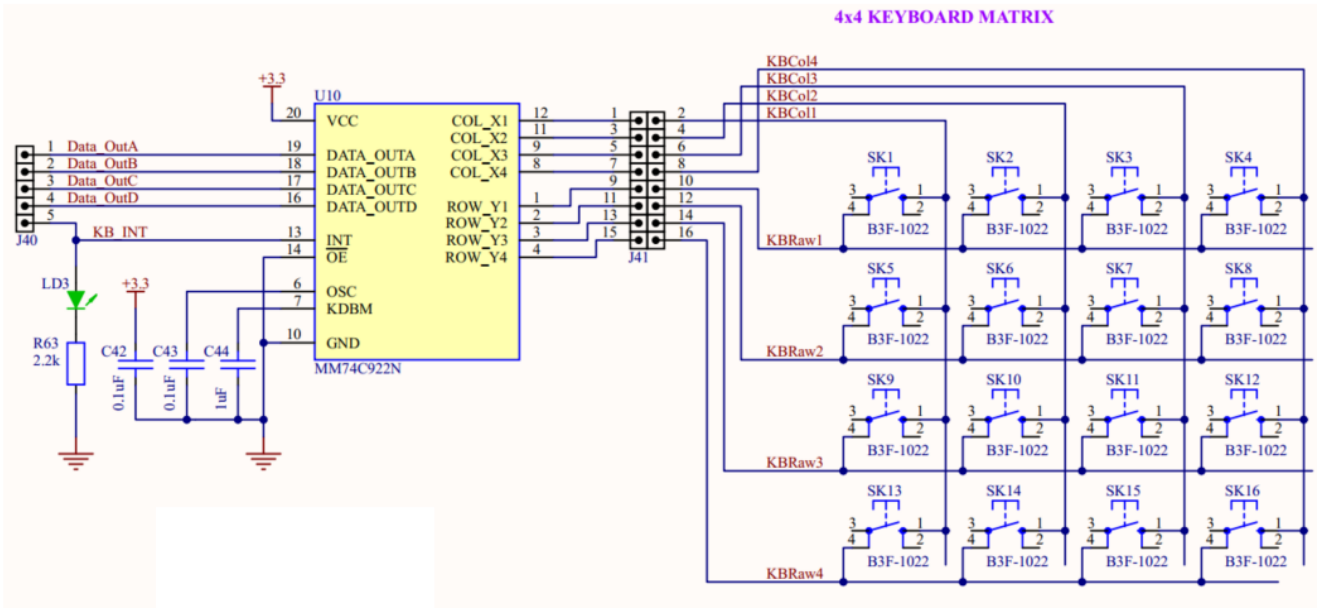
בתוך ה-ISR, המיקרו-בקר יכול לבצע פעולות או משימות ספציפיות הקשורות לאירוע ההפסקה, ולספק תגובה מהירה לגירויים חיצוניים.

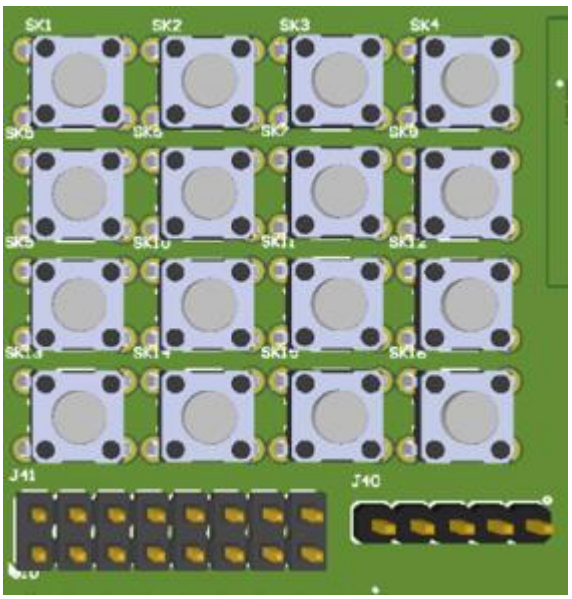
```
#define INTERRUPT_PIN 2 // Replace with actual GPIO pin number
volatile bool interruptFlag = false;

void IRAM_ATTR optoInterrupt() {
  interruptFlag = true;
}

void setup() {
  pinMode(INTERRUPT_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), optoInterrupt, FALLING);

  Serial.begin(115200);
}

void loop() {
  if (interruptFlag) {
    // Handle interrupt event
    Serial.println("Interrupt received!");
    interruptFlag = false; // Reset flag
  }
  // Main program loop
}
```

הפעלת מונה עם כל מעבר יד מעל זוג אופטי.

```
#define INTERRUPT_PIN 2 // Replace with actual GPIO pin number
volatile int counter = 0;
void IRAM_ATTR optoInterrupt() {
  counter++;
}
void setup() {
  pinMode(INTERRUPT_PIN, INPUT_PULLUP);
  attachInterrupt(digitalPinToInterrupt(INTERRUPT_PIN), optoInterrupt, FALLING);
  Serial.begin(115200);
}
void loop() {
  // Print counter value
  Serial.println(counter);
  delay(1000); // Delay for better readability, adjust as needed
}
```

קליטת מצב של לוח מקשים בעזרת פסיקה.





גרסה ראשונה עם גרסה חיצונית.

```
#include <Keypad.h>
#define ROWS 4
#define COLS 4
char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};

byte rowPins[ROWS] = {2, 3, 4, 5}; // Replace these with your actual row pin numbers
byte colPins[COLS] = {6, 7, 8, 9}; // Replace these with your actual column pin numbers
volatile bool keyPressed = false;
char key = '\0';
void IRAM_ATTR keypadISR() {
  keyPressed = true;
}
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
```

```
void setup() {
  pinMode(10, INPUT_PULLUP); // Interrupt pin connected to GPIO pin 10
  attachInterrupt(digitalPinToInterrupt(10), keypadISR, FALLING);
  Serial.begin(115200);
}
void loop() {
  if (keyPressed) {
    key = keypad.getKey();
    if (key != NO_KEY) {
      Serial.println(key);
    }
    keyPressed = false;
  }
}
```

גרסה אחרת עם פסיקה חיצונית.

```
#include <Keypad.h>
#define ROWS 4
#define COLS 4
char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {2, 3, 4, 5}; // Replace with your actual row pin numbers
byte colPins[COLS] = {6, 7, 8, 9}; // Replace with your actual column pin numbers
volatile bool dataAvailable = false;
byte keypadData = 0;
void IRAM_ATTR keypadISR() {
  dataAvailable = true;
}
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
void setup() {
  pinMode(10, INPUT_PULLUP); // External interrupt pin connected to GPIO pin 10
  attachInterrupt(digitalPinToInterrupt(10), keypadISR, FALLING);
  Serial.begin(115200);
}
void loop() {
  if (dataAvailable) {
    // Read 4-bit input from decoder
    keypadData = 0;
    for (int i = 0; i < 4; i++) {
      if (digitalRead(colPins[i]) == HIGH) {
        keypadData |= (1 << i);
      }
    }
    // Decode key pressed
    char key = '\0';
    switch (keypadData) {
      case 0x01: key = '1'; break;
      case 0x02: key = '2'; break;
      case 0x04: key = '3'; break;
      case 0x08: key = 'A'; break;
      case 0x10: key = '4'; break;
      case 0x20: key = '5'; break;
      case 0x40: key = '6'; break;
      case 0x80: key = 'B'; break;
```

```
      case 0x11: key = '7'; break;
      case 0x22: key = '8'; break;
      case 0x44: key = '9'; break;
      case 0x88: key = 'C'; break;
      case 0x21: key = '*'; break;
      case 0x42: key = '0'; break;
      case 0x84: key = '#'; break;
      case 0x18: key = 'D'; break;
    }
    // Output key pressed
    if (key != '\0') {
      Serial.println(key);
    }
    // Reset flag
    dataAvailable = false;
  }
}
```

גרסה עם 4 סיביות ופסיקה.

```
#include <Keypad.h>
#define ROWS 4
#define COLS 4
char keys[ROWS][COLS] = {
  {'1', '2', '3', 'A'},
  {'4', '5', '6', 'B'},
  {'7', '8', '9', 'C'},
  {'*', '0', '#', 'D'}
};
byte rowPins[ROWS] = {2, 3, 4, 5}; // Replace with your actual row pin numbers
byte colPins[COLS] = {6, 7, 8, 9}; // Replace with your actual column pin numbers
byte dataPins[4] = {10, 11, 12, 13}; // Replace with your actual data pin numbers
byte interruptPin = 14; // Replace with your actual interrupt pin number
volatile bool dataAvailable = false;
byte keypadData = 0;
void IRAM_ATTR keypadISR() {
  dataAvailable = true;
}
Keypad keypad = Keypad(makeKeymap(keys), rowPins, colPins, ROWS, COLS);
void setup() {
  for (int i = 0; i < 4; i++) {
    pinMode(dataPins[i], OUTPUT);
    digitalWrite(dataPins[i], LOW);
  }
  pinMode(interruptPin, INPUT_PULLUP); // External interrupt pin connected to GPIO pin 14
  attachInterrupt(digitalPinToInterrupt(interruptPin), keypadISR, FALLING);
  Serial.begin(115200);
}
void loop() {
  if (dataAvailable) {
    // Read 4-bit input from decoder
    keypadData = 0;
    for (int i = 0; i < 4; i++) {
      if (digitalRead(dataPins[i]) == HIGH) {
        keypadData |= (1 << i);
      }
    }
    // Decode key pressed
    char key = '\0';
```

```
    switch (keypadData) {
      case 0x01: key = '1'; break;
      case 0x02: key = '2'; break;
      case 0x04: key = '3'; break;
      case 0x08: key = 'A'; break;
      case 0x10: key = '4'; break;
      case 0x20: key = '5'; break;
      case 0x40: key = '6'; break;
      case 0x80: key = 'B'; break;
      case 0x11: key = '7'; break;
      case 0x22: key = '8'; break;
      case 0x44: key = '9'; break;
      case 0x88: key = 'C'; break;
      case 0x21: key = '*'; break;
      case 0x42: key = '0'; break;
      case 0x84: key = '#'; break;
      case 0x18: key = 'D'; break;
    }
    // Output key pressed
    if (key != '\0') {
      Serial.println(key);
    }
    // Reset flag
    dataAvailable = false;
  }
}
```

תרגול בנושא פסיקות עם לוח מקשים:

שנה קוד הבא עם שימוש בפסיקה דרך רכיב 74C922.

תרגיל מס' 1. סיסמת נעילת מקלדת

```
#include <Keypad.h>
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

#define ROWS  4
#define COLS  4

char keyMap[ROWS][COLS] = {
  {'1','2','3', 'A'},
  {'4','5','6', 'B'},
  {'7','8','9', 'C'},
  {'*','0','#', 'D'}
};

uint8_t rowPins[ROWS] = {14, 27, 26, 25}; // GIOP14, GIOP27, GIOP26, GIOP25
uint8_t colPins[COLS] = {33, 32, 18, 19}; // GIOP33, GIOP32, GIOP18, GIOP19
uint8_t LCD_CursorPosition = 0;
String PassWord = "134679";
String InputStr = "";

Keypad keypad = Keypad(makeKeymap(keyMap), rowPins, colPins, ROWS, COLS );
LiquidCrystal_I2C I2C_LCD(0x27, 16, 2);  // set the LCD address to 0x27 for a 16 chars and 2 line display

void setup(){
  Serial.begin(115200);
  // Initialize The I2C LCD
```

```
  I2C_LCD.begin();
  // Turn ON The Backlight
  I2C_LCD.backlight();
  // Clear The Display
  I2C_LCD.clear();
  I2C_LCD.setCursor(0, 0);
  I2C_LCD.print("Enter PassWord:");
}

void loop(){

  char key = keypad.getKey();

  if (key) {
    InputStr += key;
    I2C_LCD.setCursor(LCD_CursorPosition++, 1);
    if(LCD_CursorPosition == 6)
    {
      String message;
      if(InputStr == PassWord) {
        message = "Access Granted!";
      }
      else {
        message = "Wrong PassWord!";
      }
      InputStr = "";
      I2C_LCD.clear();
      LCD_CursorPosition = 0;
      I2C_LCD.print(message);
    }
    else if(key == 'D')
    {
      InputStr = "";
      I2C_LCD.clear();
      LCD_CursorPosition = 0;
      I2C_LCD.print("Enter PassWord:");
    }
    else
    {
      I2C_LCD.print(key);
    }
  }
}
```

תרגיל מס' 2. פסיקת לחצן:

חבר מתג לחיצה לפין GPIO של ה-ESP32.

הגדר את פין GPIO כדי להפעיל פסיקה בקצה העולה של האות.

השתמש בפסיקה כדי לשנות את המצב של נורית LED.

תרגיל מס' 3. פסיקת חיישן PIR:

חבר חיישן תנועה אינפרא אדום פסיבי (PIR) לפין GPIO של ה-ESP32.

הגדר את פין GPIO כדי להפעיל פסיקה כאשר מזוהה תנועה.

השתמש בפסיקה כדי לשלוח הודעה (למשל, להדפיס הודעה לצג טורי) או להפעיל אזעקה.

תרגיל מס' 4. פסיקת מקודד רוטרי:

חבר מקודד סיבובי לשני פיני GPIO של ה-ESP32 (אחד עבור כל פלט מקודד).

הגדר את פיני ה-GPIO כך שיפעילו פסיקות הן בקצוות העולים והן בירידה של אותות המקודד.

השתמש בפסיקות כדי לעקוב אחר כיוון הסיבוב והספירה של המקודד.

תרגיל מס' 5. פסיקת חיישן אפקט הול (צריכים לקנות בנפרד).

חבר חיישן אפקט הול לפין GPIO של ה-ESP32.

הגדר את פין GPIO כדי להפעיל פסיקה כאשר מזוהה שדה מגנטי.

השתמש בפסיקה כדי למדוד את המהירות או ספירת הסיבובים של עצם ממוגנט.

תרגיל מס' 6. פסיקת חיישן מגע (צריכים לקנות בנפרד).

חבר חיישן מגע קיבולי לפין GPIO של ה-ESP32.

הגדר את סיכת ה-GPIO כדי להפעיל פסיקה בעת נגיעה.

השתמש בפסיקה כדי לשלוט בפלט (למשל, להדליק נורית) או להפעיל פעולה (למשל, לשלוח הודעה).

תרגיל מס' 7. פסיקת חיישן אור.

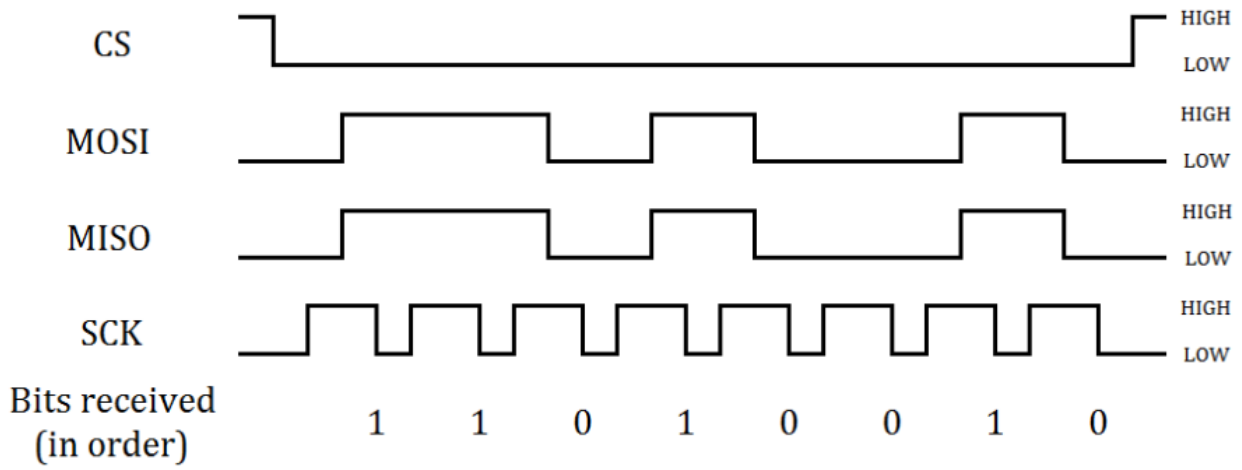
חבר נגד תלוי אור (LDR) לפין כניסה אנלוגי של ה-ESP32.

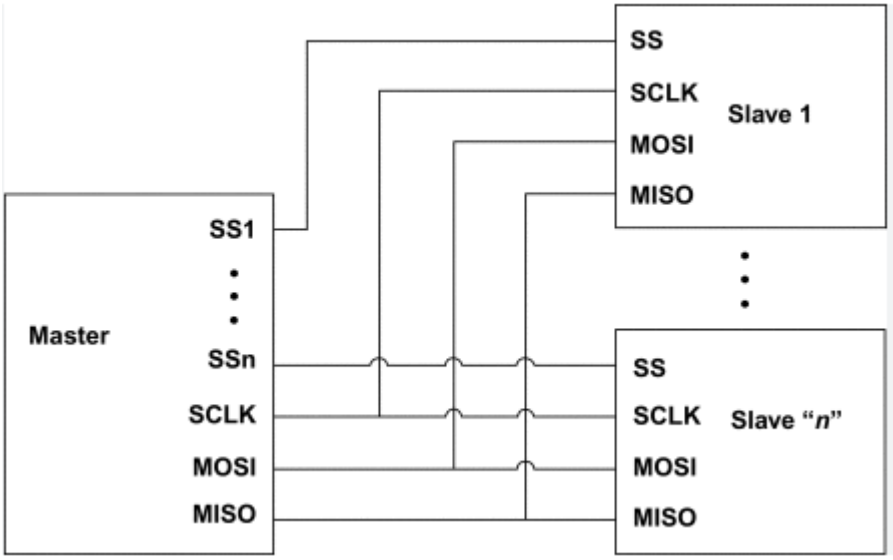
הגדר את ה-ADC כדי להפעיל פסיקה כאשר רמת האור חוצה סף מסוים.

השתמש בפסיקה כדי לשלוט בפלט (למשל, הדלקת LED) על סמך תנאי האור הסביבתי.

פרק 4. פרוטוקול תקשורת SPI.

פרוטוקול Serial Peripheral Interface (SPI) הוא פרוטוקול תקשורת טורית סינכרוני המשמש לתקשורת בין מיקרו-בקרים והתקנים היקפיים. הוא מאפשר תקשורת במהירות גבוהה, דופלקס מלא בין התקן מאסטר (כגון המיקרו-בקר ESP32) לבין רכיב "עבד" (נשלט) אחד או יותר. בהסבר מפורט זה, נפרק את פרוטוקול SPI ביט אחרי ביט ונבדוק כיצד הוא מיושם במיקרו-בקר ESP32.





1. תקשורת מאסטר-עבד:

תקשורת SPI כוללת רכיב מאסטר אחד (ESP32) ורכיב "עבד" אחד או יותר. התקן המאסטר שולט בתקשורת על ידי הפקת פולסי שעון ובחירת מכשירי העבד איתם הוא רוצה לתקשר.

2. שעון (SCLK):

תקשורת SPI היא סינכרונית, כלומר נתונים מועברים בין מכשירים באופן סינכרוני עם אות שעון (SCLK). ההתקן הראשי מייצר פעימות שעון על קו SCLK כדי לסנכרן את העברת הנתונים.

3. קווי נתונים (MOSI/MISO):

SPI משתמש בשני קווי נתונים לתקשורת:

Master Out Slave In (MOSI): ההתקן הראשי שולח נתונים למכשיר/ים העבדים בקו זה.

Master In Slave Out (MISO): מכשירי העבד שולחים נתונים למכשיר הראשי בקו זה.

4. בחירת שבב (CS):

לכל רכיב עבד המחובר לאפיק SPI יש קו ייעודי לבחירת שבב (CS). ההתקן הראשי בוחר התקן עבד ספציפי לתקשורת על ידי משיכת קו ה-CS שלו נמוך.

5. העברת נתונים (דופלקס מלא):

תקשורת SPI היא דופלקס מלא, כלומר ניתן להעביר ולהתקבל נתונים בו-זמנית בין המכשירים הראשיים והעבדים. התקן המאסטר שולח נתונים על קו MOSI ובו זמנית מקבל נתונים על קו MISO.

6. סדר סיביות וקוטביות/שלב שעון:

פרמטרי תקשורת SPI, כגון סדר הסיביות (MSB-first או LSB-first) וקוטביות/פאזה שעון, יכולים להשתנות בהתאם לתצורה הספציפית של אפיק ה-SPI. פרמטרים אלה ניתנים בדרך כלל להגדרה וחייבים להיות עקביים בין התקני המאסטר והעבד לתקשורת מוצלחת.

יישום ב-ESP32:

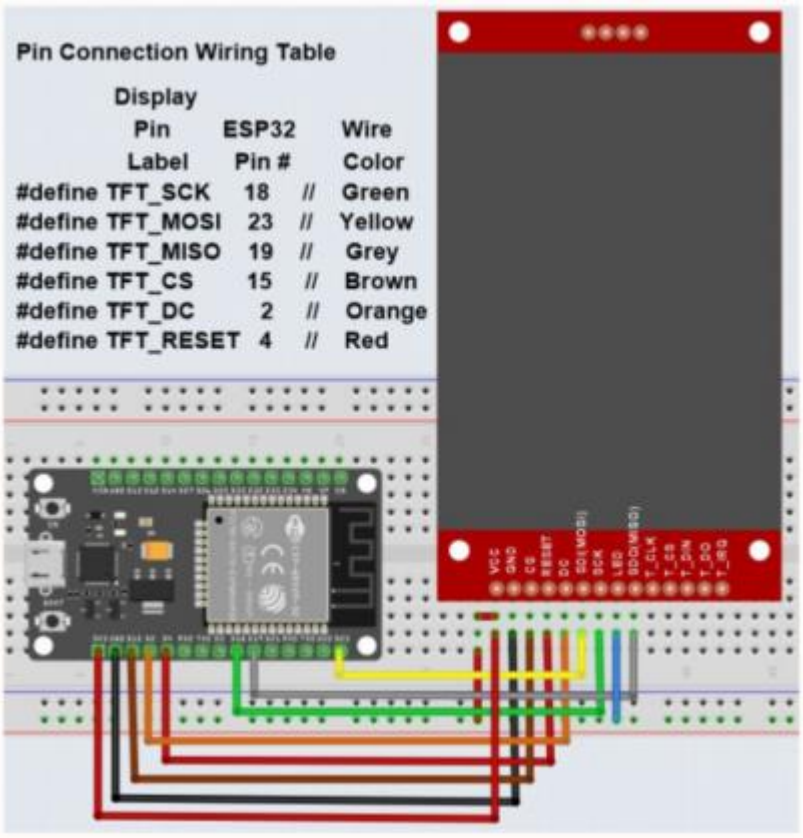
המיקרו-בקר ESP32 כולל תמיכה מובנית בפרוטוקול SPI, מה שמקל על היישום בפרויקטים של ESP32. על ידי הכללת ספריית SPI ושימוש בפונקציות שלה, מפתחים יכולים לאתחל תקשורת SPI, להגדיר פרמטרים של תקשורת ולהעביר נתונים בין המכשירים ההיקפיים של ESP32 ו-SPI.

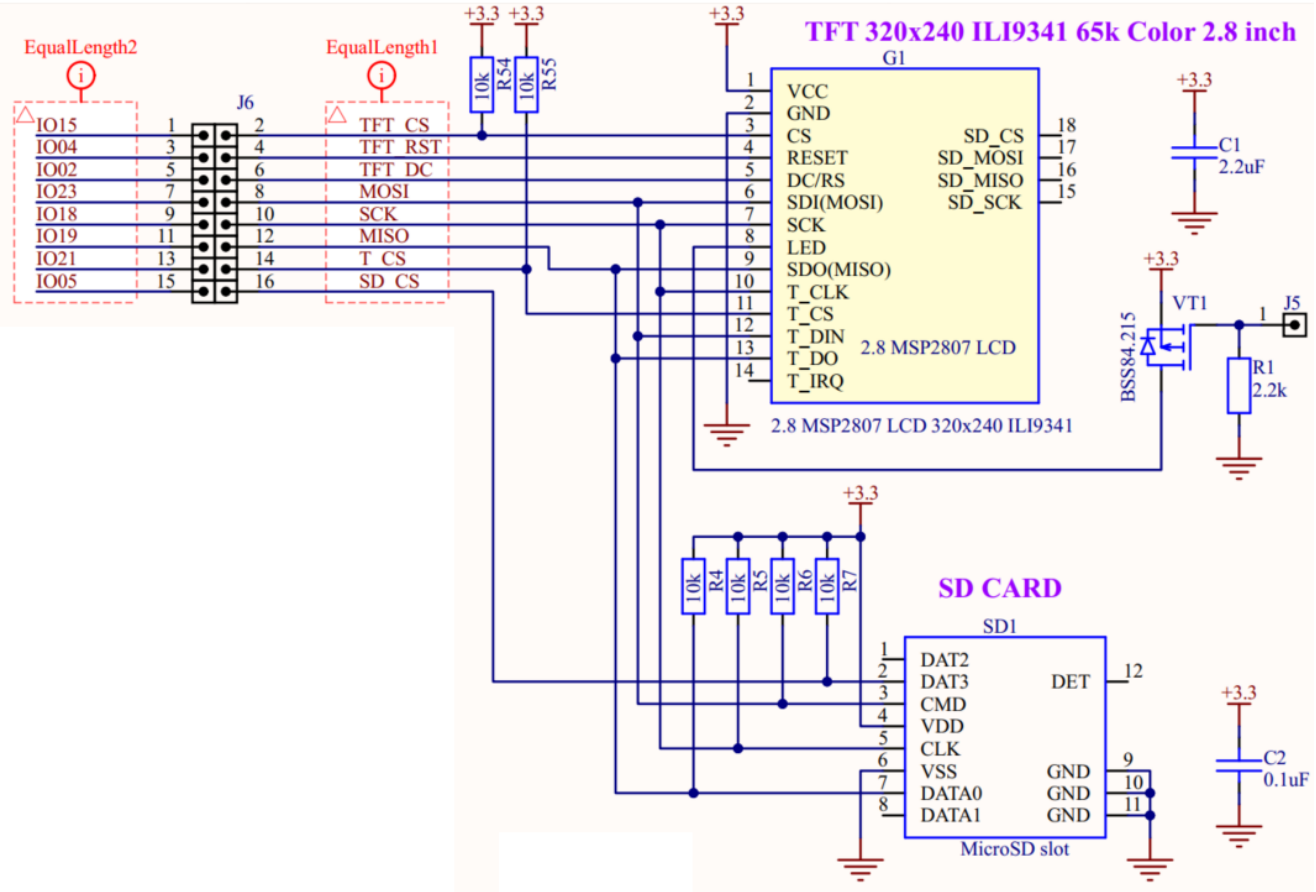
לסיכום:

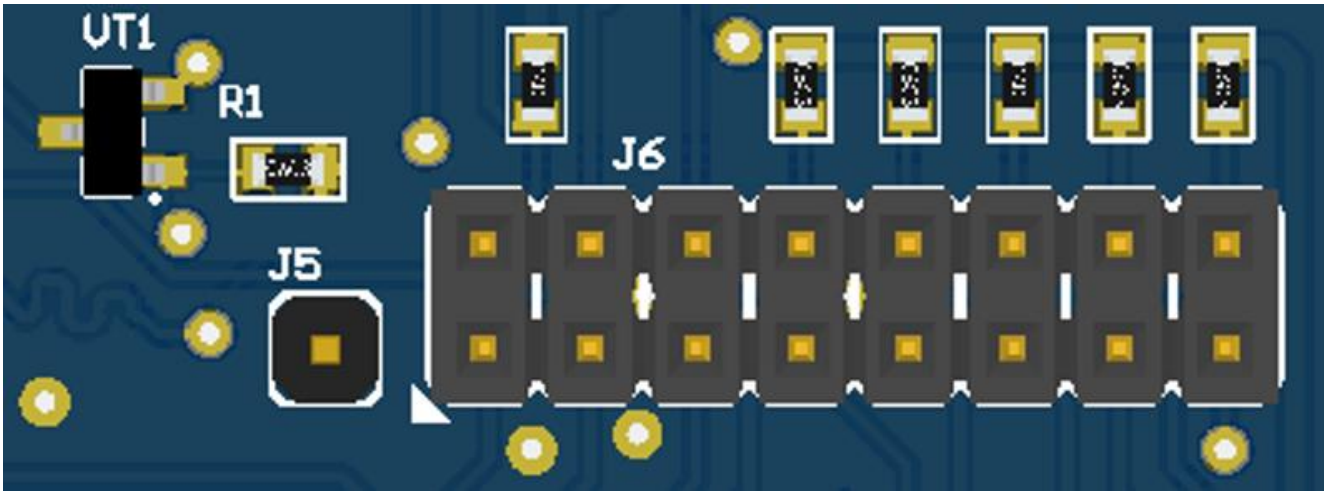
פרוטוקול SPI הוא פרוטוקול תקשורת טורית רב תכליתי ורב שימוש המתאים לתקשורת במהירות גבוהה, דופלקס מלא בין מיקרו-בקרים והתקנים היקפיים. על ידי הבנת פעולתו ביט אחר ביט והטמעתה בפרויקטים של ESP32, מפתחים יכולים למנף את הגמישות והביצועים של פרוטוקול SPI עבור מגוון רחב של יישומים.

(למידע מלא https://ww1.microchip.com/downloads/en/devicedoc/spi.pdf).

הפעלת מסך TFT.







צגי TFT (Thin-Film Transistor), יחד עם בקר ILI9341, מספקים פתרון תוסס ורב-תכליתי עבור ממשקים חזותיים במערכות משובצות. סעיף זה יעמיק בממשק של תצוגת TFT ILI9341 עם המיקרו-בקר ESP32 באמצעות פרוטוקול התקשורת SPI (Serial Peripheral Interface).

צג TFT ILI9341:

תצוגת TFT ILI9341 היא בחירה פופולרית בשל הרזולוציה הגבוהה, עומק הצבע וקצב הרענון המהיר שלה. הוא כולל רזולוציה של x320240 פיקסלים ויכול להציג עד 262,144 צבעים. התצוגה מונעת על ידי בקר ILI9341, התומך בתכונות שונות כגון גלילת חומרה, תיקון גמא ומצב שינה.

ממשק TFT ILI9341 עם ESP32:

כדי לממשק את צג TFT ILI9341 עם המיקרו-בקר ESP32 באמצעות SPI, בצע את השלבים הבאים:

התקנת ספרייה:

התקן את ספריית TFT ILI9341 עבור ESP32, המספקת פונקציות לשליטה בתצוגה.

ספרייה זו כוללת בדרך כלל פונקציות לאתחול התצוגה, הגדרת פיקסלים, ציור צורות והצגת טקסט.

אתחול:

אתחל את ממשק התקשורת SPI ב-ESP32.

אתחול תצוגת ה-TFT באמצעות פרמטרי האתחול המתאימים (למשל, רזולוציית תצוגה, סיבוב).

בקרת תצוגה:

השתמש בפונקציות ספרייה כדי לשלוט בתצוגת ה-TFT, כגון ציור צורות (קווים, מלבנים, עיגולים), הצגת תמונות ועיבוד טקסט.

השתמש בעסקאות SPI להעברת נתונים יעילה לתצוגה.

נראה את הקוד הבסיסי להפעלת מסך TFT.

```
#include <SPI.h>
#include <TFT_ILI9341_ESP.h>
#define TFT_CS   5
#define TFT_DC   15
#define TFT_RST  -1  // Set to -1 if not used, otherwise connect to a GPIO pin
```

```
TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);  // Adjust rotation if necessary
  tft.fillScreen(ILI9341_BLACK);
}
void loop() {
  // Display your content here
}
```

הצגת הודעה על המסך:

```
#include <SPI.h>
#include <TFT_ILI9341_ESP.h>
#define TFT_CS    5
#define TFT_DC    15
#define TFT_RST   -1
TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
  tft.setCursor(10, 10);
  tft.setTextSize(2);
  tft.setTextColor(ILI9341_WHITE);
  tft.println("Hello, World!");
}

void loop() {
  // Your code here
}
```

הצגת צורות הנדסיות על המסך.

```
#include <SPI.h>
#include <TFT_ILI9341_ESP.h>
#define TFT_CS    5
#define TFT_DC    15
#define TFT_RST   -1
TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
  tft.fillRect(10, 10, 50, 50, ILI9341_RED);
  tft.fillCircle(100, 100, 30, ILI9341_GREEN);
}
void loop() {
  // Your code here
}
```

הצגת תמונות על המסך.

```
#include <SPI.h>
#include <TFT_ILI9341_ESP.h>
#include "image.h" // Include image data
#define TFT_CS   5
#define TFT_DC   15
#define TFT_RST  -1
TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
  tft.drawBitmap(0, 0, image_data, 240, 320, ILI9341_WHITE);
}
void loop() {
  // Your code here
}
```

הצגת גל סינוס על המסך.

```
#include <SPI.h>
#include <TFT_ILI9341_ESP.h>
#include <math.h>
#define TFT_CS   5
#define TFT_DC   15
#define TFT_RST  -1
TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
}
void loop() {
  static int xOffset = 0;
  float y = 160 + 100 * sin(xOffset * 0.1);
  tft.drawPixel(xOffset, y, ILI9341_WHITE);
  xOffset = (xOffset + 1) % 240;
  delay(50);
}
```

הגדרת פונט חדש להצגה על המסך.

```
// Include necessary libraries and define TFT pins

TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
void setup() {
  // Initialize serial communication and TFT display
```

```
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
  tft.setFont(ArialMT_Plain_24); // Use custom font
  tft.setTextColor(ILI9341_WHITE);
  tft.drawString("Custom Font", 20, 50);
}
void loop() {
  // Your code here
}
```

הצגת גרף של מדידות.

```
// Include necessary libraries and define TFT pins
TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
void setup() {
  // Initialize serial communication and TFT display
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
}
void loop() {
  // Read sensor data and plot on the TFT display as a graph
  int sensorValue = analogRead(A0); // Example sensor reading
  tft.drawPixel(dataPointX, map(sensorValue, 0, 1023, 0, tft.height()), ILI9341_WHITE);
  dataPointX = (dataPointX + 1) % tft.width();
  delay(1000);
}
```

הצגת טקסט רץ על המסך.

```
// Include necessary libraries and define TFT pins
TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
void setup() {
  // Initialize serial communication and TFT display
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
}
void loop() {
  // Display scrolling text
  static int xPos = 0;
  tft.setCursor(xPos, 50);
  tft.setTextSize(2);
  tft.setTextColor(ILI9341_WHITE);
  tft.print("This is scrolling text!");
  xPos = (xPos + 1) % (tft.width() + 100); // Adjust scrolling speed
  delay(50);
}
```

ריכוז פקודות להפעלת מסך TFT:

הסבר	פונקציה
אתחול ה-LCD.	`begin()`
סיבוב מסך 0, 90, 180, 270 מעלות.	`setRotation(rotation)`
הכנסת למצב שינה	`sleep()`

wake()	מתעורר ממצב שינה.
fillScreen(color)	ממלא את כל המסך בצבע שצוין.
setCursor(x, y)	הגדר את מיקום הסמן לפי קואורדינטות שצוינו.
setTextColor(color)	הגדר צבע גופן
setTextSize(size)	הגדר גודל גופן
drawPixel(x, y, color)	מצייר פיקסל בקואורדינטות שצוינו עם הצבע שצוין.
drawLine(x0, y0, x1, y1, color)	מצייר קו מ-(x0, y0) ל-(x1, y1) עם הצבע שצוין.
drawRect(x, y, width, height, color)	מצייר מלבן עם הרוחב, הגובה והצבע שצוינו.
fillRect(x, y, width, height, color)	מצייר מלבן מלא עם הרוחב, הגובה והצבע שצוינו.
drawCircle(x0, y0, radius, color)	מצייר עיגול עם הרדיוס והצבע שצוינו.
fillCircle(x0, y0, radius, color)	מצייר עיגול מלא עם הרדיוס והצבע שצוינו.
drawTriangle(x0, y0, x1, y1, x2, y2, color)	מצייר משולש עם קואורדינטות והצבע שצוינו.
fillTriangle(x0, y0, x1, y1, x2, y2, color)	מצייר משולש מלא עם קואורדינטות והצבע שצוינו.
drawChar(x, y, char, color, bg, size)	מצייר תו במיקום שצוין עם הצבע וצבע הרקע שצוינו.
drawBitmap(x, y, bitmap, width, height, color)	מצייר תמונת מפת סיביות בקואורדינטות שצוינו עם הרוחב, הגובה והצבע שצוינו.
scrollTo(y)	גלגל את התצוגה לקואורדינטת ה-y שצוינה.
scroll(x, y)	גלגל בתצוגה לפי הזזה של x ו-y שצוינו.
touchRead()	קריאת קואורדינטות של מגע.
setFont(font)	הגדר גופן
readPixel(x, y)	קורא את צבע הפיקסל בקואורדינטות שצוינו.
drawRGBBitmap(x, y, bitmap, width, height)	מצייר תמונת מפת סיביות RGB בקואורדינטות שצוינו עם הרוחב והגובה שצוינו.

```
#include <Adafruit_ILI9341.h>

#define TFT_CS   10
#define TFT_RST   9
#define TFT_DC    8
#define TFT_MOSI  11
#define TFT_CLK   13
#define TFT_MISO  12

Adafruit_ILI9341 tft = Adafruit_ILI9341(TFT_CS, TFT_DC, TFT_RST);

void setup() {
  Serial.begin(9600);

  tft.begin();

  tft.setRotation(3); // Set display rotation (0-3)

  tft.fillScreen(ILI9341_BLACK); // Clear screen with black color

  // Display Control Functions
  tft.sleep(); // Enter sleep mode
  delay(1000);
  tft.wake(); // Wake up from sleep mode
  delay(1000);

  // Drawing Functions
  tft.fillScreen(ILI9341_WHITE); // Fill screen with white color
  delay(1000);
```

```
  tft.drawLine(0, 0, 100, 100, ILI9341_RED); // Draw a red line from (0,0) to (100,100)
  delay(1000);

  tft.drawRect(50, 50, 50, 50, ILI9341_GREEN); // Draw a green rectangle at (50,50) with width 50 and
height 50
  delay(1000);

  tft.drawCircle(100, 100, 30, ILI9341_BLUE); // Draw a blue circle at (100,100) with radius 30
  delay(1000);

  // Text Display Functions
  tft.setTextColor(ILI9341_YELLOW); // Set text color to yellow
  tft.setTextSize(2); // Set text size to 2

  tft.setCursor(10, 10); // Set cursor position
  tft.print("Hello, World!"); // Print a string
  delay(1000);

  // Display various types of variables
  int num = 123;
  float pi = 3.14;
  char ch = 'A';

  tft.setCursor(10, 40);
  tft.print("Number: ");
  tft.print(num);
  delay(1000);

  tft.setCursor(10, 60);
  tft.print("Pi: ");
  tft.print(pi);
  delay(1000);

  tft.setCursor(10, 80);
  tft.print("Char: ");
  tft.print(ch);
  delay(1000);

  // Clear screen
  tft.fillScreen(ILI9341_BLACK);
}

void loop() {
  // Your code here
}
```

הפעלת מנגנון קלט של המסך.

צגי TFT (Thin-Film Transistor) עם פונקציונליות מגע, כגון מסך ILI9341, מספקים ממשק משתמש אינטואיטיבי ליישומים אינטראקטיביים. בסעיף זה, נחקור כיצד להשתמש בפונקציונליות המגע במסך TFT ILI9341 עם המיקרו-בקר ESP32, כולל הסברים על עקרונות חישת מגע ודוגמאות קוד ליישום.

עקרון חישת מגע:

פונקציונליות המגע בצגי TFT מסתמכת בדרך כלל על חיישני מגע התנגדות או קיבולית. מסך ILI9341 משלב לעיתים קרובות חיישן מגע קיבולי, המזהה שינויים בקיבול כאשר חפץ מוליך, כמו אצבע, מתקרב למסך. שינוי זה בקיבול מתורגם לקואורדינטות מגע (X, Y) על ידי בקר המגע.

חיישן מגע ממשק עם ESP32:

כדי לממשק את חיישן המגע של מסך TFT ILI9341 עם המיקרו-בקר ESP32, בצע את השלבים הבאים:

התקנת ספרייה:

התקן את ספריית חיישני המגע המתאימה עבור ה-ESP32.

ספריות כגון XPT2046_Touchscreen משמשות בדרך כלל להתממשקות חיישני מגע עם ה-ESP32.

אתחול:

אתחל את חיישן המגע בפונקציית setup() של סקיצת ה-Arduino שלך.

כייל את חיישן המגע במידת הצורך כדי להבטיח זיהוי מגע מדויק.

זיהוי מגע:

השתמש בפונקציות הספרייה כדי לקרוא קואורדינטות מגע (X, Y) מחיישן המגע.

יישם היגיון כדי להגיב לאירועי מגע, כגון לחיצות על כפתורים, תנועות מחוון או בחירות תפריט.

דוגמה לקוד:

קוד להצגת טקסט מהקובץ:

```
#include <SPI.h>
#include <SD.h>
#include <TFT_ILI9341_ESP.h>

#define TFT_CS     5
#define TFT_DC     15
#define TFT_RST    -1
#define SD_CS      13

TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);

void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);

  if (!SD.begin(SD_CS)) {
    Serial.println("SD card initialization failed.");
    return;
  }

  File txtFile = SD.open("/text.txt");
  if (txtFile) {
    while (txtFile.available()) {
      tft.println(txtFile.readStringUntil('\n'));
    }
    txtFile.close();
  } else {
    Serial.println("Error opening text.txt");
  }
}
```

```
void loop() {
  // Your code here
}
```

הנה דוגמה פשוטה של קוד המדגימה את פונקציונליות המגע במסך TFT ILI9341 עם ה-ESP32 באמצעות ספריית מסך המגע XPT2046_:

```
#include <SPI.h>
#include <TFT_ILI9341_ESP.h>
#include <XPT2046_Touchscreen.h>

#define TFT_CS   5
#define TFT_DC   15
#define TFT_RST  -1
#define TOUCH_CS  21

TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
XPT2046_Touchscreen ts(TOUCH_CS);

void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
}

void loop() {
  TS_Point p = ts.getPoint();
  if (p.z > 0) {
    // Touch detected, get touch coordinates
    int touchX = map(p.x, 0, 4095, 0, tft.width());
    int touchY = map(p.y, 0, 4095, 0, tft.height());
    // Handle touch event, e.g., draw a circle at the touch position
    tft.fillCircle(touchX, touchY, 5, ILI9341_WHITE);
    delay(200); // Debounce delay
  }
}
```

עוד דוגמה להפעלת מנגנון מגע.

```
#include <SPI.h>
#include <TFT_ILI9341_ESP.h>
#include <XPT2046_Touchscreen.h>

#define TFT_CS   5
#define TFT_DC   15
#define TFT_RST  -1
#define TOUCH_CS  21

TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
XPT2046_Touchscreen ts(TOUCH_CS);
```

```
void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
}

void loop() {
  TS_Point p = ts.getPoint();
  if (p.z > 0) {
    tft.fillRect(50, 50, 100, 100, ILI9341_RED);
    delay(500); // Debounce delay
  }
}
```

בניית תפריט על המסך.

```
// Include necessary libraries and define TFT and touch pins
TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
XPT2046_Touchscreen ts(TOUCH_CS);
void setup() {
  // Initialize serial communication, TFT display, and touch sensor
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
  // Draw menu options
}

void loop() {
  // Check for touch input and handle menu navigation
  TS_Point p = ts.getPoint();
  if (p.z > 0) {
    // Determine which menu option was selected and take appropriate action
  }
}
```

ציור צורות הנדסיות בהתאם ללחיצה על המסך.

```
#include <SPI.h>
#include <TFT_ILI9341_ESP.h>
#include <XPT2046_Touchscreen.h>

#define TFT_CS     5
#define TFT_DC     15
#define TFT_RST    -1
#define TOUCH_CS   21

TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);
XPT2046_Touchscreen ts(TOUCH_CS);

void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
}
```

```
void loop() {
  TS_Point p = ts.getPoint();
  if (p.z > 0) {
    // Draw a shape at the touch coordinates
    int touchX = map(p.x, 0, 4095, 0, tft.width());
    int touchY = map(p.y, 0, 4095, 0, tft.height());
    tft.fillCircle(touchX, touchY, 10, ILI9341_WHITE); // Example: draw a circle
    delay(200); // Debounce delay
  }
}
```

הפעלת כרטיס SD.

בפרויקטים משובצים רבים, חיוני שיהיו יכולות אחסון עבור רישום נתונים, קבצי תצורה או אחסון נכסים עבור גרפיקה ותמונות. שימוש בכרטיס SD הוא פתרון נפוץ ונוח. בשילוב עם מסך TFT ILI9341 ומיקרו-בקר ESP32, כרטיס SD יכול לספק אחסון בשפע למטרות שונות. בסעיף זה, נחקור כיצד לשלב כרטיס SPI SD עם מסך TFT ILI9341 ב-ESP32, כולל הסברים על פונקציות, פרטי יישום ודוגמאות קוד.

פונקציונליות והסברים:

יסודות כרטיס SPI SD:

כרטיסי Secure Digital (SD) נמצאים בשימוש נרחב לאחסון נתונים במערכות משובצות בשל גודלם הקומפקטי ויכולת האחסון הגבוהה שלהם.

פרוטוקול SPI (Serial Peripheral Interface) משמש בדרך כלל לתקשורת עם כרטיסי SD.

מודול כרטיס SPI SD מספק בדרך כלל פונקציות לאתחול הכרטיס, קריאה וכתיבה של נתונים וניהול מערכות קבצים.

ממשק כרטיס SD עם ESP32:

חבר את מודול כרטיס SPI SD ל-ESP32, תוך הבטחת חיווט תקין עבור פיני MOSI, MISO, CLK ו-CS.

התקן את הספריות המתאימות לתקשורת כרטיסי SD ב-ESP32, כגון ספריית SD עבור Arduino.

אתחל את כרטיס ה-SD בפונקציית setup() של הסקיצה של Arduino באמצעות הפונקציה SD.begin() .

קריאה וכתיבה של נתונים:

השתמש בפונקציות ספריית SD כדי לקרוא ולכתוב קבצים בכרטיס ה-SD.

כדי לקרוא נתונים מקובץ, פתח את הקובץ באמצעות SD.open() והשתמש בפונקציות כמו available(), read() ו-seek() כדי לקרוא נתונים.

כדי לכתוב נתונים לקובץ, פתח את הקובץ במצב כתיבה באמצעות SD.open() והשתמש בפונקציות כמו write() ו-println() כדי לכתוב נתונים.

טעינת תמונות/גרפיקה:

אחסן נכסי גרפיקה כגון תמונות, סמלים או גופנים בכרטיס ה-SD.

טען נכסים גרפיים מכרטיס ה-SD לזיכרון המיקרו-בקר בעת הצורך לתצוגה על מסך ה-TFT.

נראה את הדוגמה:

```
#include <SPI.h>
#include <TFT_ILI9341_ESP.h>
#include <SD.h>

#define TFT_CS    5
#define TFT_DC    15
#define TFT_RST   -1
#define SD_CS     13

TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);

void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);
  tft.fillScreen(ILI9341_BLACK);
```

```
  if (!SD.begin(SD_CS)) {
    Serial.println("SD card initialization failed.");
    return;
  }

  File dataFile = SD.open("/data.txt");
  if (dataFile) {
    while (dataFile.available()) {
      tft.println(dataFile.readStringUntil('\n'));
    }
    dataFile.close();
  } else {
    Serial.println("Error opening data.txt");
  }
}

void loop() {
  // Your code here
}
```

עוד דוגמאות:

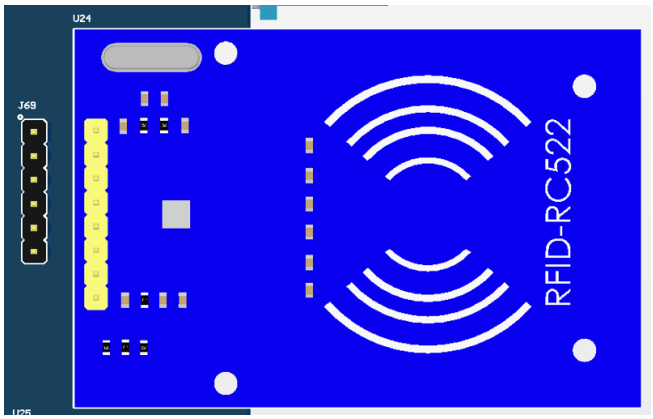
הצגת תמונה ששמורה על כרטיס SD.

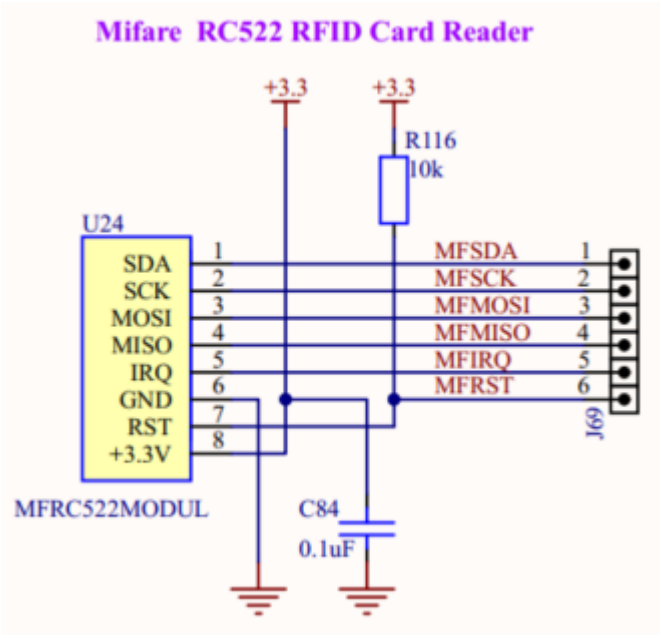
```
#include <SPI.h>
#include <TFT_ILI9341_ESP.h>
#include <SD.h>

#define TFT_CS     5
#define TFT_DC     15
#define TFT_RST    -1
#define SD_CS      13

TFT_ILI9341_ESP tft = TFT_ILI9341_ESP(TFT_CS, TFT_DC, TFT_RST);

void setup() {
  Serial.begin(115200);
  SPI.begin();
  tft.init();
  tft.setRotation(1);

  if (!SD.begin(SD_CS)) {
    Serial.println("SD card initialization failed.");
    return;
  }

  File imgFile = SD.open("/image.bmp");
  if (imgFile) {
    tft.drawBmp("/image.bmp", 0, 0);
    imgFile.close();
  } else {
    Serial.println("Error opening image.bmp");
  }
}

void loop() {
  // Your code here
}
```

הכנסת מדידות לתוך כרטיס SD.

```
#include <SPI.h>
#include <SD.h>

#define SD_CS      13

File dataFile;

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

  if (!SD.begin(SD_CS)) {
    Serial.println("SD card initialization failed.");
    return;
  }

  dataFile = SD.open("/data.txt", FILE_WRITE);
  if (dataFile) {
    dataFile.println("Timestamp, SensorValue");
    dataFile.close();
  } else {
    Serial.println("Error opening data.txt");
  }
}

void loop() {
  int sensorValue = analogRead(32);
  dataFile = SD.open("/data.txt", FILE_WRITE);
  if (dataFile) {
    dataFile.print(millis());
    dataFile.print(",");
    dataFile.println(sensorValue);
    dataFile.close();
  } else {
    Serial.println("Error opening data.txt");
  }
  delay(1000);
}
```

רכיב SPI RFID.





טכנולוגיית RFID (זיהוי תדר רדיו) מצאה יישום נרחב בתחומים שונים, מבקרת גישה וניהול מלאי ועד למערכות מעקב ואימות. מודול RC522 הוא מודול קורא וכותב RFID פופולרי הפועל בתדר 13.56 מגה-הרץ. במאמר זה נעמיק בארכיטקטורה, במבנה, בפונקציות וביישומים המעשיים של מודול RFID RC522. בנוסף, נספק דוגמאות קוד כדי להדגים את השימוש בו עם מיקרו-בקר ESP32.

מודול RFID RC522 מורכב ממספר מרכיבי מפתח:

קורא/כותב RFID: המרכיב העיקרי האחראי לתקשורת עם תגי RFID. הוא פולט גלי רדיו ומזהה תגובות מתגי RFID בטווח שלו.

אנטנה: האנטנה מאפשרת שידור וקליטה של אותות רדיו בין מודול RC522 ותגי RFID.

ממשק מיקרו-בקר: מודול RC522 מתממשק בדרך כלל עם מיקרו-בקר (למשל, ESP32) באמצעות פרוטוקול תקשורת SPI (Serial Peripheral Interface).

ממשק בקרה: פינים לאספקת חשמל (VCC ו-GND) ותקשורת SPI (MISO, MOSI, SCK ו-SS).

פונקציות ופעולות:

מודול RFID RC522 מאפשר את הפעולות הבאות:

זיהוי תגים: המודול יכול לזהות נוכחות של תגי RFID בטווח הפעולה שלו.

זיהוי תג: לאחר זיהוי תג, מודול RC522 יכול לקרוא את המזהה הייחודי (UID) שתוכנת בזיכרון התג.

החלפת נתונים: זה מאפשר קריאה וכתיבה של נתונים למגזרי זיכרון ספציפיים של תגי RFID תואמים.

אימות: תגי RFID מסוימים תומכים במנגנוני אימות כדי להבטיח גישה מאובטחת לנתונים שלהם. מודול RC522 יכול לבצע פעולות אימות בעת הצורך.

קוד לאתחול RFID.

```
#include <SPI.h>
#include <MFRC522.h>
#define SS_PIN 5
#define RST_PIN 9
```

```
MFRC522 mfrc522(SS_PIN, RST_PIN); // Create MFRC522 instance
void setup() {
  Serial.begin(115200); // Initialize serial communication
  SPI.begin(); // Initialize SPI bus
  mfrc522.PCD_Init(); // Initialize RC522
}
void loop() {
  // Your code here
}
```

קוד לסריקת תג RFID.

```
void loop() {
  // Look for new RFID cards
  if ( ! mfrc522.PICC_IsNewCardPresent() || ! mfrc522.PICC_ReadCardSerial() ) {
    delay(50);
    return;
  }
  // A card is present, print its UID
  Serial.print("Found card with UID: ");
  String content= "";
  for (byte i = 0; i < mfrc522.uid.size; i++) {
    content.concat(String(mfrc522.uid.uidByte[i] < 0x10 ? " 0" : " "));
    content.concat(String(mfrc522.uid.uidByte[i], HEX));
  }
  content.toUpperCase();
  Serial.println(content);
}
```

קריאת נתונים מתג RFID.

```
void loop() {
  // Look for new RFID cards
  if ( ! mfrc522.PICC_IsNewCardPresent() || ! mfrc522.PICC_ReadCardSerial() ) {
    delay(50);
    return;
  }
  // Read data from a sector of the tag
  MFRC522::StatusCode status;
  byte buffer[18];
  byte bufferSize = sizeof(buffer);
  status = mfrc522.MIFARE_Read(1, buffer, &bufferSize);
  if (status == MFRC522::STATUS_OK) {
    Serial.println("Data read successfully:");
    Serial.println((char *)buffer);
  } else {
    Serial.println("Error reading data from tag.");
  }
  delay(2000);
}
```

פרק 5. פרוטוקול תקשורת I2C.

פרוטוקול Inter-Integrated Circuit (I2C) הוא פרוטוקול תקשורת טורית בשימוש נרחב המיועד לתקשורת יעילה בין מעגלים משולבים (ICs) בתוך רכיבים אלקטרוניים. פרק זה נועד לספק הבנה מקיפה של פרוטוקול I2C, כולל ההיסטוריה, העקרונות, הפעולה, היישומים, היתרונות והמגבלות שלו.

הִיסטוֹרִיָה:

I2C פותחה על ידי Philips Semiconductor (כיום NXP Semiconductors) בתחילת שנות ה-80 כדי לספק אמצעי תקשורת פשוט ויעיל בין IC שונים בלוח מעגלים. הוא תוכנן בתחילה לתקשורת בין מיקרו-בקר להתקנים היקפיים כמו חיישנים, EEPROMs, שעוני זמן אמת ו-ICs אחרים.

עקרונות:

תקשורת טורית: I2C הוא פרוטוקול תקשורת טורית, כלומר הוא מעביר נתונים ביט אחד בכל פעם על חוט או אוטובוס בודד.

ארכיטקטורת מאסטר-עבד: במערכת I2C, רכיב אחד פועל כמאסטר, יוזם תקשורת ושולט באפיק, בעוד מכשיר אחד או יותר פועלים כעבדים, המגיבים לפקודות מאסטר.

תקשורת דו-כיוונית: I2C תומכת בתקשורת דו-כיוונית, המאפשרת הן למכשירי המאסטר והן להתקני העבד לשדר ולקבל נתונים על אותו אפיק.

עיקרון פעולה:

תנאי התחלה: התקשורת באפיק I2C מתחילה בהתקן הראשי שיוצר מצב התחלה, המסמן את תחילת העברת הנתונים.

כתובת: המכשיר הראשי שולח את הכתובת של התקן העבד איתו הוא רוצה לתקשר.

העברת נתונים: נתונים מועברים בין המכשירים הראשיים והעבדים בסדרה של מחזורי שעון, כאשר כל ביט מקבל אישור על ידי המכשיר המקבל.

תנאי עצירה: התקשורת מסתיימת עם תנאי עצירה שנוצר על ידי ההתקן הראשי.

יישומים:

רשתות חיישנים: I2C משמש בדרך כלל ברשתות חיישנים להתממשקות עם חיישנים שונים כמו חיישני טמפרטורה, חיישני לחות ומד תאוצה.

התקני זיכרון: הוא משמש להתממשקות עם התקני זיכרון כמו EEPROMs ושעוני זמן אמת.

ממשקי תצוגה: I2C משמש בממשקי תצוגה לשליטה בתצוגות LCD ו-OLED.

מערכות משובצות: הוא מוצא יישומים במערכות משובצות לתקשורת בין שבבים ובקרת התקנים.

יתרונות:

פשטות: I2C קל ליישום ודורש רק שני חוטים לתקשורת.

יכולת ריבוי מאסטרים: הוא תומך בתצורות ריבוי מאסטר, המאפשר למספר התקני מאסטר לתקשר באותו אפיק.

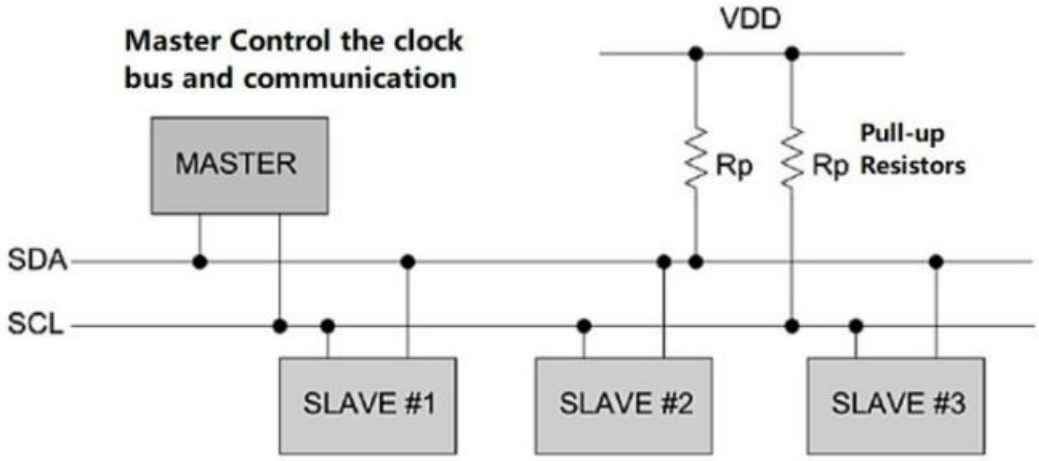
כתובת מובנית: ניתן לטפל בנפרד בהתקנים המחוברים לאפיק I2C, מה שמאפשר תקשורת יעילה במערכות מרובות התקנים.

מגבלות:

מרחק מוגבל: בשל הסתמכותו על איתות ניקוז פתוח או אספן פתוח, I2C אינו מתאים לתקשורת למרחקים ארוכים.

מהירויות נמוכות יותר: בהשוואה לפרוטוקולים אחרים כמו SPI, I2C פועל בדרך כלל במהירויות נמוכות יותר, מה שהופך אותו לפחות מתאים ליישומים במהירות גבוהה.

פרוטוקול I2C (Inter-Integrated Circuit) הוא פרוטוקול תקשורת טורית פופולרי המשמש לתקשורת בין מעגלים משולבים (ICs). הוא מאפשר למספר מכשירים לתקשר אחד עם השני באמצעות שני חוטים בלבד - קו שעון (SCL) וקו נתונים (SDA). בהסבר מפורט זה, נפרק את פרוטוקול I2C ביט אחרי ביט ונבדוק כיצד הוא מיושם במיקרו-בקר ESP32.



1. ביט התחלה (Start bit):

התקשורת באפיק I2C מתחילה בהתקן הראשי (ESP32) שמתחיל מצב התחלה. זה מסומן על ידי משיכת קו הנתונים (SDA) נמוך בעוד קו השעון (SCL) נשאר גבוה. תנאי ההתחלה מציין את תחילתה של העברת נתונים.

2. כתובת (Address):

לאחר תנאי ההתחלה, רכיב ראשי (מיקרו בקר) שולח את הכתובת של התקן העבד, איתו הוא רוצה לתקשר. הכתובת מועברת דרך קו הנתונים (SDA) ב-7 או 10 סיביות, תלוי אם ההתקן תומך במצב כתובת 7-bit או 10-bit. הסיבית הפחות משמעותית (LSB) של בית הכתובת מציינת אם המאסטר מתכוון לכתוב אל (LSB = 0) או לקרוא מהתקן העבד (LSB = 1).

3. תגובה (ACK/NACK):

לאחר שידור הכתובת, רכיב ראשי (מיקרו בקר) משחרר את קו הנתונים (SDA) והתקן העבד התואם לכתובת שולח אות אישור (ACK) על ידי משיכת קו הנתונים נמוך. אם אף התקן עבד לא מאשר את הכתובת, המכשיר הראשי מקבל אות לא מאומת (NACK).

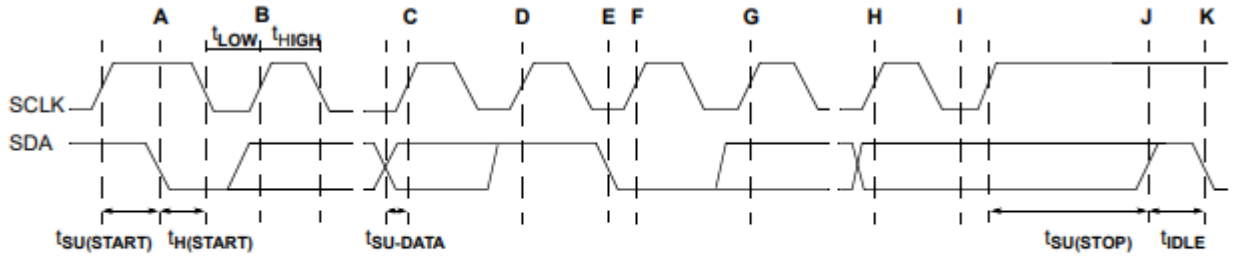
4. העברת נתונים (Data):

ברגע שהעבד יאשר את הכתובת, העברת נתונים יכולה להתרחש. שני המכשירים הראשיים והעבדים יכולים לשדר ולקבל נתונים על קו הנתונים (SDA) בזמן שקו השעון (SCL) מדופק. נתונים מועברים ומתקבלים בייט אחד בכל פעם, כאשר כל בייט מלווה באות אישור מהמכשיר המקבל.

5. מצב עצירה (Stop bit):

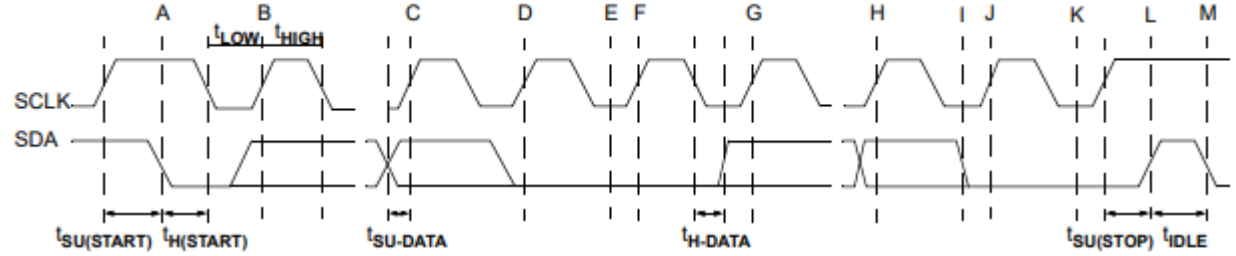
התקשורת בקו I2C מסתיימת עם ביט עצירה. זה מסומן על ידי ההתקן הראשי מושך את קו הנתונים (SDA) גבוה בעוד קו השעון (SCL) נשאר גבוה. תנאי העצירה מציין את סיום העברת הנתונים.

צורה גרפית של תהליך קריאה.



A = Start Condition
B = MSB of Address Clocked into Slave
C = LSB of Address Clocked into Slave
D = R/W Bit Clocked into Slave
E = Slave Pulls SDA Line Low
F = Acknowledge Bit Clocked into Master
G = MSB of Data Clocked into Master
H = LSB of Data Clocked into Master
I = Acknowledge Clock Pulse
J = Stop Condition
K = New Start Condition

צורה גרפית של תהליך כתיבה.



A = Start Condition
B = MSB of Address Clocked into Slave
C = LSB of Address Clocked into Slave
D = R/$\overline{W}$ Bit Clocked into Slave
E = Slave Pulls SDA Line Low
F = Acknowledge Bit Clocked into Master
G = MSB of Data Clocked into Slave
H = LSB of Data Clocked into Slave
I = Slave Pulls SDA Line Low
J = Acknowledge Clocked into Master
K = Acknowledge Clock Pulse
L = Stop Condition, Data Executed by Slave
M = New Start Condition

צורה לוגית של תהליך קריאה.

Write Byte Format

S	Address	WR	ACK	Command	ACK	Data	ACK	P
	7 Bits			8 Bits		8 Bits		

Slave Address

Command Byte: selects which register you are writing to.

Data Byte: data goes into the register set by the command byte.

צורה לוגית של תהליך כתיבה.

Read Byte Format

S	Address	WR	ACK	Command	ACK	S	Address	RD	ACK	Data	NACK	P
	7 Bits			8 Bits			7 Bits			8 Bits		

Slave Address

Command Byte: selects which register you are reading from.

Slave Address: repeated due to change in data-flow direction.

Data Byte: reads from the register set by the command byte.

(שרטוטים מדפי נתונים של TC74 https://ww1.microchip.com/downloads/en/DeviceDoc/21462D.pdf)

(למידע מלא על פרוטוקול https://www.nxp.com/docs/en/user-guide/UM10204.pdf).

יישום ב-ESP32:

המיקרו-בקר ESP32 כולל תמיכה מובנית בפרוטוקול I2C, מה שמקל על הטמעה בפרויקטים של ESP32. על ידי הכללת ספריית Wire ושימוש בפונקציות שלה, מפתחים יכולים לאתחל תקשורת I2C, לשלוח ולקבל נתונים ולהתממשק עם התקני I2C שונים.

מסך I2C LCD 16x2.

הבנת צגי LCD 16x2 I2C: סקירה מקיפה

תצוגות LCD (Liquid Crystal Display) 16x2 I2C נמצאות בשימוש נרחב בפרויקטים אלקטרוניים שונים בשל הפשטות, הקריאות וקלות השימוש שלהם. בשילוב עם ממשק I2C (Inter-Integrated Circuit), הצגים הללו הופכים אפילו יותר נוחים לשילוב בפרויקטים מבוססי מיקרו-בקר כמו ESP32. במאמר זה, נחקור את תכונות המפתח, עקרון העבודה, היתרונות והיישומים של צגי LCD 16x2 I2C.

תכונות עיקריות:

תצוגת תווים בגודל x216: צגי LCD בגודל x216 יכולים להראות 16 תווים בכל שורה ויש להם שתי שורות, המספקות סך של 32 תווים להצגת טקסט וגרפיקה פשוטה.

תאורה אחורית: צגי LCD רבים בגודל x216 מגיעים עם תאורה אחורית משולבת, המשפרת את הקריאה בתנאי תאורה חלשה.

ממשק I2C: על ידי שילוב של ממשק I2C (בדרך כלל מבוסס על PCF8574 או ICs דומים), צגים אלה דורשים רק כמה חיבורים למיקרו-בקר, מה שמפשט את החיווט ומפחית את השימוש ב-GPIO.

צריכת חשמל נמוכה: צגי LCD צורכים חשמל מינימלי, מה שהופך אותם למתאימים ליישומים המופעלים על ידי סוללה.

תווים מותאמים אישית: צגי LCD תומכים ביצירת תווים מותאמים אישית, המאפשרים הצגת סמלים ואייקונים מותאמים אישית.

עקרון עבודה:

צגי LCD בגודל x216 מורכבים מרשת של תאי גביש נוזליים המסודרים בשורות ובעמודות. כל תא פועל כפיקסל וניתן לשלוט בו בנפרד כדי להציג תווים או גרפיקה. התצוגה נשלטת על ידי בקר LCD, המקבל פקודות ונתונים מהמיקרו-בקר דרך ממשק I2C. ממשק I2C משתמש בפרוטוקול תקשורת טורית להעברת אותות בקרה ונתונים בין המיקרו-בקר לבקר ה-LCD.

יתרונות:

פשטות: צגי LCD 16x2 I2C דורשים חיווט מינימלי ופחות פיני GPIO בהשוואה לצגים עם ממשקים מקבילים, מה שמפשט את שילוב החומרה.

חיסכון במקום: עם גודלם הקומפקטי ודרישות החיווט המינימליות שלהם, צגים אלו חוסכים מקום על ה-PCB, מה שהופך אותם מתאימים לעיצובים קומפקטיים.

קלות שימוש: ספריות וקודים לדוגמה זמינים בקלות עבור מיקרו-בקרים פופולריים כמו ESP32, מה שמאפשר יצירת אב טיפוס מהיר ופיתוח.

קריאות: צגי LCD מציעים קריאה טובה בתנאי תאורה שונים, מה שהופך אותם למתאימים ליישומים פנימיים.

מחיר סביר: צגי LCD 16x2 I2C הם חסכוניים וזמינים, מה שהופך אותם לבחירה פופולרית עבור חובבים ויצרנים.

יישומים:

מערכות משובצות: תצוגות LCD משמשות בדרך כלל במערכות משובצות להצגת מצב המערכת, קריאות חיישנים ורכיבי ממשק משתמש.

פרויקטי עשה זאת בעצמך אלקטרוניקה: תחביבים ויצרנים משתמשים בתצוגות LCD בפרויקטים שונים של עשה זאת בעצמך, כולל שעונים, תחנות מזג אוויר, מדי חום דיגיטליים ומערכות אוטומציה ביתית.

כלים חינוכיים: צגי LCD משמשים בערכות חינוכיות ומדריכים כדי ללמד מושגי תכנות ויסודות אלקטרוניקה.

לוחות בקרה תעשייתיים: צגי LCD משמשים בלוחות בקרה תעשייתיים וממשקי אדם-מכונה (HMIs) לניטור ובקרה של תהליכים תעשייתיים.

סיכום:

צגי LCD 16x2 I2C מציעים פתרון רב-תכליתי וידידותי להצגת טקסט וגרפיקה פשוטה במגוון רחב של פרויקטים אלקטרוניים. עם הפשטות, הקריאה וקלות האינטגרציה שלהם, צגים אלה מתאימים היטב ליישומים הדורשים רכיבי ממשק משתמש בסיסיים ומחווני מצב. על ידי הבנת התכונות, עקרון העבודה, היתרונות והיישומים שלהם, מפתחים יכולים למנף ביעילות תצוגות LCD 16x2 I2C כדי לשפר את העיצובים שלהם וליצור חוויות משתמש מרתקות.

ממשק תצוגת LCD בגודל 16 x2 עם מיקרו בקר ESP32 באמצעות פרוטוקול I2C היא משימה נפוצה בפרויקטים משובצים. הסבר מפורט זה יכסה הן את חיבורי החומרה והן את יישום התוכנה הנדרש כדי להשתמש בהצלחה ב- I2C LCD 16x2 עם ESP32.

הגדרת חומרה:

תצוגת LCD בגודל 16x2 תצוגת ה-LCD כוללת בדרך כלל 16 עמודות ו-2 שורות של תווים. יש לו תאורה אחורית ופוטנציומטר לכוונון ניגודיות.

מיקרו-בקר ESP32: ה-ESP32 ישלוט בתצוגת ה-LCD באמצעות פרוטוקול I2C.

תרמיל I2C: תרמיל I2C (או מודול I2C) מפשט את החיווט על ידי המרת נתונים מקבילים מה-LCD לנתונים טוריים עבור ה-ESP32 באמצעות פרוטוקול I2C.

חיווט: חבר את פין SDA של תרמיל I2C לפין SDA של ESP32, את פין SCL של תרמיל I2C לפין SCL של ESP32. חבר את VCC ו-GND כראוי.

יישום תוכנה:

התקנת ספריות:

התקן את ספריית "LiquidCrystal_I2C" עבור ESP32. ספרייה זו מספקת פונקציות לשליטה בתצוגת ה-LCD באמצעות פרוטוקול I2C.

קריאת ספריות:

```
#include <Wire.h>

#include <LiquidCrystal_I2C.h>
```

הגדרת תכונות המסך:

```
#define LCD_ADDRESS 0x27 // I2C address of the LCD

#define LCD_COLUMNS 16   // Number of columns in the LCD

#define LCD_ROWS 2       // Number of rows in the LCD
```

הגדרת אובייקט LCD:

```
LiquidCrystal_I2C lcd(LCD_ADDRESS, LCD_COLUMNS, LCD_ROWS);
```

פונקציות עיקריות להגדרת המסך:

```
void setup() {
  Wire.begin();      // Initialize I2C communication
  lcd.init();        // Initialize the LCD
  lcd.backlight();   // Turn on the backlight
  lcd.setCursor(0, 0); // Set cursor to the first column of the first row
  lcd.print("Hello!"); // Print message
}
```

להלן רשימה מלאה של כל הפונקציות של המסך 2x16 I2C LCD:

פונקציה	הסבר
`begin(cols, rows, charsize)`	אתחול ה-LCD עם מספר העמודות, השורות וגודל התווים.
`init()`	אתחול ה-LCD עם פרמטרי ברירת מחדל (16 x2, 5x8 תווים בגודל).
`backlight()`	הפעל את התאורה האחורית של ה-LCD
`noBacklight()`	כבה את התאורה האחורית של ה-LCD
`setCursor(col, row)`	הגדר את מיקום הסמן לעמודה ולשורה שצוינו.
`home()`	החזר את הסמן למיקום הבית (עמודה ראשונה, שורה ראשונה).
`clear()`	נקה את התצוגה והחזר את הסמן למיקום הבית.
`display()`	הפעל את התצוגה (הצג תווים).
`noDisplay()`	כבה את התצוגה (הסתר תווים).
`cursor()`	הפעל את הסמן (הצג את הסמן).
`noCursor()`	כבה את הסמן (הסתר סמן).
`blink()`	הפעל את הסמן מהבהב.
`noBlink()`	כבה את הסמן מהבהב.
`write(value)`	כתוב תו בודד לתצוגה.
`print(string)`	הדפס מחרוזת לתצוגה.
`print(value)`	הדפס ערך מספרי לתצוגה.
`println(string)`	הדפס מחרוזת ואחריה תו חדש.
`println(value)`	הדפס ערך מספרי ואחריו תו חדש.
`scrollDisplayLeft()`	גלגל את התצוגה עמודה אחת שמאלה.
`scrollDisplayRight()`	גלגל את התצוגה עמודה אחת ימינה.
`createChar(location, charmap)`	הגדר תו מותאם אישית במיקום הזיכרון שצוין.
`write(customCharIndex)`	כתוב תו מותאם אישית לתצוגה.
`autoscroll()`	אפשור גלילה אוטומטית של התצוגה.
`noAutoscroll()`	נטרול גלילה אוטומטית של התצוגה.
`leftToRight()`	הגדר כיוון כתיבת טקסט משמאל לימין (ברירת מחדל).
`rightToLeft()`	הגדר את כיוון כתיבת הטקסט מימין לשמאל.

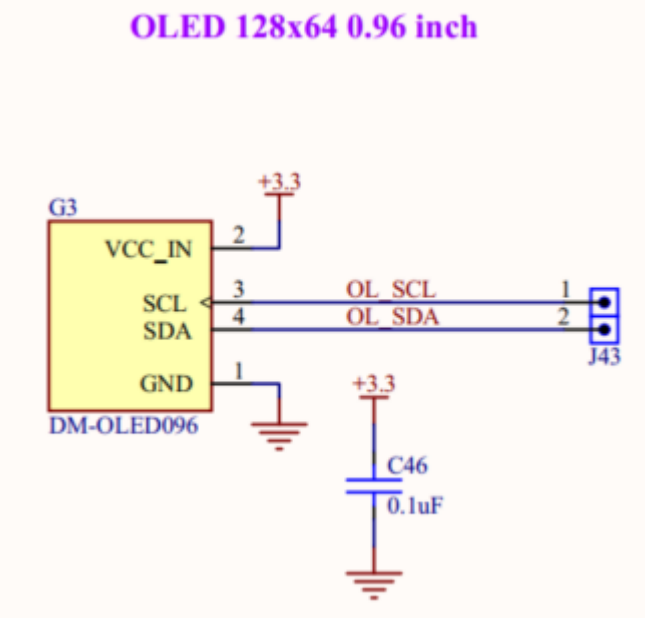
קוד הדגמה של כל הפונקציות:

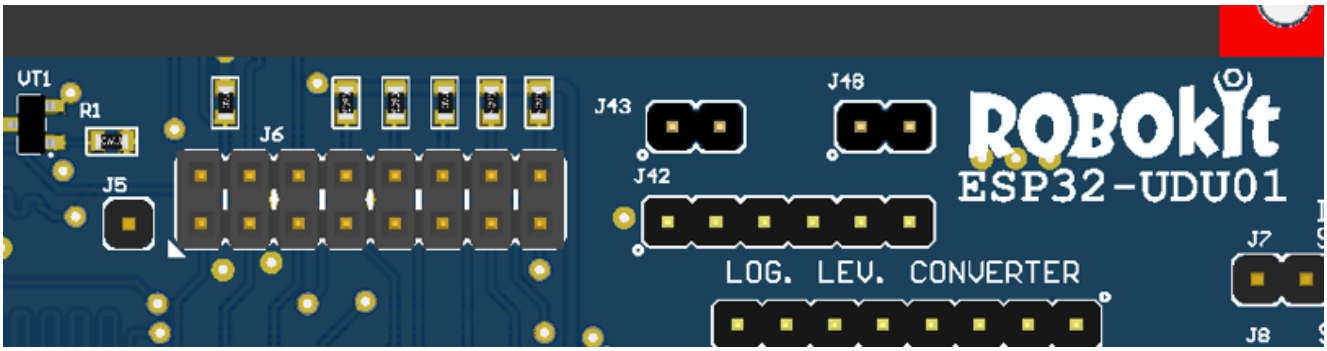
```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>

// Define I2C address of the LCD
#define LCD_ADDRESS 0x27
// Define number of columns and rows of the LCD
#define LCD_COLUMNS 16
#define LCD_ROWS 2

// Create an LCD object
```

```
LiquidCrystal_I2C lcd(LCD_ADDRESS, LCD_COLUMNS, LCD_ROWS);

void setup() {
  // Initialize I2C communication
  Wire.begin();
  // Initialize the LCD
  lcd.init();
  // Turn on the backlight
  lcd.backlight();
}

void loop() {
  // Initialization Functions
  lcd.begin(16, 2); // Initialize the LCD with 16 columns and 2 rows
  delay(1000);

  // Cursor Control Functions
  lcd.setCursor(0, 0); // Set the cursor to column 0, row 0
  delay(1000);

  lcd.home(); // Move the cursor to the home position (0, 0)
  delay(1000);

  lcd.clear(); // Clear the LCD screen
  delay(1000);

  // Display Control Functions
  lcd.display(); // Turn on the display
  delay(1000);

  lcd.noDisplay(); // Turn off the display
  delay(1000);

  // Text Display Functions
  lcd.print("Hello, World!"); // Print a string to the LCD
  delay(1000);

  lcd.write('A'); // Write a single character to the LCD
  delay(1000);

  lcd.setCursor(0, 1); // Set the cursor to column 0, row 1
  lcd.print("Line 2"); // Print a string to the second line of the LCD
  delay(1000);

  // Special Characters Functions
  byte customChar[8] = {
    B00000,
    B00000,
    B00000,
    B00000,
    B00000,
    B00000,
    B00000,
    B00000
  };
  lcd.createChar(0, customChar); // Create a custom character
  lcd.write(byte(0)); // Display the custom character
  delay(1000);

  // Advanced Control Functions
```

```
  lcd.autoscroll(); // Enable auto-scrolling of the display
  delay(1000);

  lcd.noAutoscroll(); // Disable auto-scrolling of the display
  delay(1000);
}
```

מסך OLED I2C.





חקר צגי OLED I2C: מדריך מקיף

צגי OLED (Organic Light-Emitting Diode) הם בחירה פופולרית עבור פרויקטים אלקטרוניים שונים בשל הניגודיות הגבוהה שלהם, זווית הצפייה הרחבה וצריכת החשמל הנמוכה שלהם. בשילוב עם ממשק I2C (מעגל משולב), צגי OLED הופכים אפילו יותר צדדיים וקלים לשימוש עם מיקרו-בקרים כמו ESP32. במאמר זה, נתעמק בתכונות המפתח, עקרון העבודה, היתרונות והיישומים של צגי OLED I2C.

תכונות עיקריות:

ניגודיות גבוהה: צגי OLED מייצרים צבעים מרהיבים ושחור עמוק, וכתוצאה מכך ניגודיות ואיכות תמונה מצוינת.

זווית צפייה רחבה: בניגוד לצגי LCD מסורתיים, לצגי OLED יש זווית צפייה רחבה, המבטיחה נראות ברורה מנקודות מבט שונות.

זמן תגובה מהיר: ניתן להפעיל ולכבות פיקסלים של OLED בנפרד, מה שמאפשר זמני תגובה מהירים והנפשות חלקות.

צריכת חשמל נמוכה: טכנולוגיית OLED אינה דורשת תאורה אחורית, מה שמוביל לצריכת חשמל נמוכה יותר בהשוואה לצגי LCD.

גמיש וקל משקל: ניתן לייצר צגי OLED על מצעים גמישים, מה שהופך אותם למתאימים למכשירים מעוקלים ולבישים.

עקרון עבודה:

צגי OLED מורכבים משכבות אורגניות של סרט דק הכרוכות בין שתי אלקטרודות מוליכות. כאשר מתח מופעל על פני האלקטרודות הללו, זרם זורם דרך השכבות האורגניות, וגורם להן לפלוט אור. הצבע והעוצמה של האור הנפלט תלויים בחומרים האורגניים המשמשים בתצוגה. בתצוגת I2C OLED, בקר ממשק I2C (למשל, SSD1306) משולב במודול התצוגה, המאפשר תקשורת טורית עם מיקרו-בקרים.

יתרונות:

פשטות: צגי I2C OLED דורשים רק כמה חיבורים (מתח, הארקה, SDA ו-SCL) למיקרו-בקר, מה שמפשט את שילוב החומרה.

חיסכון במקום: עם הגודל הקומפקטי ודרישות החיווט המינימליות שלו, צג I2C OLED חוסך מקום על ה-PCB, מה שהופך אותו מתאים לעיצובים קומפקטיים.

קלות שימוש: ספריות וקודים לדוגמה זמינים בקלות עבור מיקרו-בקרים פופולריים כמו ESP32, מה שמאפשר יצירת אב טיפוס מהיר ופיתוח.

התאמה אישית: צגי OLED מציעים גמישות בהצגת גרפיקה, טקסט ואנימציות, ומאפשרות ממשקי משתמש יצירתיים ואינטראקטיביים.

יישומים:

התקנים לבישים: צגי OLED הם אידיאליים עבור שעונים חכמים, מעקבי כושר ומכשירים לבישים אחרים בשל אופיים קל משקל וחסכוני בצריכת החשמל.

התקני IoT: ביישומי IoT, צגי I2C OLED יכולים לספק הדמיית נתונים בזמן אמת, מחווני מצב ומשוב משתמשים.

מכשירים ניידים: מכשירים כף יד כגון מולטימטרים, אוסילוסקופים ומדדי טמפרטורה/לחות יכולים להפיק תועלת מצגי OLED קומפקטיים להצגת נתונים.

פרויקטי עשה זאת בעצמך: תחביבים ויצרנים משתמשים לעתים קרובות בתצוגות I2C OLED בפרויקטי אלקטרוניקה עשה זאת בעצמך, כולל שעונים, תחנות מזג אוויר ונגני מדיה.

סיכום:

צגי I2C OLED מציעים פתרון רב-תכליתי וידידותי למשוב ויזואלי ותצוגת מידע במגוון רחב של פרויקטים אלקטרוניים. עם הניגודיות הגבוהה שלהם, זווית הצפייה הרחבה וצריכת החשמל הנמוכה שלהם, צגים אלה מתאימים היטב ליישומים הדורשים גודל קומפקטי, צריכת חשמל נמוכה וחזותי תוססת. על ידי הבנת התכונות, עקרון העבודה, היתרונות והיישומים שלהם, מפתחים יכולים למנף ביעילות את צגי I2C OLED כדי לשפר את העיצובים שלהם וליצור חוויות משתמש מרתקות.

הסבר	פונקציה
אתחול ה-LCD.	**begin()**
הפעל את התצוגה.	**display()**
כבה את התצוגה.	**noDisplay()**
הופך את התצוגה (true עבור הפוך, false עבור רגיל).	**invertDisplay(boolean)**
מגדיר את בהירות ה-OLED (true עבור עמום, false עבור בהירות מלאה).	**dim(boolean)**
הגדר את מיקום הסמן לפי קואורדינטות שצוינו.	**setCursor(x, y)**
החזר את הסמן למיקום הבית (עמודה ראשונה, שורה ראשונה).	**home()**

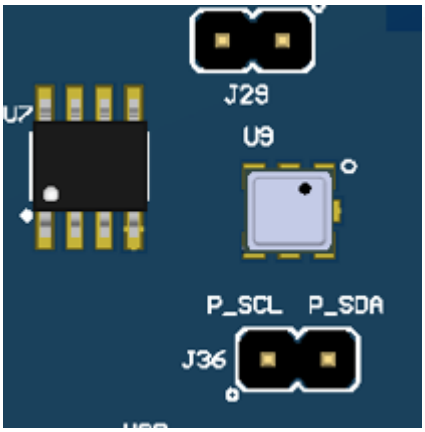
clear()	נקה את התצוגה והחזר את הסמן למיקום הבית.
write(char)	כתוב תו בודד לתצוגה.
print(string)	הדפס מחרוזת לתצוגה.
print(number)	הדפס ערך מספרי לתצוגה.
drawPixel(x, y, color)	מצייר פיקסל בקואורדינטות שצוינו עם הצבע שצוין.
drawLine(x0, y0, x1, y1, color)	מצייר קו מ-(x0, y0) ל-(x1, y1) עם הצבע שצוין.
drawRect(x, y, width, height, color)	מצייר מלבן עם הרוחב, הגובה והצבע שצוינו.
fillRect(x, y, width, height, color)	מצייר מלבן מלא עם הרוחב, הגובה והצבע שצוינו.
drawCircle(x0, y0, radius, color)	מצייר עיגול עם הרדיוס והצבע שצוינו.
fillCircle(x0, y0, radius, color)	מצייר עיגול מלא עם הרדיוס והצבע שצוינו.
drawBitmap(x, y, bitmap, width, height, color)	מצייר תמונת מפת סיביות בקואורדינטות שצוינו עם הרוחב, הגובה והצבע שצוינו.
startscrollright()	גלגל את התצוגה עמודה אחת ימינה.
startscrollleft()	גלגל את התצוגה עמודה אחת שמאלה.
startscrolldiagright()	גלגל את התצוגה באלכסון ימינה.
startscrolldiagleft()	גלגל את התצוגה באלכסון שמאלה.
stopscroll()	נטרול גלילה של התצוגה.
displayOn()	הפעל את התצוגה.
displayOff()	כבה את התצוגה.
setContrast(contrast)	מגדיר את ניגודיות ה-OLED (0-255).

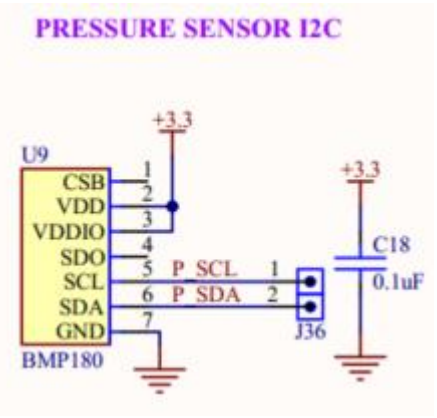
קוד לדוגמה: שימוש בכל הפונקציות מהטבלה.

```
#include <Adafruit_GFX.h>
#include <Adafruit_SSD1306.h>

#define SCREEN_WIDTH 128
#define SCREEN_HEIGHT 64

#define OLED_RESET -1
#define SCREEN_ADDRESS 0x3C

Adafruit_SSD1306 display(SCREEN_WIDTH, SCREEN_HEIGHT, &Wire, OLED_RESET);

void setup() {
  Wire.begin();
  display.begin(SSD1306_SWITCHCAPVCC, SCREEN_ADDRESS);
  display.display(); // Display is on by default, but in case it's not
  delay(2000); // Pause for 2 seconds
}

void loop() {
  // Display Control Functions
  display.display(); // Turns on the OLED display
  delay(2000);

  display.noDisplay(); // Turns off the OLED display
  delay(2000);

  display.invertDisplay(true); // Inverts the display
  delay(2000);

  display.invertDisplay(false); // Restores normal display
  delay(2000);

  display.dim(true); // Sets dim display
  delay(2000);
```

```
  display.dim(false); // Restores full brightness
  delay(2000);

  // Cursor Control Functions
  display.clear(); // Clears the OLED display
  delay(2000);

  display.setCursor(0, 0); // Sets cursor to position (0, 0)
  display.print("Hello!"); // Prints "Hello!" to the OLED display
  delay(2000);

  display.home(); // Moves cursor to the home position (0, 0)
  delay(2000);

  // Text Display Functions
  display.print("ESP32"); // Prints "ESP32" to the OLED display
  delay(2000);

  display.write('A'); // Writes the character 'A' to the OLED display
  delay(2000);

  display.print(123); // Prints the number 123 to the OLED display
  delay(2000);

  // Graphics Drawing Functions
  display.drawPixel(10, 10, WHITE); // Draws a white pixel at position (10, 10)
  delay(2000);

  display.drawLine(0, 0, 20, 20, WHITE); // Draws a white line from (0, 0) to (20, 20)
  delay(2000);

  display.drawRect(30, 30, 10, 10, WHITE); // Draws a white rectangle at position (30, 30) with width 10
and height 10
  delay(2000);

  display.fillCircle(50, 50, 5, WHITE); // Draws a filled white circle at position (50, 50) with radius 5
  delay(2000);

  // Bitmap Display Functions
  display.drawBitmap(70, 10, logo_bitmap, 32, 32, WHITE); // Draws a bitmap image at position (70, 10)
  delay(2000);

  // Scrolling Functions
  display.startscrollright(0x00, 0x0F); // Scrolls the display to the right
  delay(2000);

  display.stopscroll(); // Stops scrolling
  delay(2000);
}
```

חיישן לחץ ברומטרי BMP180.





חקר חיישן BMP180: מדריך מפורט

חיישן BMP180 הוא חיישן לחץ ברומטרי דיגיטלי המיוצר על ידי Bosch Sensortec. הוא מספק מדידות מדויקות של לחץ וטמפרטורה אטמוספריים, מה שהופך אותו למרכיב חיוני במערכות ניטור מזג אוויר, התקני מעקב גבהים ויישומים שונים אחרים. במאמר זה, נתעמק בתכונות המפתח, עקרון העבודה, היישומים ושיקולי השימוש של חיישן BMP180.

תכונות עיקריות:

ממשק דיגיטלי: חיישן BMP180 מתקשר עם מיקרו-בקרים באמצעות פרוטוקול I2C (Inter-Integrated Circuit), מה שמפשט את האינטגרציה בפרויקטים.

מדידת לחץ: עם טווח מדידה רחב של 300 עד 1100 hPa (הקטופסקלים), חיישן BMP180 יכול לזהות במדויק שינויים בלחץ האטמוספרי.

מדידת טמפרטורה: בנוסף ללחץ, חיישן זה מספק גם קריאת טמפרטורה ברמת דיוק גבוהה.

צריכת חשמל נמוכה: חיישן BMP180 פועל בהספק נמוך, מתאים ליישומים המופעלים על ידי סוללה, כולל תחנות מזג אוויר ניידות ומד גובה.

גודל קומפקטי: חיישן BMP180 זמין בפורמט קטן, מתאים לעיצובים מוגבלי מקום במכשירים אלקטרוניים שונים.

עקרון עבודה:

חיישן BMP180 משתמש בטכנולוגיה piezoresistive למדידת לחץ אטמוספרי. הוא מכיל קוביית חיישן לחץ עם חיישן טמפרטורה מובנה. כאשר הם נחשפים ללחץ אוויר, האלמנטים הפייזוריסטיים בתוך החיישן מתעוותים, וגורמים לשינויים בהתנגדות. שינויים אלו מומרים לקריאת לחץ דיגיטלית באמצעות ממיר אנלוגי לדיגיטלי (ADC). פיצוי טמפרטורה מוחל כדי להבטיח מדידות לחץ מדויקות על פני טווח טמפרטורות רחב.

יישומים:

ניטור מזג אוויר: חיישן BMP180 נמצא בשימוש נפוץ בתחנות מזג אוויר ובמערכות ניטור סביבתיות למדידת לחץ וטמפרטורה אטמוספריים, המאפשרים חיזוי של דפוסי מזג אוויר.

מעקב אחר גבהים: על ידי מדידת שינויים בלחץ האטמוספרי, חיישן BMP180 יכול להעריך שינויים בגובה, מה שהופך אותו למתאים עבור מדי גובה בתעופה, טיולים וספורט חוץ.

ניווט פנימי: בשילוב עם חיישנים אחרים כגון מדי תאוצה וג'ירוסקופים, חיישן BMP180 יכול לשמש ליישומי ניווט ולוקליזציה פנימיים.

התקני IoT: עם צריכת החשמל הנמוכה והממשק הדיגיטלי שלו, חיישן BMP180 אידיאלי לשילוב בהתקני IoT (האינטרנט של הדברים) לניטור סביבתי מרחוק ורישום נתונים.

שיקולי שימוש:

כיול: בעוד שחיישן BMP180 מגיע מכויל מראש מהמפעל, ייתכן שיהיה צורך בכיול מחדש תקופתי כדי לשמור על דיוק לאורך זמן, במיוחד בסביבות קשות.

תנאי הפעלה: ודא שהחיישן מופעל בטווחי הטמפרטורה והלחות שצוינו כדי להבטיח מדידות מדויקות ואמינות.

כיוון הרכבה: כיוון נכון של החיישן ביחס לקרקע חיוני למדידות גובה מדויקות. התקן את החיישן בצורה אופקית לקבלת התוצאות הטובות ביותר.

ספק כוח: ספק כוח יציב לחיישן בטווח המתח שצוין כדי למנוע נזק ולהבטיח פעולה תקינה.

סיכום:

חיישן BMP180 מציע פתרון רב תכליתי ואמין למדידת לחץ וטמפרטורה אטמוספריים במגוון רחב של יישומים. עם הממשק הדיגיטלי, הדיוק הגבוה והעיצוב הקומפקטי שלו, חיישן זה מתאים לניטור מזג אוויר, מעקב גבהים, ניווט פנימי ויישומי IoT. על ידי הבנת התכונות שלו, עקרון העבודה, היישומים ושיקולי השימוש שלו, מפתחים יכולים למנף ביעילות את חיישן BMP180 כדי לשפר את העיצובים שלהם וליצור פתרונות חדשניים לצרכים מגוונים.

התקנת ספריות:

פתח את Arduino IDE ונווט אל Sketch -> Include Library <- נהל ספריות.

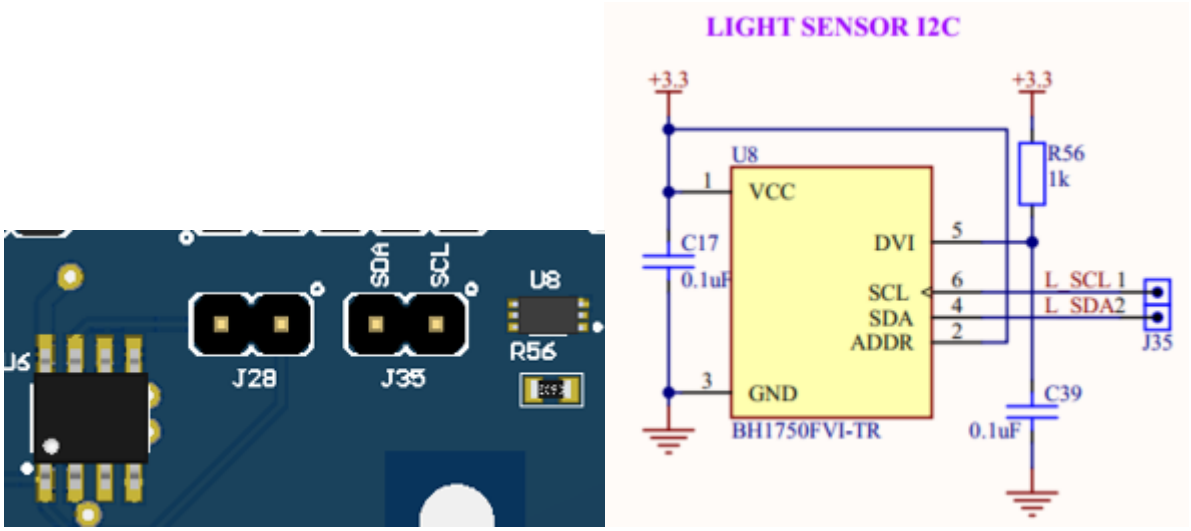
חפש את "Adafruit BMP085" והתקן את הספרייה של Adafruit.

לחלופין, אתה יכול להשתמש בספריית "Adafruit BMP280", התואמת לחיישן BMP180.

קוד נראה כך:

```
#include <Wire.h>
#include <Adafruit_BMP085.h> // or <Adafruit_BMP280.h>
Adafruit_BMP085 bmp;
void setup() {
    Serial.begin(9600);
    if (!bmp.begin()) {
        Serial.println("BMP180 initialization failed!");
        while (1);
    }
}
void loop() {
    float pressure = bmp.readPressure();
    float temperature = bmp.readTemperature();

    Serial.print("Pressure: ");
    Serial.print(pressure);
    Serial.print(" Pa\t");

    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.println(" °C");

    delay(1000); // Delay for 1 second
}
```

חיישן אור I2C.



הבנת חיישן BH1750FVI-TR: סקירה מקיפה

ה-BH1750FVI-TR הוא חיישן אור סביבה דיגיטלי המספק מדידות מדויקות של עוצמת האור בלוקס. חיישן זה, המיוצר על ידי ROHM Semiconductor, נמצא בשימוש נרחב ביישומים שונים שבהם חיוני זיהוי אור מדויק. במאמר זה, נחקור את תכונות המפתח, עקרון העבודה, היישומים ושיקולי השימוש של חיישן BH1750FVI-TR.

תכונות עיקריות:

ממשק דיגיטלי: ה-BH1750FVI-TR מתקשר עם מיקרו-בקרים באמצעות ממשק I2C, מה שמקל על שילובו בפרויקטים.

טווח מדידה רחב: חיישן זה מסוגל לזהות עוצמת אור בטווח רחב, בדרך כלל בין 0 ל-65535 לוקס.

רזולוציה גבוהה: ברזולוציה של 1 לוקס, ה-BH1750FVI-TR מספק קריאות מדויקות גם בתנאי תאורה חלשים.

צריכת חשמל נמוכה: חיישן זה פועל בהספק נמוך, מתאים ליישומים המופעלים על ידי סוללה.

גודל קומפקטי: ה-BH1750FVI-TR זמין בפורמט קטן, המאפשר שילוב קל בעיצובים קומפקטיים.

עקרון עבודה:

חיישן BH1750FVI-TR משתמש במערך פוטודיודות כדי להמיר אור חודר לאותות חשמליים. האותות הללו מעובדים לאחר מכן על ידי ממיר אנלוגי לדיגיטלי (ADC) כדי לקבל ערכים דיגיטליים המייצגים את עוצמת האור בלוקס. המעגל הפנימי של החיישן כולל אוגרי כיול להתאמת הרגישות וטווח המדידה.

יישומים:

בקרת תאורה אוטומטית: ה-BH1750FVI-TR משמש בדרך כלל במערכות תאורה חכמות כדי להתאים את הבהירות בהתבסס על תנאי האור הסביבה, לשיפור יעילות האנרגיה ונוחות המשתמש.

בקרת תאורה אחורית לתצוגה: במכשירים אלקטרוניים כגון סמארטפונים, טאבלטים ומסכים, חיישן זה עוזר לייעל את תאורת התצוגה האחורית לחשיפה אופטימלית בסביבות תאורה שונות.

ניטור מזג אוויר: בתחנות מזג אוויר ובמערכות ניטור סביבתיות, ה-BH1750FVI-TR מספק נתונים חשובים להערכת רמות ומגמות של אור יום.

התקני IoT: עם גודלו הקומפקטי וצריכת החשמל הנמוכה, חיישן זה מתאים לשילוב במכשירי IoT עבור אוטומציה של בית חכם, טכנולוגיה לבישה ועוד.

שיקולי שימוש:

כיול: למרות שה-BH1750FVI-TR מגיע מכויל מראש מהמפעל, ייתכן שיהיה צורך בכיול מחדש תקופתי כדי לשמור על דיוק לאורך זמן.

כיוון הרכבה: כיוון נכון של החיישן ביחס למקור האור חיוני למדידות מדויקות. הימנע מחסימת שדה הראייה של החיישן.

סביבת הפעלה: ודא שהחיישן מוגן מאור שמש ישיר וממקורות אחרים של אור עז כדי למנוע רוויה וקריאות לא מדויקות.

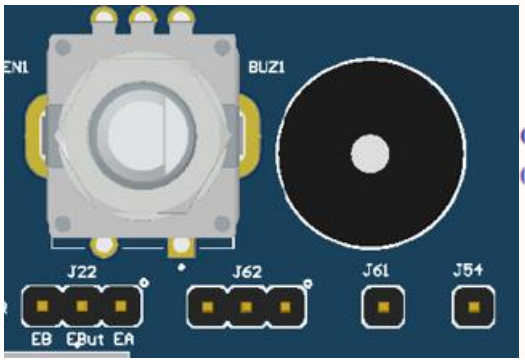
ספק כוח: ספק כוח יציב לחיישן בטווח המתח שצוין כדי להבטיח פעולה אמינה.

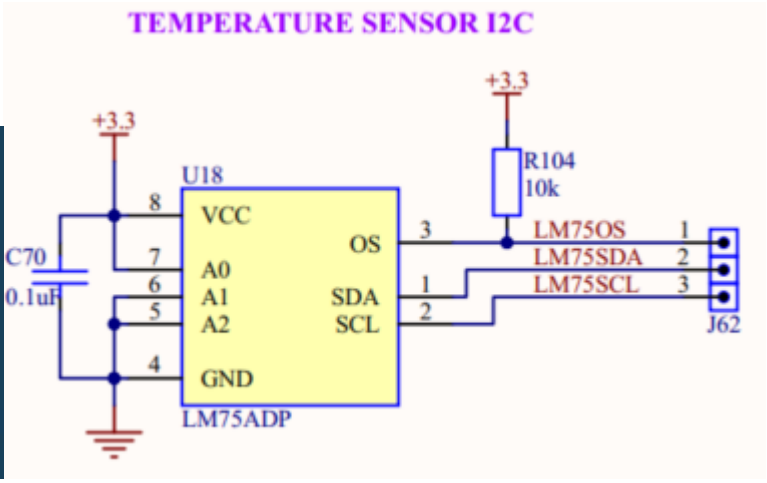
סיכום:

חיישן BH1750FVI-TR מציע פתרון נוח ואמין למדידת עוצמת האור הסביבתי במגוון רחב של יישומים. עם הממשק הדיגיטלי, הדיוק הגבוה והעיצוב הקומפקטי שלו, חיישן זה מתאים לפרויקטים שונים הדורשים יכולות זיהוי אור. על ידי הבנת התכונות, עקרון העבודה, היישומים ושיקולי השימוש שלו, מפתחים יכולים למנף ביעילות את חיישן BH1750FVI-TR כדי לשפר את העיצובים שלהם וליצור פתרונות חדשניים לצרכים מגוונים.

```
#include <Wire.h>
#include <BH1750.h>

BH1750 lightMeter(0x23); // Define the I2C address of the BH1750 sensor

void setup() {
  Serial.begin(9600);

  // Initialize the BH1750 sensor
  lightMeter.begin(BH1750_CONTINUOUS_HIGH_RES_MODE);
}

void loop() {
  // Read light intensity
  uint16_t lux = lightMeter.readLightLevel();

  // Print light intensity to Serial Monitor
  Serial.print("Light Intensity: ");
  Serial.print(lux);
  Serial.println(" lux");

  delay(1000); // Delay for 1 second
}
```

חיישן טמפרטורה I2C.





חקור את חיישן הטמפרטורה LM75: מדריך מקיף

ה-LM75 הוא חיישן טמפרטורה דיגיטלי המיוצר על ידי Texas Instruments. הוא מספק מדידות טמפרטורה מדויקות ברזולוציה גבוהה, מה שהופך אותו לבחירה פופולרית עבור יישומים שונים באלקטרוניקה, אוטומציה תעשייתית ואלקטרוניקה צריכה. במאמר זה נתעמק בתכונות המפתח, עקרון העבודה, היתרונות והיישומים של חיישן הטמפרטורה LM75.

תכונות עיקריות:

ממשק דיגיטלי: ה-LM75 מתקשר עם מיקרו-בקרים באמצעות ממשק I2C (Inter-Integrated Circuit), המאפשר שילוב קל במערכות דיגיטליות.

טווח טמפרטורות רחב: ה-LM75 יכול למדוד טמפרטורות הנעות בין -55 מעלות צלזיוס עד +125 מעלות צלזיוס, מה שהופך אותו למתאים הן ליישומים בטמפרטורה נמוכה והן בטמפרטורה גבוהה.

רזולוציה גבוהה: ברזולוציה של 0.125°C, ה-LM75 מספק מדידות טמפרטורה מדויקות לניטור ובקרה מדויקים.

צריכת חשמל נמוכה: פועל בהספק נמוך, ה-LM75 מתאים ליישומים המופעלים על ידי סוללה ועיצובים חסכוניים באנרגיה.

חבילה קומפקטית: זמין בגורמי צורה קטנים כגון SOT23 ו-SOIC, ניתן לשלב את ה-LM75 בקלות בעיצובים מוגבלי מקום.

עקרון עבודה:

ה-LM75 מנצל את העיקרון של מתח תלוי טמפרטורה למדידת טמפרטורה. הוא מכיל אלמנט חיישן טמפרטורה, ממיר אנלוגי לדיגיטלי (ADC) ובקר ממשק I2C. כשהוא מופעל, ה-LM75 ממיר את המתח האנלוגי מחיישן הטמפרטורה לערך דיגיטלי באמצעות ה-ADC המובנה. ערך דיגיטלי זה מועבר לאחר מכן אל המיקרו-בקר באמצעות ממשק I2C, שם ניתן לקרוא ולעבד אותו.

יתרונות:

דיוק: ה-LM75 מספק מדידות טמפרטורה מדויקות ברזולוציה גבוהה, מה שמבטיח ביצועים אמינים בתנאי הפעלה שונים.

פשטות: עם הממשק הדיגיטלי וה-ADC המובנה שלו, ה-LM75 מפשט את חישת הטמפרטורה והניטור במערכות דיגיטליות.

צדדיות: ניתן להשתמש ב-LM75 במגוון רחב של יישומים, כולל ניטור סביבתי, אוטומציה תעשייתית, מערכות HVAC ואלקטרוניקה לצרכן.

יכולת פעולה הדדית: ה-LM75 תואם לממשקי I2C סטנדרטיים, ומאפשר לו לתקשר עם מגוון רחב של מיקרו-בקרים והתקנים דיגיטליים.

עלות-יעילות: ה-LM75 מציע פתרון חסכוני עבור חישת טמפרטורה, מה שהופך אותו מתאים לפרויקטים מסחריים ותחביבים כאחד.

יישומים:

ניטור סביבתי: ה-LM75 משמש בתחנות מזג אוויר, מערכות בקרת אקלים ומכשירי ניטור סביבתיים למדידת טמפרטורת הסביבה.

אוטומציה תעשייתית: ביישומי אוטומציה תעשייתית ובקרת תהליכים, ה-LM75 מספק חישת טמפרטורה לניטור ובקרה של תהליכי ייצור.

מוצרי אלקטרוניקה: ה-LM75 משולב במוצרי אלקטרוניקה כגון סמארטפונים, מחשבים ניידים ומכשירי חשמל ביתיים לניטור טמפרטורה וניהול תרמי.

מכשירים רפואיים: במכשירים רפואיים וביישומי בריאות, ה-LM75 מבטיח מדידות טמפרטורה מדויקות לניטור ואבחון מטופלים.

מערכות רכב: ה-LM75 מועסק במערכות רכב עבור חישת טמפרטורה בניהול מנוע, בקרת אקלים ומערכות ניהול סוללות.

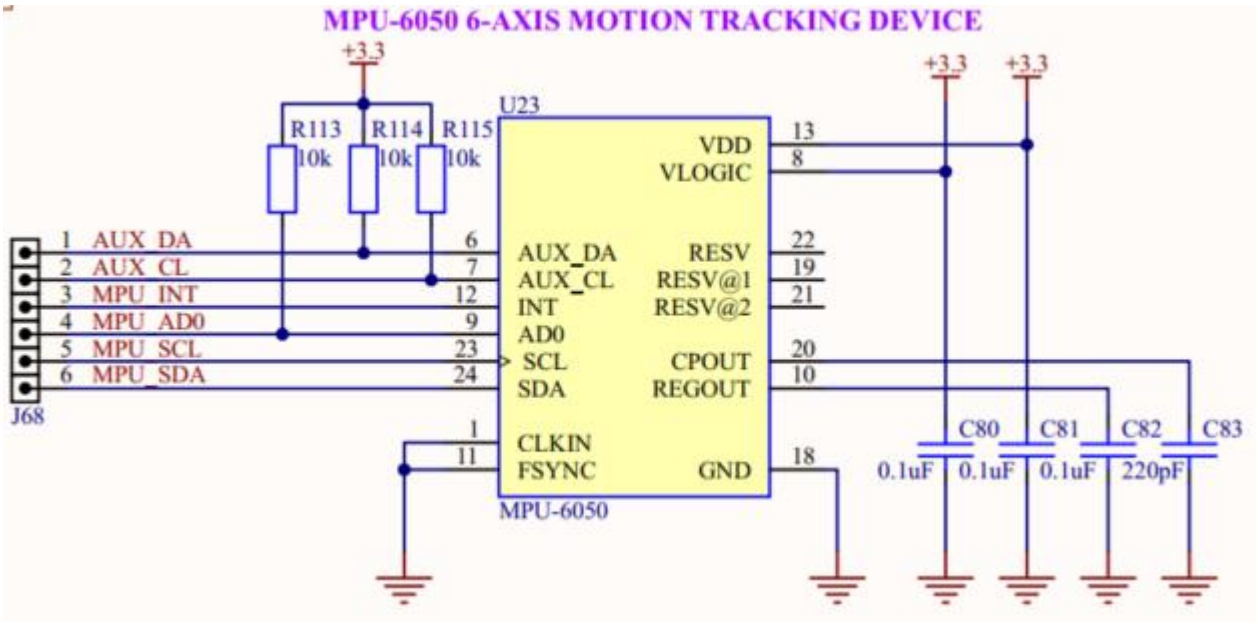
סיכום:

חיישן הטמפרטורה LM75 מציע פתרון רב תכליתי ואמין עבור חישת טמפרטורה ביישומים שונים. עם הממשק הדיגיטלי, הדיוק הגבוה והעיצוב הקומפקטי שלו, ה-LM75 מתאים היטב לניטור ובקרה של טמפרטורה במערכות דיגיטליות. על ידי הבנת התכונות שלו, עקרון העבודה, היתרונות והיישומים שלו, מפתחים יכולים למנף ביעילות את חיישן הטמפרטורה LM75 כדי לשפר את העיצובים שלהם וליצור פתרונות חדשניים לצרכים מגוונים.

שימו לב שקוד של רכיב שלנו הוא 1110 100

```
#include <Wire.h>
#define LM75_ADDR 0x4E // LM75 I2C address
void setup() {
  Serial.begin(9600);
  Wire.begin();
}
void loop() {
  // Request temperature data from LM75
  Wire.beginTransmission(LM75_ADDR);
  Wire.write(0x00); // Register address for temperature data
  Wire.endTransmission();
  // Read 2 bytes of temperature data
  Wire.requestFrom(LM75_ADDR, 2);
  if (Wire.available() == 2) {
    int16_t tempRaw = Wire.read() << 8 | Wire.read();
    // Convert raw temperature data to Celsius
    float temperature = tempRaw / 256.0;
    // Print temperature to Serial Monitor
    Serial.print("Temperature: ");
    Serial.print(temperature);
    Serial.println(" °C");
  }
  delay(1000); // Delay for 1 second
}
```

מד תאוצה I2C.



חקר חיישן MPU-6050: מדריך מקיף

ה-MPU-6050 הוא מודול חיישן מד תאוצה וגירוסקופ פופולרי בשימוש נרחב בפרויקטים אלקטרוניים שונים, רובוטיקה ויישומי חישת תנועה. מיוצר על ידי InvenSense (כיום TDK), מודול חיישן זה מציע דיוק גבוה, צריכת חשמל נמוכה ויכולות מגוונות. במאמר זה, נתעמק בתכונות המפתח, עקרון העבודה, היתרונות, היישומים ושיקולי השימוש של חיישן MPU-6050.

תכונות עיקריות:

מד תאוצה: ה-MPU-6050 כולל מד תאוצה תלת צירים המסוגל למדוד תאוצה לאורך שלושה צירים אורתוגונליים (X, Y, Z).

ג'ירוסקופ: בנוסף למד התאוצה, ה-MPU-6050 כולל ג'ירוסקופ תלת צירים למדידת מהירות זוויתית סביב אותם שלושה צירים.

היתוך חיישן משולב: ה-MPU-6050 משלב נתוני מד תאוצה וג'ירוסקופ כדי לספק מעקב אחר תנועה ואומדן כיוון מדויקים.

ממשק דיגיטלי: החיישן מתקשר עם מיקרו-בקרים באמצעות ממשק I2C (Inter-Integrated Circuit), מה שמפשט את האינטגרציה במערכות דיגיטליות.

צריכת חשמל נמוכה: פועל בהספק נמוך, ה-MPU-6050 מתאים ליישומים המופעלים על ידי סוללה ועיצובים חסכוניים באנרגיה.

חיישן טמפרטורה: ה-MPU-6050 כולל חיישן טמפרטורה משולב למדידת טמפרטורת הסביבה.

עקרון עבודה:

מודול החיישן MPU-6050 מכיל חיישני MEMS (מיקרו-אלקטרו-מכניים) עבור מד התאוצה והגירוסקופ. כאשר הם נתונים לתנועה או תאוצה, חיישני ה-MEMS מייצרים אותות חשמליים פרופורציונליים לכוחות המופעלים או למהירויות הזוויתיות. האותות הללו מעובדים לאחר מכן על ידי המעגלים הפנימיים של החיישן, כולל ממירים אנלוגיים לדיגיטליים (ADC) ואלגוריתמים לעיבוד אותות דיגיטלי (DSP). לאחר מכן, הנתונים המעובדים זמינים למיקרו-בקר באמצעות ממשק I2C.

יתרונות:

קומפקטי וקל משקל: מודול החיישן MPU-6050 הוא קומפקטי וקל משקל, מה שהופך אותו למתאים ליישומים ניידים ולבישים.

דיוק גבוה: עם אלגוריתמי היתוך החיישנים המשולבים שלו, ה-MPU-6050 מספק מעקב תנועה מדויק ואומדן כיוון.

צדדיות: מודול החיישן מתאים למגוון רחב של יישומים, כולל חישת תנועה, זיהוי מחוות, רובוטיקה, מל"טים ומציאות מדומה.

קלות אינטגרציה: ה-MPU-6050 מתקשר דרך ממשק I2C בשימוש נרחב, ומאפשר שילוב קל במערכות דיגיטליות.

עלות-יעילות: מודול החיישן MPU-6050 מציע פתרון חסכוני ליישומי חישת תנועה, מה שהופך אותו לנגיש לחובבים, יצרנים ומקצוענים כאחד.

יישומים:

רובוטיקה: ה-MPU-6050 נמצא בשימוש נפוץ ברובוטיקה לייצוב ושליטה בתנועה של פלטפורמות ומניפולטורים רובוטיים.

חישת תנועה: במוצרי אלקטרוניקה ובמכשירים ניידים, ה-MPU-6050 מאפשר תכונות חישת תנועה כגון סיבוב מסך, זיהוי מחוות ובקרות משחקים.

ניווט אינרציאלי: מודול החיישן מופעל במערכות ניווט ובמל"טים (כלי טיס בלתי מאוישים) עבור ניווט אינרציאלי והערכת גישה.

בריאות וכושר: במכשירים לבישים כגון עוקבי כושר ושעונים חכמים, ה-MPU-6050 מספק יכולות מעקב אחר תנועה לניטור פעילות ואינטראקציות מבוססות מחוות.

אוטומציה תעשייתית: ה-MPU-6050 משמש במערכות אוטומציה ובקרה תעשייתיות לניטור וניתוח רעידות ותנועות מכונות.

שיקולי שימוש:

כיול: ייתכן שיהיה צורך בכיול מודול החיישן של MPU-6050 כדי לפצות על סחיפה והטיות בחיישנים, כדי להבטיח מעקב אחר תנועה ואומדן כיוון מדויק.

כיוון הרכבה: כיוון הרכבה נכון של מודול החיישן חיוני למדידה מדויקת של תנועה וכיוון. עיין בגיליון הנתונים לקבלת הנחיות לגבי תצורות הרכבה אופטימליות.

עיבוד נתונים: ייתכן שיידרשו יישום אלגוריתמי היתוך חיישנים וטכניקות סינון כדי לחלץ מידע משמעותי מנתוני החיישן הגולמיים, במיוחד בסביבות דינמיות ורועשות.

ניהול צריכת חשמל: הטמעת אסטרטגיות לניהול צריכת חשמל, כגון מצבי צריכת חשמל נמוכה ומצבי שינה של חיישנים, יכולה לעזור לייעל את צריכת החשמל ביישומים המופעלים על ידי סוללה.

סיכום:

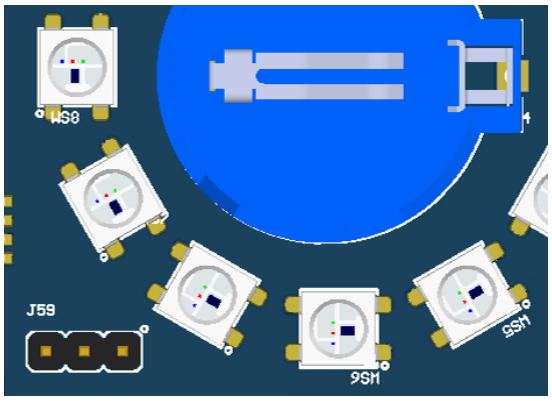
מודול החיישן MPU-6050 מציע פתרון רב תכליתי ואמין עבור חישת תנועה ואומדן כיוון ביישומים שונים. עם מד התאוצה המשולב, הגירוסקופ, יכולות היתוך חיישנים והממשק הדיגיטלי, ה-MPU-6050 מתאים לרובוטיקה, מוצרי צריכה אלקטרוניים, מכשירי בריאות וכושר, אוטומציה תעשייתית ועוד. על ידי הבנת התכונות שלו, עקרון העבודה, היתרונות,

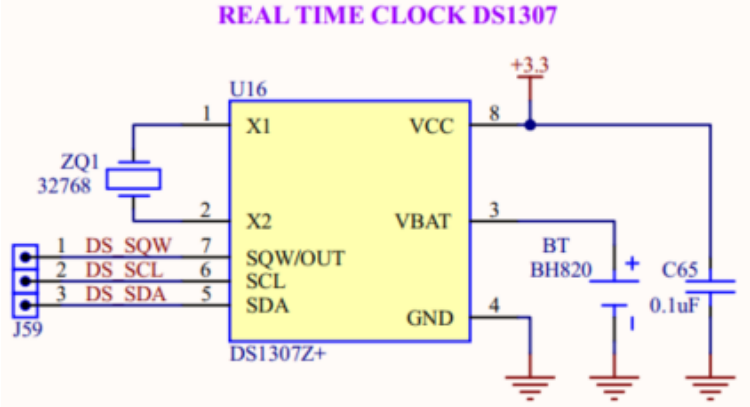
היישומים ושיקולי השימוש שלו, מפתחים יכולים למנף ביעילות את מודול החיישן MPU-6050 כדי לשפר את העיצובים שלהם וליצור פתרונות חדשניים לצרכים מגוונים.

להלן קוד הפעלת רכיב MPU-6050.

```
#include <Wire.h>
#include <MPU6050.h>

MPU6050 mpu;
void setup() {
  Serial.begin(9600);
    Wire.begin();
  mpu.initialize();
    // Initialize the MPU-6050 sensor
  while (!mpu.testConnection()) {
    Serial.println("MPU-6050 connection failed");
    delay(1000);
  }
  Serial.println("MPU-6050 connection successful");
}
void loop() {
  // Read accelerometer and gyroscope data
  int16_t ax, ay, az, gx, gy, gz;
  mpu.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);
    // Print data to Serial Monitor
  Serial.print("Accelerometer (mg): ");
  Serial.print(ax); Serial.print(", ");
  Serial.print(ay); Serial.print(", ");
  Serial.println(az);
   Serial.print("Gyroscope (deg/s): ");
  Serial.print(gx); Serial.print(", ");
  Serial.print(gy); Serial.print(", ");
  Serial.println(gz);
    delay(1000); // Delay for 1 second
}
```

שעון זמן אמת I2C.





הבנת ה-RTC DS1307: מדריך מקיף

ה-DS1307 הוא מודול שעון זמן אמת (RTC) המיוצר על ידי Maxim Integrated. הוא מספק פונקציות מדויקות של שמירת זמן ולוח שנה, מה שהופך אותו לבחירה פופולרית עבור פרויקטים אלקטרוניים שונים, כולל שעונים, יומני נתונים ויישומי תזמון. במאמר זה, נתעמק בתכונות המפתח, עקרון העבודה, היתרונות, היישומים ושיקולי השימוש של מודול RTC DS1307.

תכונות עיקריות:

שעון זמן אמת: מודול ה-DS1307 כולל שעון זמן אמת/שבב לוח שנה עם מעגל מתנד מובנה לשמירת זמן מדויקת.

גיבוי סוללה: סוללת מטבע קטנה מספקת כוח גיבוי ל-DS1307, ומבטיחה שנתוני שמירת זמן נשמרים גם כאשר אספקת החשמל הראשית מנותקת.

ממשק I2C: ה-DS1307 מתקשר עם מיקרו-בקרים באמצעות ממשק I2C (מעגל משולב), המאפשר שילוב פשוט במערכות דיגיטליות.

צריכת חשמל נמוכה: פועל בהספק נמוך, ה-DS1307 מתאים ליישומים המופעלים על ידי סוללה ועיצובים חסכוניים באנרגיה.

אזעקה ניתנת לתכנות: ה-DS1307 כולל שתי אזעקות הניתנות לתכנות שיכולות להפעיל אותות פסיקה בזמנים או בתאריכים ספציפיים.

פלט גל ריבועי: בנוסף לשמירת זמן, ה-DS1307 יכול להפיק אות גל ריבועי עם תדר שניתן לתכנות עבור יישומי שעון/לוח שנה.

עקרון עבודה:

מודול DS1307 מכיל שבב שעון זמן אמת (DS1307) וסוללת מטבע קטנה. שבב DS1307 משתמש במעגל מתנד גבישי קוורץ משולב כדי ליצור פולסי שעון במרווחים מדויקים. פעימות השעון הללו נספרות על ידי ה-DS1307 כדי לשמור על מידע מדויק על זמן, כולל שניות, דקות, שעות, יום, תאריך, חודש ושנה. נתוני שמירת הזמן נגישים למיקרו-בקרים דרך ממשק I2C, ומאפשרים פעולות קריאה וכתיבה כדי להגדיר או לאחזר מידע על זמן ותאריך.

יתרונות:

דיוק: ה-DS1307 מספק שמירת זמן מדויקת עם דיוק טיפוסי של ±2ppm (חלקים למיליון), מה שמבטיח ביצועים אמינים ביישומים שונים.

קלות שילוב: עם ממשק I2C וספריות זמינות רבות, ניתן לשלב את ה-DS1307 בקלות במערכות דיגיטליות באמצעות מיקרו-בקרים פופולריים כמו Arduino.

גיבוי סוללה: גיבוי הסוללה המובנה מבטיח שנתוני שמירת זמן נשמרים גם בהיעדר מתח חיצוני, ומונע אובדן נתונים במהלך הפסקות חשמל או הפרעות.

צדדיות: מודול DS1307 מתאים למגוון רחב של יישומים, כולל שעונים, רותי נתונים, טיימרים ומתזמני אירועים.

עלות נמוכה: מודול DS1307 מציע פתרון חסכוני להוספת פונקציונליות של שעון זמן אמת לפרויקטים אלקטרוניים, מה שהופך אותו לנגיש לחובבים, יצרנים ואנשי מקצוע כאחד.

יישומים:

שעונים וטיימרים: מודול DS1307 נמצא בשימוש נפוץ בשעונים דיגיטליים, שעונים מעוררים וטיימרים לשמירה על זמן ותזמון מדויקים.

יומני נתונים: ביישומי רישום נתונים, ה-DS1307 מספק פונקציונליות של חותמת זמן להקלטה וניתוח נתונים שנאספו מחיישנים וממכשירים.

מערכות אבטחה: ניתן להשתמש ב-DS1307 במערכות אבטחה והתקני בקרת גישה לרישום אירועים והחתמת זמן של אירועים הקשורים לאבטחה.

אלקטרוניקה לרכב: ביישומי רכב, ה-DS1307 מועסק במערכות מעקב אחר רכב, תצוגות לוח מחוונים ומחשבים מובנים עבור משימות רגישות לזמן.

אוטומציה תעשייתית: מודול DS1307 משמש באוטומציה ובקרה תעשייתית לתזמון משימות, רישום אירועים וסנכרון פעולות.

שיקולי שימוש:

החלפת סוללה: ייתכן שיהיה צורך בהחלפה תקופתית של סוללת המטבע כדי לשמור על פעולת גיבוי אמינה של הסוללה.

כיול: ייתכן שיידרש כיול של מודול DS1307 כדי לפצות על סחיפת השעון ולהבטיח שמירת זמן מדויקת לאורך תקופות ממושכות.

תנאי הפעלה: ודא שמודול DS1307 מופעל בטווחי הטמפרטורה והמתח שצוינו כדי לשמור על ביצועי שמירת זמן מדויקים.

כתובת I2C: כתובת ברירת המחדל של I2C של מודול DS1307 היא 0x68. אם מספר התקני I2C מחוברים לאותו אוטובוס, ודא שאין התנגשויות כתובות.

אתחול: אתחול נכון של מודול DS1307, כולל הגדרת השעה והתאריך הראשוניים, חיוני לפעולה נכונה.

סיכום:

מודול DS1307 RTC מציע פתרון רב-תכליתי ואמין עבור שמירת זמן מדויקת ופונקציות לוח שנה בפרויקטים אלקטרוניים שונים. עם שבב שעון זמן אמת משולב, גיבוי סוללה, ממשק I2C ותכונות הניתנות לתכנות, ה-DS1307 מתאים היטב לשעונים, לוגרי נתונים, טיימרים ויישומים רגישים לזמן אחרים. על ידי הבנת התכונות שלו, עקרון העבודה, היתרונות, היישומים ושיקולי השימוש שלו, מפתחים יכולים למנף ביעילות את מודול DS1307 RTC כדי לשפר את העיצובים שלהם וליצור פתרונות חדשניים לצרכים מגוונים.

רשימת פונקציות של RTC:

begin(): מאתחל תקשורת עם מודול DS1307 RTC.

isrunning(): בודק אם מודול DS1307 RTC פועל ומגיב ב-true אם כן, או ב-false אם לא.

now(): מחזירה אובייקט DateTime המייצג את התאריך והשעה הנוכחיים ממודול DS1307 RTC.

read(): קורא את התאריך והשעה הנוכחיים ממודול DS1307 RTC ומאחסן אותם במבנה tmElements_t.

write(): כותב את התאריך והשעה שסופקו למודול DS1307 RTC.

set(): מגדיר את התאריך והשעה של מודול DS1307 RTC באמצעות רכיבי זמן בודדים (שנה, חודש, יום, שעה, דקה, שנייה).

adjust(): מכוונן את הזמן של מודול DS1307 RTC באמצעות רכיבי זמן בודדים (שנה, חודש, יום, שעה, דקה, שנייה).

chipPresent(): בודק אם מודול DS1307 RTC קיים ומגיב ב-true אם כן, או ב-false אם לא.

להלן קוד להפעלת RTC.

```
#include <Wire.h>
#include <DS1307RTC.h>

void setup() {
  Serial.begin(9600);
  while (!Serial) ; // Wait for Serial Monitor to open

  // Initialize the I2C communication
  Wire.begin();

  // Check if the RTC is running
  if (!RTC.isrunning()) {
    Serial.println("RTC is NOT running!");
    // Uncomment the line below to set the RTC to the date and time
this sketch was compiled
    //RTC.begin(DateTime(__DATE__, __TIME__));
  }
}

void loop() {
  // Get the current time from the RTC
  DateTime now = RTC.now();

  // Print the current date and time to Serial Monitor
  Serial.print(now.year(), DEC);
  Serial.print('/');
  Serial.print(now.month(), DEC);
  Serial.print('/');
  Serial.print(now.day(), DEC);
  Serial.print(" ");
  Serial.print(now.hour(), DEC);
  Serial.print(':');
  Serial.print(now.minute(), DEC);
  Serial.print(':');
  Serial.print(now.second(), DEC);
  Serial.println();

  delay(1000); // Delay for 1 second
}
```

פרק 6. פרוטוקול תקשורת One Wire.

מבוא לפרוטוקול חוט אחד:

פרוטוקול One-Wire הוא פרוטוקול תקשורת טורית שפותח על ידי Dallas Semiconductor (כיום Maxim Integrated) המאפשר העברת נתונים על חוט בודד (בנוסף להארקה). הוא נמצא בשימוש נרחב לתקשורת בין מיקרו-בקרים (כגון ESP32) והתקנים כגון חיישני טמפרטורה, שבבי זיכרון ואסימוני אימות. בהסבר מפורט זה, נפרק את פרוטוקול One-Wire טיפין טיפין ונבדוק כיצד ניתן ליישם אותו במיקרו-בקר ESP32.

1. תקשורת חוט יחיד:

כפי שהשם מרמז, פרוטוקול One-Wire דורש רק חוט בודד לתקשורת, בנוסף לחיבור הארקה. זה הופך אותו לפתרון חסכוני עבור יישומים שבהם רצוי לצמצם את מספר החוטים.

2. ריבוי תחום זמן:

אחת התכונות המרכזיות של פרוטוקול One-Wire היא השימוש שלו בריבוי תחום זמן. הנתונים מקודדים באמצעות וריאציות של משך הפולסים על החוט היחיד, מה שמאפשר גם העברת נתונים וגם אספקת חשמל למכשירים על אותו חוט.

3. דופק נוכחות (אתחול):

התקשורת באפיק ה-One-Wire מתחילה בפולס נוכחות, המשמש כאות אתחול. כדי ליזום תקשורת, התקן הראשי (ESP32) מושך לרגע את קו הנתונים נמוך ואז משחרר אותו. הנוכחות של מכשיר עבד מסומנת על ידי תגובת דופק הנוכחות מהתקן העבד.

4. פקודה והעברת נתונים:

לאחר פעימת הנוכחות, המכשיר הראשי יכול להנפיק פקודות ולשדר נתונים למכשיר העבד על ידי קידוד מידע בתזמון הפולסים בקו הנתונים. נתונים בינאריים מועברים ביט אחר סיביות, כאשר כל סיביות מיוצגות על ידי תבנית תזמון מסוימת.

5. אספקת כוח (כוח טפילי):

בנוסף להעברת נתונים, פרוטוקול One-Wire תומך גם באספקת חשמל למכשירים על אותו חוט. חלק מהתקני One-Wire יכולים לפעול במצב חשמל טפילי, שבו הם שואבים חשמל מקו הנתונים במהלך התקשורת.

6. נגד משיכה:

נגד משיכה מחובר בין קו הנתונים לאספקת החשמל (VCC) כדי להבטיח שקו הנתונים יישאר ברמה לוגית גבוהה כאשר הוא אינו מופעל באופן פעיל נמוך על ידי התקן. הערך של הנגד המשוך משפיע על המהירות והאמינות של התקשורת באוטובוס One-Wire.

יישום ב-ESP32:

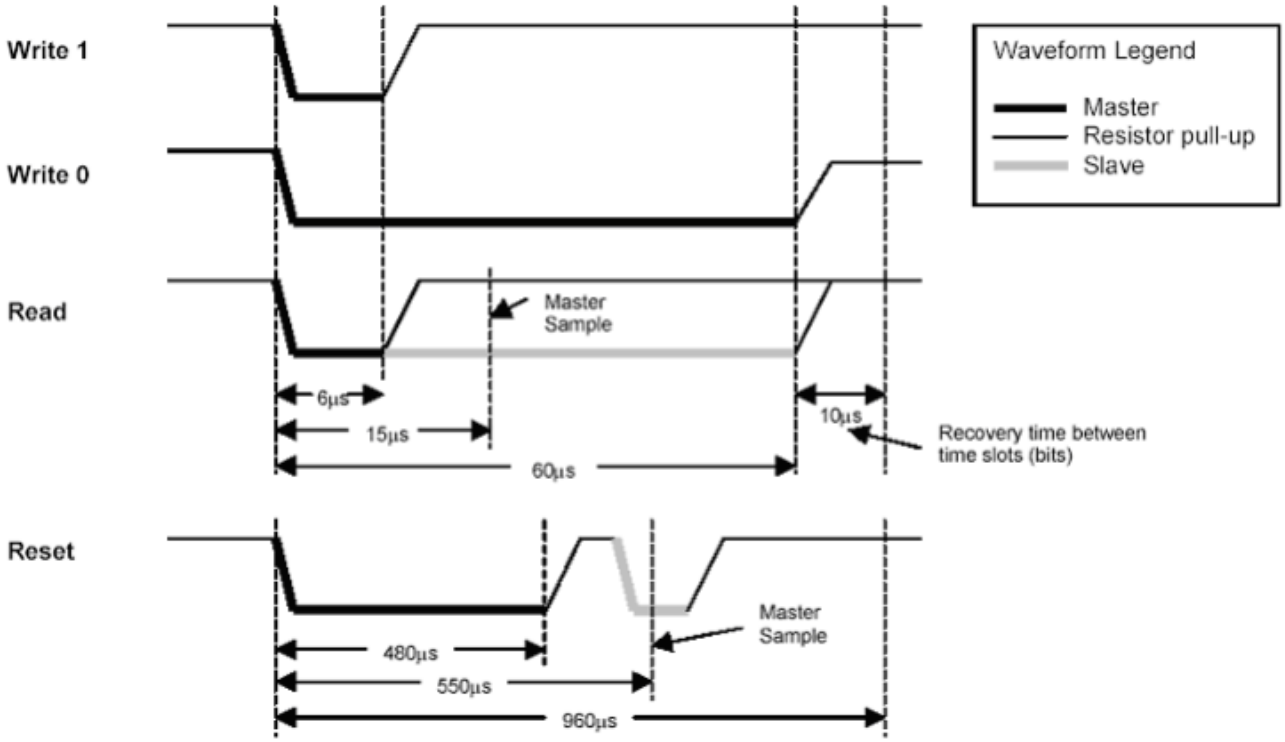
הטמעת פרוטוקול One-Wire במיקרו-בקר ESP32 כרוכה בדרך כלל בהגדרת אחד מפיני ה-GPIO לפעול כקו הנתונים ושליטה במצבו לשדר ולקבל נתונים בהתאם לדרישות התזמון של הפרוטוקול. ניתן להשתמש בספריות כגון ספריית OneWire כדי לפשט את התקשורת עם התקני One-Wire בפרויקטים של ESP32.

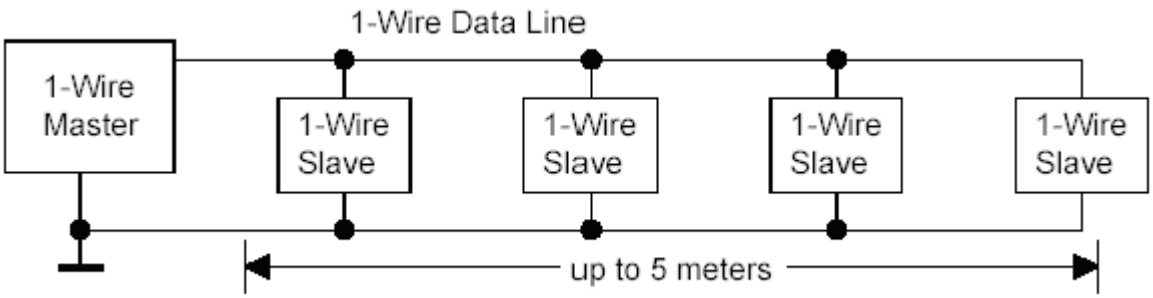
לסיכום:

פרוטוקול One-Wire הוא פרוטוקול תקשורת טורית פשוט ויעיל המאפשר העברת נתונים על חוט בודד. על ידי הבנת פעולתו ביט אחר ביט והטמעתה בפרויקטים של ESP32, מפתחים יכולים להתממשק עם מגוון רחב של התקני One Wire עבור יישומים שונים, כולל חישת טמפרטורה, רישום נתונים ואימות.

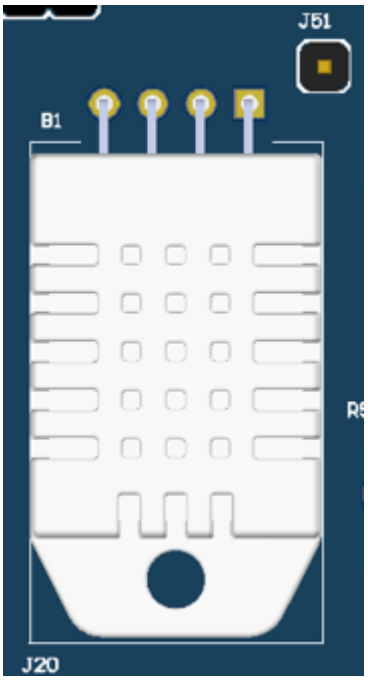
(למידע מלא: https://www.mikrocontroller.net/attachment/239139/One-Wire-Bussysteme.pdf).

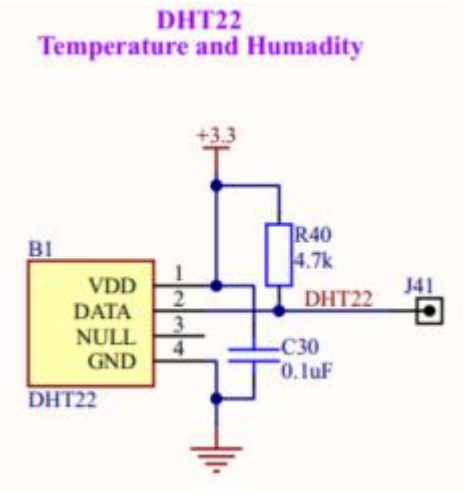
Operation	Description	Implementation
Write 1 bit	Send a '1' bit to the 1-Wire slaves (Write 1 time slot)	Drive bus low, delay 6µs Release bus, delay 64µs
Write 0 bit	send a '0' bit to the 1-Wire slaves (Write 0 time slot)	Drive bus low, delay 60µs Release bus, delay 10µs
Read bit	Read a bit from the 1-Wire slaves (Read time slot)	Drive bus low, delay 6µs Release bus, delay 9µs Sample bus to read bit from slave Delay 55µs
Reset	Reset the 1-Wire bus slave devices and ready them for a command	Drive bus low, delay 480µs Release bus, delay 70µs Sample bus, 0 = device(s) present, 1 = no device present Delay 410µs





חיישן טמפרטורה ולחות DHT22.





הבנת חיישן DHT22: מדריך מקיף

חיישן DHT22, הידוע גם בשם AM2302, הוא חיישן טמפרטורה ולחות דיגיטלי רב תכליתי ובעל שימוש נרחב. הוא מספק קריאות מדויקות של טמפרטורה ולחות יחסית, מה שהופך אותו מתאים ליישומים שונים בניטור סביבתי, מערכות HVAC, תחנות מזג אוויר ועוד. במאמר זה, נחקור את תכונות המפתח, עקרון העבודה, היתרונות, היישומים ושיקולי השימוש של חיישן DHT22.

תכונות עיקריות:

חיישן דיגיטלי: חיישן DHT22 הוא חיישן דיגיטלי שמוציא נתוני טמפרטורה ולחות בפורמט דיגיטלי, מה שמקל על התממשק עם מיקרו-בקרים.

דיוק גבוה: עם דיוק טמפרטורה של C±0.5° ודיוק לחות של ±2%, ה-DHT22 מספק מדידות אמינות לניטור תנאי סביבה.

טווח פעולה רחב: חיישן DHT22 יכול למדוד טמפרטורות הנעות בין -C40° עד C80° ולחות יחסית הנעה בין 0% ל-100%.

זמן תגובה מהיר: לחיישן DHT22 יש זמן תגובה מהיר, המאפשר לו לספק קריאות טמפרטורה ולחות בזמן אמת.

צריכת חשמל נמוכה: חיישן DHT22 פועל בהספק נמוך, מתאים ליישומים המופעלים על ידי סוללה ועיצובים חסכוניים באנרגיה.

עקרון עבודה:

חיישן DHT22 משתמש באלמנט חישת לחות קיבולי ותרמיסטור למדידת טמפרטורה. כאשר הוא נחשף לאוויר, אלמנט חישת הלחות סופג או משחרר לחות, מה שגורם לשינויים בקיבול. החיישן ממיר את השינויים הללו בקיבול לאותות דיגיטליים המייצגים לחות יחסית. במקביל, התרמיסטור מודד את הטמפרטורה, והחיישן ממיר את אות הטמפרטורה האנלוגי לפורמט דיגיטלי. חיישן DHT22 משלב את האותות הדיגיטליים הללו כדי לספק קריאות מדויקות של טמפרטורה ולחות.

יתרונות:

דיוק גבוה: חיישן DHT22 מציע דיוק גבוה במדידות טמפרטורה ולחות, מה שמבטיח ביצועים אמינים ביישומים שונים.

קל לשימוש: עם הפלט הדיגיטלי והממשק הפשוט שלו, חיישן DHT22 קל לשילוב בפרויקטים מבוססי מיקרו-בקר.

חסכוני: חיישן DHT22 מספק מדידות מדויקות של טמפרטורה ולחות במחיר סביר, מה שהופך אותו לנגיש לחובבים, יצרנים ומקצוענים כאחד.

צדדיות: חיישן DHT22 מתאים למגוון רחב של יישומים, לרבות ניטור אקלים פנימי, בקרת חממה, מערכות HVAC, תחנות מזג אוויר ועוד.

עיצוב קומפקטי: העיצוב הקומפקטי והקל משקל של חיישן DHT22 מאפשר התקנה קלה בסביבות מוגבלות בחלל.

יישומים:

ניטור סביבתי: חיישן DHT22 נמצא בשימוש נפוץ במערכות ניטור סביבתיות למדידת רמות טמפרטורה ולחות בסביבות פנימיות וחיצוניות.

מערכות HVAC: במערכות חימום, אוורור ומיזוג אוויר (HVAC), חיישן DHT22 מספק משוב על תנאי הסביבה לבקרת טמפרטורה ולחות.

תחנות מזג אוויר: חיישן DHT22 משמש בתחנות מזג אוויר למדידת תנאי אטמוספירה ומתן נתוני מזג אוויר בזמן אמת.

בקרת חממה: ביישומי חקלאות וגננות, חיישן DHT22 משמש לניטור ובקרה של רמות טמפרטורה ולחות בחממות ובחדרי גידול.

בקרת אקלים פנימית: חיישן DHT22 עוזר לשמור על אקלים פנימי נוח בבתים, משרדים ובניינים מסחריים על ידי ניטור רמות טמפרטורה ולחות.

שיקולי שימוש:

כיול: חיישן DHT22 עשוי לדרוש כיול תקופתי כדי לשמור על דיוק לאורך זמן, במיוחד בסביבות קשות או לאחר שימוש ממושך.

תנאי הפעלה: ודא שחיישן DHT22 מופעל בטווחי הטמפרטורה והלחות שצוינו כדי למנוע נזק ולהבטיח מדידות מדויקות.

מיקום הרכבה: מיקום הרכבה נכון של חיישן DHT22 חיוני למדידות מדויקות של טמפרטורה ולחות. הימנע מהצבת החיישן בקרבת מקורות חום, אור שמש ישיר או מקורות לחות שעלולים להשפיע על קריאותיו.

יציבות אות: הבטח אותות מתח ותקשורת יציבים לחיישן DHT22 כדי למנוע השחתת נתונים או הפרעות אות.

עיבוד נתונים: הטמע אלגוריתמים מתאימים לעיבוד נתונים כדי לסנן רעשים וחריגים בקריאות החיישנים, במיוחד בסביבות דינמיות עם שינויים מהירים בטמפרטורה ולחות.

סיכום:

חיישן DHT22 מציע פתרון אמין וחסכוני למדידת טמפרטורה ולחות ביישומים שונים. עם הדיוק הגבוה, קלות השימוש והרבגוניות שלו, חיישן DHT22 מתאים היטב לניטור סביבתי, מערכות HVAC, תחנות מזג אוויר ועוד. על ידי הבנת התכונות שלו, עקרון העבודה, היתרונות, היישומים ושיקולי השימוש שלו, מפתחים יכולים למנף ביעילות את חיישן DHT22 כדי לשפר את העיצובים שלהם וליצור פתרונות חדשניים לצרכים מגוונים.

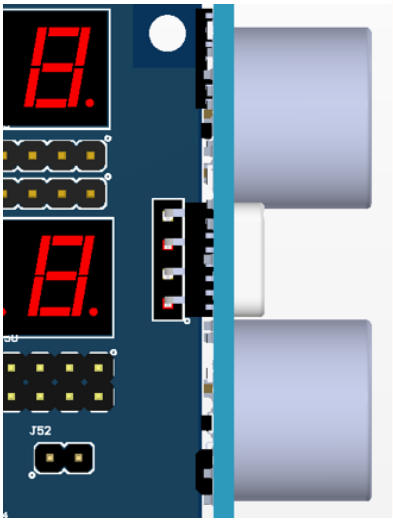
להלן קוד הפעלה לחיישן DHT22.

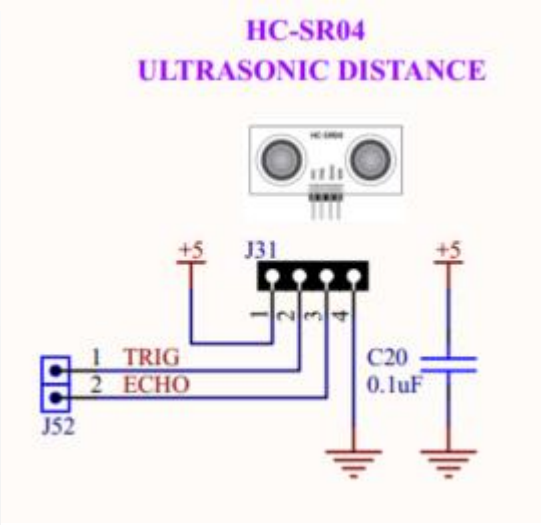
```
#include <DHT.h>

#define DHT_PIN 2       // Pin connected to the DHT22 sensor
#define DHT_TYPE DHT22 // DHT sensor type

DHT dht(DHT_PIN, DHT_TYPE);

void setup() {
  Serial.begin(9600);
```

```
  dht.begin();
}

void loop() {
  delay(2000); // Wait for 2 seconds between readings

  // Read temperature and humidity data from the sensor
  float temperature = dht.readTemperature();
  float humidity = dht.readHumidity();

  // Check if any reads failed
  if (isnan(temperature) || isnan(humidity)) {
    Serial.println("Failed to read from DHT sensor!");
    return;
  }

  // Print temperature and humidity to Serial Monitor
  Serial.print("Temperature: ");
  Serial.print(temperature);
  Serial.print(" °C, Humidity: ");
  Serial.print(humidity);
  Serial.println(" %");
}
```

פרק 7. הפעלת רכיבים מיוחדים.

חיישן מרחק אולטרה סוני.





הבנת החיישן האולטראסוני SRF05: מדריך מקיף

ה-SRF05 הוא חיישן מרחק קולי פופולרי הידוע בפשטותו, הדיוק והרבגוניות שלו. הוא נמצא בשימוש נרחב ברובוטיקה, אוטומציה ופרויקטים אלקטרוניים שונים הדורשים יכולות מדידת מרחק. במאמר זה, נחקור את תכונות המפתח, עקרון העבודה, היתרונות, היישומים ושיקולי השימוש של החיישן האולטראסוני SRF05.

תכונות עיקריות:

חישת מרחק קולי: חיישן SRF05 מודד מרחק על ידי פליטת פולסים קוליים וחישוב הזמן שלוקח לפולסים לקפוץ מאובייקט ולחזור לחיישן.

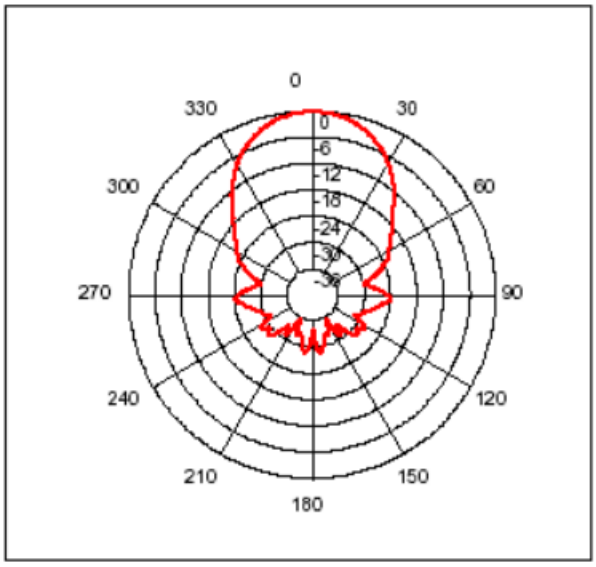
דיוק גבוה: עם רזולוציה של 1 ס"מ ודיוק של עד 3 מ"מ, ה-SRF05 מספק מדידות מרחק מדויקות בסביבות שונות.

טווח פעולה רחב: החיישן יכול לזהות עצמים בטווח של 2 ס"מ עד 450 ס"מ, מה שהופך אותו מתאים ליישומי חישת מרחק קצר וארוך כאחד.

ממשק פשוט: ה-SRF05 מתקשר עם מיקרו-בקרים באמצעות ממשק טריגר והד פשוט, מה שמקל על שילובו בפרויקטים אלקטרוניים.

צריכת חשמל נמוכה: פועל בהספק נמוך, ה-SRF05 מתאים ליישומים המופעלים על ידי סוללה ותכנונים חסכוניים באנרגיה.

שימו לב על סביבה שחיישן דוגם:



עקרון עבודה:

החיישן האולטראסוני SRF05 פועל על בסיס עקרון ההד. הוא פולט פרץ של פולסים על-קוליים ומודד את הזמן שלוקח לפולסים לעבור לעצם, להשתקף ממנו ולחזור לחיישן. לאחר מכן החיישן מחשב את המרחק לאובייקט בהתבסס על הזמן שלוקח לסיבוב הלוך ושוב של הפולסים האולטראסוניים. על ידי מדידת ההשהיה בין שליחת דופק ההדק לבין קבלת דופק ההד, ה-SRF05 יכול לקבוע במדויק את המרחק לאובייקטים בשדה הראייה שלו.

Distance (in cm) = (elapsed time * sound velocity (340 m/s)) / 100 / 2

יתרונות:

דיוק: חיישן SRF05 מציע דיוק ורזולוציה גבוהים במדידות מרחק, מה שהופך אותו למתאים למשימות מיקום וניווט מדויקים.

צדדיות: עם טווח הפעולה הרחב והממשק הפשוט שלו, חיישן SRF05 הוא רב תכליתי וניתן להשתמש בו במגוון רחב של יישומים, לרבות רובוטיקה, זיהוי מכשולים, סיוע בחניה ועוד.

עלות-תועלת: חיישן SRF05 מספק יכולות חישת מרחק אמינות במחיר סביר, מה שהופך אותו לנגיש לחובבים, יצרנים ומקצוענים כאחד.

קלות שימוש: חיישן SRF05 קל לשימוש ומשתלב בפרויקטים אלקטרוניים, הודות לממשק ההדק וההד הפשוטים שלו וספריות זמינות וקודים לדוגמה.

עיצוב קומפקטי: העיצוב הקומפקטי וקל המשקל של חיישן SRF05 מאפשר התקנה והרכבה קלה בתצורות וסביבות שונות.

יישומים:

רובוטיקה: חיישן SRF05 נמצא בשימוש נפוץ ברובוטיקה למשימות הימנעות ממכשולים, ניווט וזיהוי עצמים.

אוטומציה: באוטומציה ויישומים תעשייתיים, חיישן SRF05 משמש למדידת מרחק ומיקום במערכות מסועים, ציוד לטיפול בחומרים ועוד.

סיוע בחניה: חיישן SRF05 משמש ביישומי רכב עבור מערכות סיוע בחניה, ועוזר לנהגים לאמוד את המרחק למכשולים בעת חניה או תמרון.

מערכות אבטחה: במערכות אבטחה ויישומי מעקב, חיישן SRF05 מספק יכולות חישת קרבה לאיתור פולשים או גישה לא מורשית.

אלקטרוניקה לצרכן: חיישן SRF05 משמש במכשירי אלקטרוניקה לצרכן כגון גאדג'טים לבית חכם, רחפנים ומדדי מרחק כף יד עבור פונקציות שונות של חישת מרחק.

שיקולי שימוש:

מיקום הרכבה: מיקום ההרכבה והכיוון הנכון של חיישן SRF05 חיוניים למדידת מרחק מדויקת. הימנע מחסימות והבטח קו ראייה ברור בין החיישן לבין העצמים הנמדדים.

הפרעות אות: צמצמו למינימום הפרעות אות מחיישנים קוליים אחרים או גורמים סביבתיים כגון רעש אקוסטי, רוח או שינויים בטמפרטורה כדי לשמור על דיוק המדידה.

כיול: כייל את חיישן SRF05 במידת הצורך כדי לפצות על סטיות או אי דיוקים במדידות מרחק, במיוחד בתנאי הפעלה לא אידיאליים.

ספק כוח: ודא אספקת חשמל יציבה לחיישן SRF05 כדי למנוע תנודות מתח או תקלות שעלולות להשפיע על הביצועים והדיוק שלו.

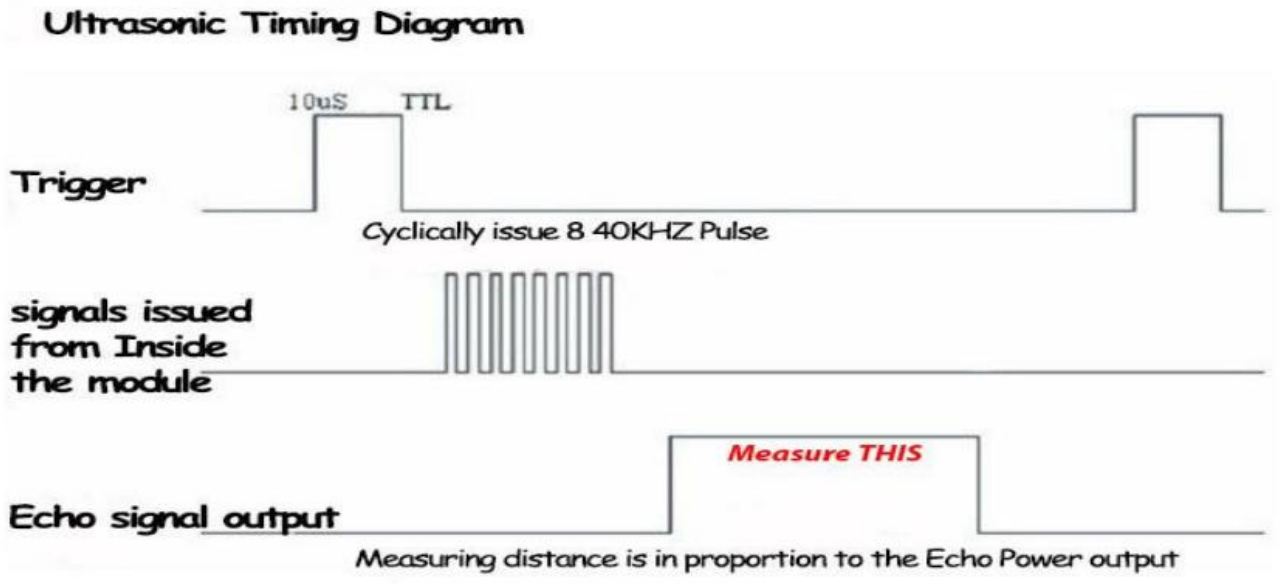
תנאים סביבתיים: קחו בחשבון גורמים סביבתיים כגון טמפרטורה, לחות ותכונות פני השטח (למשל, רפלקטיביות) בעת השימוש בחיישן SRF05, מכיוון שהם יכולים להשפיע על דיוק מדידת המרחק.

סיכום:

החיישן האולטראסוני SRF05 מציע פתרון אמין וחסכוני עבור חישת מרחק ביישומים שונים. עם הדיוק הגבוה, טווח הפעולה הרחב וקלות השימוש שלו, חיישן SRF05 מאומץ באופן נרחב בתעשיות רובוטיקה, אוטומציה, רכב ואלקטרוניקה צריכה. על ידי הבנת התכונות, עקרון העבודה, היתרונות, היישומים ושיקולי השימוש שלה, מפתחים יכולים למנף ביעילות.

נראה את דפי נתונים של הרכיב:

https://www.micros.com.pl/mediaserver/M_HY-SRF05_0003.pdf



נראה את הדוגמה של הקוד עם ספריות:

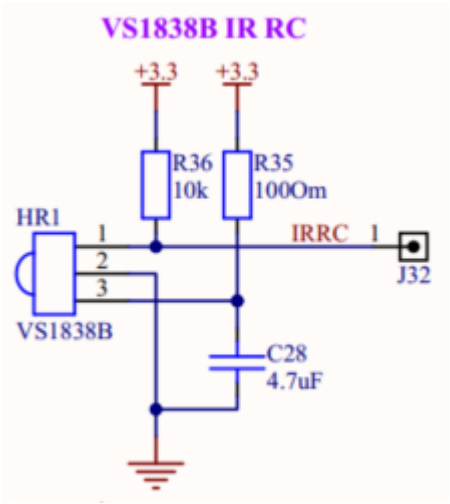
```
#include <NewPing.h>
```

```
#define TRIGGER_PIN  12  // Pin connected to the SRF05 trigger
#define ECHO_PIN     14  // Pin connected to the SRF05 echo
#define MAX_DISTANCE 200 // Maximum distance in centimeters

NewPing sonar(TRIGGER_PIN, ECHO_PIN, MAX_DISTANCE);

void setup() {
  Serial.begin(9600);
}

void loop() {
  delay(50); // Wait between measurements
  unsigned int distance = sonar.ping_cm(); // Send a ping and get the
distance in centimeters
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
}
```

ועכשיו קוד ללא ספרייה, אלא עם גישה ישירה לפורטים.

```
#define TRIGGER_PIN  12  // Pin connected to the SRF05 trigger
#define ECHO_PIN     14  // Pin connected to the SRF05 echo
void setup() {
  Serial.begin(9600);
  pinMode(TRIGGER_PIN, OUTPUT);
  pinMode(ECHO_PIN, INPUT);
}
void loop() {
  // Send a 10us pulse to the trigger pin
  digitalWrite(TRIGGER_PIN, LOW);
  delayMicroseconds(2);
  digitalWrite(TRIGGER_PIN, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIGGER_PIN, LOW);
  // Measure the duration of the echo pulse
  unsigned long duration = pulseIn(ECHO_PIN, HIGH);
  // Calculate distance in centimeters
  unsigned int distance = duration / 58;
  Serial.print("Distance: ");
  Serial.print(distance);
  Serial.println(" cm");
  delay(100); // Wait between measurements
}
```

חיישן אינפרה אדום.



הבנת פרוטוקול NEC (RECS80) למקלטי IR: מדריך מקיף

פרוטוקול NEC (ביפנית: Hepburn: Sekigaisen Tsūshin Kikaku, 赤外線通信規格, מילולית "תקן תקשורת אינפרא אדום"), הידוע גם כפרוטוקול RECS80, הוא פרוטוקול תקשורת אינפרא אדום בשימוש נרחב עבור יישומי שלט רחוק. הוא מגדיר שיטה סטנדרטית לקידוד ופענוח נתונים המועברים באמצעות אותות אינפרא אדום בין שלטים רחוקים והתקנים אלקטרוניים כגון טלוויזיות, מזגנים ומערכות בידור ביתיות. במאמר זה, נעמיק בתכונות המפתח, פורמט הקידוד, פרמטרי התזמון, היתרונות, היישומים ושיקולי השימוש של פרוטוקול NEC עבור מקלטי IR.

תכונות עיקריות:

פשוט ואיתן: פרוטוקול NEC פשוט ליישום ומספק תקשורת חזקה בין שלטים רחוקים והתקנים אלקטרוניים.

אומץ נרחב: זהו אחד מפרוטוקולי ה-IR הנפוצים ביותר בעולם, מה שהופך אותו לתואם למגוון רחב של מוצרי אלקטרוניקה לצרכן.

תדר אפנון: פרוטוקול NEC משתמש בדרך כלל בתדר נשא של כ-38 קילו-הרץ להעברת אותות אינפרא אדום.

מבנה מנות: נתונים המועברים באמצעות פרוטוקול NEC מאורגנים במנות המורכבות מסיבית התחלה, סיביות כתובת, סיביות פקודה וסכום ביקורת לזיהוי שגיאות.

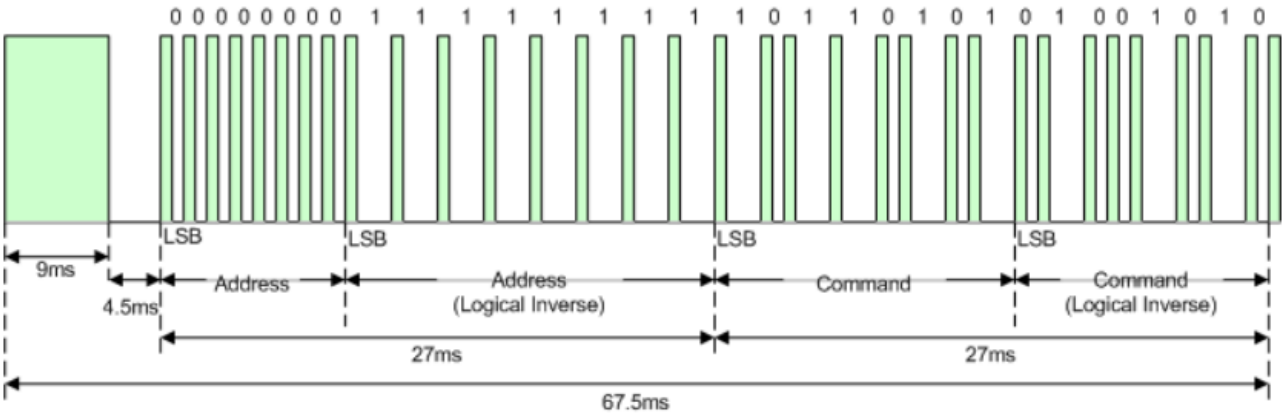
פורמט קידוד:

Start Bit: דופק של 9 ms ואחריו רווח של 4.5 ms כדי לציין את התחלת שידור.

סיביות כתובת: בדרך כלל 8 סיביות המייצגות את כתובת ההתקן או המזהה.

סיביות פקודה: בדרך כלל 8 סיביות המייצגות את הפקודה או הפעולה שיש לבצע על ידי המכשיר המקבל.

Checksum: סכום ביקורת של 1 סיביות המחושב מסיביות הכתובות והפקודה כדי לאמת את שלמות הנתונים.



https://sibotic.files.wordpress.com/2013/12/adoh-necinfraredtransmissionprotocol-281113-1713-47344.pdf

פרמטרי תזמון:

משך סיביות: כל סיביות בפרוטוקול NEC מקודד כפולס של 562.5 µs עבור '1' לוגי או רווח של 562.5 µs עבור '0' לוגי.

משך מסגרת: מסגרת NEC שלמה מורכבת מ-32 ביטים המשודרים ברצף, עם משך כולל של כ-25 ms.

יתרונות:

תאימות: פרוטוקול NEC זוכה לתמיכה רחבה על ידי יצרני מוצרי אלקטרוניקה, מה שמבטיח תאימות בין שלטים רחוקים והתקנים ממותגים שונים.

אמינות: עם מנגנון זיהוי השגיאות החזק שלו, פרוטוקול NEC מספק תקשורת אמינה בסביבות רועשות ולמרחקים ארוכים יותר.

פשטות: פורמט הקידוד הפשוט ופרמטרי התזמון של פרוטוקול NEC מקלים על היישום בחומרה ובתוכנה.

יעילות: השימוש בטכניקת אפנון תדר נושא ו-PPM (PPM) מביא לשידור וקליטה יעילים של נתונים.

עלות נמוכה: מקלטי IR ומפענחי IR תואמי NEC זמינים בקלות בעלות נמוכה, מה שהופך אותם לנגישים עבור פרויקטים של חובבים ועשה זאת בעצמך.

יישומים:

שלט רחוק: פרוטוקול NEC נמצא בשימוש נרחב בשלטים לטלוויזיות, נגני DVD, ממירים, מזגנים ומכשירי אלקטרוניקה אחרים.

אוטומציה ביתית: הוא מנוצל במערכות אוטומציה ביתיות לשליטה על אורות, וילונות ומכשירי חשמל ביתיים אחרים באמצעות שלט אינפראולוגי.

מקלטי משדר אינפרא אדום: מקלטי IR ומשדרים תואמי NEC משמשים במודולי מקלטי אינפרא אדום לתקשורת בין מכשירים ביישומי בית חכם ויישומי IoT.

אוטומציה תעשייתית: פרוטוקול NEC מוצא יישומים באוטומציה תעשייתית ורובוטיקה לשליטה מרחוק של מכונות, ציוד ותהליכי ייצור.

פרויקטים חינוכיים: הוא נמצא בשימוש פופולרי בפרויקטים חינוכיים ובערכות אלקטרוניקה עשה זאת בעצמך ללימוד על תקשורת אינפרא אדום וטכנולוגיית שלט רחוק.

שיקולי שימוש:

קו ראייה: ודא קו ראייה ברור בין משדר IR (שלט רחוק) למקלט IR (התקן) לתקשורת אמינה.

מרחק: הטווח האפקטיבי של פרוטוקול NEC תלוי בגורמים כמו הספק של משדר ה-IR, רגישות מקלט ה-IR ותנאי האור הסביבתי.

הפרעות: צמצמו למינימום הפרעות ממקורות אור אחרים ומכשירים אלקטרוניים הפולטים קרינת אינפרא אדומה כדי למנוע עיוות או אובדן של האות.

כתובת: הקצה כתובות מכשיר ייחודיות כדי למנוע התנגשויות ולוודא שכל מכשיר מגיב רק לפקודות המיועדות לו.

טיפול בשגיאות: הטמע מנגנוני זיהוי שגיאות ותיקון שגיאות כדי לטפל בחבילות נתונים פגומות או לא חוקיות ולהבטיח שלמות הנתונים.

סיכום:

פרוטוקול NEC (RECS80) הוא פרוטוקול תקשורת אינפרא אדום בשימוש נרחב עבור יישומי שלט רחוק, המציע פשטות, אמינות ותאימות במגוון רחב של מכשירים אלקטרוניים לצרכן. על ידי הבנת התכונות שלו, פורמט הקידוד, פרמטרי

התזמון, היתרונות, היישומים ושיקולי השימוש שלו, מפתחים יכולים להשתמש ביעילות בפרוטוקול NEC לצורך הטמעת פונקציונליות שלט רחוק בפרויקטים ויישומים אלקטרוניים שונים.

הבנת חיישן IR VS183 עבור שלטים: מדריך מקיף

ה- VS183 הוא מודול מקלט אינפרא אדום (IR) הנפוץ ביישומי שלט רחוק. הוא נועד לקבל ולפענח אותות אינפרא אדום המועברים על ידי שלטים רחוקים, מה שמאפשר להתקנים להגיב לפקודות קלט של המשתמש. במאמר זה, נחקור את התכונות העיקריות, עקרון העבודה, היתרונות, היישומים ושיקולי השימוש של חיישן ה-IR VS183 לשלטים.

תכונות עיקריות:

רגישות גבוהה: חיישן ה-IR VS183 מציע רגישות גבוהה, המאפשר לו לזהות אותות אינפרא אדום משלטים רחוקים גם במרחקים ארוכים.

זווית קליטה רחבה: יש לו בדרך כלל זווית קליטה רחבה, המאפשרת לו ללכוד אותות IR מכיוונים שונים, ומספקת גמישות ביישומי שלט רחוק.

דמודולטור משולב: מודול VS183 משלב מעגל דמודולטור, המחלץ את הנתונים הדיגיטליים המקודדים באותות ה-IR המאופנים המתקבלים משלטים רחוקים.

תאימות: הוא תואם למגוון רחב של שלטים באמצעות פרוטוקולי שידור IR סטנדרטיים כגון NEC, RC5, RC6, Sony SIRC וכו'.

צריכת חשמל נמוכה: חיישן VS183 פועל בהספק נמוך, מתאים למכשירים המופעלים על ידי סוללה ועיצובים חסכוניים באנרגיה.

עקרון עבודה:

חיישן IR VS183 פועל על בסיס העיקרון של זיהוי פוטודיודות. כאשר נחשפת לקרינת אינפרא אדומה הנפלטת על ידי שלט רחוק, הפוטודיודה בתוך מודול VS183 מייצרת זרם חשמלי קטן פרופורציונלי לעוצמת אות ה-IR המתקבל. לאחר מכן מעגל הדמודולטור מעבד את האות החשמלי הזה, מחלץ את הנתונים הדיגיטליים המקודדים ומוציא אותם בפורמט מתאים לעיבוד נוסף על ידי מיקרו-בקרים או מכשירים אלקטרוניים אחרים.

יתרונות:

פונקציונליות שלט רחוק: חיישן IR VS183 מאפשר למכשירים לקבל ולפרש פקודות משלטים, מה שמקל על אינטראקציה ושליטה של המשתמש.

צדדיות: הוא תואם למגוון רחב של שלט רחוק ופרוטוקולי שידור IR, מה שהופך אותו מתאים ליישומים והתקנים שונים.

קלות שילוב: גורם הצורה הקומפקטי של מודול VS183 והממשק הפשוט מקלים על שילובו בפרויקטים אלקטרוניים והתקנים הדורשים פונקציונליות של שלט רחוק.

אמינות: עם הרגישות הגבוהה ויכולות הדמודולציה החזקות שלו, חיישן VS183 מספק ביצועים אמינים בתנאי הפעלה מגוונים.

עלות-יעילות: חיישן ה-IR VS183 מציע פתרון חסכוני להטמעת יכולות שלט רחוק באלקטרוניקה צריכה, מערכות אוטומציה ביתית, רובוטיקה ועוד.

יישומים:

אלקטרוניקה לצרכן: חיישן IR VS183 נמצא בשימוש נרחב בטלוויזיות, נגני DVD, ממירים, מזגנים ומכשירי אלקטרוניקה אחרים עם פונקציונליות שלט רחוק.

אוטומציה ביתית: היא מועסקת במערכות אוטומציה ביתיות לשליטה על אורות, מכשירי חשמל, תרמוסטטים ומכשירים חכמים אחרים באמצעות שלט אינפראולוגי.

מערכות בידור: חיישן VS183 משמש במערכות אודיו/וידאו, קונסולות משחקים ונגני מדיה לתפעול וניווט מרחוק.

רובוטיקה: ביישומי רובוטיקה, חיישן ה-IR VS183 מאפשר שליטה מרחוק על רובוטים ופלטפורמות רובוטיות להפעלה טלפונית וביצוע פקודות.

בקרה תעשייתית: הוא מוצא יישומים במערכות בקרה תעשייתיות לשליטה מרחוק במכונות, ציוד ותהליכי ייצור באמצעות שלטי IR.

שיקולי שימוש:

קו ראייה: ודא קו ראייה ברור בין חיישן ה-IR VS183 לשלט הרחוק כדי להקל על קליטת אות ופענוח אמינים.

מרחק: הטווח האפקטיבי של חיישן VS183 תלוי בגורמים כמו עוצמת השלט הרחוק, רגישות החיישן ותנאי אור הסביבה.

הפרעות: צמצמו למינימום הפרעות ממקורות אור אחרים ומכשירים אלקטרוניים הפולטים קרינת אינפרא אדומה כדי למנוע עיוות או אובדן של האות.

תאימות פרוטוקול: ודא שחיישן VS183 תואם לפרוטוקול שידור IR המשמש את השלט הרחוק כדי להבטיח פענוח אותות ופרשנות פקודות תקינים.

ספק כוח: ספק אספקת חשמל יציבה למודול VS183 כדי להבטיח ביצועים עקביים והפעלה אמינה ביישומי שלט רחוק.

סיכום:

חיישן IR VS183 הוא רכיב רב תכליתי ואמין ליישום פונקציונליות שלט רחוק במכשירים ויישומים אלקטרוניים שונים. על ידי הבנת התכונות שלו, עקרון העבודה, היתרונות, היישומים ושיקולי השימוש שלו, מפתחים יכולים למנף ביעילות את חיישן VS183 כדי לשפר את האינטראקציה של המשתמשים, השליטה והאוטומציה בפרויקטים ובמוצרים שלהם.

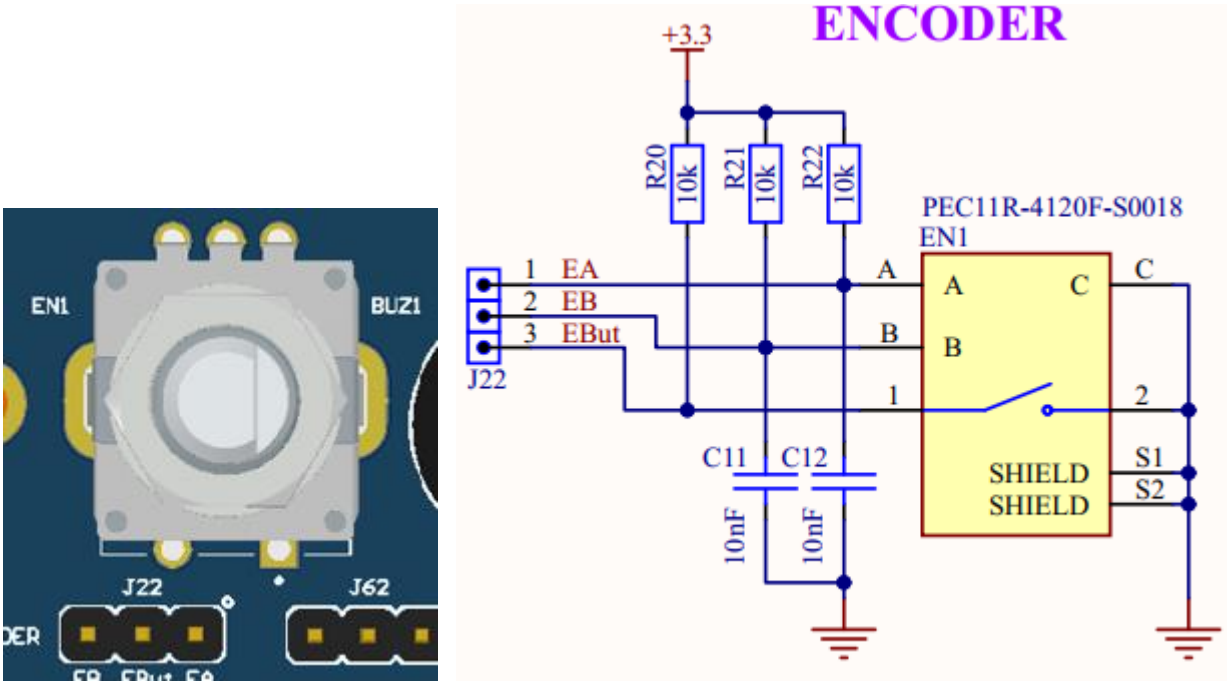
כדי להשתמש בחיישן IR VS183 עם ה-ESP32 ב-Arduino IDE, תצטרך להשתמש בספרייה התומכת בתקשורת IR. ספרייה פופולרית אחת למטרה זו היא ספריית IRremoteESP8266, התואמת ל-ESP32. להלן קוד לדוגמה המדגים כיצד לקבל ולפענח אותות IR באמצעות חיישן VS183 עם ESP32:

(כדי לבדוק כל כפתור יש לבדוק קוד של כל לחצן ואז לשייך אותו לפעולה מסוימת).

```
#include <IRremoteESP8266.h>
#include <IRrecv.h>
#include <IRutils.h>
#define IR_RECEIVE_PIN  14  // Pin connected to the VS183 IR sensor
IRrecv irrecv(IR_RECEIVE_PIN);
decode_results results;
void setup() {
  Serial.begin(115200);
  irrecv.enableIRIn(); // Start the IR receiver
}
void loop() {
  if (irrecv.decode(&results)) {
    // Print the received IR code
    Serial.print("Received IR code: 0x");
    Serial.println(results.value, HEX);

    irrecv.resume(); // Receive the next IR code
  }
```

```
}
```

קליטת מידה מאנקודר.



הבנת מקודדים רוטריים והטמעתם עם ESP32: מדריך מקיף

מבוא:

מקודדים סיבוביים הם מכשירים אלקטרומכניים המשמשים להמרת המיקום הזוויתי או הסיבוב של אובייקט לאותות דיגיטליים. הם מוצאים שימוש נרחב ביישומים שונים כמו רובוטיקה, בקרות תעשייתיות, פוטנציומטרים דיגיטליים והתקני קלט משתמשים כמו ידיות וגלגלי ריצה. במאמר זה, נחקור מקודדים סיבוביים, סוגיהם, עקרונות העבודה, היתרונות וכיצד לממשק אותם עם המיקרו-בקר ESP32 לשימוש בפרויקטים.

תכונות עיקריות:

זיהוי סיבוב מכני: מקודדים סיבוביים יכולים לזהות הן את הכיוון והן את גודל התנועה הסיבובית.

קידוד מצטבר: הם מספקים קידוד מצטבר, כאשר כל צעד או עצירה תואמים לשינוי מיקום ספציפי.

פעולה ללא מגע: חלק מהמקודדים הסיבוביים משתמשים בחישה אופטית או מגנטית לפעולה ללא מגע, מה שמבטיח עמידות ואמינות.

רזולוציה גבוהה: מקודדים סיבוביים יכולים להציע רזולוציה גבוהה, המאפשרת מדידה מדויקת של תנועות סיבוביות.

צדדיות: הם מגיעים בסוגים שונים, כולל מכאני, אופטי ומגנטי, העונים על דרישות יישומים שונות.

סוגי מקודדים סיבוביים:

מקודדים סיבוביים מכניים: מקודדים אלה משתמשים ברכיבים מכניים כגון גלגלי שיניים ומגעים כדי לזהות תנועה סיבובית.

מקודדים סיבוביים אופטיים: מקודדים אופטיים משתמשים בחיישני אור ובדיסק מקודד כדי לזהות סיבוב, ומספקים רזולוציה ודיוק גבוהים.

מקודדים סיבוביים מגנטיים: מקודדים מגנטיים משתמשים בשדות מגנטיים ובחיישנים כדי לזהות תנועה סיבובית, ומציעים חוסן ואמינות.

עקרון עבודה:

מקודדים סיבוביים מורכבים בדרך כלל מפיר, דיסק מסתובב עם חריצים או סימונים וחיישנים לזיהוי הסיבוב. כאשר ציר המקודד מסתובב, הוא גורם לשינויים במיקום החריצים או הסימונים על הדיסק, המתגלים על ידי החיישנים. החיישנים יוצרים אז אותות דיגיטליים התואמים לכיוון וגודל הסיבוב, ומאפשרים למיקרו-בקר לעקוב אחר מיקום המקודד.

חיבור למקודדים סיבוביים עם ESP32:

כדי לממשק מקודד סיבובי עם המיקרו-בקר ESP32, בצע את השלבים הבאים:

הגדרת חומרה: חבר את פיני המוצא של המקודד הסיבובי (בדרך כלל פינים A ו-B) לפיני GPIO ב-ESP32. בנוסף, חבר את הפין המשותף לאדמה והשתמש בנגדים משיכה על פיני A ו-B.

יישום תוכנה: השתמש בשיטות מבוססות פסיקה או סקר כדי לזהות שינויים במיקום המקודד. ביישום מבוסס פסיקות, הגדר פסיקות בשני הקצוות העולים והיורדים של אחד מפיני המקודד (למשל, פין A). ביישום מבוסס סקר, עקוב ברציפות אחר המצב של שני פיני המקודד ועקוב אחר שינויים במיקום.

יתרונות השימוש במקודדים סיבוביים עם ESP32:

דיוק גבוה: מקודדים רוטריים מציעים דיוק ורזולוציה גבוהים, מה שהופך אותם למתאימים ליישומים הדורשים מעקב אחר מיקום מדויק.

משוב בזמן אמת: הם מספקים משוב בזמן אמת על תנועות סיבוביות, ומאפשרים שליטה ואינטראקציה תגובה בפרויקטים.

התקן קלט רב-תכליתי: מקודדים סיבוביים יכולים לשמש כהתקני קלט רב-תכליתיים עבור אינטראקציה עם המשתמש, ניווט בתפריט, בקרת עוצמת הקול והתאמת פרמטרים.

פעולה ללא מגע: חלק מהמקודדים הסיבוביים פועלים ללא מגע פיזי, מפחית בלאי ומאריך את תוחלת החיים שלהם.

שילוב קל: עם הטמעת תוכנה נכונה, ניתן לשלב בקלות מקודדים סיבוביים עם פרויקטים מבוססי ESP32, ולשפר את הפונקציונליות ואת חווית המשתמש שלהם.

יישומים:

ממשקי משתמש: מקודד סיבובי משמש בדרך כלל בממשקי משתמש לשליטה בפרמטרים, הגדרות וניווט בתפריטים במכשירים אלקטרוניים.

בקרת מנוע: הם מוצאים יישומים במערכות בקרת מנוע לשליטה במהירות, מיקום וכיוון של מנועים.

רובוטיקה: מקודדים סיבוביים משמשים ברובוטיקה למשוב מיקום במפרקי רובוט, מניפולטורים ומפעילים.

ציוד שמע: הם מועסקים בציוד שמע כגון כפתורי עוצמת קול, גלגלי ריצה וכפתורי כוונון לשליטה והתאמה מדויקת.

אוטומציה תעשייתית: מקודדים סיבוביים משמשים באוטומציה תעשייתית עבור חישת מיקום, בקרת מהירות וניטור תנועה במכונות וציוד.

סיכום:

מקודדים סיבוביים הם מכשירים מגוונים המשמשים לאיתור ומדידה של תנועה סיבובית ביישומים שונים. על ידי הבנת הסוגים, עקרונות העבודה, היתרונות ושיטות ההתממשקות שלהם עם המיקרו-בקר ESP32, מפתחים יכולים למנף ביעילות מקודדים סיבוביים בפרויקטים שלהם כדי לשפר את האינטראקציה של המשתמשים, השליטה והמשוב. בין אם נעשה בהם שימוש בממשקי משתמש, מערכות בקרת מנוע, רובוטיקה או אוטומציה תעשייתית, מקודדים סיבוביים מציעים יכולות מדויקות ואמינות של מעקב אחר מיקום, מה שהופך אותם לרכיבים יקרי ערך במערכות והתקנים אלקטרוניים מודרניים.

כדי לממש את האנקודר אפשר להשתמש בשיטת פסיקות או בשיטת Polling.

נראה קודם את השיטה של פסיקות.

```
#define ENCODER_PIN_A  12  // Pin connected to encoder channel A
```

```
#define ENCODER_PIN_B  13  // Pin connected to encoder channel B

volatile int encoderPos = 0;  // Current position of the encoder
volatile int lastEncoded = 0; // Last position of the encoder
volatile unsigned long lastMicros = 0;
// Last time the encoder position was updated

void setup() {
  Serial.begin(9600);
  pinMode(ENCODER_PIN_A, INPUT);
  pinMode(ENCODER_PIN_B, INPUT);
  attachInterrupt(digitalPinToInterrupt(ENCODER_PIN_A), updateEncoder,
CHANGE);
  attachInterrupt(digitalPinToInterrupt(ENCODER_PIN_B), updateEncoder,
CHANGE);
}

void loop() {
  // Print current position of the encoder
  Serial.println(encoderPos);
  delay(100);
}

void updateEncoder() {
  unsigned long nowMicros = micros();

  // Debounce the encoder input
  if (nowMicros - lastMicros < 2000) {
    return;
  }
  lastMicros = nowMicros;

  int MSB = digitalRead(ENCODER_PIN_A); // Most significant bit
  int LSB = digitalRead(ENCODER_PIN_B); // Least significant bit

  int encoded = (MSB << 1) | LSB;
// Combine MSB and LSB to create encoded value
  int sum = (lastEncoded << 2) | encoded;
// Combine previous and current encoded value

  if (sum == 0b1101 || sum == 0b0100 || sum == 0b0010 || sum == 0b1011)
{
    encoderPos++; // Clockwise rotation
  } else if (sum == 0b1110 || sum == 0b0111 || sum == 0b0001 || sum ==
0b1000) {
    encoderPos--; // Counter-clockwise rotation
  }

  lastEncoded = encoded;
// Store current encoded value for next iteration
```

```
}
```

דוגמה השנייה היא דוגמה לפי שיטת Polling.

```
#define ENCODER_PIN_A  12  // Pin connected to encoder channel A
#define ENCODER_PIN_B  13  // Pin connected to encoder channel B

int encoderPos = 0; // Current position of the encoder
int lastEncoded = 0; // Last position of the encoder

void setup() {
  Serial.begin(9600);
  pinMode(ENCODER_PIN_A, INPUT);
  pinMode(ENCODER_PIN_B, INPUT);
}

void loop() {
  int MSB = digitalRead(ENCODER_PIN_A); // Most significant bit
  int LSB = digitalRead(ENCODER_PIN_B); // Least significant bit

  int encoded = (MSB << 1) | LSB;
// Combine MSB and LSB to create encoded value

  if (encoded != lastEncoded) { // Check if encoder position changed
    if ((lastEncoded == 0b00 && encoded == 0b01) || (lastEncoded ==
0b11 && encoded == 0b10)) {
      encoderPos++; // Clockwise rotation
    }
else if ((lastEncoded == 0b01 && encoded == 0b00) || (lastEncoded ==
0b10 && encoded == 0b11)) {
      encoderPos--; // Counter-clockwise rotation
    }
    Serial.println(encoderPos);
// Print current position of the encoder
    lastEncoded = encoded; // Update last encoded value
  }
}
```

צג 4 ספרות 7 סגמנט TM1637 בחיבור CA.

ה-TM1637 הוא מודול תצוגת LED פופולרי בן 4 ספרות בן 7 סגמנטים המשלב שבב TM1637. הוא נמצא בשימוש נפוץ בפרויקטים אלקטרוניים שונים בשל הפשטות, קלות השימוש והעלות הנמוכה שלו. מודול TM1637 כולל ארבע ספרות של 7 מקטעים (0-9), נורית נקודתיים וכמה פיני בקרה להתממשקות עם מיקרו-בקרים.

תכונות עיקריות של מודול תצוגה TM1637:

תצוגה בת 4 ספרות: המודול מורכב מארבע ספרות LED בנות 7 מקטעים, המאפשרות הצגת ערכים מספריים מ-0 עד 9999.

שבב TM1637 משולב: שבב TM1637 משמש כמנהל התקן/בקר עבור התצוגה בת 7 סגמנטים, מה שמפשט את הממשק עם מיקרו-בקרים.

תקשורת טורית: תקשורת עם מודול TM1637 מושגת באמצעות ממשק טורי פשוט, מה שמקל על שילוב עם מיקרו-בקרים כמו ESP32.

בקרת בהירות: המודול תומך בדרך כלל ברמות בהירות מתכווננות עבור תצוגת LED, המאפשרת נראות בתנאי תאורה שונים.

צריכת חשמל נמוכה: מודול TM1637 תוכנן לפעול ביעילות, מה שהופך אותו למתאים ליישומים המופעלים על ידי סוללה.

אינטראקציה עם ESP32 ב- Arduino IDE:

חיבור חומרה:

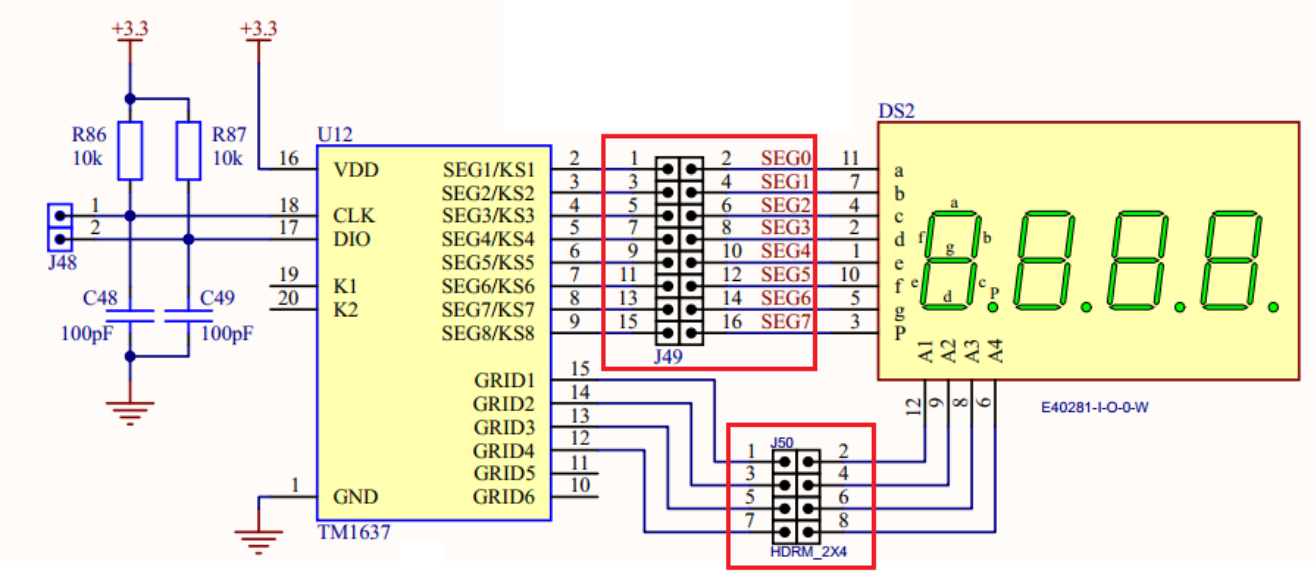
כדי לממשק את מודול התצוגה TM1637 עם בקר מיקרו ESP32 ב- Arduino IDE, תצטרך לבצע את החיבורים הבאים:

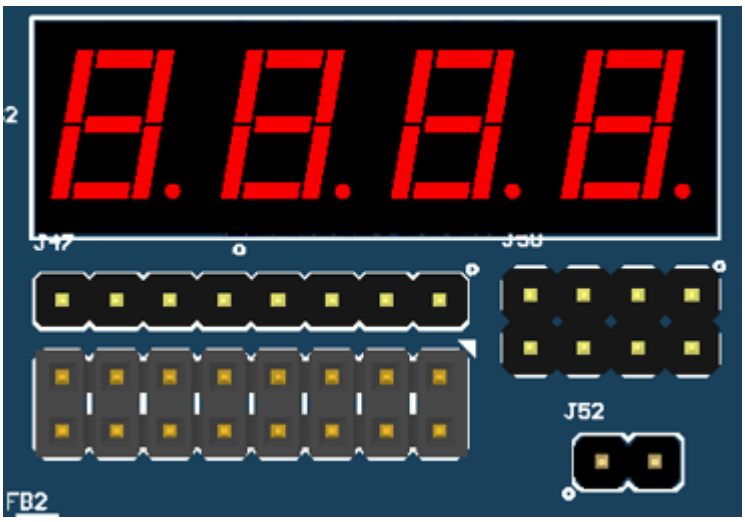
פין TM1637 CLK (שעון) לפין GPIO ב-ESP32 (למשל, GPIO 14).

TM1637 DIO (קלט/פלט נתונים) לפין GPIO אחר ב-ESP32 (למשל, GPIO 13).

פיני VCC (כוח) ו-GND (הארקה) של מודול TM1637 מחוברים לפיני מתח והארקה מתאימים ב-ESP32.

כדי להפעיל את הרכיב יש לקצר בעזרת חוטים או ג'מפרים של מחבר J49 ומחבר J50 כמו שמתואר בשרטוט.





שילוב תוכנה:

כדי לשלוט במודול התצוגה TM1637 ממיקרו-בקר ESP32 ב- Arduino IDE, תשתמש בדרך כלל בספריית TM1637. הנה דוגמה בסיסית לשימוש בספריית TM1637 כדי להציג מספרים במודול TM1637:

```
#include <TM1637Display.h>
// Define the pins for TM1637 (CLK, DIO)
#define CLK_PIN 14
#define DIO_PIN 13
// Create an instance of TM1637Display
TM1637Display display(CLK_PIN, DIO_PIN);
void setup() {
  // Initialize the TM1637 display
  display.setBrightness(7); // Set brightness level (0-7, 0 being the lowest)
  display.showNumberDec(1234); // Display a decimal number (e.g., 1234)
}
void loop() {
  // Code for main program loop
}
```

באמצעות ספריית TM1637, אתה יכול לשלוט בקלות במודול התצוגה כדי להציג מספרים, תווים או סמלים מותאמים אישית. הספרייה מספקת פונקציות להגדרת רמת הבהירות, הצגת מספרים עשרוניים או הקסדצימליים, הצגת ערכי טמפרטורה ועוד. בנוסף, אתה יכול לשלוט ב-LED של המעי הגס ולהגדיר אפשרויות תצוגה אחרות לפי הצורך.

מודול התצוגה TM1637 מציע פתרון נוח להצגת מידע מספרי בפרויקטים אלקטרוניים. כאשר הוא מתממשק עם מיקרו-בקר ESP32 ב-Arduino IDE, הוא הופך לכלי רב תכליתי ליצירת סוגים שונים של צגים, שעונים, טיימרים ועוד. עם פרוטוקול התקשורת הטורית הפשוט והספרייה הקלה לשימוש, שילוב מודול TM1637 בפרויקטים מבוססי ESP32 הוא פשוט ויעיל.

רשימת הפונקציות של הספרייה TM1637:

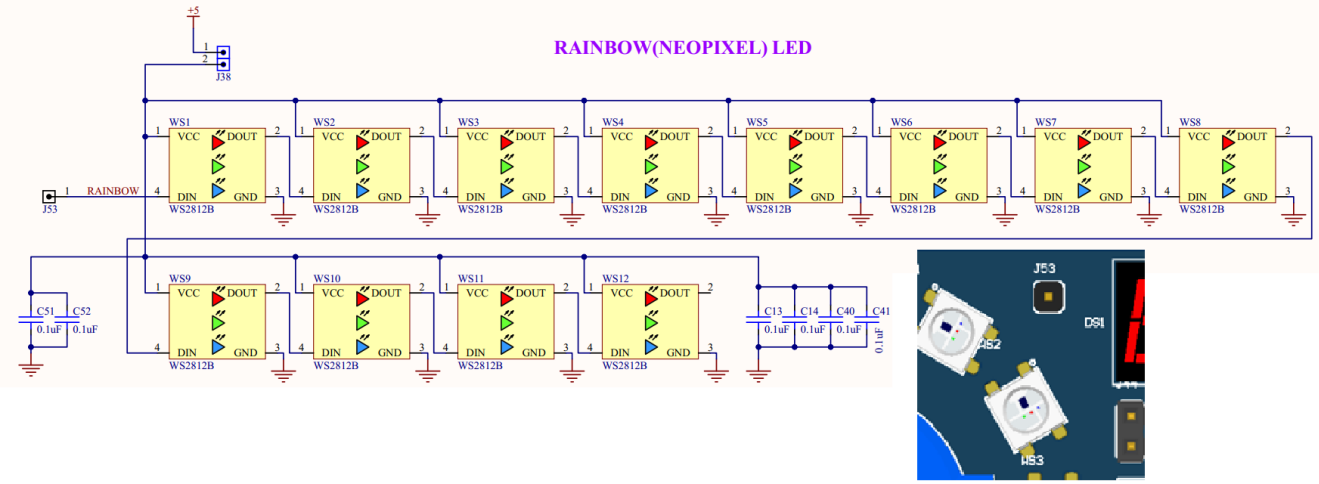
```
display.showNumberDec(-12);         // displayed _-12
display.showNumberDec(-999);        // displayed -999
display.showNumberDec(42);            // displayed __42
display.showNumberDec(42, false);     // displayed __42
display.showNumberDec(42, false, 2, 0);
// displayed 42__ => display 2 digit at position 0
display.showNumberDec(42, true);      // displayed 0042 => zero padding
display.showNumberDec(14, false, 2, 1);  // displayed _14_
display.showNumberDec(-5, false, 3, 0);  // displayed _-5_
display.showNumberDec(1234);          // displayed 1234
```

דוגמה לשימוש בפונקציות בסיסיות:

```
#include <TM1637Display.h>

#define CLK  22 // The ESP32 pin GPIO22 connected to CLK
#define DIO  23 // The ESP32 pin GPIO23 connected to DIO

// create a display object of type TM1637Display
TM1637Display display = TM1637Display(CLK, DIO);

// an array that sets individual segments per digit to display the word "dOnE"
const uint8_t done[] = {
  SEG_B | SEG_C | SEG_D | SEG_E | SEG_G,         // d
  SEG_A | SEG_B | SEG_C | SEG_D | SEG_E | SEG_F, // O
  SEG_C | SEG_E | SEG_G,                         // n
  SEG_A | SEG_D | SEG_E | SEG_F | SEG_G          // E
};

// degree celsius symbol
const uint8_t celsius[] = {
  SEG_A | SEG_B | SEG_F | SEG_G,  // Degree symbol
  SEG_A | SEG_D | SEG_E | SEG_F   // C
};

void setup() {
  display.clear();
  display.setBrightness(7); // set brightness to 7 (0:dimmest, 7:brightest)
}

void loop() {
  // show counter 0-9
  int i;
  for (i = 0; i < 10; i++) {
    display.showNumberDec(i);
    delay(500);
  display.clear();
  }

  display.showNumberDec(-91);            // displayed _-91
  delay(2000);
  display.clear();

  display.showNumberDec(-109);           // displayed -109
  delay(2000);
  display.clear();

  display.showNumberDec(21, false);      // displayed __21
  delay(2000);
  display.clear();

  display.showNumberDec(21, true);       // displayed 0021
  delay(2000);
  display.clear();

  display.showNumberDec(28, false, 2, 1); // displayed _28_
  delay(2000);
  display.clear();
```

```
  display.showNumberDec(-9, false, 3, 0); // displayed _-9_
  delay(2000);
  display.clear();

  // displayed 15:30
  display.showNumberDecEx(1530, 0b11100000, false, 4, 0);
  delay(2000);
  display.clear();

  // displayed 23°C
  int temperature = 23; // or read from temperature sensor
  display.showNumberDec(temperature, false, 2, 0);
  display.setSegments(celsius, 2, 2);
  delay(2000);
  display.clear();

  // displayed letters: dOnE
  display.setSegments(done);
  delay(2000);
  display.clear();
}
```

הפעלת לד NeoPixel.



הבנת נוריות ה-NeoPixel והטמעתם עם ESP32: מדריך מקיף

מבוא:

נורות LED של NeoPixel, הידועות גם בשם WS2812 או WS2812B LED, הן נוריות RGB הניתנות לטיפול בנפרד, המאפשרות אפקטי תאורה מורכבים וצבעוניים בפרויקטים אלקטרוניים שונים. במאמר זה, נחקור את התכונות העיקריות, עקרונות העבודה, היתרונות וכיצד לממשק נוריות NeoPixel עם המיקרו-בקר ESP32 לשימוש בפרויקטים.

תכונות עיקריות:

ניתן להתייחסות בנפרד: ניתן לשלוט בכל NeoPixel LED בשרשרת בנפרד, מה שמאפשר דפוסים והנפשות מורכבות.

ערבוב צבעים RGB: נורות LED של NeoPixel כוללות נוריות LED אדומות, ירוקות וכחולות (RGB) בתוך כל חבילה, מה שמאפשר להציג מגוון רחב של צבעים.

ניתן לשרשרת: ניתן לחבר מספר נוריות של NeoPixel יחד בשרשרת, מה שמאפשר יצירת צגים גדולים ומתקנים תאורה.

בהירות גבוהה: לנורות ה-NeoPixel יש בדרך כלל רמות בהירות גבוהות, מה שהופך אותן למתאימות ליישומים פנימיים וחיצוניים כאחד.

בקר משולב: כל LED של NeoPixel מכיל שבב בקר מובנה, המפשט את תהליך ההתממשקות ומפחית את מספר הרכיבים החיצוניים הנדרשים.

עקרון עבודה:

נוריות NeoPixel פועלות באמצעות פרוטוקול תקשורת חוט אחד, שבו הנתונים נשלחים באופן סדרתי לכל LED בשרשרת. הנתונים מורכבים מזרם של פולסים המייצגים את הצבע והבהירות הרצויים עבור כל LED. שבב הבקר בתוך כל LED מקבל את הנתונים, מפרש אותם ומתאים את בהירות הנוריות האדומות, הירוקות והכחולות בהתאם, מה שמביא לפלט הצבע הרצוי.

ממשק נוריות NeoPixel עם ESP32:

כדי לממשק את נוריות ה-NeoPixel עם המיקרו-בקר ESP32, בצע את השלבים הבאים:

הגדרת חומרה: חבר את פין קלט הנתונים של ה-NeoPixel LED הראשון בשרשרת לפין GPIO ב-ESP32. בנוסף, חבר את פיני ספק הכוח (VCC) והארקה (GND) של נוריות ה-NeoPixel למקורות מתח מתאימים.

יישום תוכנה: השתמש בספרייה תואמת כגון ספריית Adafruit NeoPixel כדי לשלוט בנורות ה-NeoPixel. אתחל את אובייקט NeoPixel עם מספר נוריות ה-LED בשרשרת והפין GPIO המחובר לכניסת הנתונים.

יתרונות השימוש ב-NeoPixel LED עם ESP32:

צדדיות: ניתן להשתמש בנורות LED של NeoPixel במגוון רחב של יישומים, כולל תאורה דקורטיבית, אלקטרוניקה לבישה, שילוט והתקנות אמנות.

התאמה אישית: עם נוריות LED הניתנות לטיפול בנפרד, המשתמשים יכולים ליצור אפקטי תאורה מותאמים אישית, אנימציות ודפוסים המותאמים לצרכים ולהעדפות הספציפיות שלהם.

קלות שימוש: ספריות כמו ספריית Adafruit NeoPixel מפשטות את תהליך השליטה בנורות ה-NeoPixel, ומאפשרות למשתמשים להתמקד בעיצוב יצירתי ולא בפרטי חומרה ברמה נמוכה.

אינטגרציה: ניתן לשלב בקלות את נורות ה-NeoPixel עם פרויקטים מבוססי ESP32, ולספק אפקטי תאורה דינמיים ומושכי עין כדי לשפר את חווית המשתמש והאסתטיקה.

מדרגיות: נוריות ה-NeoPixel ניתנות לשרשרת, מה שמאפשר למשתמשים להתאים את הגדרות התאורה שלהם על ידי הוספת נוריות LED נוספות לפי הצורך ללא צורך בחיווט מורכב או חומרה נוספת.

יישומים:

תאורה דקורטיבית: נוריות ה-NeoPixel משמשות בדרך כלל ביישומי תאורה דקורטיבית, כולל תצוגות חג, קישוטים למסיבות ותאורת סביבה לחללים בבית ובמשרד.

מוצרי אלקטרוניקה לבישים: הם מועסקים בפרויקטי אלקטרוניקה לבישים כגון תלבושות, אביזרים ובגדים אינטראקטיביים כדי להוסיף אפקטי תאורה צבעוניים ודינמיים.

התקנות אמנות: נורות LED של NeoPixel פופולריות במיצבי אמנות ובתערוכות אינטראקטיביות, שבהן הן יכולות ליצור חוויות סוחפות ומרשימות לצופים.

ציוד היקפי למשחקים: הם משמשים בציוד היקפי למשחקים כגון מקלדות, עכברים ובקרים כדי לספק אפקטי תאורת RGB הניתנים להתאמה אישית ומשוב ויזואלי.

אב טיפוס וחינוך: נורות LED של NeoPixel הן כלים יקרי ערך ליצירת אב טיפוס ולמטרות חינוכיות, המאפשרות למשתמשים להתנסות עם אפקטים של תאורה, ערבוב צבעים ומושגי תכנות.

סיכום:

נורות LED של NeoPixel מציעות פתרון רב-תכליתי וניתן להתאמה אישית ליצירת אפקטי תאורה תוססים בפרויקטים אלקטרוניים. על ידי הבנת התכונות, עקרונות העבודה, היתרונות ושיטות ההתממשקות שלהם עם המיקרו-בקר ESP32, מפתחים יכולים לשחרר את היצירתיות שלהם ולעצב התקנות תאורה שובות לב, ציוד לביש ותצוגות אינטראקטיביות. בין אם משתמשים בהם לתאורה דקורטיבית, אלקטרוניקה לבישה, התקנות אמנות או ציוד היקפי למשחקים, נוריות ה-NeoPixel מספקות אינסוף אפשרויות להוספת צבע ודינמיות לפרויקטים וסביבות.

קוד להפעלת NeoPixel עם שימוש בספריה סטנדרטית:

```
#include <Adafruit_NeoPixel.h>

#define LED_PIN    12 // Pin connected to the NeoPixel data input
#define NUM_LEDS   8  // Number of NeoPixel LEDs in the strip

Adafruit_NeoPixel strip(NUM_LEDS, LED_PIN, NEO_GRB + NEO_KHZ800);

void setup() {
  strip.begin();           // Initialize the NeoPixel strip
  strip.show();            // Initialize all pixels to 'off'
```

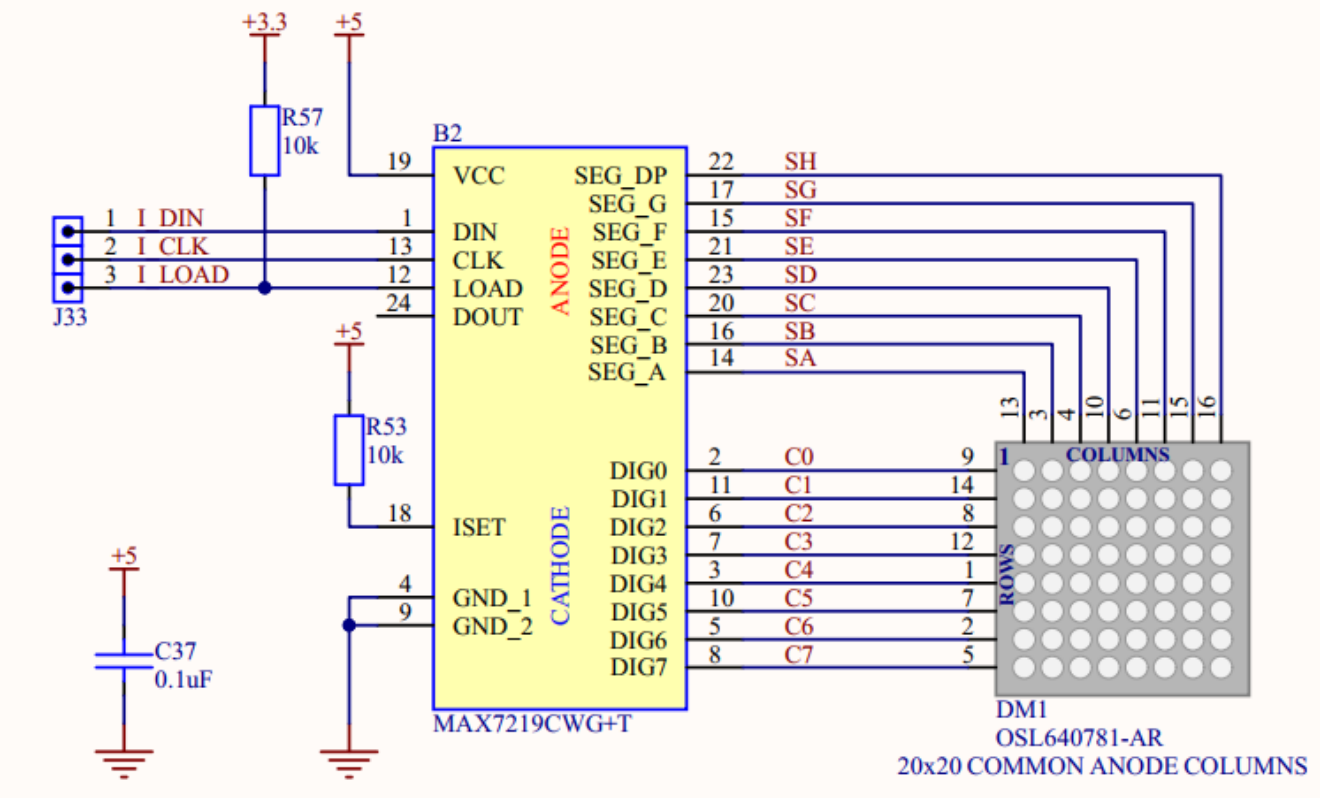
```
}

void loop() {
  // Fill the strip with a color (red, green, blue)
  colorWipe(strip.Color(255, 0, 0), 50); // Red
  delay(1000);
  colorWipe(strip.Color(0, 255, 0), 50); // Green
  delay(1000);
  colorWipe(strip.Color(0, 0, 255), 50); // Blue
  delay(1000);
}

// Fill the dots one after the other with a color
void colorWipe(uint32_t color, int wait) {
  for(int i=0; i<strip.numPixels(); i++) {
    strip.setPixelColor(i, color); // Set pixel color
    strip.show(); // Update LED strip
    delay(wait); // Pause for a moment
  }
}
```

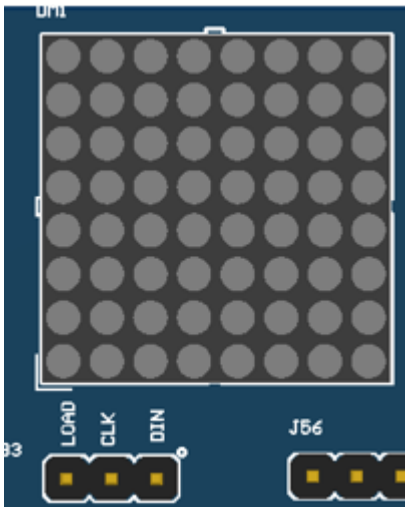
קוד להפעלת NeoPixel ללא שימוש בספריה סטנדרטית:

```
#define LED_PIN    12 // Pin connected to the NeoPixel data input
#define NUM_LEDS   8  // Number of NeoPixel LEDs in the strip

void setup() {
  pinMode(LED_PIN, OUTPUT); // Set pin as output
}

void loop() {
  // Fill the strip with a color (red, green, blue)
  colorWipe(255, 0, 0, 50); // Red
  delay(1000);
  colorWipe(0, 255, 0, 50); // Green
  delay(1000);
  colorWipe(0, 0, 255, 50); // Blue
  delay(1000);
}

// Fill the strip with a specified RGB color
void colorWipe(int red, int green, int blue, int wait) {
  for(int i=0; i<NUM_LEDS; i++) {
    setColor(i, red, green, blue); // Set LED color
    delay(wait); // Pause for a moment
  }
}

// Set the color of a specific LED
void setColor(int led, int red, int green, int blue) {
```

```
  for(int j=7; j>=0; j--) {
    digitalWrite(LED_PIN, LOW); // Start bit
    delayMicroseconds(9); // Delay to define logic '0'
    if(bitRead(led, j)) { // Bit is 1
      digitalWrite(LED_PIN, HIGH); // Logic '1'
      delayMicroseconds(6); // Delay to define logic '1'
    } else { // Bit is 0
      digitalWrite(LED_PIN, LOW); // Logic '0'
      delayMicroseconds(6); // Delay to define logic '0'
    }
  }
  // Send color data (GRB format)
  for(int j=7; j>=0; j--) {
    digitalWrite(LED_PIN, LOW); // Start bit
    delayMicroseconds(9); // Delay to define logic '0'
    if(bitRead(green, j)) { // Bit is 1
      digitalWrite(LED_PIN, HIGH); // Logic '1'
      delayMicroseconds(6); // Delay to define logic '1'
    } else { // Bit is 0
      digitalWrite(LED_PIN, LOW); // Logic '0'
      delayMicroseconds(6); // Delay to define logic '0'
    }
  }
  // Repeat the process for red and blue color data
}
```

הפעלת מטריצת לדים Dot Matrix טורי.





הבנת תצוגות Dot Matrix 8x8 במצב מקביל רגיל ועם MAX7219: מדריך מקיף

מבוא:

תצוגות מטריצת נקודות משמשות בדרך כלל בפרויקטים אלקטרוניים שונים להצגת תווים אלפאנומריים, סמלים, גרפיקה ואנימציות. במאמר זה, נסקור תצוגות מטריצת נקודות בתצורת x88, הן במצב מקבילי רגיל והן בממשק עם שבב מנהל ההתקן MAX7219. נעמיק בתכונות שלהם, עקרונות העבודה, היתרונות וכיצד לממשק אותם עם מיקרו-בקרים כגון ESP32 לשימוש בפרויקטים.

תצוגת Dot Matrix 8x8 במצב מקביל רגיל:

מאפיינים:

רזולוציה: צג מטריצת נקודות בגודל 8x8 מורכב מ-8 שורות ו-8 עמודות של נוריות LED, המספקות סך של 64 פיקסלים הניתנים לשליטה בנפרד.

צדדיות: הוא יכול להציג מגוון רחב של תווים, סמלים ותבניות גרפיות, מה שהופך אותו למתאים ליישומים שונים.

קלות שימוש: צגי מטריצת נקודות קלים יחסית לממשק עם מיקרו-בקרים המשתמשים בתקשורת מקבילה, ודורשים רק כמה פינים דיגיטליים לשליטה.

התאמה אישית: משתמשים יכולים ליצור דמויות מותאמות אישית, אנימציות ואפקטים של גלילה על ידי הפעלה או כיבוי סלקטיבית של נוריות LED בודדות במטריצה.

עקרון עבודה:

במצב מקבילי רגיל, כל שורה של תצוגת הנקודות מחוברת לפין פלט דיגיטלי נפרד של המיקרו-בקר, בעוד שהעמודים מחוברים לאדמה באמצעות נגדים מגבילי זרם. כדי להציג דפוס או תו ספציפי, המיקרו-בקר מגדיר את המצב הרצוי (ON/OFF) עבור כל פיקסל LED על ידי הנעה סלקטיבית של פיני השורה המתאימים גבוה ונמוך, תוך שמירה על פיני העמודה נמוכים.

יתרונות:

רזולוציה גבוהה: תצוגות מטריצת נקודות מציעות רזולוציה גבוהה ובהירות, המאפשרות הצגת גרפיקה וטקסט מפורטים.

עלות נמוכה: הם חסכוניים בהשוואה לטכנולוגיות תצוגה אחרות, מה שהופך אותם למתאימים לפרויקטים ידידותיים לתקציב.

התאמה אישית: למשתמשים יש שליטה מלאה על התוכן המוצג במטריצה, מה שמאפשר אפקטים חזותיים יצירתיים ומותאמים אישית.

נראות: צגי מטריצת נקודות נראים היטב, אפילו בתנאי תאורה חלשה, מה שהופך אותם למתאימים ליישומים פנימיים וחיצוניים.

צג Dot Matrix 8x8 עם MAX7219:

מאפיינים:

מנהל התקן משולב: ה-MAX7219 הוא שבב מנהל התקן משולב של תצוגת LED שנועד לפשט את השליטה בתצוגות מטריצת נקודות.

ממשק טורי: הוא מתקשר עם מיקרו-בקרים באמצעות ממשק טורי פשוט (SPI), הדורש פחות פינים לשליטה.

חיבור מדורג: ניתן לשדר שבבי MAX7219 מרובים כדי לשלוט על מערכים גדולים של צגי מטריצת נקודות מבלי לצרוך פינים נוספים של מיקרו-בקר.

בקרת בהירות: ה-MAX7219 מספק רמות בהירות מתכווננות עבור הצגים המחוברים, המאפשרים נראות אופטימלית בתנאי תאורה שונים.

עקרון עבודה:

בתצורת MAX7219, צגי מטריצת נקודות מרובים מחוברים בסדרה (משרשרת), כאשר כל צג מקבל נתונים משבב מנהל ההתקן MAX7219. המיקרו-בקר מתקשר עם שבב MAX7219 באמצעות פרוטוקול SPI, ושולח פקודות לעדכון התוכן המוצג בתצוגות הנקודות. ה-MAX7219 מטפל בריבוי והנעה של נוריות ה-LED, ומפשט את תהליך הבקרה עבור המיקרו-בקר.

יתרונות:

חיווט פשוט: שימוש ב-MAX7219 מפחית את מספר החוטים הנדרשים לשליטה על תצוגות נקודות מרובות, ומייעל את הגדרת החומרה.

בקרה יעילה: שבב הנהג המשולב מוריד את משימות בקרת התצוגה מהמיקרו-בקר, ומאפשר לו להתמקד במשימות אחרות.

מדרגיות: ניתן להוסיף או להסיר בקלות תצוגות מטריצות נקודות מרובות מהמערכת על ידי שילוב שבבי MAX7219 נוספים, המספקים מדרגיות וגמישות.

תכונות מובנות: ה-MAX7219 מציע תכונות נוספות כגון בקרת בהירות, מצב בדיקת תצוגה ופענוח ספרות, מה שמשפר את הפונקציונליות והרבגוניות של מערכת תצוגת הנקודות.

יישומים:

גלילה של לוחות הודעות: תצוגות מטריצת נקודות משמשות בדרך כלל בגלילה בלוחות הודעות להצגת הודעות טקסט, הודעות ופרסומות.

שעונים וטיימרים: הם מועסקים בשעונים דיגיטליים, טיימרים ותצוגות ספירה לאחור כדי לציין זמן ומשכים שחלפו.

תצוגות מידע: תצוגות מטריצת נקודות מוצאות יישומים בתצוגות מידע, לוחות תוצאות ומערכות שילוט להעברת נתונים ועדכונים בזמן אמת.

מכשירי משחק: הם משמשים במכשירי משחק, מכונות ארקייד וקונסולות כף יד להצגת גרפיקה של משחק, ציונים ואנימציות.

כלים חינוכיים: תצוגות מטריצת נקודות משמשות ככלים חינוכיים להוראת מושגי תכנות, עיצוב ממשק משתמש גרפי (GUI) ומערכות משוב חזותי.

סיכום:

תצוגות Dot Matrix בתצורת 8x8 מציעות פתרון רב תכליתי וניתן להתאמה אישית להצגת טקסט, גרפיקה ואנימציות בפרויקטים אלקטרוניים. בין אם נעשה בהם שימוש במצב מקבילי רגיל או בממשק עם שבב מנהל ההתקן MAX7219, צגי מטריצת נקודות מספקים רזולוציה גבוהה, נראות וגמישות עבור מגוון רחב של יישומים. על ידי הבנת התכונות, עקרונות העבודה, היתרונות והיישומים שלהם, מפתחים יכולים לשלב ביעילות תצוגות מטריצת נקודות בפרויקטים שלהם כדי לשפר את אינטראקציית המשתמש, תצוגת המידע והאסתטיקה החזותית.

כדי לשלוט בצג Dot Matrix 8x8 עם מנהל ההתקן MAX7219 עבור ESP32, אתה יכול להשתמש בספריית MAX7219 עבור ESP32 או ליישם את פרוטוקול התקשורת באופן ידני מבלי להשתמש בספריות כלשהן. להלן דוגמאות לשתי הגישות:

עם ספריית MAX7219:

```
#include <MD_MAX72xx.h>
#include <SPI.h>

#define MAX_DEVICES 4  // Number of MAX7219 devices in the chain
#define DATA_IN  23     // Data input pin
#define CLK      18     // Clock pin
#define CS       5      // Chip select pin

MD_MAX72XX mx = MD_MAX72XX(HARDWARE_TYPE, CS, MAX_DEVICES);
```

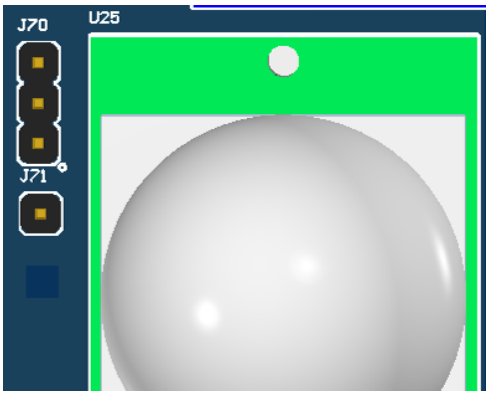
```
void setup() {
  mx.begin();
  mx.control(MD_MAX72XX::INTENSITY, 2); // Set brightness level (0-15)
  mx.setTextAlignment(PA_CENTER, PB_CENTER); // Text alignment
}

void loop() {
  mx.clear(); // Clear display

  // Display text "HELLO" scrolling horizontally
  mx.print("HELLO", PA_CENTER, 0, 50); // Text, Alignment, Delay
  delay(1000);
}
```

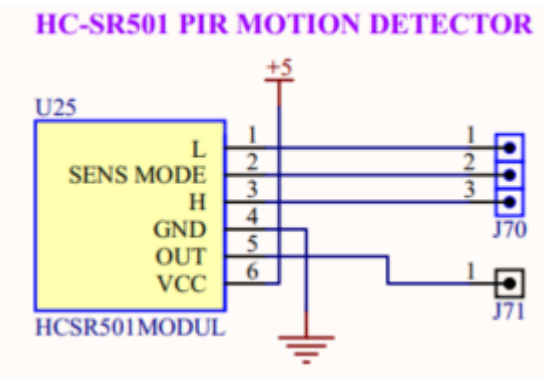
ללא ספריית MAX7219:

```
#define MAX_DEVICES 1    // Number of MAX7219 devices in the chain
#define DATA_IN  23      // Data input pin
#define CLK      18      // Clock pin
#define CS       5       // Chip select pin

void setup() {
  pinMode(DATA_IN, OUTPUT);
  pinMode(CLK, OUTPUT);
  pinMode(CS, OUTPUT);
  digitalWrite(CS, HIGH); // Deselect all devices
  sendCommand(0x09, 0xFF); // Decode mode: BCD for all digits
  sendCommand(0x0A, 0x0F); // Intensity: High intensity
  sendCommand(0x0B, 0x07); // Scan limit: All digits
  sendCommand(0x0C, 0x01); // Shutdown: Normal operation
}

void loop() {
  clearDisplay();
  // Display text "HELLO" scrolling horizontally
  for (int i = -8; i < 8; i++) {
    displayChar('H', i, 0);
    displayChar('E', i + 8, 0);
    displayChar('L', i + 16, 0);
    displayChar('L', i + 24, 0);
    displayChar('O', i + 32, 0);
    delay(200);
  }
}

void clearDisplay() {
  for (int i = 1; i <= 8; i++) {
    sendCommand(i, 0);
  }
}
```

```
void displayChar(char c, int col, int row) {
  int font_index = c - 32; // Offset ASCII code
  if (col < -7 || col >= 8) return;
  for (int i = 0; i < 8; i++) {
    if (col + i < 0 || col + i >= 8) continue;
    sendCommand(col + i + 1, pgm_read_byte_near(font[font_index] + i));
  }
}

void sendCommand(byte address, byte data) {
  digitalWrite(CS, LOW); // Select device
  for (int i = MAX_DEVICES; i > 0; i--) {
    shiftOut(DATA_IN, CLK, MSBFIRST, address); // Send address
    shiftOut(DATA_IN, CLK, MSBFIRST, data);    // Send data
  }
  digitalWrite(CS, HIGH); // Deselect device
}
```

חיישן תנוע HC-SR501 PIR.





הבנת חיישן תנועה PIR HC-SR501: מדריך מקיף

מבוא:

חיישן התנועה HC-SR501 פסיבי אינפרא אדום (PIR) הוא רכיב אלקטרוני בשימוש נרחב המזהה תנועה על ידי חישת שינויים בקרינת אינפרא אדום הנפלטת מעצמים נעים. במאמר זה, נתעמק בתכונות, עקרונות העבודה, היישומים והטיפים לשימוש יעיל בחיישן התנועה HC-SR501 PIR בפרויקטים אלקטרוניים שונים.

מאפיינים:

טווח זיהוי: לחיישן HC-SR501 טווח זיהוי של עד כ-7 מטרים, מה שהופך אותו למתאים לזיהוי תנועה בתוך אזור מוגדר.

רגישות מתכוונת: הוא כולל פוטנציומטר מובנה להתאמת רגישות החיישן, המאפשר למשתמשים להתאים אישית את טווח הזיהוי בהתאם לדרישות הספציפיות שלהם.

מתח הפעלה: החיישן פועל על מגוון רחב של מתחים, בדרך כלל בין 5 V ל-DC 20V , מה שהופך אותו לתואם למגוון מיקרו-בקרים, לוחות ארדואינו והתקנים אלקטרוניים אחרים.

אות פלט: כאשר מזוהה תנועה, החיישן מוציא אות גבוה (רמה לוגית 1), שיכול להפעיל מעגל חיצוני או מיקרו-בקר לבצע פעולות מוגדרות מראש.

עקרון עבודה:

חיישן התנועה HC-SR501 PIR מזהה תנועה באמצעות חיישן פירואלקטרי החש שינויים בקרינת אינפרא אדומה הנפלטת על ידי בעלי חיים בעלי דם חם, בני אדם או עצמים פולטי חום אחרים בטווח הזיהוי שלו. החיישן מורכב מאלמנט חיישן פירואלקטרי, עדשת Fresnel ומעגלים תומכים. כאשר מזוהה תנועה, החיישן הפירואלקטרי מייצר אות חשמלי קטן, המוגבר ומעובד על ידי המעגלים המשולבים. אות הפלט עולה גבוה, מה שמצביע על נוכחות של תנועה.

יישומים:

מערכות אבטחה: חיישן HC-SR501 PIR נמצא בשימוש נפוץ במערכות אבטחה כדי לזהות פולשים או תנועה לא מורשית בבתים, במשרדים ובמקומות אחרים.

תאורה אוטומטית: היא משמשת במערכות תאורה אוטומטיות כדי להדליק אורות כאשר מזוהה תנועה ולכבות אותם לאחר תקופה מסוימת של חוסר פעילות, מה שמשפר את יעילות האנרגיה.

התקני בית חכם: חיישני PIR משולבים במכשירי בית חכם כגון חיישני תפוסה, מצלמות המופעלות בתנועה ותרמוסטטים חכמים כדי לספק אוטומציה ונוחות.

התקנות אמנות אינטראקטיביות: אמנים ויוצרים משתמשים בחיישני PIR במיצבי אמנות ותערוכות אינטראקטיביות כדי ליצור סביבות דינמיות ומגיבות המגיבות לנוכחות הצופים או המשתתפים.

ניטור חיות בר: חיישני PIR משמשים בניטור ומחקר של חיות בר כדי לזהות נוכחות של בעלי חיים ולחקור את התנהגותם בבתי גידול טבעיים.

טיפים לשימוש בחיישן PIR HC-SR501:

מיקום: הרכב את החיישן בגובה ובזווית מתאימים כדי להבטיח כיסוי אופטימלי של אזור הזיהוי הרצוי תוך מזעור טריגרים שווא של עצמים נעים מחוץ לטווח המיועד.

התאם את הרגישות: כוונן את הרגישות של החיישן באמצעות הפוטנציומטר המשולב כדי להשיג ביצועי זיהוי אמינים מבלי להיות רגישים מדי לשינויים סביבתיים.

ספק כוח: ספק אספקת חשמל יציבה בטווח המתח שצוין כדי להבטיח פעולה תקינה של החיישן ולמנוע התנהגות לא סדירה או טריגרים שווא.

הגדרות השהיה: הגדר את המגשר המובנה או המעגל החיצוני כדי להתאים את עיכוב הזיהוי, הקובע כמה זמן החיישן יישאר מופעל לאחר זיהוי תנועה.

בדיקה וכיול: בדוק את החיישן בתנאי תאורה ובסביבות שונות כדי לאמת את הביצועים שלו ולבצע התאמות נדרשות לרגישות ולהגדרות אחרות.

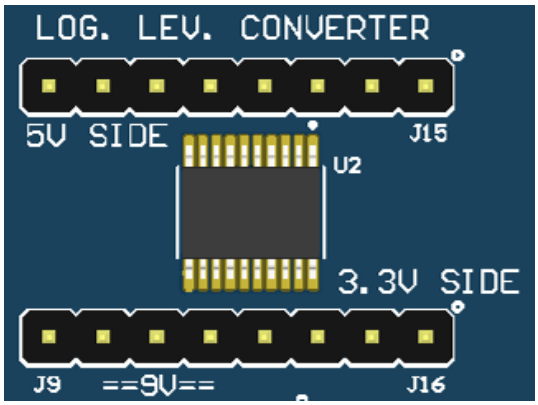
סיכום:

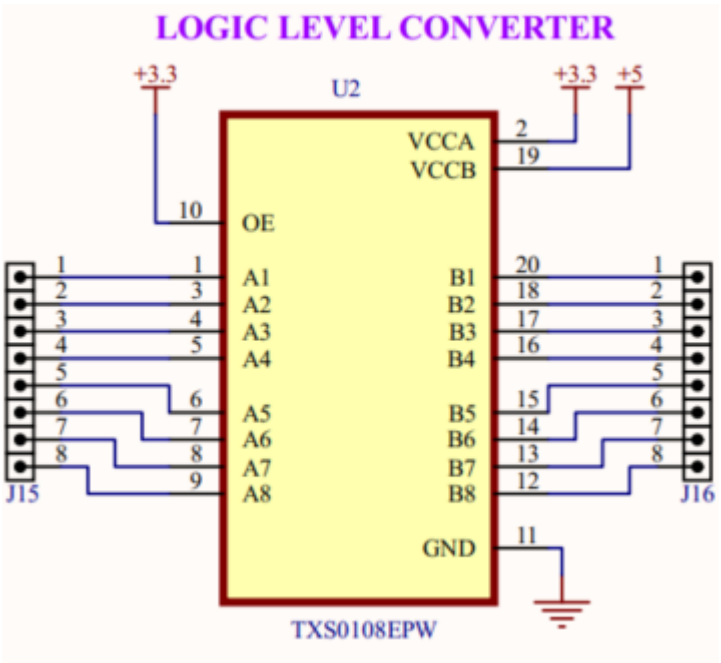
חיישן התנועה HC-SR501 PIR הוא רכיב רב תכליתי ואמין לזיהוי תנועה בפרויקטים אלקטרוניים, מערכות אבטחה, תאורה אוטומטית, מכשירי בית חכם ועוד. על ידי הבנת התכונות שלו, עקרונות העבודה, היישומים ושיטות העבודה המומלצות לשימוש, מפתחים וחובבים יכולים לשלב ביעילות את חיישן HC-SR501 בפרויקטים שלהם כדי לשפר את הפונקציונליות, האוטומציה והאינטראקטיביות. בין אם נעשה בו שימוש לאבטחת בית, יעילות אנרגטית או התקנות יצירתיות, חיישן PIR HC-SR501 מציע פתרון חסכוני לזיהוי תנועה עם מגוון רחב של יישומים ואפשרויות.

כדי להפעיל חיישן זה אנו צריכים להמיר רמת מתח מ-5V ל-3.3V בעזרת רכיב TXS0108EPW או בעזרת מחלק מתח (נגדים 1k,2k).

```
const int pirPin = 2; // Pin connected to the OUT pin of the HC-SR501
sensor
bool motionDetected = false;
void setup() {
  Serial.begin(9600);
  pinMode(pirPin, INPUT);
}

void loop() {
  int sensorValue = digitalRead(pirPin);
  if (sensorValue == HIGH) {
    if (!motionDetected) {
      Serial.println("Motion detected!");
      motionDetected = true;
    }
  } else {
    if (motionDetected) {
      Serial.println("Motion stopped.");
      motionDetected = false;
    }
  }
  delay(100); // Adjust delay as needed for responsiveness
}
```

המרת מתחים.





ה-TXS0108EPW הוא מתרגם רמות מתח דו-כיווני שנועד להקל על תקשורת בין מכשירים הפועלים ברמות מתח שונות. הוא משמש כממשק בין מערכות עם רמות מתח בלתי תואמות, ומבטיח חילופי נתונים חלקים ללא סיכון לנזק לרכיבים המחוברים. בסעיף זה, נספק סקירה כללית של מחליף הרמה TXS0108EPW ונחקור כמה דוגמאות מעשיות לשימוש בו.

סקירה כללית של TXS0108EPW:

מאפיינים:

תרגום דו-כיווני: ה-TXS0108EPW תומך בתרגום מתח דו-כיווני, המאפשר לנתונים לזרום בשני הכיוונים בין הצדדים של מתח נמוך ומתח גבוה.

טווח מתח רחב: הוא יכול לתרגם רמות מתח מ- 0.7 V עד ל- 5.5 V, מה שהופך אותו לתואם למגוון רחב של מכשירים.

שמונה ערוצים: המכשיר מספק שמונה ערוצים לתרגום אותות בודדים או קווי נתונים מרובים בו זמנית.

חישת כיוון אוטומטית: ה-TXS0108EPW כולל חישת כיוון אוטומטית, המבטל את הצורך באותות בקרה נוספים כדי לקבוע את כיוון זרימת הנתונים.

הפעלה במהירות גבוהה: עם קצב נתונים מרבי של עד 100 Mbps, מעביר הרמה מתאים לממשקי תקשורת מהירים כגון SPI, I2C, UART ו-GPIO.

יישומים:

ממשק מיקרו-בקרים: הוא מאפשר תקשורת בין מיקרו-בקרים הפועלים ברמות מתח שונות, ומקל על שילוב במערכות מתח מעורבות.

ממשקי חיישנים: מחליף הרמה מאפשר לחיישנים הפועלים ברמות מתח שונות לתקשר עם יחידות בקרה או מיקרו-בקרים.

המרת רמה לוגית: ניתן להשתמש בו כדי להמיר אותות לוגיים בין התקנים עם רמות מתח לא תואמות, תוך הבטחת שלמות האות תקינה.

דוגמאות מעשיות:

דוגמה 1: תקשורת Arduino-to-ESP32

נניח שיש לך מיקרו-בקר Arduino הפועל ב-V5 ומחשב ESP32 עם לוח יחיד הפועל על 3.3 V. אתה יכול להשתמש ב-TXS0108EPW כדי לאפשר תקשורת דו-כיוונית בין שני המכשירים, מה שמאפשר להם להחליף נתונים בצורה חלקה.

דוגמה 2: ממשק חיישן

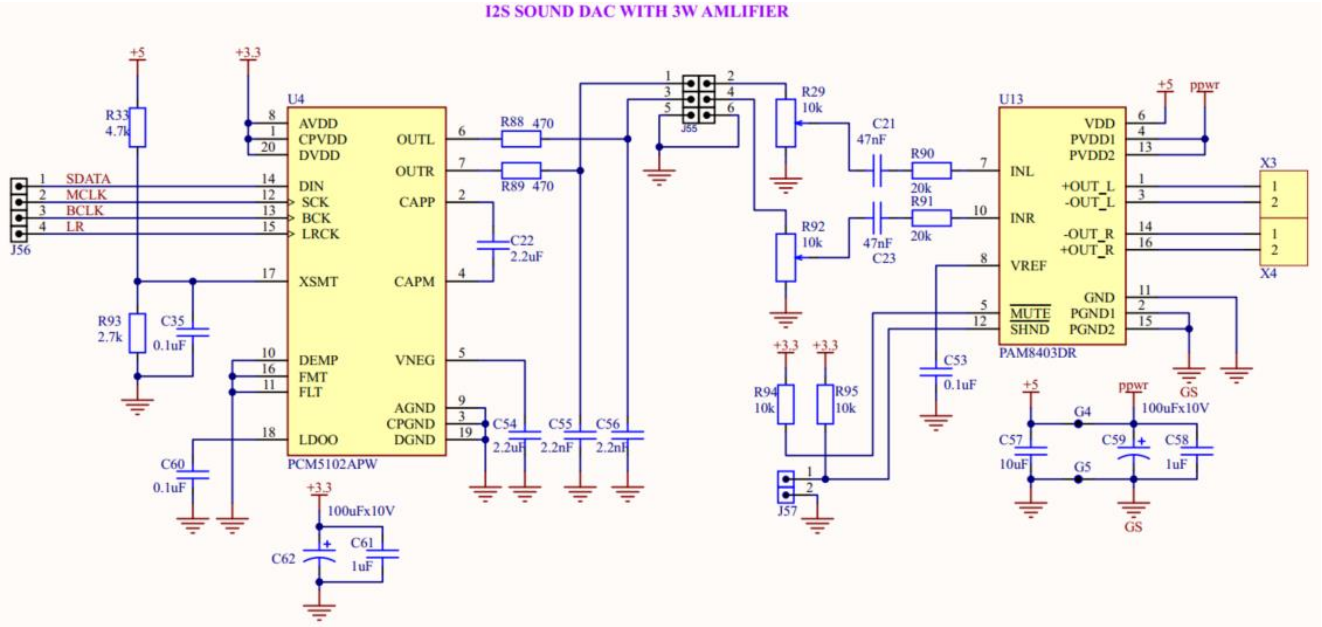
שקול תרחיש שבו יש לך חיישן טמפרטורה של 3.3 V ומיקרו-בקר של 5. V. על ידי שימוש ב-TXS0108EPW, אתה יכול לממשק את החיישן עם המיקרו-בקר ללא בעיות תאימות מתח, מה שמבטיח מדידות טמפרטורה מדויקות.

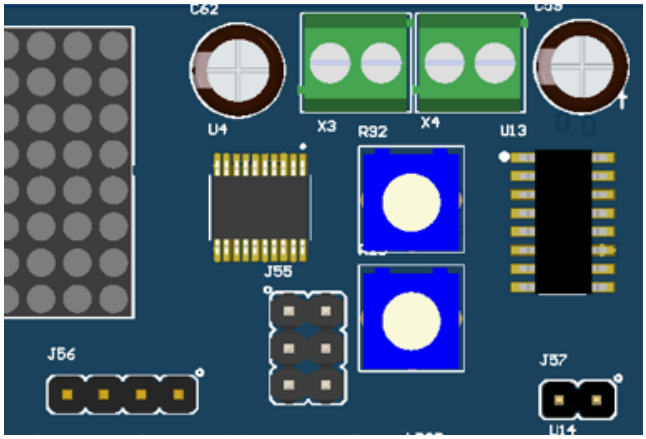
דוגמה 3: אינטגרציה היקפית במתח מעורב

תאר לעצמך מערכת שבה אתה צריך לממשק ציוד היקפי מרובים, כגון צגי LCD, נוריות ודרייברי מנוע, הפועלת ברמות מתח שונות עם מיקרו-בקר יחיד. ה-TXS0108EPW יכול לשמש כגשר בין המיקרו-בקר לציוד היקפי, ומאפשר תקשורת חלקה על פני כל המערכת.

לסיכום, מחליף הרמות TXS0108EPW מציע פתרון רב-תכליתי להתממשקות התקנים עם רמות מתח לא תואמות במערכות מתח מעורבות. יכולת התרגום הדו-כיווני שלו, טווח המתח הרחב והפעולה המהירה שלו הופכים אותו למרכיב חיוני ביישומים אלקטרוניים שונים. בין אם אתה משלב חיישנים, מיקרו-בקרים או התקנים היקפיים, ה-TXS0108EPW מספק תרגום רמת מתח אמין, המבטיח חילופי נתונים חלקים ויכולת פעולה הדדית של המערכת.

הפעלת אודיו (שמע) בעזרת I2S.





הבנת טכנולוגיית I2S: מדריך מקיף

מבוא ל-I2S:

Inter-IC Sound (I2S) הוא תקן ממשק אפיק טורי המשמש לחיבור התקני שמע דיגיטליים. זה מקל על העברה של נתוני אודיו דיגיטליים בין מעגלים משולבים (ICs) במערכות שמע כגון קודקים אודיו, מעבדי אותות דיגיטליים (DSPs), מיקרו-בקרים וממירים דיגיטליים לאנלוגיים (DAC). במאמר זה, נתעמק ביסודות טכנולוגיית I2S, תכונותיה, היישומים והיישום שלה במיקרו-בקר ESP32.

יסודות I2S:

I2S פועלת על ארכיטקטורת מאסטר-עבד ומורכבת משלושה קווים עיקריים:

קו נתונים טורי (SD): משדר נתוני אודיו דיגיטליים בפורמט טורי.

שעון טורי (SCK): מספק את אות השעון לסנכרון שידור נתונים.

בחירת מילים (WS או LRCLK): מציינת את ההתחלה והסוף של כל מסגרת לדגימת אודיו.

תכונות של I2S:

שמע באיכות גבוהה: I2S תומך בפורמטי שמע ברזולוציה גבוהה עם קצבי דגימה של עד 192 קילו-הרץ ורזולוציה של עד 32 סיביות.

תקשורת סינכרונית: העברת הנתונים ב-I2S היא סינכרונית, מה שמבטיח יישור תזמון מדויק בין דגימות אודיו.

תמיכה מרובה ערוצים: I2S יכול להתמודד עם ערוצי אודיו מרובים, מה שמאפשר תצורות סטריאו, צליל היקפי ותצורות אודיו רב-ערוציות.

חביון נמוך: I2S מציע תקשורת עם חביון נמוך, מה שהופך אותו מתאים ליישומי עיבוד אודיו בזמן אמת כגון ערבוב אודיו דיגיטלי, סינון ועיבוד אפקטים.

יישום I2S ב-ESP32:

המיקרו-בקר ESP32 כולל תמיכה מובנית בתקשורת I2S, מה שהופך אותו לאידיאלי עבור פרויקטים הקשורים לאודיו. היבטים מרכזיים של הטמעת I2S ב-ESP32 כוללים:

תמיכת חומרה: ה-ESP32 מספק ציוד היקפי ייעודי של I2S עם פינים והגדרות הניתנים להגדרה לשידור וקבלה של נתוני אודיו דיגיטליים.

גמישות: ה-ESP32 תומך בפורמטים שונים של שמע, קצבי דגימה ועומקי סיביות, מה שמאפשר תצורה גמישה כדי לעמוד בדרישות של יישומי שמע שונים.

תכונות משולבות: לוחות פיתוח ESP32 כוללים לרוב תכונות נוספות הקשורות לאודיו כגון DACs, ADCs, מגברים ו-codec שמע, המאפשרים אינטגרציה מקיפה של מערכת שמע.

ספריות תוכנה: ESP-IDF (Espressif IoT Development Framework) מספקת ספריות תוכנה וממשקי API לקביעת תצורה ושליטה של ציוד היקפי I2S, ומפשטת את הפיתוח של יישומי אודיו בפלטפורמת ESP32.

יישומים של I2S ב-ESP32:

השמעת אודיו: מכשירים מבוססי ESP32 יכולים לנגן אודיו ממקורות שונים כגון כרטיסי SD, זיכרון פלאש SPI, מודולי Bluetooth ושירותי סטרימינג באמצעות תקשורת I2S.

הקלטת אודיו: ה-ESP32 יכול ללכוד קלט שמע ממיקרופונים חיצוניים או מקורות ליין-אין ולעבד אותו בזמן אמת באמצעות ממשקי I2S.

זיהוי קולי: I2S מאפשר למכשירי ESP32 לבצע זיהוי קולי ועיבוד פקודות קוליות עבור יישומים כגון אוטומציה ביתית, עוזרי קול והמרת דיבור לטקסט.

עיבוד אותות דיגיטלי: מערכות מבוססות ESP32 יכולות ליישם אלגוריתמים של עיבוד אותות דיגיטליים (DSP) כגון שוויון, סינון והפחתת רעש באמצעות I2S עבור קלט ופלט אודיו.

דוגמאות להפעלת I2S.

הקלט אודיו ממיקרופון I2S ואחסן אותו בכרטיס SD

```
#include <Arduino.h>
#include <driver/i2s.h>
#include "SD.h"

const int SAMPLE_RATE = 44100;
const int BITS_PER_SAMPLE = I2S_BITS_PER_SAMPLE_16BIT;
const int CHANNELS = 2;

void setup() {
  Serial.begin(115200);
  i2s_config_t i2s_config = {
```

```
    .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_RX),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = BITS_PER_SAMPLE,
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = i2s_comm_format_t(I2S_COMM_FORMAT_I2S |
I2S_COMM_FORMAT_I2S_MSB),
    .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
  };
  i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
}

void loop() {
  size_t bytes_read;
  int16_t data[64];
  i2s_read(I2S_NUM_0, data, sizeof(data), &bytes_read, portMAX_DELAY);
  // Process audio data here

  // Example: Store audio data to SD card
  File file = SD.open("/audio.raw", FILE_APPEND);
  if (file) {
    file.write((uint8_t*)data, bytes_read);
    file.close();
  } else {
    Serial.println("Error opening file");
  }
}
```

הפעלת אודיו המאוחסן בכרטיס SD דרך I2S DAC

```
#include <Arduino.h>
#include <driver/i2s.h>
#include "SD.h"

const int SAMPLE_RATE = 44100;
const int BITS_PER_SAMPLE = I2S_BITS_PER_SAMPLE_16BIT;
const int CHANNELS = 2;

void setup() {
  Serial.begin(115200);
  i2s_config_t i2s_config = {
    .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_TX),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = BITS_PER_SAMPLE,
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = i2s_comm_format_t(I2S_COMM_FORMAT_I2S |
I2S_COMM_FORMAT_I2S_MSB),
    .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
```

```
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
  };
  i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
  SD.begin(5);
}

void loop() {
  // Read audio data from SD card
  File file = SD.open("/audio.raw");
  if (file) {
    size_t bytes_read;
    int16_t data[64];
    while (file.available()) {
      bytes_read = file.read((uint8_t*)data, sizeof(data));
      i2s_write(I2S_NUM_0, data, bytes_read, portMAX_DELAY);
    }
    file.close();
  } else {
    Serial.println("Error opening file");
  }
}
```

יצירת גל סינוס והפעל אותו דרך I2S DAC.

```
#include <Arduino.h>
#include <driver/i2s.h>
#include <math.h>
const int SAMPLE_RATE = 44100;
const int BITS_PER_SAMPLE = I2S_BITS_PER_SAMPLE_16BIT;
const int CHANNELS = 2;
void setup() {
  Serial.begin(115200);
  i2s_config_t i2s_config = {
    .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_TX),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = BITS_PER_SAMPLE,
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = i2s_comm_format_t(I2S_COMM_FORMAT_I2S |
I2S_COMM_FORMAT_I2S_MSB),
    .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
  };
  i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
}
void loop() {
```

```
  size_t bytes_written;
  const int samples_per_cycle = SAMPLE_RATE / 440; // 440 Hz sine wave
  int16_t data[64];
  for (int i = 0; i < sizeof(data) / 2; i++) {
    float angle = 2 * M_PI * i / samples_per_cycle;
    data[i] = 32767 * sin(angle); // 16-bit amplitude
  }
  i2s_write(I2S_NUM_0, data, sizeof(data), &bytes_written,
portMAX_DELAY);
}
```

קוד שלכוד נתוני אודיו ממיקרופון I2S ולעבד אותם בזמן אמת:

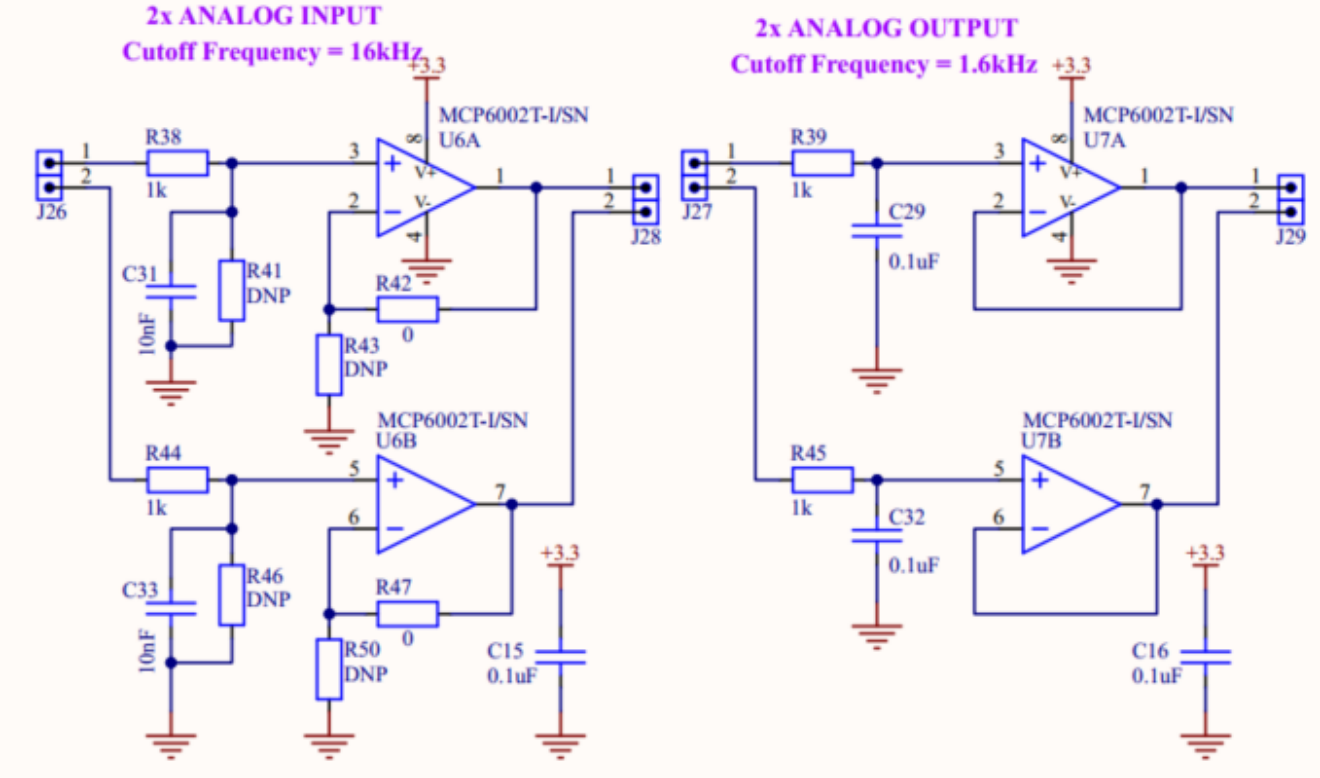
```
#include <Arduino.h>
#include <driver/i2s.h>

const int SAMPLE_RATE = 44100;
const int BITS_PER_SAMPLE = I2S_BITS_PER_SAMPLE_16BIT;
const int CHANNELS = 2;

void setup() {
  Serial.begin(115200);
  i2s_config_t i2s_config = {
    .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_RX),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = BITS_PER_SAMPLE,
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = i2s_comm_format_t(I2S_COMM_FORMAT_I2S |
I2S_COMM_FORMAT_I2S_MSB),
    .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
  };
  i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
}

void loop() {
  size_t bytes_read;
  int16_t data[64];
  i2s_read(I2S_NUM_0, data, sizeof(data), &bytes_read, portMAX_DELAY);
  // Process audio data here
}
```

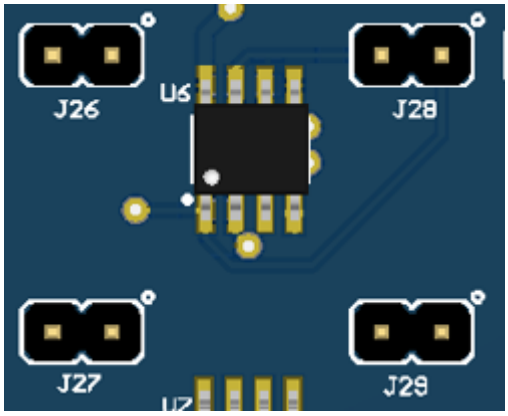
השמע אודיו המאוחסן ב-PROGMEM (זיכרון תוכנית) דרך I2S DAC

```
#include <Arduino.h>
#include <driver/i2s.h>

const int SAMPLE_RATE = 44100;
const int BITS_PER_SAMPLE = I2S_BITS_PER_SAMPLE_16BIT;
```

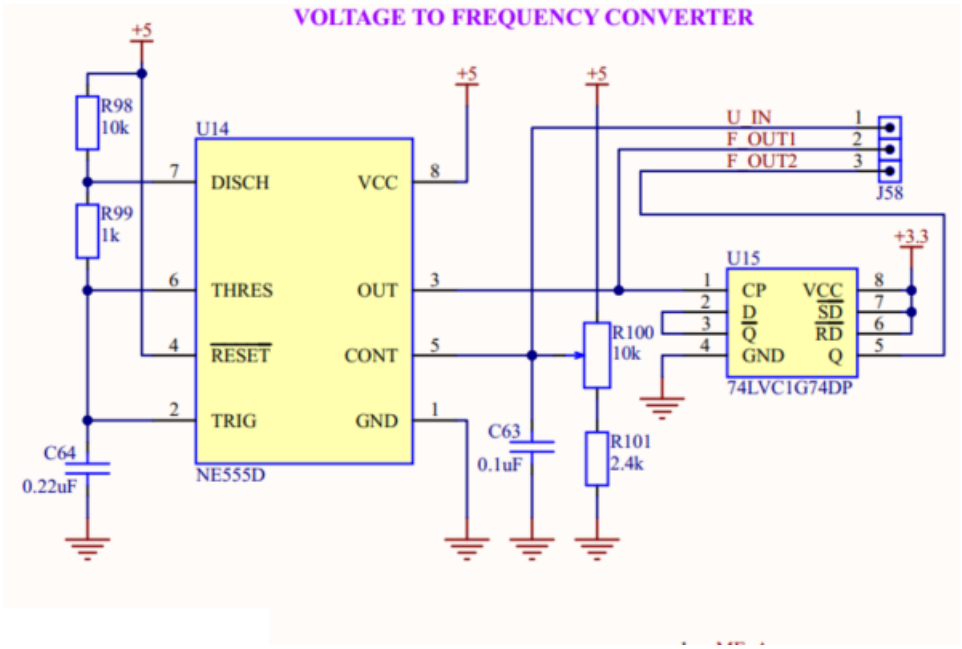
```
const int CHANNELS = 2;

const int16_t audio_data[] PROGMEM = { /* Insert audio data here */ };

void setup() {
  Serial.begin(115200);
  i2s_config_t i2s_config = {
    .mode = i2s_mode_t(I2S_MODE_MASTER | I2S_MODE_TX),
    .sample_rate = SAMPLE_RATE,
    .bits_per_sample = BITS_PER_SAMPLE,
    .channel_format = I2S_CHANNEL_FMT_RIGHT_LEFT,
    .communication_format = i2s_comm_format_t(I2S_COMM_FORMAT_I2S |
I2S_COMM_FORMAT_I2S_MSB),
    .intr_alloc_flags = ESP_INTR_FLAG_LEVEL1,
    .dma_buf_count = 8,
    .dma_buf_len = 64,
    .use_apll = false
  };
  i2s_driver_install(I2S_NUM_0, &i2s_config, 0, NULL);
}

void loop() {
  size_t bytes_written;
  i2s_write(I2S_NUM_0, audio_data, sizeof(audio_data), &bytes_written,
portMAX_DELAY);
}
```

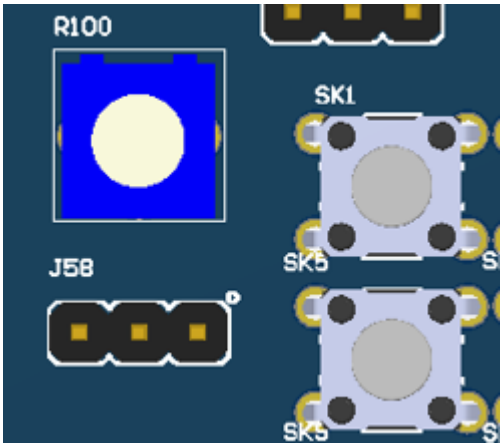
מגברים עם מסננים.





ממיר מתח לתדר.





הבנת המרה תדר למתח (FtoV) והמרת מתח לתדר (VtoF): יישומים בפרויקטים של מיקרו-בקרים

מבוא:

המרות של תדר למתח (FtoV) ומתח לתדר (VtoF) הן טכניקות חיוניות המשמשות בפרויקטים שונים מבוססי מיקרו-בקר להתממשקות עם חיישנים, הפקת אותות אנלוגיים ומדידה של תדרים. במאמר זה, נחקור את המושגים של המרות FtoV ו-VtoF, היישומים שלהם וכיצד הם מיושמים בפרויקטים של מיקרו-בקרים.

המרת תדר למתח (FtoV):

המרת FtoV כוללת המרת אות תדר לאות מתח פרופורציונלי. המרה זו משמשת בדרך כלל בפרויקטים שבהם נדרשות מדידות או בקרה מבוססות תדר. ממירי FtoV משתמשים בדרך כלל ב-IC של המרת תדר למתח או במעגלים כגון מתנדים מבוקרי מתח (VCOs) ולולאות נעולות פאזה (PLLs). מתח המוצא של ממיר FtoV הוא פרופורציונלי ישירות לתדר הכניסה, מה שמאפשר התממשקות קלה לממירים אנלוגיים לדיגיטליים (ADC) או מעגלים אנלוגיים אחרים בפרויקטים של מיקרו-בקרים.

המרת מתח לתדר (VtoF):

המרת VtoF, לעומת זאת, כוללת המרת אות מתח לאות תדר פרופורציונלי. המרה זו שימושית להפקת אותות עם אפנון תדר, אותות אפנון רוחב דופק (PWM), או למדידת מתחים אנלוגיים באמצעות טכניקות מבוססות תדר. ניתן ליישם ממירי VtoF באמצעות מיקרו-בקרים עם טיימרים מובנים ומחוללי פולסים, או עם ICs ייעודיים של ממירי מתח לתדר. תדר המוצא של

ממיר VtoF עומד ביחס ישר למתח הכניסה, מה שהופך אותו למתאים ליישומים כגון המרה אנלוגית לדיגיטלית, ממשק חיישנים ויצירת צורות גל בפרויקטים של מיקרו-בקרים.

יישומים בפרויקטים של מיקרו-בקרים:

ממשק חיישן: המרות FtoV ו-VtoF משמשות בדרך כלל להתממשק עם חיישנים אנלוגיים המוציאים אותות תדר או מתח, כגון חיישני טמפרטורה, חיישני לחץ ומדדי זרימה. על ידי המרת פלטי חיישנים לאותות דיגיטליים, מיקרו-בקרים יכולים לעבד ולנתח נתוני חיישנים עבור יישומים שונים.

יצירת אותות אנלוגיים: ניתן להשתמש בממירי VtoF ליצירת אותות מאופנים בתדר או אותות PWM לשליטה במנועים, מפעילים והתקנים אנלוגיים אחרים במערכות מבוססות מיקרו-בקר. על ידי שינוי מתח הכניסה, ניתן להתאים את תדירות אות המוצא כדי להשיג שליטה ואפנון מדויקים.

מדידת תדר: ניתן להשתמש בממירי FtoV כדי למדוד את התדירות של אותות חיצוניים או צורות גל בפרויקטים של מיקרו-בקר. יכולת זו שימושית למשימות כגון ניתוח תדרים, ניטור אותות ואבחון מערכת.

המרה אנלוגית לדיגיטלית: ניתן להשתמש בממירי VtoF כחלק מטכניקות המרה אנלוגית לדיגיטלית, כאשר מתח הכניסה מומר לאות דיגיטלי על ידי מדידת התדר של צורת גל שנוצרת. גישה זו מציעה יתרונות כגון פשטות, דיוק וחסינות נגד רעש במערכות מדידה מבוססות מיקרו-בקר.

סיכום:

המרות של תדר למתח (FtoV) ומתח לתדר (VtoF) הן טכניקות רב-תכליתיות שמוצאות שימוש נרחב בפרויקטים של מיקרו-בקרים להתממשקות עם חיישנים, הפקת אותות אנלוגיים וביצוע מדידות מבוססות תדר. על ידי הבנת העקרונות של המרת FtoV ו-VtoF והיישומים שלהם, מפתחים יכולים לתכנן וליישם פרויקטים חדשניים הממנפים את היכולות של מיקרו-בקרים לעיבוד אותות אנלוגי, ממשק חיישנים ויישומי בקרה. בין אם מדידת יציאות חיישנים, הפקת צורות גל אנלוגיות או ביצוע המרה אנלוגית לדיגיטלית, המרות FtoV ו-VtoF ממלאות תפקיד מכריע במתן שילוב של פונקציות אנלוגיות ודיגיטליות במערכות מבוססות מיקרו-בקר.