

# Digital Dice

Author: Voicu Teodora-Andreea, 333CA

## Introducere

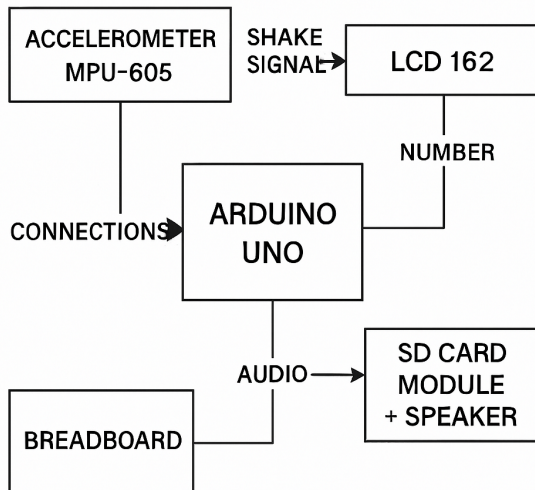
- Generează două număr aleatoare între 1 și 6, le afișează pe un display LCD și redă sunetul de aruncare a zarului.
- Oferă un exemplu practic de integrare a senzorilor și a modulelor audio/video cu un microcontroller Arduino, într-un proiect interactiv și educațional.
- Am vrut să creez un zar digital, care folosește un accelerometru pentru detectarea „shake”-ului și un LCD pentru afișarea rezultatului, completat de un efect sonor autentic.
- Demonstrează concepte de hardware și software embedded, este distractiv și poate fi extins la alte aplicații care folosesc detecția de mișcare și redarea de conținut multimedia.

## Descriere Generala

- **Hardware:**
  - **Arduino Uno** – unitatea centrală care primește date de la accelerometru, generează numărul și controlează LCD-ul și modulul audio.
  - **Modul accelerometru** – detectează gestul de „shake” pentru a declanșa aruncarea zarului.
  - **Display LCD** – afișează rezultatul numeric al aruncării zarului.
  - **Modul SD card + difuzor** – stochează și redă fișierul audio WAV cu efectul sonor de aruncare a zarului.
  - **Breadboard și cabluri de conexiune** – realizează legăturile electrice între toate modulele.
- **Software:**
  - Citire accelerometru și detectare „shake”.
  - Generare numere aleatorii 1-6.
  - Afișare mesaje și rezultate pe OLED.
  - Inițializare SD și redare WAV pe buzzer

- **Interacțiuni:**

1. Dispozitivul e zguduit → accelerometrul detectează mișcare peste SHAKE\_THRESHOLD
2. Stare SHAKING → afișează „Shake!” și porneste redarea DICE.WAV pe buzzer
3. După SHAKE\_TIMEOUT fără mișcare → oprește WAV, generează două valori random (1-6) și le afișează pe OLED
4. După RESULT\_HOLD → revine la „READY!” și așteaptă următorul shake



## Hardware Design

Lista Componente:

- Arduino:  
[https://www.optimusdigital.ro/ro/placi-avr/2563-placa-de-dezvoltare-compatibila-cu-arduino-uno-atmega328p-i-ch340-si-cablu-50-cm.html?search\\_query=arduino+uno&results=129](https://www.optimusdigital.ro/ro/placi-avr/2563-placa-de-dezvoltare-compatibila-cu-arduino-uno-atmega328p-i-ch340-si-cablu-50-cm.html?search_query=arduino+uno&results=129)
- Breadboard:  
[https://www.optimusdigital.ro/ro/prototipare-breadboard-uri/13245-breadboard-750-puncte.html?search\\_query=breadboard&results=126](https://www.optimusdigital.ro/ro/prototipare-breadboard-uri/13245-breadboard-750-puncte.html?search_query=breadboard&results=126)
- Accelerometer:  
[https://www.optimusdigital.ro/ro/senzori-senzori-inertiali/4407-modul-cu-accelerometru-triaxial-lis2dh12tr.html?search\\_query=accelerometru&results=68](https://www.optimusdigital.ro/ro/senzori-senzori-inertiali/4407-modul-cu-accelerometru-triaxial-lis2dh12tr.html?search_query=accelerometru&results=68)
- Display:  
<https://www.emag.ro/afisaj-oled-0-96-i2c-iic-ssd1306-128x64px-3-5v-e498/pd/DX0LYDYBM/>
- MicroSDSlot:  
[https://www.optimusdigital.ro/ro/memorii/1516-modul-slot-card-microsd.html?search\\_query=microsd&results=67](https://www.optimusdigital.ro/ro/memorii/1516-modul-slot-card-microsd.html?search_query=microsd&results=67)
- PassiveBuzzer:  
[https://www.optimusdigital.ro/ro/componente-electronice/12598-modul-buzzer-pasiv.html?search\\_query=buzzer+pasiv&results=14](https://www.optimusdigital.ro/ro/componente-electronice/12598-modul-buzzer-pasiv.html?search_query=buzzer+pasiv&results=14)
- Wires:  
[https://www.optimusdigital.ro/ro/fire-fire-mufate/890-set-fire-tata-tata-40p-30-cm.html?search\\_query=fire&results=428](https://www.optimusdigital.ro/ro/fire-fire-mufate/890-set-fire-tata-tata-40p-30-cm.html?search_query=fire&results=428)

Functional Overview:

- Când este zguduit accelerometrul LIS2DH12, acesta detectează mișcarea și declanșează Arduino-ul să genereze două numere aleatoare de la 1 la 6. Aceste numere sunt afisate pe ecranul OLED SSD1306, în timp ce, simultan, Arduino-ul redă un fișier WAV cu sunet de zaruri de pe cardul microSD prin buzzerul pasiv.



- GND → GND

Rețeaua de masă comună.

- A4 → SDA

Linia de date I<sup>2</sup>C: aici circulă octeți de la Arduino către pixelii OLED-ului.

- A5 → SCL

Linia de ceas I<sup>2</sup>C: sincronizează transferul de date pe SDA.

De ce OLED? – Afișează mesaje și rezultatul zarurilor într-un format grafic compact, cu consum redus de energie.

#### \* **Modul SD-card (SPI)**

- 5 V → VCC

Alimentează convertorul de nivel și interfața SPI.

- GND → GND

Referința de masă pentru comunicația SPI.

- D10 → CS (Chip Select)

Selectează modulul SD pe magistrala SPI (când e LOW, modulul răspunde).

- D11 → MOSI (Master Out Slave In)

Trimite datele din Arduino spre SD (de exemplu, comenzi de citire).

- D12 → MISO (Master In Slave Out)

Primește datele de la SD spre Arduino (conținutul fișierului WAV).

- D13 → SCK (Serial Clock)

Ceasul pentru sincronizarea transferului de biți MOSI/MISO.

De ce SD-card? – Permite stocarea fișierului WAV de sunet de zaruri, fără să încarce memoria internă a plăcii.

#### \* **Buzzer pasiv**

- D6 (PWM) → S

Pinul PWM generează semnalul audio propriu-zis (tonuri variabile conform valorilor citite din WAV).

- 5 V → +

Redă amplitudinea semnalului PWM către buzzer.

- GND → -

Întoarcerea curentului și referința de masă.

Ce face PWM? – Pulse Width Modulation modulează lățimea impulsurilor pentru a crea frecvențe audio perceptibile ca sunet.

### \* Modul LIS2DH12 (I<sup>2</sup>C)

- 3.3 V → VCC

Alimentează senzorul cu tensiune stabilă de 3.3 V.

- GND → GND

Stabilirea referinței zero pentru măsurări.

- A4 → SDA

Linia de date I<sup>2</sup>C: transmite valorile brute de accelerație.

- A5 → SCL

Linia de ceas I<sup>2</sup>C pentru sincronizarea transferului de date.

- 5 V → SDO/SA0

Configurează adresa I<sup>2</sup>C la 0x19 (pull-up intern), astfel evitând conflicte pe bus.

- (neconectat) → CS

CS rămâne HIGH; senzorul rămâne mereu activ pe I<sup>2</sup>C.

De ce accelerometru LIS2DH12? – Detectează gestul de „shake” cu precizie, fără butoane suplimentare, și are consum redus.

## Software Design

```
/*
 * DIGITAL DICE - Arduino UNO
 * – Shake → pornește DICE.WAV pe pin 9
 * – Stop Shake → afișează două numere pe OLED
 *
 * HARDWARE:
 * LIS2DH12 (I2C) | SDA/SCL = A4/A5
 * SSD1306 OLED 128×64 | SDA/SCL = A4/A5
 * Micro-SD (SPI) | CS = 10
 * Buzzer pasiv | pin 9 (Timer1 PWM)
 */

#include <Wire.h> // permite comunicarea I2C
#include "SparkFun_LIS2DH12.h" // driverul pentru accelerometru
```

```

#include <SPI.h> // pentru SD (comunicarea SPI)
#include <SD.h> // pt citire/scrie fisiere pe cardul SD
#include <TMRpcm.h> // redarea fisierelor WAV mono printr-un PWM pe buzzer
#include <U8g2lib.h> // desene pe OLED

#define SD_CS_PIN          10 // chip select pt modulul SD
#define SPEAKER_PIN        9 // buzzerul pasiv conectat pe pinul 9 (pwm timer)

#define SHAKE_THRESHOLD 2000 // daca diferenta de citiri depaseste
pragul e considerat shake
#define SHAKE_TIMEOUT     300 // ms fără miscare = aruncare terminata
#define RESULT_HOLD       1500 // ms pastram rez pe ecran inainte sa
schimbam iar la Ready

const char WAV_NAME[] = "DICE.WAV";

SPARKFUN_LIS2DH12 accel; // cream accelerometrul
TMRpcm          audio; // playerul care genereaza pwm ul pt a reda
fisierul wav pe buzzer
U8G2_SSD1306_128X64_NONAME_1_HW_I2C u8g2(U8G2_R0, U8X8_PIN_NONE); //
obiectul care controleaza ecranul

enum class Stare : uint8_t { GATA, SHAKING, REZULTAT }; // logicile posibile
Stare stareCurenta = Stare::GATA; // starea de start

unsigned long tStartStare; // mom cand s-a schimbat ultima data starea
int16_t lastX, lastY, lastZ; // valorile de pe fiecare axa a
accelometrului

// primeste un sir si un font, si scrie acel text centrat orizontal la linia
y pe ecran
void drawCentered(const char* text, uint8_t y, const uint8_t* font) {
    u8g2.setFont(font); // font select
    int w = u8g2.getStrWidth(text); // latime txt
    u8g2.firstPage(); do { u8g2.drawStr((128 - w) / 2, y, text); } while
(u8g2.nextPage()); // plaseaza incep in centrul ecranului
}

void setup() {
    // Serial.begin(9600); // pt debug
    if (!accel.begin()) while (1); // start accelerometru si se blocheaza
daca lipseste
    u8g2.begin(); // initializeaza ecranul
    drawCentered("READY!", 32, u8g2_font_ncenB14_tr); // afis ready pt inceput

    audio.speakerPin = SPEAKER_PIN; // spune playerului pe ce pin scoate
semnal pwm catre buzzer
    audio.CSPin      = SD_CS_PIN; // ce pin e chipselect pt sd

    if (!SD.begin(SD_CS_PIN)) while (1); // stop dacă SD fail

```

```
if (!SD.exists(WAV_NAME)) while (1); // stop dacă lipsește DICE.wav

audio.setVolume(5); // 0-6, 5 a fost optim
audio.quality(1); // pt sunet mai clar

randomSeed(analogRead(A0)); // folosește zgomot de pe pinul A0 ca baza pt
generarea numerelor aleatoare
lastX = accel.getRawX(); lastY = accel.getRawY(); lastZ =
accel.getRawZ(); // citește și salvează primele valori de la accelro
}

void loop() {
  int16_t x = accel.getRawX(), y = accel.getRawY(), z = accel.getRawZ();
  // cit accelerația pe fiecare axă
  int16_t dX = abs(x - lastX), dY = abs(y - lastY), dZ = abs(z - lastZ);
  // dif față de ultima dată
  lastX = x; lastY = y; lastZ = z; // actualizare valori

  switch (stareCurenta) { // alegem ce stare urm în fct de starea curenta

  case Stare::GATA:
    if (dX > SHAKE_THRESHOLD || dY > SHAKE_THRESHOLD || dZ > SHAKE_THRESHOLD)
    { // am detectat mișcarea
      stareCurenta = Stare::SHAKING; // deci trecem în starea shaking
      tStartStare = millis();
      drawCentered("Shake!", 32, u8g2_font_ncenB14_tr); // afișăm shake
      audio.play(WAV_NAME); // și redăm sunetul DICE
    }
    break;

  case Stare::SHAKING:
    if (dX > SHAKE_THRESHOLD || dY > SHAKE_THRESHOLD || dZ > SHAKE_THRESHOLD)
    { // dacă începe să scuturăm accelero
      tStartStare = millis(); // resetăm timeoutul
    }
    if (millis() - tStartStare > SHAKE_TIMEOUT) { // dacă de la ultima
miscare a trecut mai mult de shake timeout
      audio.stopPlayback(); // oprim sunetul
      uint8_t d1 = random(1,7), d2 = random(1,7); // afișăm 2 nre random
      char buf[6]; sprintf(buf, sizeof(buf), "%d %d", d1, d2);
      drawCentered(buf, 50, u8g2_font_logisoso32_tn);

      stareCurenta = Stare::REZULTAT; // trecem la starea de result
      tStartStare = millis();
    }
    break;

  case Stare::REZULTAT:
    if (millis() - tStartStare > RESULT_HOLD) { // dacă a trecut timpul de
cat tb tinut rez pe ecran
      drawCentered("READY!", 32, u8g2_font_ncenB14_tr); // afișăm din nou
```

```
starea de ready pt urmatoare miscare si
    stareCurenta = Stare::GATA; // actualizam si starea
}
break;
}

delay(5);           // pauza pt CPU
}
```

## Biblioteci externe utilizate

- “Wire.h”

comunicare I<sup>2</sup>C cu accelerometrul și OLED

- “SparkFun\_LIS2DH12.h”

driver pentru citirea accelerațiilor de la senzorul “LIS2DH12”

- “SPI.h”

suport hardware “SPI” pentru modulul SD

- “SD.h”

acces la sistemul de fișiere “FAT” de pe cardul micro-SD

- “TMRpcm.h”

redare fișiere “WAV” mono prin “PWM” (Timer1) pe buzzerul pasiv

- “U8g2lib.h”

comandă și desen pe “OLED SSD1306” 128×64

## Componente ale codului

Codul este împărțit în mai multe componente:

- interfațare cu hardware-ul, acoperind:

inițializare și citire continuă a accelerometrului “LIS2DH12”

- control “SPI/SD” pentru verificarea și deschiderea fișierului “DICE.WAV”
- generare “PWM” audio pe pinul buzzer-ului
- control “I<sup>2</sup>C” pentru desenul pe OLED
- State machine, ce include:

‘ ‘GATA’ ’ – așteaptă un shake și afișează “READY!”

```
    'SHAKING' – redă sunetul și afișează "Shake!" cât timp mișcarea  
    continuă
```

```
    'REZULTAT' – generează două valori aleatorii, le afișează pe OLED,  
    apoi revine la 'GATA'
```

- funcții auxiliare:

```
    'randomSeed(analogRead(A0))' pentru entropie la pornire  
    'random(1,7)' pentru fiecare zar  
    'drawCentered(text, y, font)' – calculează lățimea textului și îl  
    plasează centrat
```

## Fluxul aplicației

→ La pornire, "setup()" inițializează senzorul, OLED-ul, SD-ul, audio și RNG-ul, apoi pe ecran apare "READY!".

→ În "loop()" se citesc accelerațiile și se calculează delta față de ultima citire. Dacă "delta > SHAKE\_THRESHOLD", starea trece în "SHAKING".

→ În "SHAKING" se pornește "audio.play("DICE.WAV")" și se afișează "Shake!". La fiecare citire de mișcare, timer-ul este resetat.

→ După "SHAKE\_TIMEOUT" ms fără mișcare, "audio.stopPlayback()". Se generează două numere (1-6) și se afișează cu font mare; starea devine "REZULTAT".

→ După "RESULT\_HOLD" ms, ecranul revine la "READY!" iar starea revine în "GATA", așteptând un nou shake.

## Detalii de implementare

→ Praguri configurabile:

- "SHAKE\_THRESHOLD = 2000" - sensibilitate la mișcare
- "SHAKE\_TIMEOUT = 300 ms" - interval fără mișcare înainte de oprire
- "RESULT\_HOLD = 1500 ms" - cât timp rămâne afișat rezultatul

→ Audio:

- "audio.setVolume(5)" - volum optim fără distorsiuni
- "audio.quality(1)" - oversampling 2x pentru claritate

→ Desen pe OLED:

- re-desenările sunt minimizate - textul este redesenat doar la tranziții de stare

→ Random seed:

- folosirea unui pin analog “flotant” pentru a evita secvențe pseudo-aleatoare repetitive

## Optimizări

- Mașina de stări izolează logica de desen și audio pe blocuri distincte, eliminând “redraw”-uri inutile
- Actualizare “I<sup>2</sup>C” minimală - apeluri “drawCentered()” doar la tranziții de stare
- Sincronizare audio-I<sup>2</sup>C - “audio.stopPlayback()” este apelat înainte de “drawCentered()” pentru a evita blocaje pe magistrala comună
- Delay mic - “delay(5)” menține CPU-ul liber pentru ISR-ul audio și reduce consumul inutil

## Used labs

### Lab0: GPIO

- controlul pinilor digitali:
- D9 - ieșire PWM către buzzer
- D10 - Chip Select pentru modulul SD
- LED-uri/Serial (opțional) pentru depanare

### Lab3: Timere & PWM

- biblioteca TMRpcm folosește Timer1 în modul PWM pentru a genera semnalul audio necesar redării fișierului WAV pe buzzerul pasiv.

### Lab5: SPI

- protocolul SPI asigură comunicația cu cardul micro-SD:
- D11 → MOSI, D12 → MISO, D13 → SCK, D10 → CS
- biblioteca SD.h gestionează sistemul de fișiere FAT și citirea fișierului DICE.WAV.

### Lab6: I<sup>2</sup>C (TWI)

- magistrala I<sup>2</sup>C este partajată de:
- LIS2DH12 - trimite datele brute de accelerație (detectarea shake-ului)
- SSD1306 OLED - primește comenzile de desen (mesajele și rezultatul zarurilor)
- linii comune: A4 → SDA, A5 → SCL.

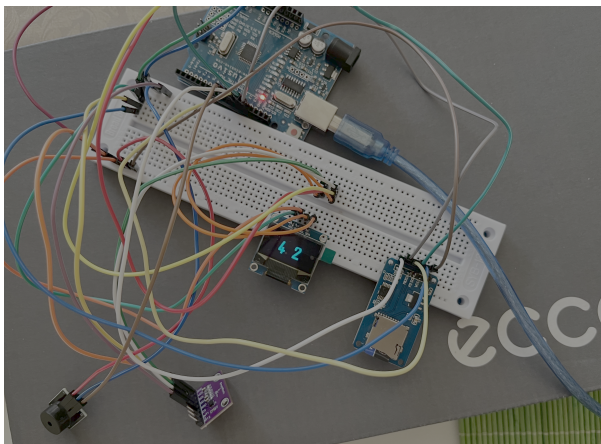
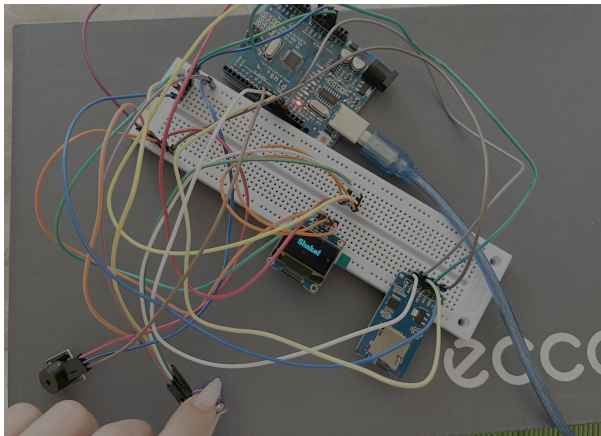
### Lab1: USART

- Serial.begin(9600) este prezent pentru diagnostic; când este decommentat, permite trimiterea de mesaje de debug în Serial Monitor.

## Rezultate Obținute

[GitHub Repository](#)

## Demo Video



## Concluzii

Acest proiect a reprezentat o ocazie excelentă de a vedea în practică conceptele teoretice și de a le pune în aplicare pentru a crea ceva distractiv.

## Bibliografie/Resurse

[Circuit Studio](#)

[Pagina oficială Arduino](#)

[Adafruit Learning System](#)

[Export to PDF](#)

From:

<http://ocw.cs.pub.ro/courses/> - **CS Open CourseWare**

Permanent link:

<http://ocw.cs.pub.ro/courses/pm/prj2025/vstoica/teodora.voicu0210>



Last update: **2025/05/29 19:57**