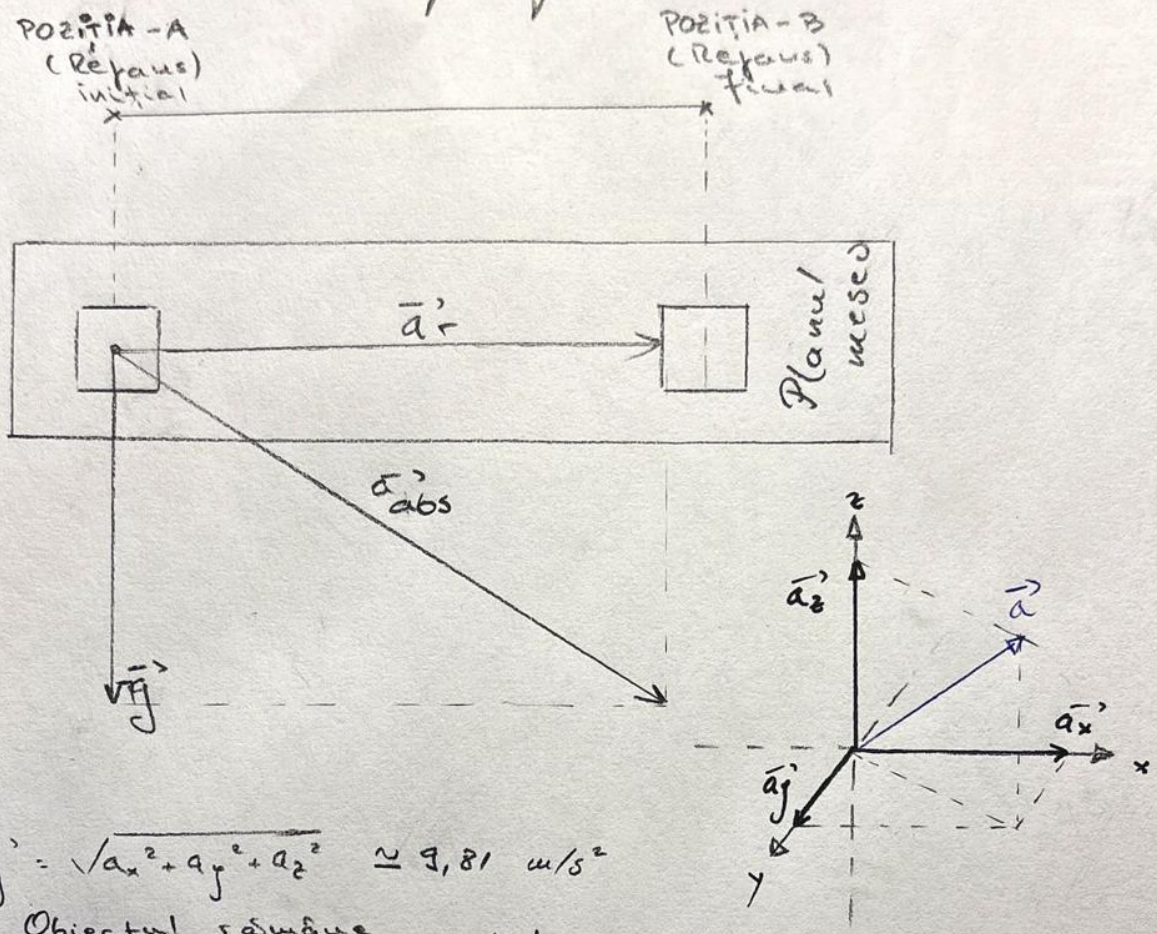
P2 **- „Giroscop digital și accelerometru cu MPU6050 pe STM32 – achiziție la 100ms, afișare UART, buffer circular EEPROM”**

VLADISLAV NORIS – VICTOR
&
Miron Marta – Florentina

Cuprins

Concept fizic



$\bar{g}' = \sqrt{a_x^2 + a_y^2 + a_z^2} \simeq 9{,}81\ m/s^2$

Obiectul rămâne aralel cu planul mesei, deci în mișcare ar trebui să vedem schimbări semnificative pt. ax și mici schimbări în a_y, a_z.

- Dorim să aflăm $\bar{a}$ (accelerația relativă față de Pământ - <u>accelerația efectivă</u> pe care o vedem)
- $a_r = \sqrt{a_{abs}^2 - g^2}$, unde $a_{abs} = \sqrt{a_x'^2 + a_y'^2 + a_z'^2}$
- După calculul t ar putem afla viteza:

$$v(t) = v_0 + \int_{t_0}^{t} a\,dt = v_0 + a \cdot \Delta t$$

- După calculul vitezei putem afla distanța:

$$d(t) = d_0 + \int_{t_0}^{t} v(t)\,dt = d_0 + v \cdot \Delta t$$

* Procesul conține scalări aferente ale parametrilor

Implementarea conceptului fizic prin cod sursă

In main.c am modificat functia HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim) astfel incat la fiecare 100 ms sa citim datele de la senzorul nostru MPU6050 – ax, ay, az (acceleratia pe fiecare axa).

```
void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim)
{
  if (++ticks == 1000) ticks = 0;

  if (ticks % 10 == 0) { // EVERY 100MS

    //INIT
    typedef enum { STATE_IDLE, STATE_MOVING, STATE_STOPPED } MotionState;
    static MotionState motion_state = STATE_IDLE;



    uint32_t now = HAL_GetTick();

    // READ PARAMETER INIT
    int16_t ax_raw, ay_raw, az_raw;
    int16_t gx_raw, gy_raw, gz_raw;
    MPU6050_Read_XYZ(&hi2c1, &ax_raw, &ay_raw, &az_raw, &gx_raw, &gy_raw, &gz_raw);

    // END READ PARAMETER INIT

    // === Conversie în unități fizice ===
    float ax_g = ax_raw / 16384.0f;
    float ay_g = ay_raw / 16384.0f;
    float az_g = az_raw / 16384.0f;
    float g_bun = 9.81f;
```

Acest lucru se face prin functia MPU6050_Read_XYZ(&hi2c1, &ax_raw, &ay_raw, &az_raw, &gx_raw, &gy_raw, &gz_raw). Aceasta este predefinita in driver-ul senzorului, creata pentru a putea comunica cu el si atasat in sursele proiectului.

Prin protocolul de communicate I2C va primi datele de la sensor si ulterior le scalam pentru a fi rezultate plauzibile cu Realitatea. Scalarea se

face prin impartirea la 16384, valoarea luata din DATASHEET.

```
float acc_abs = sqrtf(ax_g * ax_g + ay_g * ay_g + az_g * az_g);
float acc_abs_scaled = acc_abs * 12.87f;


float val_raw = acc_abs_scaled * acc_abs_scaled - g * g;
float val = 0.0f;
acc = 0.0f;

if (val_raw > 0.0f) {
    val = sqrtf(val_raw);
    if (val > 1.7f) {
        acc = val;
    }
}
```

In continuare calculam valoarea acceleratiei absolute prin functia prezentata in conceptul fizic. Valoarea este inmultita cu 12.87 fiindca acesta este prescaler-ul calculat de noi astfel:

- Cand obiectul este pe loc si orizontal pe masa citim ax, ay, az
- In repaus pe masa stim ca singura acceleratia a corpului nostrum este cea gravitationala
- Astfel calculam acceleratia absoluta si vedem cu cat trebuie inmultita pentru a avea 9.81 (acceleratia gravitationala)
- Numarul cu care am inmultit devine prescaler -ul nostru pentru calculul acceleratiei si il utilizam si cand calculam acceleratia absoluta a corpului aflat in miscare.

Pentru a minimiza erorile acceleratia relativa, calculate prin diferenta patratelor dintre acceleratia absoluta si cea gravitationala, va fi 0 daca nu este

mai mare decat un prag 1.7 m/s^2.

```
if (acc > 0.2f) {
    v_x += acc * dt;
```

Dupa calculam viteza prin formula fizica expus mai sus, unde dt este definit asftel:

```
float dt = (now - last_time) / 1000.0f;
last_time = now;
```

Pentru calculul distantei implementam un automat cu 3 stari: STATE_IDLE, STATE_MOVING, STATE_STOPPED

```
// Dacă începe să se miște, trecem în starea MOVING
if (motion_state != STATE_MOVING) {
    d_x = 0.0f;  // resetăm distanța
    motion_state = STATE_MOVING;
}
else {
v_x *= 0.5f; // reducere viteză
// Dacă viteza devine foarte mică, trecem în starea STOPPED
// și păstrăm distanța înghețată

if (fabs(v_x) < 0.1f) {
    v_x = 0.0f;

    if (motion_state == STATE_MOVING) {
        motion_state = STATE_STOPPED;  // tocmai s-a oprit
    } else if (motion_state == STATE_STOPPED) {
        // Rămâne în STOPPED, distanța înghețată
    } else {
        // Stă de la început, nimic de făcut
    }
}
```

🔸 1. Detectare început mișcare

```c
if (motion_state != STATE_MOVING) {
    d_x = 0.0f;                 // Resetăm distanța (se pornește de la 0)
    motion_state = STATE_MOVING; // Trecem în starea MOVING
}
```

🟡 Dacă detectăm că viteza a crescut (presupus mai sus în cod), schimbăm starea din STATIONAR/STOPPED în MOVING și **resetăm distanța**.

🔸 2. Încetinire (simulare de fricțiune)

```c
else {
    v_x *= 0.5f;  // Reducem viteza artificial (frânare)
}
```

🔵 Dacă **obiectul deja se mișcă**, dar fără forță nouă, îi scădem viteza progresiv. Este o **simulare de frecare** care îl va aduce la oprire.

🔸 3. Detectare oprire completă

```c
if (fabs(v_x) < 0.1f) {
    v_x = 0.0f;
```

🟢 Dacă viteza devine **suficient de mică**, o anulăm complet pentru a evita drift-ul numeric.

🔸 4. Tranziții de stare în funcție de viteză zero

```c
if (motion_state == STATE_MOVING) {
    motion_state = STATE_STOPPED;  // Tocmai s-a oprit
}
else if (motion_state == STATE_STOPPED) {
    // Rămâne în STOPPED, distanța înghețată
}
else {
    // Stă de la început, nimic de făcut
}
```

🔴 Aceste blocuri decid:

- Dacă eram în mișcare → trecem în `STATE_STOPPED` (dar păstrăm deplasarea atinsă)
- Dacă eram deja oprit → nu modificăm nimic
- Dacă eram în repaus total de la început → ignor

In aceeasi functie HAL_TIM_PeriodElapsedCallback (TIM_HandleTypeDef *htim) utilizam o noua clauza in functie de numarul de tick pentru a afisa datele calculate anterior – acceleratia, viteza si distanta la 100ms

```
if (ticks % 100 == 0) {  // la fiecare 1s

  // Initializare variabile pentru transmitere
  int16_t acc_int = (int16_t)acc;
  int16_t acc_frac = (int16_t)((fabs(acc - acc_int)) * 1000);

  int16_t v_x_int = (int16_t)v_x;
  int16_t v_x_frac = (int16_t)((fabs(v_x - v_x_int)) * 1000);

  int16_t d_x_int = (int16_t)d_x;
  int16_t d_x_frac = (int16_t)((fabs(d_x - d_x_int)) * 1000);


  char buf [256];

  sprintf(buf, "Acceleration: acc=%d.%03.d m/s^2 || Velocity: v_x=%d.%03.d m/s || Distance: d_x=%d.%03.d cm\r\n",
  acc_int, acc_frac, v_x_int, v_x_frac, d_x_int, d_x_frac);
  HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), HAL_MAX_DELAY);
```

Utilizarea memoriei EEPROM AT24C256

Citim datele de la sensor la fiecare 100 ms si le afisam la fiecare 1 s. In continuare dorim sa stocam la fiecare 1 s acceleratia intr-un EEPROM extern AT24C256. Acesta va avea disponibile 20 de slot-uri care vor fi actualizate la fiecare secunda cu o valoare noua. Cand devine full (dupa 20s) dorim sa incepem sa surpascriem datele (de la cea mai veche la cea mei noua). EEPROM-UL se va comporta ca un buffer circular cu 20 de locatii.

1. Scrierea in EEPROM

```
// EEPROM WRITE

uint16_t eeprom_address = current_eeprom_index * EEPROM_SLOT_SIZE;
float acc_to_store = acc;
uint8_t acc_bytes[4];

memcpy(acc_bytes, &acc_to_store, sizeof(float));
AT24C256_WriteBuffer(&hi2c1, eeprom_address, acc_bytes, sizeof(float));

current_eeprom_index = (current_eeprom_index + 1) % EEPROM_SLOT_COUNT;  // buffer circular
}
```

- Calculam adressa din eeprom prin inmultirea indexului curent cu marimea slot-ului (sizeof(float) = 4)
- Functia memcpy – converteste valoarea acceleratiei in 4 octeti
- Se scrie în EEPROM extern de tip AT24C256 (256 kilobits = 32KB).

Scrierea se face: La adresa calculată (eeprom_address), cu bufferul acc_bytes (4 octeți), folosind instanța I2C hi2c1 pentru comunicația I2C

- Marim indexul curent si prin % EEPROM_SLOT_COUNT se asigura revenirea la 0 dupa cele 20 de locatii (buffer circular).

```c
void AT24C256_WriteBuffer(I2C_HandleTypeDef *hi2c, uint16_t memAddr, uint8_t *data, uint16_t length) {
    for (uint16_t i = 0; i < length; i++) {
        AT24C256_WriteByte(hi2c, memAddr + i, data[i]);
    }
}
```

2. Citirea din EEPROM

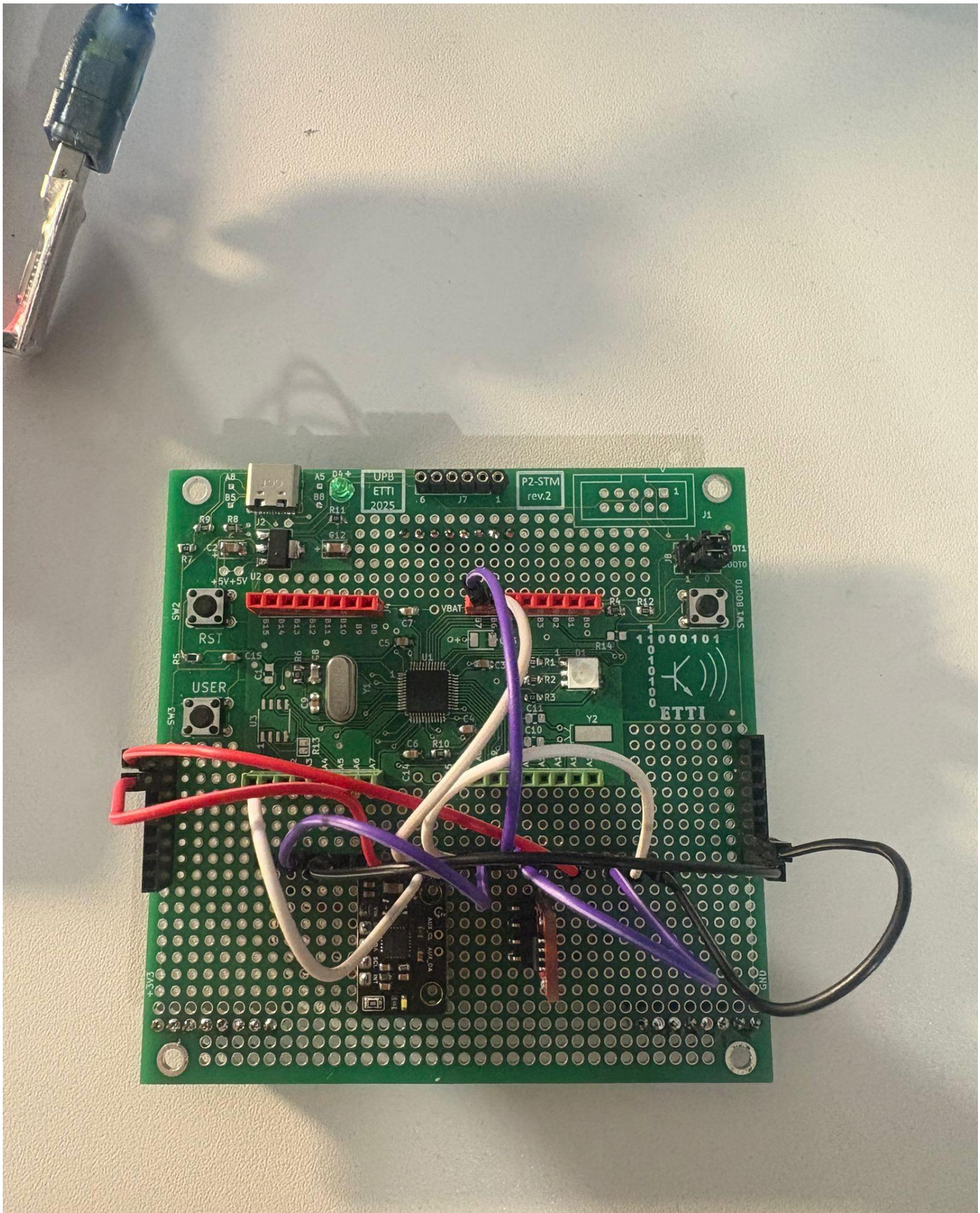
Citirea se va face la fiecare 10 s, chiar daca dupa 20 s se umple EEPROM-ul pentru a putea vizualiza shiftarea datelor, prin functia Print_All_Saved_ACC_Circular().

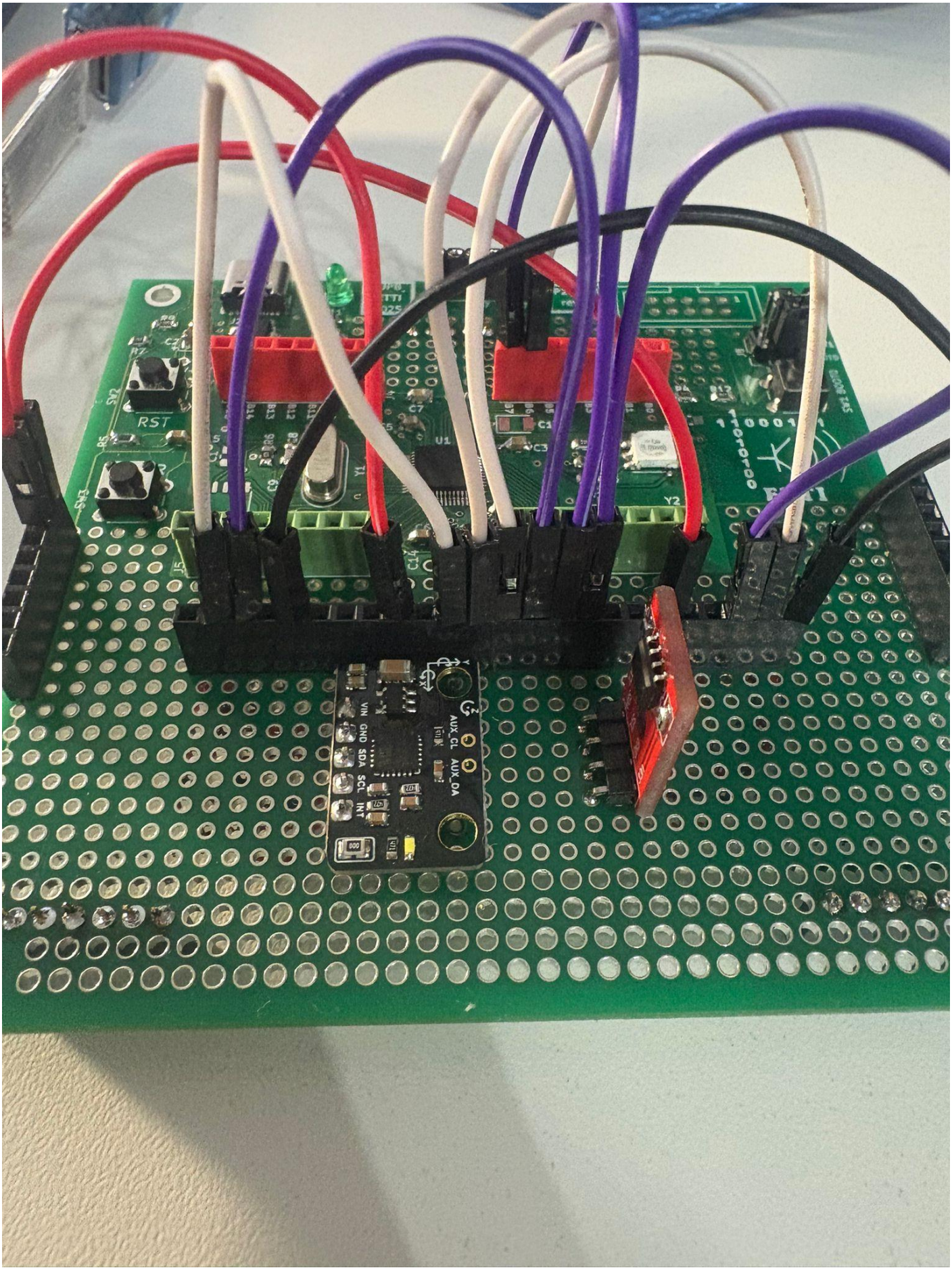
```c
if(ticks % 1000 == 0){
  Print_All_Saved_ACC_Circular();
}
```

```c
void Print_All_Saved_ACC_Circular(void) {
  uint8_t acc_bytes[4];
  float acc_value;
  char buf[64];

  uint16_t index = current_eeprom_index; // acesta e capul circular
  for (uint16_t i = 0; i < EEPROM_SLOT_COUNT; i++) {
    uint16_t slot = (index + i) % EEPROM_SLOT_COUNT;
    uint16_t eeprom_address = slot * EEPROM_SLOT_SIZE;

    AT24C256_ReadBuffer(&hi2c1, eeprom_address, acc_bytes, 4);
    memcpy(&acc_value, acc_bytes, sizeof(float));

    if (acc_value < 0.0f || acc_value > 100.0f || isnan(acc_value)) continue; // skip invalid

    int acc_int = (int)acc_value;
    int acc_frac = (int)((fabsf(acc_value - acc_int)) * 1000);  // 3 zecimale

    snprintf(buf, sizeof(buf), "EEPROM[%u] = Acceleration = %d.%03d m/s^2\r\n", eeprom_address, acc_int, acc_frac);
    HAL_UART_Transmit(&huart1, (uint8_t*)buf, strlen(buf), HAL_MAX_DELAY);
  }
```

- index este capul actual al bufferului circular – de aici începem citirea pentru a respecta ordinea cronologică (cel mai vechi → cel mai nou).
- se parcurg toate sloturile de memori, unde slot: asigură citirea circulară, cu wrap-around.
- Se reconstruiește valoarea float din octeți cu memcpy
- Dacă valoarea este: negative, prea mare (>100 m/s^2), NaN (Not a Number)
  → atunci este **ignorată**, fiind considerată coruptă/inutilă.
- La final se da DUMP la EEPROM in serial monitor

Componente hardware utilizate și schema de conectare




RST
VIN
GND
SDA
SCL
INT
AUX_CL
AUX_DA

Pentru citirea acceleratiei a fost utilizat senzorul MPU6050 un senzor inertial IMU (Inertial Measurement Unit) care integrează:

- un accelerometru pe 3 axe (X, Y, Z)
- un giroscop pe 3 axe
- un procesor de semnal digital (DMP) integrat pentru procesare internă

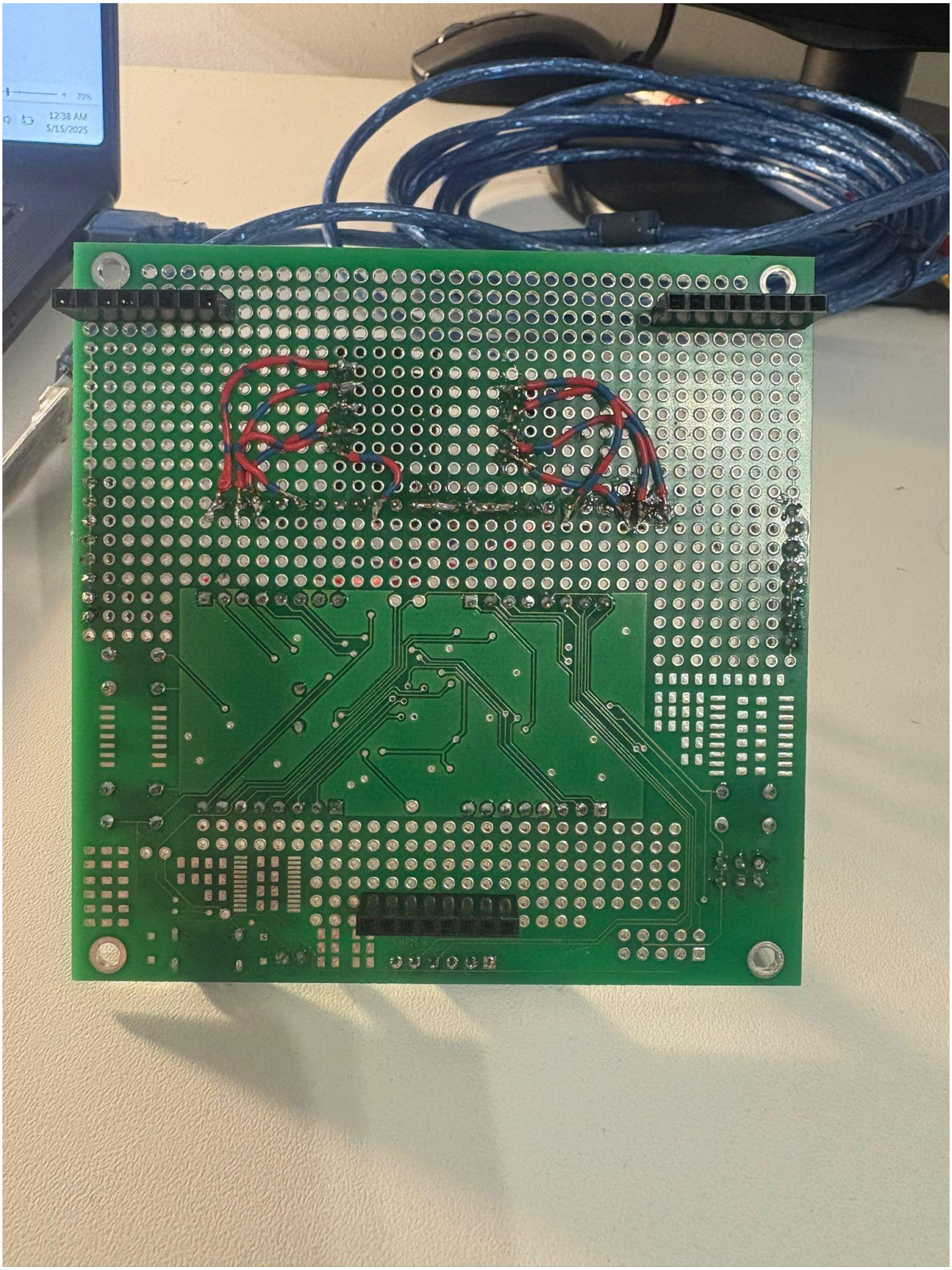
Este produs de InvenSense (acum parte din TDK) și este utilizat pe scară largă în proiecte embedded pentru măsurarea mișcării, orientării și vibrațiilor.

Senzorul este alimentat la tensiune de 3.3V si comunica cu STM-32 prin I2C (SDA + SCL).

Pentru stocarea datelor a fost utilizat o memorie EEPROM externa AT24C256 produsă de Microchip (fost Atmel). Este folosită pentru **stocarea nevolatilă a datelor** – adică reține datele chiar și după oprirea alimentării.

Memoria este alimentata la tensiune de 3.3V si comunica cu STM-32 prin I2C (SDA + SCL).

Conexiunea pe spatele placii



Citire prin HYPER-TERMINAL

```
Acceleration: acc=0.    m/s^2 || Velocity: v_x=7.608 m/s || Distance: d_x=17091.324 cm
Acceleration: acc=0.    m/s^2 || Velocity: v_x=0.    m/s || Distance: d_x=17163.511 cm
Acceleration: acc=7. 88 m/s^2 || Velocity: v_x=1.586 m/s || Distance: d_x=30.175 cm
Acceleration: acc=13.427 m/s^2 || Velocity: v_x=5.440 m/s || Distance: d_x=507.129 cm
Acceleration: acc=16.638 m/s^2 || Velocity: v_x=16.858 m/s || Distance: d_x=1670.160 cm
Acceleration: acc=14.762 m/s^2 || Velocity: v_x=9.928 m/s || Distance: d_x=2847.704 cm
Acceleration: acc=4.974 m/s^2 || Velocity: v_x=14.763 m/s || Distance: d_x=3935.828 cm
Acceleration: acc=17.168 m/s^2 || Velocity: v_x=15.203 m/s || Distance: d_x=5718.990 cm
Acceleration: acc=12.440 m/s^2 || Velocity: v_x=15.261 m/s || Distance: d_x=7088.610 cm
Acceleration: acc=4.855 m/s^2 || Velocity: v_x=17.783 m/s || Distance: d_x=8453.834 cm
```

Primul Memory dump:

```
Acceleration: acc=0.    m/s^2 || Velocity: v_x=0.    m/s || Distance: d_x=9.730 cm
Acceleration: acc=0.    m/s^2 || Velocity: v_x=0.    m/s || Distance: d_x=9.730 cm
Acceleration: acc=0.    m/s^2 || Velocity: v_x=0.    m/s || Distance: d_x=9.730 cm
Acceleration: acc=0.    m/s^2 || Velocity: v_x=0.    m/s || Distance: d_x=9.730 cm
Acceleration: acc=0.    m/s^2 || Velocity: v_x=0.    m/s || Distance: d_x=9.730 cm
Acceleration: acc=8.424 m/s^2 || Velocity: v_x=8.486 m/s || Distance: d_x=385.357 cm
Acceleration: acc=14.773 m/s^2 || Velocity: v_x=18. 85 m/s || Distance: d_x=1732.864 cm
Acceleration: acc=11.405 m/s^2 || Velocity: v_x=28.703 m/s || Distance: d_x=4103.592 cm
Acceleration: acc=6.  8 m/s^2 || Velocity: v_x=40.464 m/s || Distance: d_x=7670.448 cm
Acceleration: acc=18.956 m/s^2 || Velocity: v_x=53.890 m/s || Distance: d_x=12488.285 cm
EEPROM[40] = Acceleration = 0.000 m/s^2
EEPROM[44] = Acceleration = 0.000 m/s^2
EEPROM[48] = Acceleration = 0.000 m/s^2
EEPROM[52] = Acceleration = 0.000 m/s^2
EEPROM[56] = Acceleration = 0.000 m/s^2
EEPROM[60] = Acceleration = 0.000 m/s^2
EEPROM[64] = Acceleration = 0.000 m/s^2
EEPROM[68] = Acceleration = 0.000 m/s^2
EEPROM[72] = Acceleration = 0.000 m/s^2
EEPROM[76] = Acceleration = 0.000 m/s^2
EEPROM[0] = Acceleration = 0.000 m/s^2
EEPROM[4] = Acceleration = 0.000 m/s^2
EEPROM[8] = Acceleration = 0.000 m/s^2
EEPROM[12] = Acceleration = 0.000 m/s^2
EEPROM[16] = Acceleration = 0.000 m/s^2
EEPROM[20] = Acceleration = 8.424 m/s^2
EEPROM[24] = Acceleration = 14.773 m/s^2
EEPROM[28] = Acceleration = 11.405 m/s^2
EEPROM[32] = Acceleration = 6.008 m/s^2
EEPROM[36] = Acceleration = 18.956 m/s^2
```

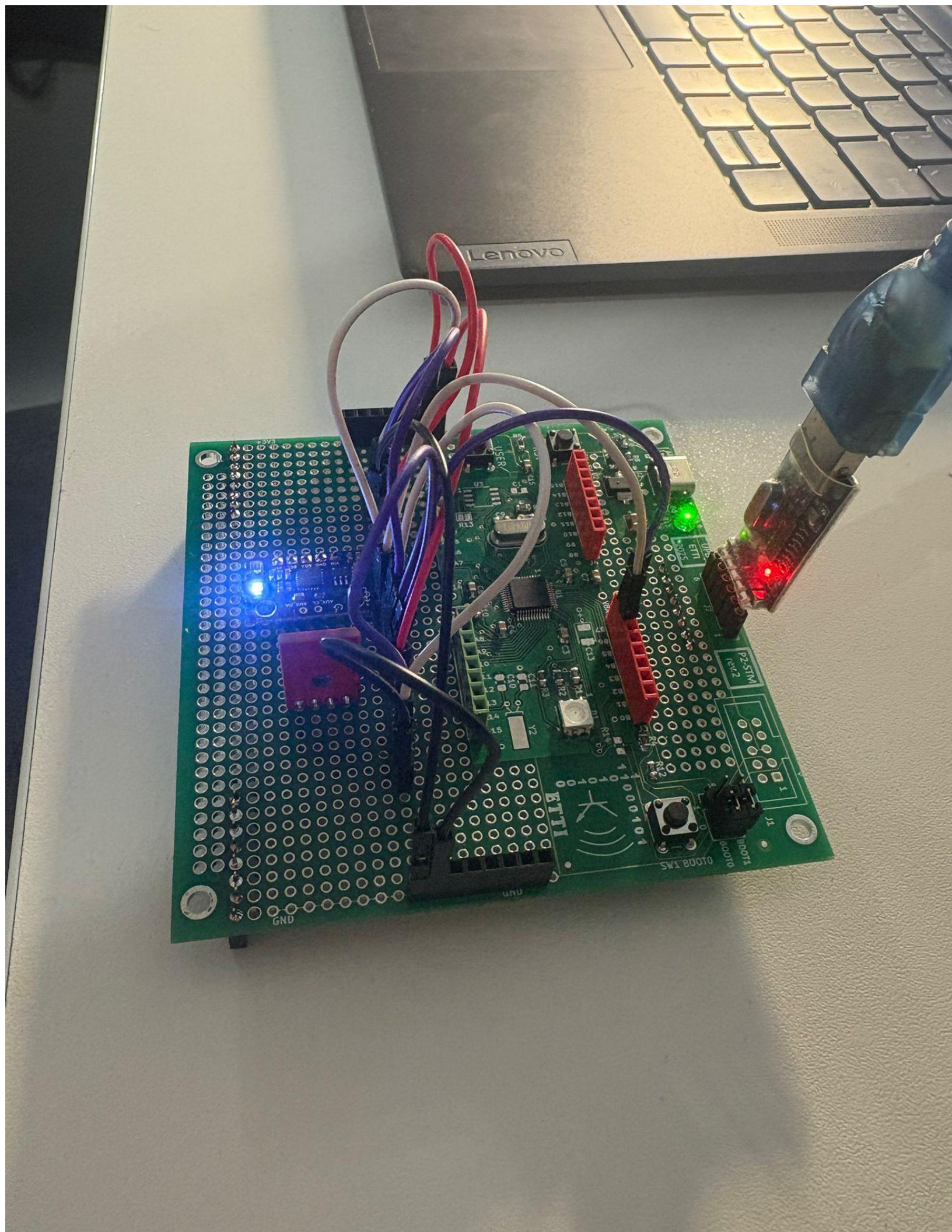
S-au memorat primele 10 valori in EEPROM

Primul Memory dump:

```
Acceleration: acc=0.    m/s^2 || Velocity: v_x=7.608 m/s || Distance: d_x=17091.324 cm
Acceleration: acc=0.    m/s^2 || Velocity: v_x=0.    m/s || Distance: d_x=17163.511 cm
Acceleration: acc=7. 88 m/s^2 || Velocity: v_x=1.586 m/s || Distance: d_x=30.175 cm
Acceleration: acc=13.427 m/s^2 || Velocity: v_x=5.440 m/s || Distance: d_x=507.129 cm
Acceleration: acc=16.638 m/s^2 || Velocity: v_x=16.858 m/s || Distance: d_x=1670.160 cm
Acceleration: acc=14.762 m/s^2 || Velocity: v_x=9.928 m/s || Distance: d_x=2847.704 cm
Acceleration: acc=4.974 m/s^2 || Velocity: v_x=14.763 m/s || Distance: d_x=3935.828 cm
Acceleration: acc=17.168 m/s^2 || Velocity: v_x=15.203 m/s || Distance: d_x=5718.990 cm
Acceleration: acc=12.440 m/s^2 || Velocity: v_x=15.261 m/s || Distance: d_x=7088.610 cm
Acceleration: acc=4.855 m/s^2 || Velocity: v_x=17.783 m/s || Distance: d_x=8453.834 cm
EEPROM[0] = Acceleration = 0.000 m/s^2
EEPROM[4] = Acceleration = 0.000 m/s^2
EEPROM[8] = Acceleration = 0.000 m/s^2
EEPROM[12] = Acceleration = 0.000 m/s^2
EEPROM[16] = Acceleration = 0.000 m/s^2
EEPROM[20] = Acceleration = 8.424 m/s^2
EEPROM[24] = Acceleration = 14.773 m/s^2
EEPROM[28] = Acceleration = 11.405 m/s^2
EEPROM[32] = Acceleration = 6.008 m/s^2
EEPROM[36] = Acceleration = 18.956 m/s^2
EEPROM[40] = Acceleration = 0.000 m/s^2
EEPROM[44] = Acceleration = 0.000 m/s^2
EEPROM[48] = Acceleration = 7.088 m/s^2
EEPROM[52] = Acceleration = 13.427 m/s^2
EEPROM[56] = Acceleration = 16.638 m/s^2
EEPROM[60] = Acceleration = 14.762 m/s^2
EEPROM[64] = Acceleration = 4.974 m/s^2
EEPROM[68] = Acceleration = 17.168 m/s^2
EEPROM[72] = Acceleration = 12.440 m/s^2
EEPROM[76] = Acceleration = 4.855 m/s^2
```

Cele 10 valori din primul dupa au fost shiftate cu 10 pozitii

S-au memorat urmatoarele 10 valori in EEPROM


Lenovo
P2-STM
rev1.2
ETTI
SW1 BOOT0
BOOT0
BOOT1
GND
USER