Arduino Graphix

Nume: Popa Andrei

Grupă: 333CA

Introducere

Arduino Graphix este un demo de pipeline grafic software implementat pe microcontrolerul ATmega328P. Se afișează pe un ecran LCD un mediu 3D, iar utilizatorul se poate folosi de un keypad pentru a modifica perspectiva de vizualiare (pentru a mișca camera).

Sistemul este format din 2 plăcuțe cu microcontroler ATmega328P:

- plăcuța 1 joacă rolul unui GPU: va fi responsabilă să deseneze scena pe un ecran LCD într-un timp cât mai scurt.
- plăcuța 2 joacă rolul unui CPU: va ține cont despre obiectele din scenă și despre cameră și va transmite informațiile relevante către “GPU” și va procesa inputul de la keypad (utilizator).

Motivul pentru care am ales acest proiect este interesul meu față de modul de funcționare a procesoarelor video și modul prin care acestea interacționează cu restul sistemului de calcul pentru a prezenta/desena informații pe un display.

Scopul meu principal este de a învăța prin experiență cum se implementează un pipeline grafic (chiar dacă este un model software foarte limitat și simplificat) și totodată, ca scop secundar, să scriu de mână un “driver” simplu pentru ecranul LCD.

Ca utilitate pentru mine (și alții), vreau ca proiectul să servească ca un exemplu de compromisuri ce trebuie făcute pentru a obține un pipeline grafic 3D funcțional, dar și utilizabil (cu framerate decent) pe un sistem hardware foarte limitat.

Schemă bloc

Hardware Design

Bill of Materials

Componentă	Cantitate	Link	Preț unitar (RON)
Placă dezvoltare Arduino Uno AtMega 328p	2	Link	30.93
Display LCD SPI	1	Link	40.62
Breadboard	1	Link	6.62
Tastatură membrană 3×4	1	Link	4.30
Fire Dupont Tată-Tată 10cm	21	Link	0.15
Rezistențe 1k	5	Link	0.19
Rezistențe 100k	1	Link	0.19
Preț total			**117.69 RON**

Schema electrică

Pinout

Display LCD:

Pin Display	Pin Arduino	Pin ATmega328p
BL	5V	
RST	5V	
DC	D10	PB2
CS	GND	
CLK	D13	PB5 / SCK
DIN	D11	PB3 / MOSI
GND	GND	
VCC	5V	

Deoarece există doar un singur periferic conectat prin SPI, am conectat pinul de CS (chip select) la GND, astfel acesta este selectat continuu. Pinul de RST (reset) este conectat la 5V (acesta funcționează în logică negată), deoarece nu am nevoie să resetez software display-ul.

Keypad 4×4:

Pin Keypad	Pin Arduino	PinATmega328p
1 (Column)	D2	PD2 / PCINT18
2 (Column)	D3	PD3 / PCINT19
3 (Column)	D4	PD4 / PCINT20
4 (Column)	D5	PD5 / PCINT21
5 (Row)	D6	PD6
6 (Row)	D7	PD7
7 (Row)	D8	PB0

8 (ROW)	D9	PB1

Ținând doar câte o coloană a keypad-ului pe high la un moment dat, în funcție de care rând este pe low, obținem ce buton a fost apăsat. Am ales pinii pentru coloană pentru a primi întreruperi în momentul apăsării unui buton. Am ales pinii pentru rând pentru că sunt următorii pini pe plăcuță.

ATmega328P:

Pin Arduino (1)	**Pin ATmega(1)**	**Pin Arduino(2)**	**Pin ATmega (2)**
D0	PD0 / RXD	D1	PD1 / TXD
D1	PD1 / TXD	D0	PD0 / RXD
GND	GND	GND	GND

Pinii aleși sunt pinii pentru USART și sunt legați RX ↔ TX.

Software Design

Mediu de dezvoltare Editor de text: Sublime, și atât.

Biblioteci utilizate

- **avr/io.h**
- **avr/interrupt.h**
- **util/delay.h**
- **stdint.h**

Justificarea utilizării funcționalităților din laborator

- **Timer 1 in mod CTC**: Prescaler CLK/64, Top=250 ⇒ Numărător cu frecvență de 1kHz, folosit pentru a număra milisecunde.
- **Întreruperi Hardware Pin Change**: activate prin regiștrii PCICR și PCMSK2 pe Pinul 2 → permit citirea asincronă a butoanelor din matrice.
- **Modulul SPI**: comunicarea cu display-ul se face prin SPI
- **Modulul USART**: comunicarea între plăcuțe se face prin UART.

Scheletul proiectului și interacțiunea funcționalităților

- **cpu.c**: Unitatea care citește input de la utilizator și trimite comenzi GPU-ului. Inputul provine de la o matrice de butoane 4×4, fiecare rând este activat alternativ continuu, iar pe fiecare coloană activează o întrerupere PCINT2 când este apăsată (și rândul e activ).
- **gpu.c**: Unitatea care primește comenzi de la CPU, le execută și apoi re-desenează scena în urma unei schimbări. Comenzile primite sunt plasate într-un buffer, iar gpu-ul va aștepta un scurt interval de timp până la procesarea unei comenzi (dacă comanda nu se află deja complet în buffer). De asemenea, gpu-ul va încerca să execute cât mai multe comenzi în același frame, dar maxim 10. Dacă o comandă nu e recunoscută, sau nu a ajuns complet în buffer, întreg buffer-ul este golit, iar comanda este ignorată. Desenarea scenei se face prin desenarea tuturor fețelor (triunghiurilor) stocate în memoria gpu-ului (maxim 35 de fețe). Înainte de desenare propriu-zisă, fețele sunt sortate progresiv, descrescător față de distanța față de observator (cameră).

- **buffer.c**: Implementarea buffer-ului folosit de gpu.
- **lcd_screen.c**: Implementarea “driver-ului” pentru display-ul spi. De asemenea conține și câteva funcții pentru desenarea formelor simple (dreptunghi, triunghi, pixel).
- **geometry.c**: Implementarea transformărilor necesare proiectării scenei 3D pe ecranul 2D al display-ului.
- **spi.c**, **uart.c**, **timer.c**: implementarea modulelor respective.

Calibrarea perifericelor și sincronizarea I/O

- **Stabilizarea electrică a intrării:** când un rând este “activat” acesta acționează ca rezistență de pull-up pentru toate butoanele de pe acel rând. Ciclarea continuă a rândurilor va genera PCINT constant, deci trebuie verificat faptul că un buton a fost apăsat (verificarea dacă pinul este tras la masă).
- **Sincronizarea CPU-ului cu GPU-ul**: La pornire, CPU-ul va trimite continuu mesajul HELLO_GPU pe uart, odată la 1 secundă. În momentul în care primește răspunsul HELLO_BACK, consideră că este conectat la GPU și intră în bucla în care poate procesa input.
- **Setup display**: Display-ul este setat să facă mirror vertical și orizontal (deoarece este conectat “invers” pe breadboard).

Optimizări arhitecturale realizate

- **Optimizare de timp**: Pentru a nu trimite la fiecare frame întreaga scenă de la CPU la GPU, GPU-ul va stoca informații despre fiecare față(poziție, culoare, id). De asemenea, dacă scena rămâne neschimbată între frame-uri, GPU-ul nu o va mai redesena. Acesta reprezintă un trade-off major, deoarece memoria RAM este foarte limitată (2KB), nu se pot păstra foarte multe fețe (maxim 35, este nevoie și de buffer de comandă, alte flag-uri și o stivă destul de liberă pentru apel de funcții).
- **Optimizare de spațiu**: Deoarece limitarea de memorie nu permite existența unui depth buffer, GPU-ul sortează fețele descrescător față de distanța față de cameră. În fiecare frame, execută 2 pași de bubble-sort.

Limitări

- **Memorie**: Din cauza dimensiunii mici a memoriei RAM, scena trebuie redesenată la fiecare schimbare, asta introduce fenomenul de “blink-ing”, ecranul fiind golit la fiecare frame, adaugă un timp vizibil în care ecranul este complet negru înainte de a desena fețele. De asemenea, sortarea fețelor este imperfectă, deoarece sortează față de distanța medie a fiecărui vârf față de cameră, în unele situații fețele se pot suprapune în mod ciudat.
- **Canal imperfect de comunicație**: Deoarece modulul USART este setat la un baud-rate ridicat(115k), pot apărea erori în transmisia unui cadru, asta induce într-o posibilă pierdere a informației. (Problema cea mai mare este trimiterea greșită sau trimiterea parțială a unei fețe, totuși asta se întâmple foarte rar).

Rezultate Obţinute

Demo

Concluzii

Proiectul demonstrează că o implementare a unui engine 3D este posibilă, chiar și într-un mediu cu resurse foarte limitate, însă compromisurile tehnice nu permit existența unor funcționalități de bază (ex: double buffering), esențiale pentru un engine modern.

De asemenea, am învățat să fac debugging aproape orb, întrucât, comunicarea dintre CPU și GPU este realizată prin USART, nu puteam comunica cu plăcuțele prin intermediul interfeței seriale (nu aveam acces la “debug prin printf”). Singurul instrument de debug pe care îl aveam la dispoziție era ecranul SPI: fie schimbam complet culoarea fundalului, fie “desenam numere” (într-un mod rudimentar). Cea mai importantă parte a fost să implementez modul prin care CPU-ul trimite comenzi către GPU, fără să “înece” canalul de comunicație (și să piardă comenzi importante în proces).

Download

https://github.com/andarkrc/Basic-Arduino-Game

popaandrei333cacod.zip

Jurnal

- 7.04.2026 - Alegere tema proiect si confirmare.
- 7.05.2026 - Creare pagină ocw pentru proiect.
- 18.05.2026 - Finalizarea parțială a codului pentru a prezenta în cadrul milestone-ului software.
- 23.05.2026 - Finalizarea codului.
- 24.05.2026 - Update final la pagina de proiect.

Bibliografie/Resurse

Resurse Hardware

Resurse Software

- Laboratoarele
- Datasheet display
- Datasheet ATmega328P

Export to PDF