

שמירת נתונים כקובץ בזיכרון ה FLASH - SPIFFS

1. מבוא

1.1 א. על מה נדבר במאמר זה ?

במאמר זה נתאר את ה - SPIFFS שהוא קיצור של Serial Peripheral Interface Flash File System - כלומר, ממשק טורי היקפי למערכת קבצים FLASH. זוהי אחת התכונות המיוחדות של ה ESP32. אפשר לתאר SPIFFS כמו כרטיס SD קטן מאוד בתוך שבב ה ESP32 עצמו. הממשק מטפל בקבצים שבזיכרון ה ESP32 כמו בקבצים בזיכרון שבמחשב. SPIFFS הוא ניהול זיכרון עבור ה ESP32. באופן ספציפי יותר, זהו אחסון זיכרון שאינו נמחק לאחר כיבוי ה ESP32. באפשרותנו להשתמש בו לרישום נתונים, שמירת קודי גישה, אחסון קבצים וזכירת הגדרות שהוזנו על-ידי המשתמש. SPIFFS ממוטב (optimized) לשימוש באירועים שבהם נדרשות פעולות קריאה וכתיבה תכופות. אני השתמשתי ב SPIFFS כאשר חיברתי תצוגת TFT חדשה לאחד מהפרויקטים שאני מנחה ורציתי לכייל את מסך המגע של התצוגה. בתחילת התוכנית, בפונקציית ה (`setup()`) כותבים קטע קוד שמצייר חץ בקודקוד התצוגה בחלק השמאלי של המסך. בעזרת פונקציית (`touch()`) בודקים שהמשתמש לוחץ על החץ, החץ עובר לקודקוד אחר. לאחר הלחיצה בכל אחד מ 4 הקודקודים של התצוגה, שומרים את נתוני ה x וה y של כל 4 הנקודות בקובץ בעזרת ה SPIFFS ואז כאשר מצירים מלבן בחירה עם הפונקציה/מתודה (`drawRect(x0,y0,w,h,color)`) ניתן לדעת האם המשתמש לחץ על נקודה כלשהי בתוך המלבן. דוגמה לקוד הכיול נמצא בקישור :

<https://github.com/DustinWatts/FreeTouchDeck/blob/master/Touch.h>

1.1 ב. מה אפשר לעשות עם SPIFFS ?

ביישומי IoT - Internet On Thing ובפרויקטים משובצים, לעתים קרובות עלינו לאחסן הגדרות תצורה (קונפיגורציה), עדכוני קושחה - Firmware, קבצי שרת אינטרנט ויומני נתוני חיישנים, בין היתר SPIFFS אידיאלי למטרה זו מכיוון שהוא מאפשר לארגן ולגשת לנתונים ביעילות.

1.1 ג. פרויקטים המשתמשים ב SPIFFS

- מכשירי IoT: מכשירים כמו תרמוסטטים חכמים, תחנות מזג אוויר ובקרי בית חכם משתמשים לעתים קרובות ב- SPIFFS לאחסון נתוני תצורה וקריאות חיישנים.
- שרתי אינטרנט: SPIFFS מאפשר לשרת דפי אינטרנט, תמונות ומשאבים אחרים ישירות מהמיקרו-בקר, וליצור יישומי אינטרנט ללא צורך באחסון חיצוני. פרויקטים עם ממשקי משתמש גרפיים (GUI) או לוחות בקרה הנגישים באמצעות דפדפן אינטרנט יכולים לאחסן את נכסי/נתוני האינטרנט שלהם ב- SPIFFS.
- עדכוני קושחה - Firmware: SPIFFS יכולים לאחסן עדכוני קושחה לפריסה מרחוק. הדבר מאפשר עדכונים דרך האוויר (OTA), ומאפשר לעדכן מרחוק את קושחת המכשיר ללא גישה פיזית.
- רישום ואחסון נתונים: ניתן להשתמש ב- SPIFFS למטרות רישום נתונים. אם הפרויקט שלנו כולל איסוף ואחסון של נתוני חיישנים, יומני אירועים או נתונים אחרים של סדרות זמן, SPIFFS מספק דרך יעילה לשמור מידע זה.

- ניטור סביבתי: תחנות מזג אוויר ומערכות ניטור סביבתי יכולות לרשום נתונים כמו טמפרטורה, לחות ואיכות אוויר ל - .SPIFFS
- אוטומציה תעשייתית: מערכות שרושמות נתוני מכונה, סטטיסטיקות ייצור או משתני תהליך.
- אוטומציה ביתית: פרויקטים המתעדים אירועים ונתוני חיישנים בבתים חכמים או בבניינים.
- מכשירים הניתנים להגדרות תצורה: מכשירים שבהם משתמשים יכולים להתאים קונפיגורציה אישית, כגון נתוני Wi Fi.
- פרופילים של משתמשים: מערכות המאחסנות פרופילים של משתמש או העדפות.

ד.1 מה קרה ל EEPROM ?

לאילו המתמצאים במיקרו-בקרים, הם בטח מכירים את EEPROM שהוא קיצור של Electrically Erasable Programmable Read-Only Memory - זיכרון לקריאה בלבד הניתן לתכנות עם מחיקה חשמלית. מבחינת ESP32 ה SPIFFS היא הגרסה המודרנית המעולה יותר של EEPROM שבו שמרנו נתונים שלא רצינו שיימחקו עם כיבוי החשמל. כדאי לשים לב ששמירת נתונים ב ESP32 עבור EEPROM יצאה משימוש ואין להשתמש בו עוד. אם נשתמש בציטוט מהקישור באתר של github : [github · espressif/arduino-esp32 · GitHub](https://github.com/espressif/arduino-esp32/blob/master/libraries/EEPROM) .

אם נתרגם את הנאמר: "EEPROM יצא משימוש. עבור יישומים חדשים ב-ESP32 – ויש להשתמש בהעדפות אחרות. ה EEPROM מסופק לתאימות לאחור עם יישומי Arduino קיימים. ה EEPROM מיושם באמצעות גוש/בלוק יחיד בתוך NVS שהוא קיצור של Non Volatile Storage – אחסון לא נדיף. שמירת נתונים זו לא צריכה להיות נעדיף בעלת ביצועים גבוהים. נעדיף להשתמש ב-nvs, ונאחסן כל ערך כאובייקט יחיד בתוכו".

בקישורים הבאים יש הסברים על סוגי הזיכרון השונים ב esp32 :

<https://docs.espressif.com/projects/esp-idf/en/latest/esp32/api-guides/memory-types.htm>

https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf#ysmem

בהתחשב בהתיישנות ה EEPROM - אילו אפשרויות אחרות קיימות לאחסון נתונים מתמשך?

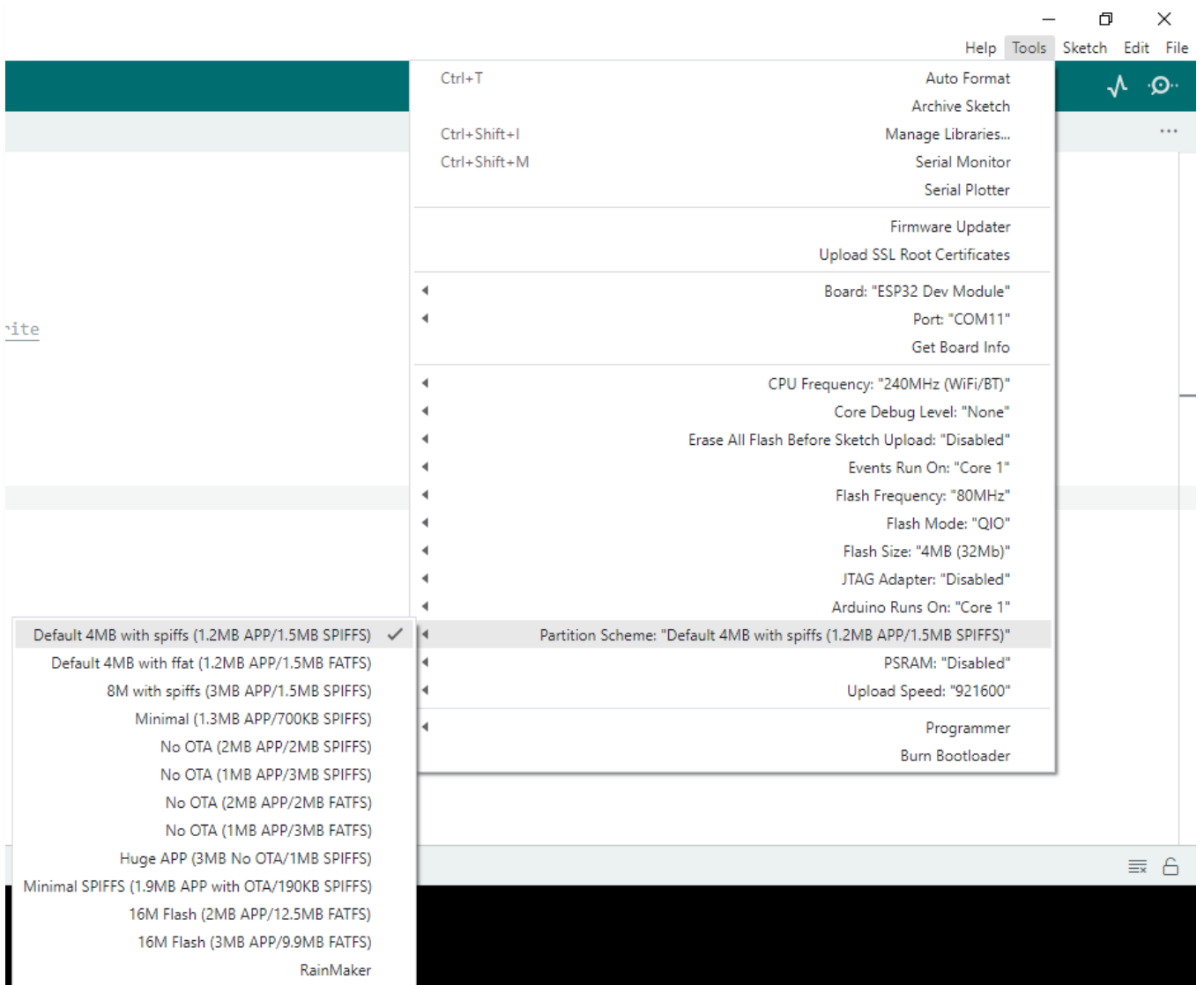
כברירת מחדל, כ- 1.5 MB מהזיכרון של ה FLASH מוקצים ל - SPIFFS. אפשר לראות את זה בתפריט של הארדואינו IDE בסכמת החלוקה למחיצות - PARTITION:

Tools -> Partition Scheme

או בעברית :

כלים - < ערכת מחיצה.

האיור הבא מתאר את אפשרויות החלוקה הפנימית בזיכרון ה FLASH .



איור 1 : אפשרויות חלוקת המחיצה בזיכרון ה FLASH.

אפשר לראות שקיימות מספר אפשרויות מחיצה זמינות אחרות. עם זאת, לא ניכנס לעומק העניין. שינוי ערכת המחיצות בכל מקרה לא יידרש עבור רוב היישומים שלנו ולכן נשאיר את ברירת המחזל שבאיור. כעת נראה את תהליך היצירה, השינוי, הקריאה והמחיקה של קובץ מ - SPIFFS באמצעות דוגמה.

2. הפונקציות/מתודות של SPIFFS

הפונקציות של SPIFFS דומות לפונקציות טיפול בקבצים בכל שפת תכנות.

2.1 SPIFFS.BEGIN()

הפונקציה SPIFFS.BEGIN() היא השער למערכת הקבצים Flash SPI (SPIFFS) ב-ESP32. פונקציה זו מאתחלת וטוענת את מערכת הקבצים SPIFFS, מה שהופך אותה לנגישה לפעולות קריאה וכתיבה. בדוגמת הקוד הבאה הוא הפונקציה נמצאת בפונקציית ה setup() של הארדוואינו ולכן היא רצה פעם אחת בלבד.

```
#include <FS.h>
#include <SPIFFS.h>
void setup()
{
  Serial.begin(115200);
  if (SPIFFS.begin()) {
    Serial.println("SPIFFS mounted successfully");
  } else {
    Serial.println("SPIFFS mount failed");
  }
}
...

```

הערה: SPIFFS.begin(true) יאתחל את מערכת הקבצים (אם היא עדיין לא מפורמטת), בעוד SPIFFS.begin() מניח שמערכת הקבצים כבר מאותחלת וטוען אותה כפי שהיא. הפירמוט ימחק את כל הנתונים/קבצים/תיקיות המאוחסנים כעת ב-ESP32. עם זאת, אם זה כבר היה מעוצב בעבר, ולאחר מכן הפעלת SPIFFS.begin(true) לא ימחק קבצים ישנים. כדי למחוק את כל הקבצים, נשתמש ב- SPIFFS.format(); משתמשים מדווחים כי SPIFFs יכולים לפעמים להיות שגיאות במהלך brown-outs – שהוא ירידה במתח ספק הכוח, וכי הגדרת true יכול לנקות ולא לתת את כל שגיאות SPIFFs של המיקום במהלך אתחול.

2.2 SPIFFS.open()

כעת, לאחר התקנת SPIFFS, השלב הבא לשימוש ב- SPIFFS הוא לפתוח קובץ. נשתמש בפונקציה SPIFFS.open() הפותחת קובץ לפני קריאה או כתיבה לקובץ כלשהו. דוגמת הקוד הבאה יוצרת קובץ, פותחת אותו ולאחר מכן מוודאת שהוא נוצר.

```
...
//declare a file (once) and then open it
File file = SPIFFS.open("/example.txt", "w");
if (!file) {
  Serial.println("Failed to open file for writing");
} else {
  Serial.println("File opened for writing");
}
...

```

SPIFFS.openNextFile() 2.3

הפונקציה מיועדת כאשר יש קבצים מרובים בספרייה. פונקציה זו שימושית במיוחד כאשר נרצה לעבוד על רשימה של קבצים ביעילות. זה עוזר לבצע איטרציה על הקבצים בתיקה ספציפית ומאפשר לנו לגשת ולתפעל כל קובץ אחד אחד. דוגמה מהירה לקוד:

```
...
File file = root.openNextFile();
while (file) {
Serial.println("File: " + String(file.name()));
file = root.openNextFile();
}
...
```



כך זה עובד: הפונקציה SPIFFS.openNextFile() נקראת כדי לפתוח את הקובץ הבא בספרייה. היא מחזירה אובייקט קובץ, המייצג את הקובץ שממנו באפשרותנו לקרוא או לכתוב אליו. בכל קריאה עוקבת אל SPIFFS.openNextFile() הוא יפתח את הקובץ הבא בספרייה, לאחר הסדר שבו הקבצים אוחסנו. השתמש בלולאה של while() בשילוב עם SPIFFS.openNextFile() כדי לעבוד על כל הקבצים בספרייה. זה מתחיל עם הקובץ הראשון ולאחר מכן עובר לקובץ הבא עד שאין קבצים נוספים לפתוח, ואז הלולאה תסתיים. שמות הקבצים עצמם אינם ממלאים תפקיד בקביעת סדר הבחירה. זה לא לפי האלפביתי! אם עלינו לגשת לקבצים בהתבסס על זמן יצירת הקובץ או קריטריונים אחרים, עלינו למיין את הקבצים לאחר פתיחתם. נראה דוגמה לכוח של הפקודה בהמשך.

SPIFFS.exsist() 2.4

הפונקציה בודקת אם קובץ קיים או לא, ומחזירה שגיאה אם לא. הקוד לדוגמה שלהלן יוצר קובץ ולאחר מכן בודק אם הקובץ וקובץ שני נוסף קיימים.

```
#include <FS.h>
#include <SPIFFS.h>
// Commenting out this line should get checkIfDirectory() to print "is not a
directory"
#define makeDir
void setup() {
Serial.begin(115200);
if (SPIFFS.begin(true)) {
```

```
Serial.println("SPIFFS mounted successfully");
#ifdef makeDir
File file = SPIFFS.open("/myfolder", "w");
#endif
// Check if a specific path exists
checkIfDirectory("/myfolder");
checkIfDirectory("/myfolder2");
} else {
Serial.println("SPIFFS mount failed");
}
}
void loop() {
// Nothing to do here
}
void checkIfDirectory(const char *path) {
if (SPIFFS.exists(path)) {
Serial.println(String(path) + " is a directory");
} else {
Serial.println(String(path) + " is not a directory");
}
}
```

הפלט שנקבל לאחר הפעלת הקוד לעיל צריך להיות:

```
SPIFFS mounted successfully
/myfolder is a directory
/myfolder2 is not a directory
```

SPIFFS.write() ו SPIFFS.print() 2.5

שתי הפונקציות משמשות לכתיבת נתונים לקובץ, אך הן עושות זאת בדרכים שונות במקצת.

SPIFFS.print() 2.5.1

פונקציה זו משמשת בצורה הטובה ביותר לכתיבת נתונים הניתנים לקריאה אנושית לקובץ, כגון מחרוזות או טקסט מעוצב. היא מטפלת בסוגי נתונים ופורמטים שונים, מה שהופך אותה מתאימה לכתיבת מחרוזות, מספרים וסוגים אחרים של נתונים ללא צורך בהמרה מפורשת. לדוגמה :

```
file.print("Are you using AI? Or is AI using you?");
```

SPIFFS.write() 2.5.2

פונקציה זו משמשת לכתיבת נתונים בינאריים גולמיים לקובץ. הוא רב-תכליתי יותר כאשר מדובר בנתונים שאינם טקסטואליים כמו תמונות, אודיו, יומני חיישנים או נתונים בינאריים טוריים. הוא מצפה למצביע על חוצץ – buffer - הנתונים ועל גודל הנתונים שייכתבו. לדוגמה:

```
uint8_t data[] = { 0x01, 0x02, 0x03, 0x04 };
file.write(data, sizeof(data));
```

2.5.3 הבדלים עיקריים בין write() ו print()

SPIFFS.print() הוא ברמה גבוהה יותר ונוח יותר לכתיבת נתונים מבוססי טקסט, טיפול בסוגי נתונים שונים ללא המרה מפורשת. אם יש לנו מחרזות, טקסט מעוצב או נתונים הניתנים לפירוש בקלות, פונקציה זו היא לעתים קרובות פשוטה יותר. SPIFFS.write() היא רב-תכליתית יותר לטיפול בנתונים גולמיים, ומאפשרת שליטה ישירה על התוכן שייכתב לקובץ. אם עובדים עם נתונים בינאריים או אם עלינו למזער את גודל הקובץ אז פונקציה זו עדיפה. להלן תוכנית דוגמה כיצד להשתמש בכל אחת משתי הפונקציות:

```
#include <FS.h>
#include <SPIFFS.h>
void setup() {
  Serial.begin(115200);
  if (SPIFFS.begin()) {
    Serial.println("SPIFFS mounted successfully");
    // Create and write binary data to one file
    writeBinaryDataToFile("/binaryFile.bin");
    // Write a sentence to another file using SPIFFS.print()
    writeFileWithPrint("/textFile.txt", "Data for text file.");
    // Read and print the contents of both files
    readAndPrintFiles();
  } else {
    Serial.println("SPIFFS mount failed");
  }
}
void loop() {
  // Nothing to do here
}
void writeBinaryDataToFile(const char *path) {
  Serial.print("Creating file with binary data: ");
```

```
Serial.println(String(path));
File file = SPIFFS.open(path, "w");
if (file) {
uint8_t binaryData[] = {0x01, 0x02, 0x03, 0x04};
file.write(binaryData, sizeof(binaryData));
file.close();
Serial.println("Binary file created and written successfully");
} else {
Serial.println("Failed to create binary file");
}
}

void writeFileWithPrint(const char *path, const char *content) {
Serial.print("Creating file with SPIFFS.print(): ");
Serial.println(String(path));
File file = SPIFFS.open(path, "w");
if (file) {
file.print(content);
file.close();
Serial.println("Text file created and written successfully");
} else {
Serial.println("Failed to create text file");
}
}

void readAndPrintFiles() {
Serial.println("Reading and printing files:");
// Read and print the contents of binaryFile.bin
readFile("/binaryFile.bin");
// Read and print the contents of textFile.txt
readFile("/textFile.txt");
}

void readFile(const char *path) {
File file = SPIFFS.open(path, "r");
if (file) {
Serial.print("File: " + String(path) + ", Content: ");
// Check if the file is binary or text based on the file extension
if (String(path).endsWith(".bin")) {
```



```
// Read and print binary data
while (file.available()) {
Serial.print(file.read(), HEX);
Serial.print(" ");
}
} else {
// Read and print text data
Serial.print(file.readString());
}
Serial.println();
file.close();
} else {
Serial.println("Failed to open file for reading: " + String(path));
}
}
```

הפלט הבא אמור להופיע לאחר הפעלה מוצלחת של תוכנית הדוגמה לעיל.

```
SPIFFS mounted successfully
Creating file with binary data: /binaryFile.bin
Binary file created and written successfully
Creating file with SPIFFS.print(): /textFile.txt
Text file created and written successfully
Reading and printing files:
File: /binaryFile.bin, Content: 1 2 3 4
File: /textFile.txt, Content: Data for text file.
```



SPIFFS.rename() 2.6

הפונקציה מאפשרת לנו לשנות את שם הקובץ או הספרייה במערכת הקבצים .SPIFFS לדוגמה תוכנית המשנה את שם הקובץ :

```
...
if (SPIFFS.rename("/oldfile.txt", "/newfile.txt")) {
Serial.println("File renamed successfully");
} else {
Serial.println("File rename failed");
}
```

```
}  
...  
}
```

SPIFFS.close() 2.7

בסיום העבודה עם קובץ/קבצים יש לסגור תמיד את הקבצים. סגירה נכונה של קובץ פתוח חיונית כדי להבטיח שהשינויים יישמרו והמשאבים ישוחררו. הפונקציה מאפשרת לסגור בבטחה קובץ פתוח. שלב זה חיוני למניעת אובדן נתונים ולהבטחת מצב עקבי של מערכת הקבצים. אם שוכחים לסגור קובץ עלולה לגרום למגוון רחב של באגים ובעיות אחרות, במיוחד בעת עבודה עם קבצים מרובים!
התוכנית הבאה מדגימה כיצד לסגור קובץ :

```
...  
File file = SPIFFS.open("/example.txt", "r");  
if (file) {  
file.close();  
Serial.println("File closed");  
}  
...  
}
```



SPIFFS.size() 2.8

כדי לעבוד ביעילות עם קבצים, לעתים קרובות עלינו לדעת את גדלי הקבצים. הפונקציה מספקת את גודל הקובץ ומאפשרת לנהל ולעבד קבצים בהתבסס על גודל הנתונים שלהם. מידע זה חשוב במיוחד בעת טיפול במערכות נתונים גדולות ויש להקפיד שלעולם לא נחרוג מהזיכרון הכולל.
קטע התוכנית הבא מראה שימוש בפונקציה:

```
...  
File file = SPIFFS.open("/example.txt", "r");  
if (file) {  
Serial.println("File size: " + String(file.size()) + " bytes");  
file.close();  
}  
...  
}
```

SPIFFS.read() 2.9

הפונקציה מאפשרת לאחזור את התוכן של קובץ. הדוגמאות הבאות מדגימות שתי דרכים שונות לקריאה ולהדפסה של תוכן קובץ, בהתאם לפורמט שבו נשמרו הנתונים.

```
...
File file = SPIFFS.open("/example.txt", "r");
//read out binary data
while (file.available()) {
Serial.print(file.read(), HEX);
Serial.print(" ");
}
file.close();
file = SPIFFS.open("/example.txt", "r");
//read text data
Serial.print(file.readString());
file.close();
...
```

עכשיו, כשאנחנו יודעים איך ליצור קבצים, לקרוא קבצים, לכתוב לקבצים ולקבל רשימה של קבצים, בואו נעשה דוגמה אחת גדולה לחבר את הכול יחד.

2.10 רישום קבצים המאוחסנים ב SPIFFS

לפעמים אנחנו עובדים עם קבצים רבים ודרושה לנו רשימה של הקבצים שנרצה לעבוד איתה. כדי להציג קבצים המאוחסנים ב-SPIFFS, נשתמש ב- SPIFFS.open(dir). להלן דוגמה לאופן שבו ניתן להציג קבצים בספרייה. התוכנית הבאה יוצרת קבצים מרובים, כותבת להם ולאחר מכן מפרטת - יוצרת רשימה שלהם עם גודל הקובץ שלהם.

```
#include <FS.h>
#include <SPIFFS.h>
void setup() {
Serial.begin(115200);
if (SPIFFS.begin()) {
Serial.println("SPIFFS mounted successfully");
// Create and write data to three different files
createAndWriteFiles();
```

```
// List files in the root directory
listFiles("");
} else {
Serial.println("SPIFFS mount failed");
}
}
void loop() {
// Nothing to do here
}
void createAndWriteFiles() {
writeFile("/file1.txt", "File 1 in the house");
writeFile("/file2.txt", "File 2 holds cabbages?!");
writeFile("/file3.txt", "Content for file 3 is the biggest of them all!");
}
void writeFile(const char *path, const char *content) {
Serial.print("Creating file: ");
Serial.println(String(path));
File file = SPIFFS.open(path, "w");
if (file) {
file.print(content);
file.close();
Serial.println("File created and written successfully");
} else {
Serial.println("Failed to create file");
}
}
void listFiles(const char *dir) {
Serial.print("Listing files in directory: ");
Serial.println(String(dir));
File root = SPIFFS.open(dir);
if (!root) {
Serial.println("Failed to open directory");
return;
}
File file = root.openNextFile();
while (file) {
```

```
Serial.println("File: " + String(file.name()) + ", Size: " + file.size());
file = root.openNextFile();
}
}
```

בדוגמה לעיל, הפונקציה `listFiles()` שלנו מקבלת נתיב ספרייה כארגומנט ומדפיסה שם קובץ וגודל עבור כל קובץ בספרייה זו. אנחנו נקבל את התמונה הבאה:

```
rst:0x1 (POWERON_RESET),boot:0x13 (SPI_FAST_FLASH_BOOT)
configsip: 0, SPIWP:0xee
clk_drv:0x00,q_drv:0x00,d_drv:0x00,cs0_drv:0x00,hd_drv:0x00,wp_drv:0x00
mode:DIO, clock div:1
load:0x3fff0030,len:1344
load:0x40078000,len:13964
load:0x40080400,len:3600
entry 0x400805f0
E (104) psram: PSRAM ID read error: 0xffffffff
SPIFFS mounted successfully
Creating file: /file1.txt
File created and written successfully
Creating file: /file2.txt
File created and written successfully
Creating file: /file3.txt
File created and written successfully
Listing files in directory: /
File: myfile.txt, Size: 7
File: file1.txt, Size: 15
File: file2.txt, Size: 16
File: file3.txt, Size: 38
```

2.10.1 התמודדות עם קבצים ישנים

ייתכן שנקבל בנוסף לקבצים שיצרנו קבצים נוספים שנוצרו בעבר באמצעות תוכניות שונות והם נשארו בזיכרון מערכת הקבצים כי לא מחקנו אותם.

לפעמים זוהי תכונה, כגון בעת עדכון קושחה – `firmware` - לגרסה העדכנית ביותר. עדיין נרצה שכל קבצי ההגדרות שנשמרו בעבר יהיו שם. אבל, לעתים קרובות זו יכולה להיות בעיה, מכיוון שעברנו לפרויקט עתידי אך הזיכרון מתבזבז באחסון קבצים ישנים שאינם בשימוש.

הערה: לפעמים בעת החלפת קושחה ישנה בקושחה גדולה יותר, קבצים ישנים עשויים להתחלף. עדכון קושחה תמיד טומן בחובו סיכון של מחיקת קבצים מאוחסנים ישנים ב-`ESP32`, במיוחד בעת שינוי גדלי הקצאת `SPIFFS`.

אם קובץ כלשהו אינו רצוי, עומדות בפנינו שלוש אפשרויות שהסברנו קודם לכן:

1. ניתן לפרמט מחדש את הזיכרון באמצעות תוכנה כדי למחוק לחלוטין את כל הנתונים הישנים כפי שהוסבר.

2. לחלופין, נוכל למחוק את הזיכרון באמצעות IDE Arduino .
3. או לבסוף, אפשר למחוק בנפרד קבצים ספציפיים באמצעות קוד.

10.2 מחיקת קבצים השמורים ב SPIFFS

מחיקת קבצים ב- SPIFFS היא פשוטה מאוד באמצעות הפונקציה `SPIFFS.remove()` . הדוגמה הבאה מוחקת את הקובץ, בודקת שגיאיות כדי לאשר שהמשימה הושלמה כראוי ומסבירה באופן אחר כל סיבה לכשלון במשימה.

```
#include <FS.h>
#include <SPIFFS.h>

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

  if (SPIFFS.begin()) {
    Serial.println("SPIFFS mounted successfully");

    // Check if the file exists before attempting to delete
    if (SPIFFS.exists("/myfile.txt")) {
      // Delete the file
      if (SPIFFS.remove("/myfile.txt")) {
        Serial.println("File deleted successfully");
      } else {
        Serial.println("Failed to delete file");
      }
    } else {
      Serial.println("File does not exist");
    }
  } else {
    Serial.println("SPIFFS mount failed");
  }
}

void loop() {
  // Nothing to do here
}
```

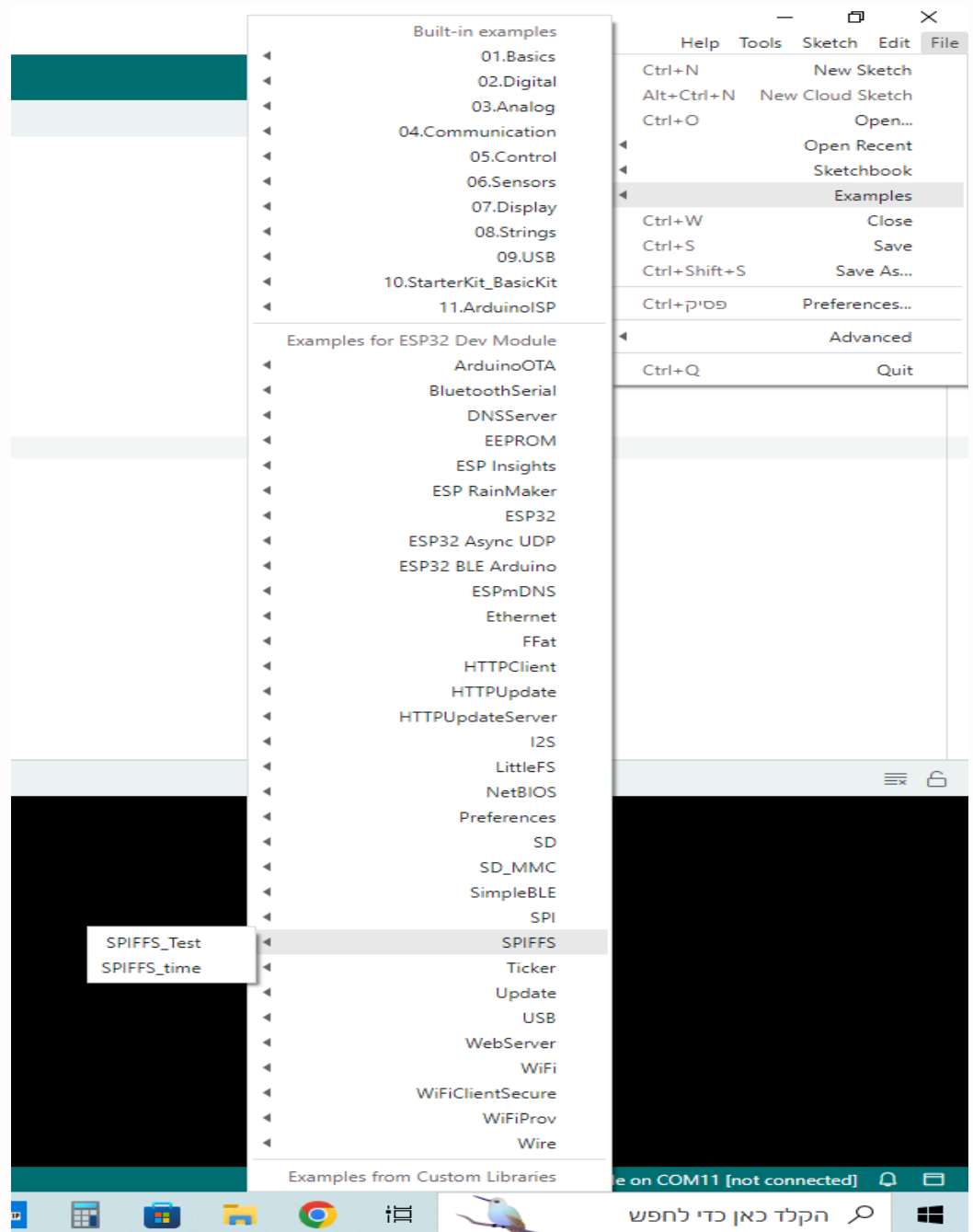
3. דוגמה נוספת מהספרייה של הארדואינו IDE

אנו נשתמש שוב בקוד לדוגמה שסופק שקיים בדוגמאות. נעבור אל :

File -> Examples -> SPIFFS -> SPIFFS_Test

או בעברית : **קובץ -> דוגמאות -> SPIFFS -> SPIFFS_Test**

קוד זה אידיאלי להבנת כל פעולות הקבצים האפשריות עם SPIFFS.



איור 2 : כיצד להגיע לקובץ SPIFFS_Test

ניתן למצוא אותו גם ב- GitHub בקישור :

[arduino-esp32/libraries/SPIFFS/examples/SPIFFS_Test/SPIFFS_Test.ino at master · espressif/arduino-esp32 · GitHub](https://github.com/arduino-esp32/libraries/SPIFFS/examples/SPIFFS_Test/SPIFFS_Test.ino)

הקובץ נראה כך :

```
#include "FS.h"
#include "SPIFFS.h"

/* You only need to format SPIFFS the first time you run a
   test or else use the SPIFFS plugin to create a partition
   https://github.com/me-no-dev/arduino-esp32fs-plugin */
#define FORMAT_SPIFFS_IF_FAILED true

void listDir(fs::FS &fs, const char * dirname, uint8_t levels) {
  Serial.printf("Listing directory: %s\r\n", dirname);

  File root = fs.open(dirname);
  if (!root) {
    Serial.println("- failed to open directory");
    return;
  }
  if (!root.isDirectory()) {
    Serial.println(" - not a directory");
    return;
  }

  File file = root.openNextFile();
  while (file) {
    if (file.isDirectory()) {
      Serial.print("  DIR : ");
      Serial.println(file.name());
      if (levels) {
        listDir(fs, file.path(), levels - 1);
      }
    } else {
      Serial.print("  FILE: ");
      Serial.print(file.name());
      Serial.print("\tSIZE: ");
      Serial.println(file.size());
    }
    file = root.openNextFile();
  }
}

void readFile(fs::FS &fs, const char * path) {
  Serial.printf("Reading file: %s\r\n", path);

  File file = fs.open(path);
  if (!file || file.isDirectory()) {
```



```
    Serial.println("- failed to open file for reading");
    return;
}

Serial.println("- read from file:");
while (file.available()) {
    Serial.write(file.read());
}
file.close();
}

void writeFile(fs::FS &fs, const char * path, const char * message) {
    Serial.printf("Writing file: %s\r\n", path);

    File file = fs.open(path, FILE_WRITE);
    if (!file) {
        Serial.println("- failed to open file for writing");
        return;
    }
    if (file.print(message)) {
        Serial.println("- file written");
    } else {
        Serial.println("- write failed");
    }
    file.close();
}

void appendFile(fs::FS &fs, const char * path, const char * message) {
    Serial.printf("Appending to file: %s\r\n", path);

    File file = fs.open(path, FILE_APPEND);
    if (!file) {
        Serial.println("- failed to open file for appending");
        return;
    }
    if (file.print(message)) {
        Serial.println("- message appended");
    } else {
        Serial.println("- append failed");
    }
    file.close();
}

void renameFile(fs::FS &fs, const char * path1, const char * path2) {
    Serial.printf("Renaming file %s to %s\r\n", path1, path2);
    if (fs.rename(path1, path2)) {
        Serial.println("- file renamed");
    } else {
```

```
        Serial.println("- rename failed");
    }
}

void deleteFile(fs::FS &fs, const char * path) {
    Serial.printf("Deleting file: %s\r\n", path);
    if (fs.remove(path)) {
        Serial.println("- file deleted");
    } else {
        Serial.println("- delete failed");
    }
}

void testFileIO(fs::FS &fs, const char * path) {
    Serial.printf("Testing file I/O with %s\r\n", path);

    static uint8_t buf[512];
    size_t len = 0;
    File file = fs.open(path, FILE_WRITE);
    if (!file) {
        Serial.println("- failed to open file for writing");
        return;
    }

    size_t i;
    Serial.print("- writing" );
    uint32_t start = millis();
    for (i = 0; i < 2048; i++) {
        if ((i & 0x001F) == 0x001F) {
            Serial.print(".");
        }
        file.write(buf, 512);
    }
    Serial.println("");
    uint32_t end = millis() - start;
    Serial.printf(" - %u bytes written in %u ms\r\n", 2048 * 512, end);
    file.close();

    file = fs.open(path);
    start = millis();
    end = start;
    i = 0;
    if (file && !file.isDirectory()) {
        len = file.size();
        size_t flen = len;
        start = millis();
        Serial.print("- reading" );
        while (flen) {
```

```
    size_t toRead = len;
    if (toRead > 512) {
        toRead = 512;
    }
    file.read(buf, toRead);
    if ((i++ & 0x001F) == 0x001F) {
        Serial.print(".");
    }
    len -= toRead;
}
Serial.println("");
end = millis() - start;
Serial.printf("- %u bytes read in %u ms\r\n", flen, end);
file.close();
} else {
    Serial.println("- failed to open file for reading");
}
}

void setup() {
    Serial.begin(115200);
    if (!SPIFFS.begin(FORMAT_SPIFFS_IF_FAILED)) {
        Serial.println("SPIFFS Mount Failed");
        return;
    }

    listDir(SPIFFS, "/", 0);
    writeFile(SPIFFS, "/hello.txt", "Hello ");
    appendFile(SPIFFS, "/hello.txt", "World!\r\n");
    readFile(SPIFFS, "/hello.txt");
    renameFile(SPIFFS, "/hello.txt", "/foo.txt");
    readFile(SPIFFS, "/foo.txt");
    deleteFile(SPIFFS, "/foo.txt");
    testFileIO(SPIFFS, "/test.txt");
    deleteFile(SPIFFS, "/test.txt");
    Serial.println( "Test complete" );
}

void loop() {
}
}
```

4. הסבר התוכנית

נסביר את התוכנית, קטע אחרי קטע. אנו מתחילים עם הכללת שתי ספריות:

```
#include "FS.h"
```

```
#include "SPIFFS.h"
```

FS הוא קיצור של - File System – מערכת הקבצים.

לאחר מכן, נראה הגדרת מאקרו : `FORMAT_SPIFFS_IF_FAILED`. בהמשך יש הערה :

```
/* You only need to format SPIFFS the first time you run a  
test or else use the SPIFFS plugin to create a partition  
https://github.com/me-no-dev/arduino-esp32fs-plugin */  
#define FORMAT_SPIFFS_IF_FAILED true
```

שאומרת שיש לפרמט את SPIFFS רק בפעם הראשונה שמפעילים בדיקה. משמעות הדבר היא שאפשר להגדיר את הערך של מאקרו זה כ- `false` לאחר ההפעלה הראשונה. לפרמט את ה- SPIFFS לוקח זמן, ואין צורך לעשות זאת בכל פעם שנפעיל תוכנית. לכן, נהוג להשתמש בקוד נפרד לפרמט SPIFFS. התוכנית הראשית איננה כוללת את פקודת הפירמוט. בדוגמה זו, שמנו במאקרו הזה `true`. לאחר מכן, ניתן לראות כי מספר פונקציות הוגדרו עבור פעולות שונות של מערכת הקבצים. הם –

- `listDir` – To list all directories
- `readFile` – To read a specific file
- `writeFile` – To write to a file (this overwrites the content already present in the file)
- `appendFile` – To append content to a file (use this when you want to add to the existing content, not overwrite it)
- `renameFile` – To change the name of a file
- `deleteFile` – To delete a file

- לפרט את כל הספריות
- לקרוא קובץ ספציפי
- לכתוב לקובץ (זה גורם לכתיבה על התוכן שנמצא בקובץ – כלומר מחליפים את התוכן של הקובץ)
- הוספה לקובץ – להוסיף לתוכן של הקובץ (משתמשים בזה כשרוצים להוסיף לקובץ ולא להחליף את התוכן שלו)
- שינוי שם הקובץ
- למחוק קובץ

```
void listDir(fs::FS &fs, const char * dirname, uint8_t levels){  
Serial.printf("Listing directory: %s\r\n", dirname);  
File root = fs.open(dirname);  
if(!root){  
Serial.println("– failed to open directory");  
return;  
  
}
```

```
if(!root.isDirectory()){ Serial.println(" – not a directory"); return;
```

```
}

File file = root.openNextFile();
while(file){ if(file.isDirectory()){
Serial.print(" DIR : "); Serial.println(file.name()); if(levels){
listDir(fs, file.name(), levels -1);

}
} else {
Serial.print(" FILE: "); Serial.print(file.name()); Serial.print("\tSIZE: "); Serial.println(file.size());

}
file = root.openNextFile();
}
}

void readFile(fs::FS &fs, const char * path){
Serial.printf("Reading file: %s\r\n", path);

File file = fs.open(path);
if(!file || file.isDirectory()){
Serial.println("- failed to open file for reading");
return;

}

Serial.println("- read from file:");
while(file.available()){ Serial.write(file.read());

}
}

void writeFile(fs::FS &fs, const char * path, const char * message){
Serial.printf("Writing file: %s\r\n", path);

File file = fs.open(path, FILE_WRITE);
```

```
if(!file){
    Serial.println("- failed to open file for writing");
    return;
}

if(file.print(message)){ Serial.println("- file written");
    }else {
    Serial.println("- frite failed");

    }
}

void appendFile(fs::FS &fs, const char * path, const char * message){
    Serial          : %s\r\n", path);
    File file = fs.open(path, FILE_APPEND);
    if(!file){
        Serial.println("- failed to open file for appending");
        return;
    }
    if(file.print(message)){ Serial.println("- message appended");
        } else {
        Serial.println("- append failed");

        }
    }

void renameFile(fs::FS &fs, const char * path1, const char * path2){
    Serial.printf("Renaming file %s to %s\r\n", path1, path2);
    if (fs.rename(path1, path2)) {
        Serial.println("- file renamed");
    } else {
        Serial.println("- rename failed");
    }
}
```

```

}
void deleteFile(fs::FS &fs, const char * path){
  Serial.printf("Deleting file: %s\r\n", path);
  if(fs.remove(path)){ Serial.println("- file deleted");
  } else {
  Serial.println("- delete failed");
}
}
}

```



יש לשים לב שכל הפונקציות לעיל אינן מבקשות שם קובץ. הם מבקשות את נתיב הקובץ המלא. כי זו מערכת קבצים. ייתכן שיש לנו ספריות, ספריות משנה וקבצים בתוך ספריות משנה אלה. לכן, ESP32 צריך לדעת את הנתיב המלא של הקובץ שבו ברצוננו לפעול. לאחר מכן מגיעה פונקציה שהיא לא בדיוק פונקציית פעולת קובץ - testFileIO. זוהי יותר פונקציית השוואת זמן. היא עושה את הדברים הבאים :

- כותבת כ - 1MB (2048 * 512 בתים) של נתונים בנתיב הקובץ שאתה מספק ומודדת את זמן הכתיבה
- קוראת את אותו קובץ ומודדת את זמן הקריאה

```

void testFileIO(fs::FS &fs, const char * path){
  Serial.printf("Testing file I/O with %s\r\n", path);

  static uint8_t buf[512];
  size_t len = 0;
  File file = fs.open(path, FILE_WRITE);
  if(!file){
    Serial.println("- failed to open file for writing");
    return;
  }

  size_t i;
  Serial.print("- writing" );
  uint32_t start = millis();
  for(i=0; i<2048; i++){
    if ((i & 0x001F) == 0x001F){
      Serial.print(".");
    }
    file.write(buf, 512);
  }
  Serial.println("");
}

```

```
uint32_t end = millis() - start;
Serial.printf(" - %u bytes written in %u ms\r\n", 2048 * 512, end);
file.close();

file = fs.open(path);
start = millis();
end = start;
i = 0;
if(file && !file.isDirectory()){
  len = file.size();
  size_t flen = len;
  start = millis();
  Serial.print("- reading" );
  while(len){
    size_t toRead = len;
    if(toRead > 512){
      toRead = 512;
    }
    file.read(buf, toRead);
    if ((i++ & 0x001F) == 0x001F){
      Serial.print(".");
    }
    len -= toRead;
  }
  Serial.println("");
  end = millis() - start;
  Serial.printf("- %u bytes read in %u ms\r\n", flen, end);
  file.close();
} else {
  Serial.println("- failed to open file for reading");
}
}
```

יש לשים לב שמערך buf אינו מאותחל עם ערך כלשהו. יכול מאוד להיות שאנחנו כותבים בייטים של זבל לקובץ. זה לא משנה כי מטרת הפונקציה היא למדוד את זמן הכתיבה ואת זמן הקריאה. לאחר הגדרת הפונקציות שלנו, אנו עוברים להתקנה, שם מוצגת ההפעלה של כל אחת מהפונקציות הללו.


```
void setup(){
  Serial.begin(115200);
  if(!SPIFFS.begin(FORMAT_SPIFFS_IF_FAILED)){
    Serial.println("SPIFFS Mount Failed");
    return;
  }
  listDir(SPIFFS, "/", 0);
  writeFile(SPIFFS, "/hello.txt", "Hello ");
  appendFile(SPIFFS, "/hello.txt", "World!\r\n");
  readFile(SPIFFS, "/hello.txt");
  renameFile(SPIFFS, "/hello.txt", "/foo.txt");
  readFile(SPIFFS, "/foo.txt");
  deleteFile(SPIFFS, "/foo.txt");
  testFileIO(SPIFFS, "/test.txt");
  deleteFile(SPIFFS, "/test.txt");
  Serial.println( "Test complete" );
}
```

ההתקנה עושה למעשה את הדברים הבאים :

- תחילה היא מאתחלת את SPIFFS באמצעות `SPIFFS.begin()`. המאקרו שהוגדר בהתחלה פועל ככן. כאשר הוא `true`, הוא מפרמט את SPIFFS (הפירמוט לוקח זמן רב); כאשר רושמים `false`, הוא מאתחל את SPIFFS ללא פירמוט.
 - לאחר מכן הוא מפרמט את כל הספריות ברמת השורש. נשים לב שציינו רמה (level) כ-0. לכן, אנחנו לא מפרטים את ספריות המשנה בתוך הספריות. באפשרותנו להגדיל את הקינון (כמה תת ספריות רוצים) על-ידי הגדלת הארגומנט `.levels`.
 - לאחר מכן הוא משנה את שם `hello.txt` ל- `foo.txt` במשפט `(renameFile....)`.
 - לאחר מכן קוראים מהקובץ `foo.txt` כדי לראות אם שינוי השם במשפט הקודם הצליח. צריך לראות "hello" מודפס מכיוון שזה מה שמאוחסן בקובץ.
 - לאחר מכן מוחקים את הקובץ `foo.txt` במשפט `(deleteFile.....)`.
 - לאחר מכן הוא מבצע את הפונקציה `testFileIO` בקובץ חדש ששמו `test.txt`.
 - בסיום הפונקציה מוחקים את הקובץ `test.txt`.
- תוכנית דוגמה זו מפרטת ובודקת את כל הפונקציות שבהן ייתכן שנרצה להשתמש עם SPIFFS. ניתן להתקדם ולשנות את הקוד הזה, ולשחק עם הפונקציות השונות.

היות ואנחנו לא רוצים לבצע כן פעילות נוספת אז הפונקציה `loop()` היא פונקציה ריקה ללא משפטים. `void loop(){ }`

