Universitatea Tehnică a Moldovei
Facultatea Calculatoare Informatică şi Microelectronică
Departamentul Ingineria Software și Automatică

RAPORT

Lucrarea de laborator nr. 2.1
La disciplina “Internetul Lucrurilor”
Tema: Sisteme de Operare - Secventiale

A efectuat: st. gr. SI-211 Adrian Chihai
A verificat: Valentina Astafi

Chișinău – 2024

Definire Problema

Realizarea unei aplicații pentru MCU care va rula minim 3 task-uri – ***Secvential***

Aplicația va rula minim 3 task-uri printre care

1. Button Led - Schimbare stare LED la detecția unei apăsări pe buton.
2. un al doilea Led Intermitent în faza în care LED-ul de la primul Task e stins
3. Incrementare/decrementare valoare a unei variabile la apăsarea a doua butoane care va reprezenta numărul de recurențe/timp în care ledul de la al doilea task se va afla într-o stare
4. Task-ul de Idle se va utiliza pentru afișarea stărilor din program, cum ar fi, afișare stare LED, și afișare mesaj la detecția apăsării butoanelor, o implementare fiind ca la apăsarea butonului sa se seteze o variabila, iar la afișare mesaj - resetare, implementând mecanismul provider/consumer.

Indicații pentru implementare

1. sa se implementeze comunicarea intre Taskuri ca provider consumer, adica:

- task-ul care generează date, provider, stochează rezultatele într-o variabila globala/semnal

- task-ul care utilizează aceste date, consumer, citește aceasta variabila/semnal.

de ex: task de UI (LCD sau Serial) preia informația din niște variabile-semnale globale și raportează

2. A se urma principiile prezentate la curs Sisteme Secvențiale

- stabilirea rezonabila a recurentei pentru a diminua incarcarea procesorului

- stabilirea ofsetului, intru a activa în ordinea cuvenita task urile

3: Task-ul de raportare pentru Secvențial cu utilizare STDIO printf() catre LCD va fi rulat in bucla infinita/IDLE deoarece este bazat pe un spin lock și ar putea bloca întreruperile deci secvențial clasic - printf & delay in main loop,

Pentru a crea un program complex și secvențial pe Arduino Uno, am împărțit codul în funcții distincte: changeLed1State(), led2Blink() și changeTime(). În funcția principală, apelăm aceste patru funcții în ordine.

Inițial, am declarat două LED-uri: unul care se aprinde la apăsarea unui buton și altul care clipește. Am definit și trei butoane: unul pentru a controla primul LED și două pentru a ajusta timpul de clipire. Variabilele asociate indică starea inițială a LED-urilor și butoanelor, precum și timpul de clipire, setat inițial la 200*10 milisecunde.

Funcția changeLed1State() schimbă starea primului LED. Verifică dacă starea butonului s-a modificat și, dacă este LOW, inversează starea LED-ului, afișând noua sa stare.

Funcția led2Blink() gestionează clipirea celui de-al doilea LED. Dacă starea LED-ului este LOW, se verifică timpul scurs de la pornire. Dacă acest timp depășește intervalul stabilit, se actualizează ultima valoare a timpului de clipire și se schimbă starea LED-ului. Dacă timpul depășește intervalul, LED-ul este setat pe LOW.

Funcția changeTime() ajustează timpul de clipire în funcție de butonul apăsat. Are două părți: prima verifică dacă starea butonului de minus s-a schimbat și, dacă intervalul nu este zero, scade 100 milisecunde din valoare, afișând un mesaj de confirmare. A doua parte face același lucru pentru butonul de plus, adăugând 100 milisecunde și afișând un mesaj de confirmare.

Am utilizat și funcții auxiliare:

- digitalRead(button) – citește valoarea unui buton sau LED.
- digitalWrite(led, LOW) – setează starea LED-ului la LOW sau HIGH.
- millis() – returnează timpul scurs de la pornirea programului.
- Serial.println("") – afișează un șir de caractere pe monitorul serial.

Pentru detalii suplimentare, consultați ANEXA A.

Figurile de mai jos ilustrează logica întregului program și a fiecărei funcții individuale. Figura 1 prezintă funcția care verifică și inversează starea primului LED la apăsarea unui buton. Figura 2 arată funcția care controlează clipirea celui de-al doilea LED la un interval specificat.

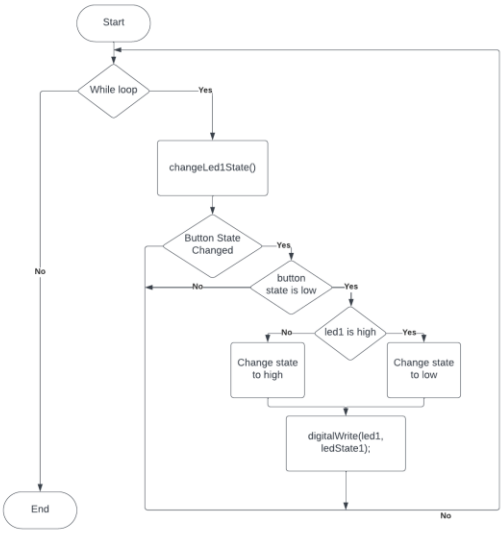

Figura 1. Schimbarea stării ledului

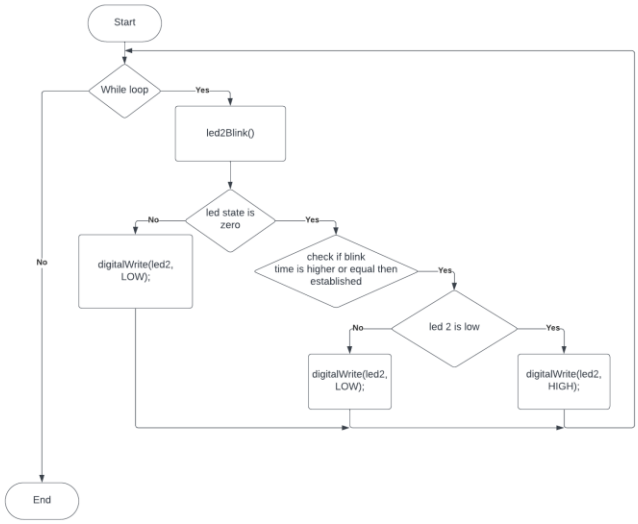

Figure 2. Blink Led

În figura 3, respectiv, este reprezentată a treia funcție care primește semnalul de la 2 butoane și, în funcție de butonul apăsat, mărește sau micșorează timpul de clipire al becului 2, iar în figura 4 putem vedea cum funcționează întregul program și ordinea de execuție a funcțiilor.

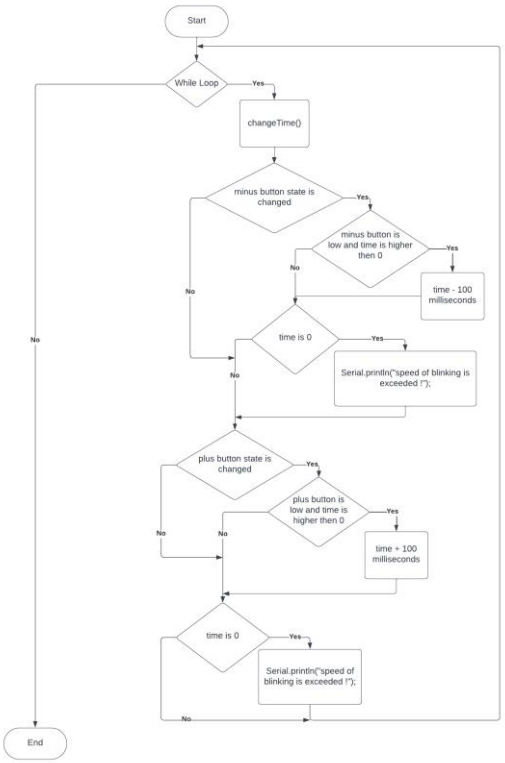

Figura 3. Schimbarea timpului de clipire

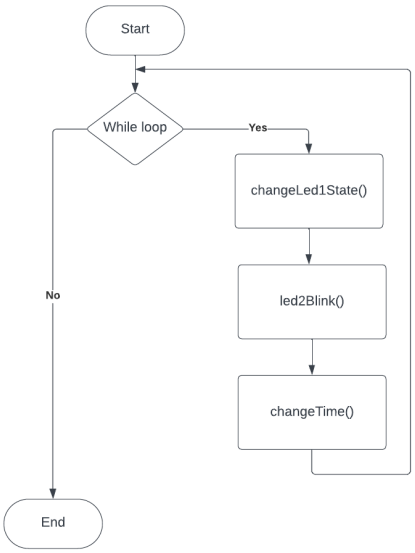

Figura 4. Programul Main

În imaginea de mai jos putem vedea cum a fost realizată simularea de laborator și cum au fost combinate elementele (placa Arduino Uno, 3 butoane, 2 becuri LED și rezistențele acestora).

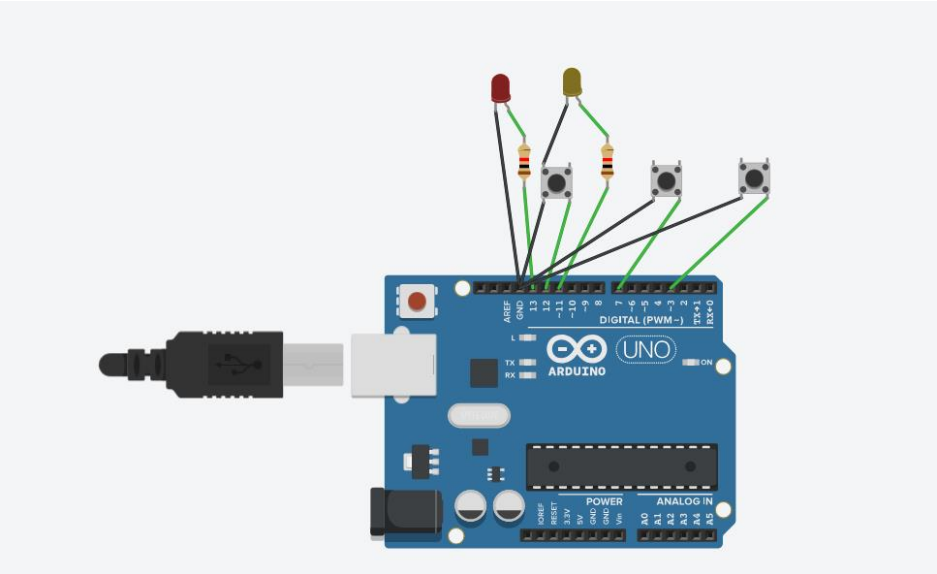

Figura 5. Arduino sequential OS

În imaginea de mai jos putem observa cum rulează programul

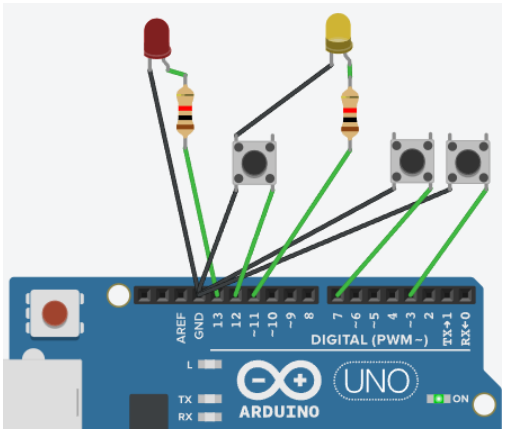

Figura 6. Rularea sequential OS

Concluzie

În cadrul acestui laborator, am învățat cum să creăm sisteme de operare secvențiale. Am descoperit cum să separăm diferite logici din blocul principal și să creăm funcții distincte pentru sarcini concrete. Am înțeles conceptul de schimbare a stării unui obiect sau a unei variabile în funcție de starea altui obiect. De asemenea, am utilizat monitorul serial pentru a afișa informații despre acțiunile noastre și timpul curent în care LED-ul clipește. Am realizat cum proiectele mari pot avea o logică de business separată pentru a simplifica complexitatea programului.

BIBLIOGRAPHY

1. TINKERCAD: *Arduino Simulator*. Online Simulator, © 2024 Autodesk, Inc [21.02.2024], Link: **https://www.tinkercad.com/dashboard**

2. LUCID: *Flow Chart Maker*. Software for Creating Flow Charts and Block Diagrams, ⓒ2023 [21.02.2024], Link: **https://lucid.app**

ANEXA 1

```
int led1 = 11;
int led2 = 13;
int button1 = 12;
int button2 = 7;
int button3 = 3;
int recurrence = 1000;
bool buttonStateBool = false;
byte lastButtonState = LOW;
byte lastButtonState1 = LOW;
byte lastButtonState2 = LOW;
byte ledState1 = LOW;
byte ledState2 = LOW;
unsigned long previousMillis = 0;
void setup() {
  Serial.begin(9600);
  pinMode(led1, OUTPUT);
  pinMode(led2, OUTPUT);
  pinMode(button1, INPUT_PULLUP);
  pinMode(button2, INPUT_PULLUP);
  pinMode(button3, INPUT_PULLUP);
}
void loop() {
  changeLed1State();
  led2Blink();
  changeTime();
}
void changeLed1State(){
  byte buttonState = digitalRead(button1);
  if (buttonState != lastButtonState)
  {
    lastButtonState = buttonState;
    if (buttonState == LOW)
    {
      ledState1 = (ledState1 == HIGH) ? LOW : HIGH;
      digitalWrite(led1, ledState1);
      Serial.println("LED 1 changed state");
    }
  }
}
void led2Blink() {
  static unsigned long previousBlinkTime = 0;
```

```
  int ledState = digitalRead(led1);
  if (ledState == 0)
  {
    unsigned long currentMillis = millis();
    if (currentMillis - previousBlinkTime >= recurrence)
    {
      previousBlinkTime = currentMillis;
      if (digitalRead(led2) == 0)
      {
        digitalWrite(led2, HIGH);
        Serial.println("LED 2 HIGH");
      }
      else{
        digitalWrite(led2, LOW);
        Serial.println("LED 2 LOW");
      }
    }
  }else {
    digitalWrite(led2, LOW);
  }
}
void changeTime(){
int buttonState1 = digitalRead(button2);
 if (buttonState1 != lastButtonState1){
   lastButtonState1 = buttonState1;
   if (buttonState1 == LOW && recurrence > 0){
     recurrence = recurrence - 100;
     Serial.println(recurrence);
   }
   if (recurrence == 0){
     Serial.println("speed of blinking is exceeded !"); }
 }
 int buttonState2 = digitalRead(button3);
 if (buttonState2 != lastButtonState2){
   lastButtonState2 = buttonState2;
   if (buttonState2 == LOW && recurrence > 0){
     recurrence = recurrence + 100;
     Serial.println(recurrence);
   }
   if (recurrence == 0){
     Serial.println("speed of blinking is exceeded !"); }
 }
}
```