ADC - ממירים מאנלוגי לדיגיטאלי ב Esp32

א. כללי

במאמר זה נסביר על ה ADC (ממיר מאנאלוגי לדיגיטאלי – Analog to Digital Converter) של ה ESP32 ונדגים יישום קריאת מתח אנאלוגי. אנחנו נעבוד עם סביבת העבודה של Arduino IDE .

ל ADC יש 2 אופני עבודה עליהם נרחיב בהמשך :

- **אופן מדידה בודדת - OneShot mode**

אופן זה תואם באופן מלא לפונקציה analogRead של Arduino. בעת קריאה לפונקציה analogRead או analogReadMillivolts היא מחזירה את התוצאה של **המרה יחידה** בהדק המבוקש.

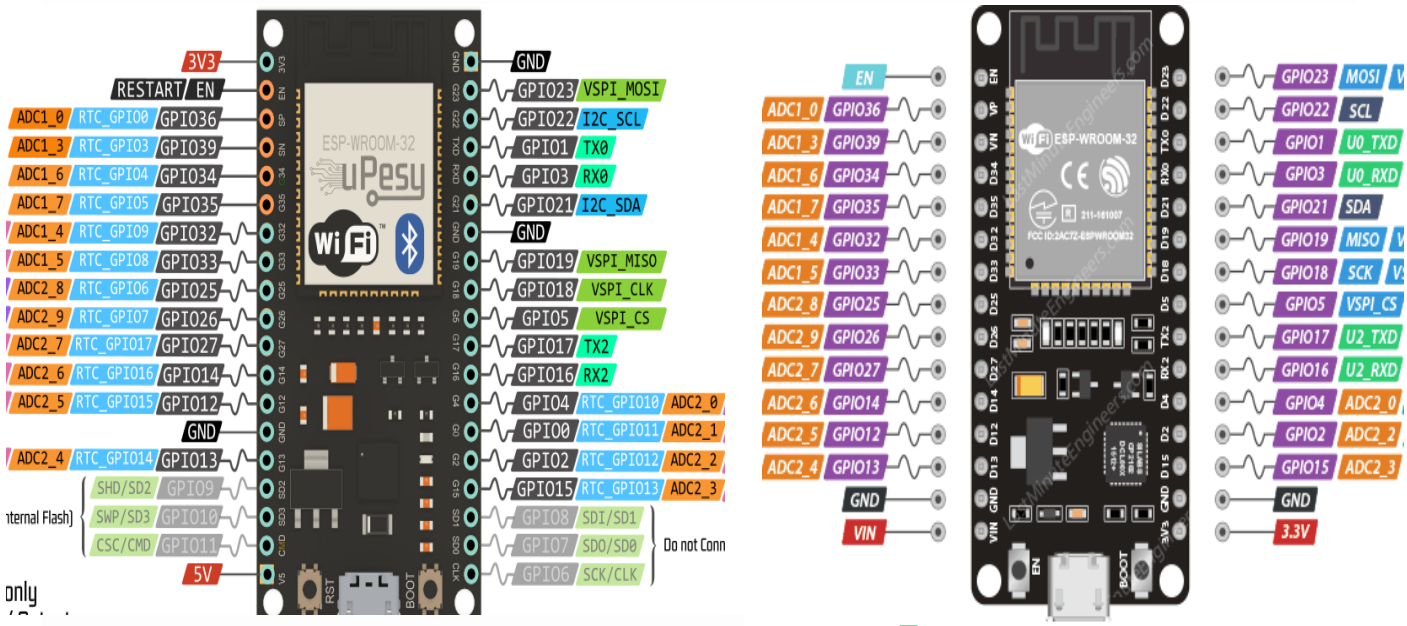
- אופן מדידה רציף - CONTINUOUS MODE

מצב רציף מיועד לביצוע המרות אנלוגיות על מספר הדקים ברקע עם התכונה של קבלת התקשרות חוזרת עם השלמת המרות אלה כדי לגשת לתוצאות. זה מאפשר לנו לציין את מספר ההמרות הרצוי לכל הדק בתוך מחזור אחד יחד עם קצב הדגימה המתאים שלו. התוצאה של הפונקציה analogContinuousRead היא מערך של מבני adc_continuous_data_t. מבנים אלה מחזיקים הן את הערך הממוצע הגולמי והן את הערך הממוצע במיליוולטים עבור כל סיכה.

ב. הדקי ה ADC של ה ESP32

ברכיב יש שני ממירי ADC מסוג SAR (Successive Approximation Register – רגיסטר הערכה מוצלחת). שני ה ADC הם של 12 ביטים כל אחד. בג'וק עצמו (לא ההדקים של כרטיסי ה ESP32 לסוגיהם השונים) יש ל ADC1 8 הדקים מ GPIO32 - GPIO39 ול ADC2 יש 10 הדקים GPIO0, GPIO2, GPIO4, GPIO12 - GPIO15, GOIO25 - GPIO27 .

עם זאת, לוח DEVKIT V1 DOIT (הגרסה עם 30 GPIOs) יש רק 15 ערוצי ADC ואילו לרכיב עם 38 ההדקים יש 16 ערוצים כאשר ההדק שנוסף הוא ADC2_1 כפי שמוצג באיור הבא (ההדקים מסומנים בצבע כתום).



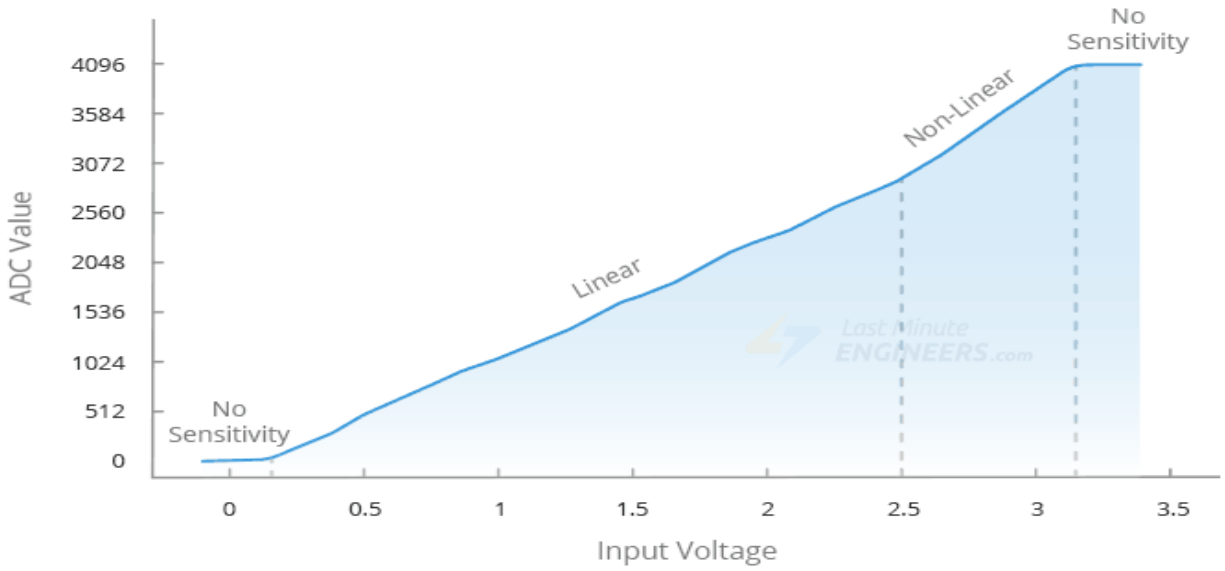
איור 1 : ההדקים של 2 ה ADCS . בצד ימין בכרטיס עם 30 ההדקים ובצד שמאל עם 38 ההדקים.

ל 2 הממירים יש רזולוציה של 12 סיביות כלומר הוא יכול לזהות 4096 (2^{12}) רמות אנלוגיות בדידות. במילים אחרות, הוא ימיר מתחי כניסה הנעים בין 0 ל- 3.3V (מתח הפעלה) לערכים שלמים הנעים בין 0 ל- 4095. התוצאה היא רזולוציה של 3.3 וולט / 4096 יחידות, או 0.0008056640625 וולט (כ 0.8mV) ליחידה. יתר על כן, ניתן להגדיר את רזולוציית ה ADC ואת טווח הערוצים בעזרת תכנות.

ג. מגבלות הממירים

ישנן מספר מגבלות לממירים :

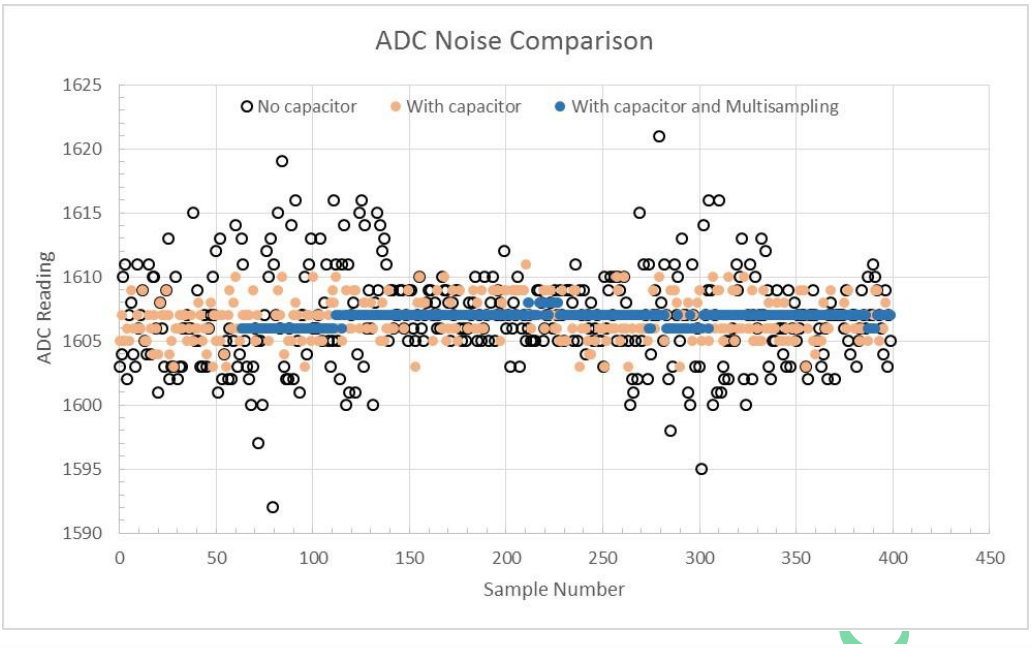
- **חלק מפיני ADC2 משמשים כ strapping pins** – הדקי קשירה - הדקים המשמשים באתחול או צריבת זיכרון הרכיב (GPIO 0, 2, 15) - ולכן לא ניתן להשתמש בהם באופן חופשי. זה המצב בערכות הפיתוח הרשמיות הבאות: ESP32 DevKitC: לא ניתן להשתמש ב-GPIO 0 עקב מעגלים חיצוניים של תוכנית אוטומטית. ESP-WROVER-KIT: לא ניתן להשתמש ב-GPIO 0, 2, 4 ו-15 עקב חיבורים חיצוניים למטרות שונות.
- **לא ניתן להשתמש בהדקי ADC2** כאשר מפעילים Wi-Fi אבל ניתן להשתמש בהדקי ADC1 .
- ניתן למדוד מתחים מ 0 ועד ל 3.3 וולט בלבד . (**לא ניתן למדוד ישירות מתחים מ 0 עד 5 וולט**) .
- **הממירים ב ESP32 אינם ליניאריים ולכן הדיוק שלהם איננו אידיאלי**. הגרף באיור הבא מסביר את אי הליניאריות של הממירים:



איור 2 : גרף ליניאריות כפונקציה של מתח הכניסה.

באיור רואים בגרף את הלא-ליניאריות בקצה התחתון והעליון של מתח הכניסה . זה בעצם אומר כי ESP32 לא יכול להבחין בין 3.2V עד 3.3V; הערך הנמדד שנקבל יהיה זהה ותוצאת המדידה תהיה 4095. באופן דומה, הוא אינו יכול להבחין בין אותות של 0 עד 0.13V והערך הנמדד יהיה זהה ותוצאת המדידה תהיה 0.

- **רעש חשמלי** - ה ADC יכול להיות רגיש לרעש, מה שמוביל לפערים גדולים בקריאות ה ADC. בהתאם ליישום ולשימוש. אפשר לחבר קבל מעקף/סינון (לדוגמה, קבל קרמי של 100 nF) קרוב ככל שניתן להדק ה ADC שבשימוש, כדי למזער את הרעש. גם פעולת multisampling – ריבוי דגימות (ביצוע מספר דגימות ובעזרת אלגוריתם לשפר את הדיוק) יכולה להפחית עוד יותר את ההשפעות של רעש.
- האיור הבא מראה דגימות של קריאות ADC ללא הפחתת רעש ועם הפחתת רעשים באמצעות קבל וריבוי דגימות.



איור 3 : גרף הממחיש הפחתת רעשים באמצעות קבל וריבוי דגימות של 64 דגימות.

עבור מתח כלשהו . הציר האופקי מראה 400 דגימות שבוצעו על מתח כלשהו (1.3 וולט שנותן קריאה של 1607) והציר האנכי מתאר את קריאת ה ADC עבור כל דגימה. העיגולים השחורים מתארים את הקריאה עבור כל דגימה ללא קבל וללא multisampling . רואים שיש הרבה רעש ואפילו קריאות של למעלה מ 1620

בדגימה 280 ו 1592 בדגימה 80 .

העיגולים בצבע כתום מתארים את הקריאות עם הקבל . רואים שיש קריאות עם הרבה פחות השפעה של רעש.

העיגולים הכחולים מתארים את הקריאות גם עם קבל וגם עם multisampling . במקרה כזה רואים שיפור ניכר בקריאות הממיר.

ד. הפונקציה analogRead()

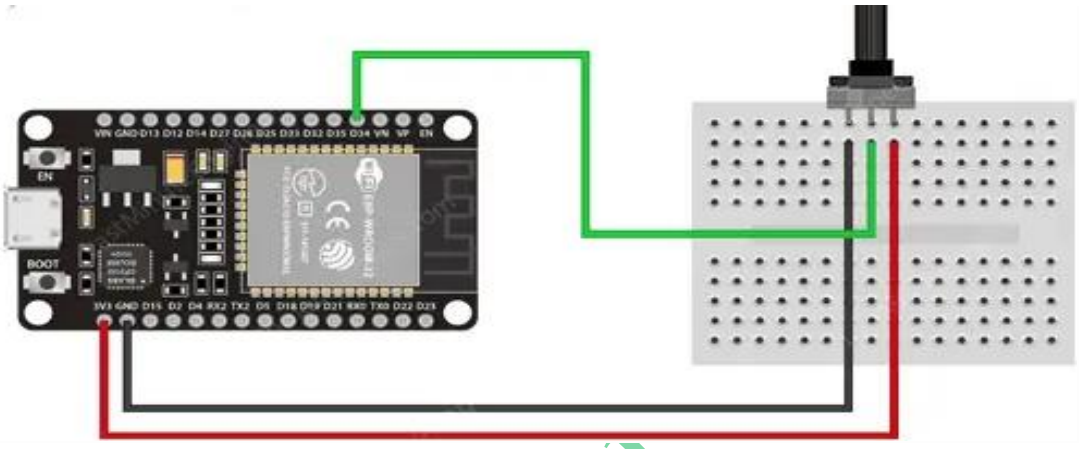
קריאת הערכים האנלוגיים מהדק GPIO היא פשוטה. ב- IDE של Arduino, נשתמש בפונקציה analogRead( ), המקבלת כארגומנט את מספר ההדק של GPIO שנרצה לקרוא. לדוגמה אם נרצה לקרוא את נערך של המתח שנכנס בהדק ADC1_0 שהוא הדק 36 נרשום :

```
analogRead(36);
```

קיימות פונקציות נוספות שנוכל להשתמש בהן בפרויקטים נוספים. הן נמצאות בהמשך המאמר.

ה. קריאה של פוטנציומטר

נשתמש בדוגמה פשוטה הקוראת ערך אנלוגי מפוטנציומטר. נחבר את המעגל הנראה באיור הבא שבו פוטנציומטר מתחבר אל לוח ESP32 . ניקח פוטנציומטר של 1 קילו אוהם (למעשה כל פוטנציומטר בסדר גודל של מאות אוהם עד עשרות קילו אוהם יהיה טוב לניסוי.) ונחבר את הזחלן שלו – הרגל האמצעית של הפוטנציומטר - להדק GPIO34 . הדבר מתואר באיור הבא:



איור 4 : חיבור פוטנציומטר אל ESP32 (התמונה מהאתר של Last Minute ENGINEERS.com).

נרשום תוכנית הקוראת את המתח בזחלן הפוטנציומטר ומדפיסה את התוצאות לצג הטורי.

```
//ADC1_6 שהוא GPIO34 הפוטנציומטר מתחבר ל
const int potPin = 34;

// משתנה לשמירת הערך הנקרא מהפוטנציומטר
int potValue = 0;

void setup()
{
  Serial.begin(115200); // אתחול התקשורת הטורית עם מסך המוניטור
  delay(1000);
}

void loop()
{
```

```
  // קריאת ערך הפוטנציומטר
  potValue = analogRead(potPin);   //  potPin = 34
  Serial.print("Analog value: ");  //  ": הערך האנאלוגי " הדפסה למסך
  Serial.println(potValue); // הדפסת הערך שקראנו
  delay(500);
}
```

לאחר שרשמנו את התוכנית נפתח את הצג הטורי בקצב שידור 115200 ונלחץ על לחצן EN ב- ESP32. אנחנו אמורים לראות ערך בין 0 ל- 4095 . כאשר הזחלן נמצא באדמה נקרא 0 וכאשר הוא בצד ה 3.3 וולט נקרא 4095. סיבוב של הזחלן ישנה את הקריאות שנקבל.

ו.פונקציות נוספות של ADC

ו.1 אופן קריאה בודדת - OneShot

- **קריאת הערך במילי וולטים.**

```
analogReadMilliVolts(מספר ההדק );
```

קבל ערך ADC עבור במילי וולטים עבור קריאה מההדק המסוים.

- **קביעת הרזולוציה**

```
analogReadResokution(מספר הביטים הרצוי );
```

מגדיר את סיביות הדגימה ואת רזולוציית הקריאה. ברירת המחדל היא רזולוציה של 12 סיביות. אם נרשום 9 הטווח יהיה מ 0 עד 511 . עבור 12 ביטים הטווח מ 0 עד 4095 .

לדוגמה : נניח שהכנסנו מתח בדיקה אנאלוגי של 3.3 וולט ורשמנו analogReadResolution(9); המספר שנקבל הוא 511 . אם נרשום analogReadResolution(10); המספר שנקבל הוא 1023 . אם נרשום analogReadResolution(12); המספר שנקבל הוא 4095 .

ב ESP32S3 ברירת המחדל של הרזולוציה היא 13 ביטים.

- **קביעת ההנחתה - analogSetAttenuation(adc_attenuation_t attenuation) ;**

פונקציה זו משמשת להגדרת ההנחתה עבור כל הערוצים. ניתן להנחית את מתחי כניסה לפני הכניסה ל- ADC. ישנן 4 אפשרויות הנחתה . ככל שההנחתה גבוהה יותר, כך מתח הכניסה הנמדד יכול להיות גבוה יותר. מתח הכניסה הנמדד משתנה עבור כל ג'וק/רכיב במשפחת ה ESP32 . הטבלה הבאה מראה את האפשרויות השונות :

ESP32

Attenuation	Measurable input voltage range
ADC_ATTEN_DB_0	100 mV ~ 950 mV
ADC_ATTEN_DB_2_5	100 mV ~ 1250 mV
ADC_ATTEN_DB_6	150 mV ~ 1750 mV
ADC_ATTEN_DB_11	150 mV ~ 3100 mV

ESP32-S2

Attenuation	Measurable input voltage range
ADC_ATTEN_DB_0	0 mV ~ 750 mV
ADC_ATTEN_DB_2_5	0 mV ~ 1050 mV
ADC_ATTEN_DB_6	0 mV ~ 1300 mV
ADC_ATTEN_DB_11	0 mV ~ 2500 mV

ESP32-C3

Attenuation	Measurable input voltage range
ADC_ATTEN_DB_0	0 mV ~ 750 mV
ADC_ATTEN_DB_2_5	0 mV ~ 1050 mV
ADC_ATTEN_DB_6	0 mV ~ 1300 mV
ADC_ATTEN_DB_11	0 mV ~ 2500 mV

ESP32-S3

Attenuation	Measurable input voltage range
ADC_ATTEN_DB_0	0 mV ~ 950 mV
ADC_ATTEN_DB_2_5	0 mV ~ 1250 mV
ADC_ATTEN_DB_6	0 mV ~ 1750 mV
ADC_ATTEN_DB_11	0 mV ~ 3100 mV

טבלה 1 : אפשרויות ההנחתה השונות עבור כל רכיב ESP32 .

- **קביעת הדק הכניסה וההנחתה שתתבצע על ההדק –**

analogSetPinAttenuation(**uint8_t** pin, adc_attenuation_t attenuation);

פונקציה זו משמשת להגדרת הנחתה עבור ערוץ PIN/ADC ספציפי. **uint8_t** pin מציין את ההדק עליו תתבצע ההנחתה.

adc_attenuation_t attenuation – מציין את גודל ההנחתה (ניתן להיעזר בפיסקה הקודמת **analogSetAttenuation**).

- ***קביעת רוחב סיביות הדגימה - ANALOGSETWIDTH(BITS)***

מגדיר את סיביות דגימת החומרה ואת רזולוציית הקריאה. ברירת המחדל היא רזולוציה של 12 סיביות. טווח: 9 עד 12 סיביות. 9 סיביות : 0-511 , 10 סיביות : 0-1023 , 11 סיביות : 0-2047 ו- 12 סיביות : 0-4095.

פונקציה זו זמינה רק עבור שבב/ג'וק ESP32 .

2.1 דוגמה לקריאת ADC באופן OneShot גם בוולטים וגם במילי וולטים בהדק 2 שהוא ADC2_2 :

הדוגמה נמצאת בקישור : ADC — Arduino-ESP32 2.0.14 documentation (readthedocs-hosted.com)

```
void setup() {
  // initialize serial communication at 115200 bits per second:
  Serial.begin(115200);

  //set the resolution to 12 bits (0-4096)
  analogReadResolution(12);
}

void loop() {
  // read the analog / millivolts value for pin 2:
  int analogValue = analogRead(2);
  int analogVolts = analogReadMilliVolts(2);

  // print out the values you read:
  Serial.printf("ADC analog value = %d\n",analogValue);
  Serial.printf("ADC millivolts value = %d\n",analogVolts);

  delay(100);  // delay in between reads for clear read from serial
}
```

3.ו אופן קריאה רציפה - Continuous mode

אופן קריאה רציפה מיועד לביצוע המרות אנלוגיות על מספר פינים ברקע עם התכונה של קבלת התקשרות חוזרת עם השלמת המרות אלה כדי לגשת לתוצאות. אופן זה מאפשר לנו לציין את מספר ההמרות הרצוי לכל הדק בתוך מחזור אחד, יחד עם קצב הדגימה המתאים שלו. התוצאה של הפונקציה analogContinuousRead( ) היא מערך של מבני adc_continuous_data_t. מבנים אלה מחזיקים הן את הערך הממוצע והן את הערך הממוצע במילי וולטים עבור כל הדק.

- **הפונקציה analogContinuous( )**

פונקציה זו משמשת לקביעת תצורה של מדידה רציפה של ציוד היקפי בפינים שנבחרו. הפונקציה היא :

```
bool analogContinuous(uint8_t pins[], size_t pins_count, uint32_t conversions_per_pin,
uint32_t sampling_freq_hz, void (*userFunc)(void));
```

הפונקציה מחזירה True אם הגדרת התצורה תצליח. אם מוחזר false, מתרחשת שגיאה ו- ADC רציף לא הוגדר.

הפונקציה מקבלת :

uint8_t pins[] - מערך של מספרי ההדקים/פינים שעליהם מבצעים דגימות,

size_t pins_count - ספירת פינים במערך .

uint32_t conversions_per_pin - מספר ההמרות לכל הדק .

sampling_freq_hz - תדירות הדגימה של ה ADC בהרצים.

void (*userFunc)(void) - מגדיר פונקציית התקשרות חוזרת לקריאה לאחר ביצוע המרת adc (ניתן להגדיר ל- NULL)

- ***פונקציית הקריאה ANALOGCONTINUOUSREAD***

פונקציה זו משמשת לקריאת נתונים רציפים של ADC למאגר התוצאות. מאגר התוצאות הוא מערך של adc_continuos_data_t.

המבנה מטיפוס adc_continuos_data_t נראה כך :

```
typedef struct {
    uint8_t pin;          /*!<ADC pin */
    uint8_t channel;      /*!<ADC channel */
    int avg_read_raw;     /*!<ADC average raw data */
    int avg_read_mvolts;  /*!<ADC average voltage in mV */
```

} adc_continuos_data_t;

הפונקציה לקריאה נראית כך :

```
bool analogContinuousRead(adc_continuos_data_t ** buffer, uint32_t timeout_ms);
```

ה buffer הוא חוצץ/מאגר תוצאות ההמרה לקריאה מ- ADC בפורמט adc_continuos_data_t.

timeout_ms הוא זמן ההמתנה לנתונים באלפיות שנייה.

פונקציה מחזירה True אם הקריאה מוצלחת והמאגר מלא בנתונים. אם מוחזר false - הקריאה נכשלה והמאגר מוגדר כ- NULL.

- ***הפונקציה ANALOGCONTINUOUSSTART ( )***

פונקציה זו משמשת להפעלת המרות רציפות של ADC. analogContinuousStart()bool; פונקציה זו תחזיר True אם ADC רציף מופעל בהצלחה. אם מוחזר false - הפעלת ADC רציף נכשלה.

- **הפונקציה analogContinuousStop**

פונקציה זו משמשת לעצירת ההמרות הרציפות של ה ADC.

```
bool analogContinuousStop();
```

פונקציה זו מחזירה True אם ADC רציף מופסק בהצלחה. אם מוחזר false - עצירת ADC רציף נכשלה.

- **הפונקציה analogContinuousDeinit( )**

פונקציה זו משמשת לביטול אתחול ADC ציוד היקפי רציף.

הפונקציה מוגדרת כך:

```
bool analogContinuousDeinit();
```

פונקציה זו תחזיר True אם ADC רציף מבוטל בהצלחה. אם מוחזר false - ביטול האתחול של ADC רציף נכשל.

- **הפונקציה analogContinuousSetAtten**

פונקציה זו משמשת להגדרת הנחתה עבור ADC רציף היקפי. לקבלת מידע נוסף ניתן לעיין ב- analogSetAttenuation בעמודים קודמים.

הפונקציה מוגדרת כך : `void analogContinuousSetAtten(adc_attenuation_t attenuation);`

attenuation – קביעת ההנחתה (ברירת המחדל היא 11db) . הסבר נוסף יש בעמודים קודמים עבור אופן העבודה OneShot .

- **הפונקציה analogContinuousSetWidth( )**

פונקציה זו משמשת להגדרת סיביות רזולוציית החומרה. ערך ברירת המחדל עבור כל השבבים הוא 12 סיביות (0 - 4095).

הפונקציה מוגדרת כך : `void analogContinuousSetWidth(uint8_t bits);`

bits – קביעת כמות הביטים של הרזולוציה.

הערה :פונקציה זו תיכנס לתוקף רק עבור שבב ESP32, שכן היא מאפשרת להגדיר רזולוציה בטווח 9-12 סיביות .

4.1 דוגמה לאופן continuous

```
// Define how many conversion per pin will happen and reading the data will be and average of all conversions
#define CONVERSIONS_PER_PIN 5

// Declare array of ADC pins that will be used for ADC Continuous mode - ONLY ADC1 pins are supported
// Number of selected pins can be from 1 to ALL ADC1 pins.
#ifdef CONFIG_IDF_TARGET_ESP32
uint8_t adc_pins[] = {36, 39, 34, 35}; //some of ADC1 pins for ESP32
#else
uint8_t adc_pins[] = {1, 2, 3, 4}; //ADC1 common pins for ESP32S2/S3 + ESP32C3/C6 + ESP32H2
#endif

// Calculate how many pins are declared in the array - needed as input for the setup function of ADC Continuous
uint8_t adc_pins_count = sizeof(adc_pins) / sizeof(uint8_t);

// Flag which will be set in ISR when conversion is done
volatile bool adc_coversion_done = false;

// Result structure for ADC Continuous reading
adc_continuos_data_t * result = NULL;

// ISR Function that will be triggered when ADC conversion is done
void ARDUINO_ISR_ATTR adcComplete() {
  adc_coversion_done = true;
```

```
www.arikporat.com

}

void setup() {
  // Initialize serial communication at 115200 bits per second:
  Serial.begin(115200);

  // Optional for ESP32: Set the resolution to 9-12 bits (default is 12 bits)
  analogContinuousSetWidth(12);

  // Optional: Set different attenaution (default is ADC_11db)
  analogContinuousSetAtten(ADC_11db);

  // Setup ADC Continuous with following input:
  // array of pins, count of the pins, how many conversions per pin in one cycle will happen, sampling frequency, callback function
  analogContinuous(adc_pins, adc_pins_count, CONVERSIONS_PER_PIN, 20000, &adcComplete);

  // Start ADC Continuous conversions
  analogContinuousStart();
}

void loop() {
  // Check if conversion is done and try to read data
  if (adc_coversion_done == true) {
    // Set ISR flag back to false
    adc_coversion_done = false;
    // Read data from ADC
    if (analogContinuousRead(&result, 0)) {

      // Optional: Stop ADC Continuous conversions to have more time to process (print) the data
      analogContinuousStop();
```

```
            for (int i = 0; i < adc_pins_count; i++) {
            Serial.printf("\nADC PIN %d data:", result[i].pin);
            Serial.printf("\n    Avg raw value = %d", result[i].avg_read_raw);
            Serial.printf("\n    Avg milivolts value = %d", result[i].avg_read_mvolts);
            }

            // Delay for better readability of ADC data
            delay(1000);

            // Optional: If ADC was stopped, start ADC conversions and wait for callback function to set adc_coversion_done flag to true
            analogContinuousStart();
        }
        else {
            Serial.println("Error occured during reading data. Set Core Debug Level to error or lower for more informations.");
        }
    }
}
```

ז. ביבליוגרפיה:

1. https://docs.espressif.com/projects/esp-idf/en/v4.4/esp32/api-reference/peripherals/adc.html
2. https://lastminuteengineers.com/esp32-basics-adc/
3. ADC — Arduino-ESP32 2.0.14 documentation (readthedocs-hosted.com)