Ministerul Educației și Cercetării al Republicii Moldova
Universitatea Tehnică a Moldovei
Facultatea Calculatoare, Informatică și Microelectronică
Departamentul Ingineria Software și Automatică

Lucrare de laborator Nr. 6

Disciplina: Internetul lucrurilor (IOT)
Tema: Control comportamental cu Automate Finite

A efectuat: Popa Cătălin, gr. TI-211

A verificat: Cristian Lupan, asist. univ.

Chișinău 2024

6.1 Problema propusă:

Realizarea applicatiei de Control comportamental cu Automate Finite - Automat de control Button-Led.

Descrierea problemei

1. Sa se realizeze o aplicatie ce va implementa Automatele finite dupa cum urmeaza:

   Proiectare Automat Finit aplicatie Button-Led.

6.2 Problema propusă:

Sa se realizeze o aplicatie ce va implementa Automatele finite dupa cum urmeaza:

Descrierea problemei

1. Proiectare Automat Finit aplicatie Semafor.

Obiective

1. Crearea unei aplicații care simulează un control simplu de tip „Button-Led” folosind concepte de automate finite.
2. Realizarea unei aplicații care simulează funcționarea unui semafor pe baza unui automat finit.

Introducere

Microcontrolerele permit controlul și monitorizarea dispozitivelor fizice, aducând tehnologia mai aproape de utilizator prin intermediul unei interfețe intuitive și simplificate. Un exemplu clasic de interacțiune între utilizator și dispozitive fizice este controlul unui LED prin apăsarea unui buton sau trimiterea de comenzi printr-o interfață serială. Această tehnologie este utilizată într-o gamă largă de aplicații, de la sisteme de iluminat inteligente până la dispozitive electronice complexe utilizate în automatizări industriale. Unul dintre avantajele microcontrolerelor este costul redus și flexibilitatea lor, permițând implementarea rapidă de prototipuri și teste pentru diverse aplicații IoT. Microcontrolerele precum cele din familia Arduino sau ESP sunt folosite frecvent pentru prototiparea sistemelor IoT datorită ecosistemului extins de biblioteci și comunități de dezvoltatori care susțin aceste platforme .

Materiale și metode

Arduino UNO - platformă de microcontroler open-source bazată pe microcontrolerul ATmega328P. Este ușor de utilizat și permite conectarea ușoară a componentelor externe prin pini digitali și analogici. Acesta este ideal pentru prototiparea de sisteme IoT și proiecte electronice interactive. Afișarea este în figura 1.1.

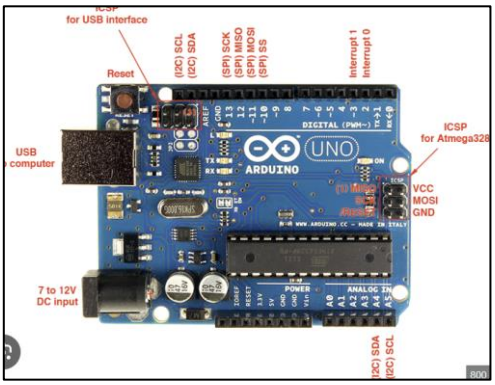


Fig. 1.1. Arduino UNO

- 14 pini digitali I/O (din care 6 sunt PWM);
- 6 intrări analogice;
- memorie flash de 32 KB;
- tensiune de operare: 5V;
- interfață de programare: Arduino IDE.

Rezistor - Rezistorul este utilizat pentru a limita curentul ce trece prin LED, prevenind arderea acestuia. Pentru a proteja LED-urile, se vor folosi rezistoare de aproximativ 220Ω. Afișarea este în figura 1.2.

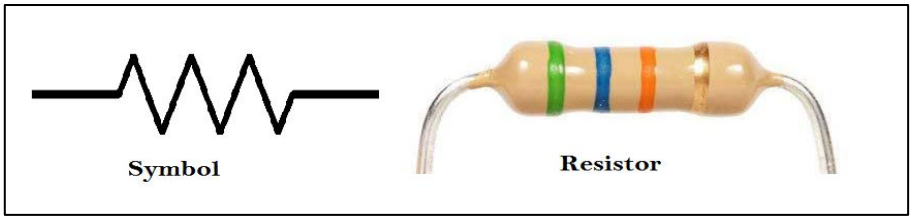


Fig. 1.2. Rezistor

Fire Jumper - Firele jumper sunt utilizate pentru a face legături electrice între diverse componente și plăcuța Arduino. Acestea permit conectarea componentelor fără a fi necesară lipirea acestora. Afișarea este în figura 1.3.

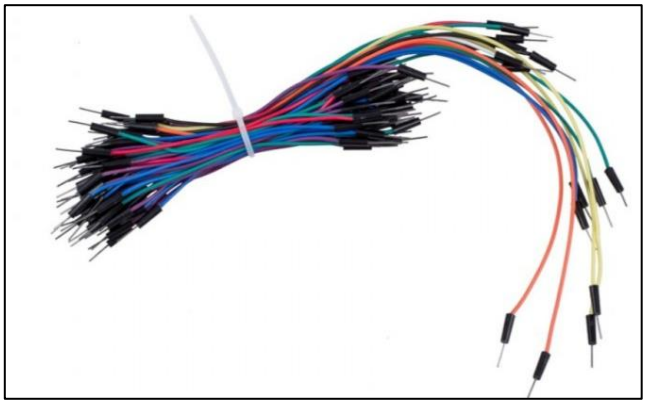


Fig. 1.3. Fire Jumper

LED-ul - (diodă emițătoare de lumină) este utilizată pentru a semnaliza starea sistemului în funcție de comenzile primite sau de validitatea codului introdus pe tastatură. Afișarea este în figura 1.4.

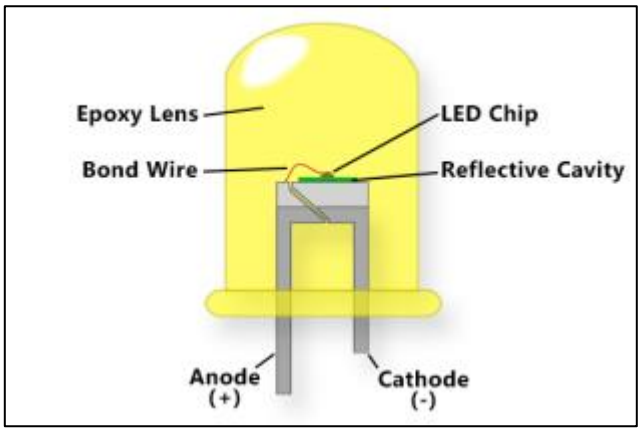


Fig. 1.4. Fire Jumper

Buton - În aplicații de acest gen, un buton (push button) este utilizat pentru a oferi utilizatorului o modalitate de a interacționa cu sistemul printr-un eveniment de apăsare. În această situație, ar putea fi utilizat pentru a reseta sistemul sau pentru a iniția o acțiune nouă. Afișarea este în figura 1.5.

Fig. 1.5. Fire Jumper

Metode

Configurarea Arduino UNO: Arduino va fi configurat folosind Arduino IDE. Se va implementa codul pentru a citi inputurile de la potențiometru și encoder, pentru ca apoi să aplice acțiuni asupra dispozitivelor externe și afișarea pe LCD.

Interacțiunea cu utilizatorul: Sistemul va primi comenzi potențiometru și sencoder și va răspunde prin intermediul LCD-ului şi serial monitor.

Validare în simulator: Codul și conexiunile vor fi testate inițial într-un simulator de circuite, cum ar fi Tinkercad, Wokwi, pentru a verifica funcționalitatea corectă. După simulare, se va testa și pe un circuit real.

6.1

Pentru prima sarcină am avut nevoie de un LED, 2 rezistoare și un buton. Simularea realizată este în figura 1.6. Schimbarea starii are loc la apăsarea butonului.

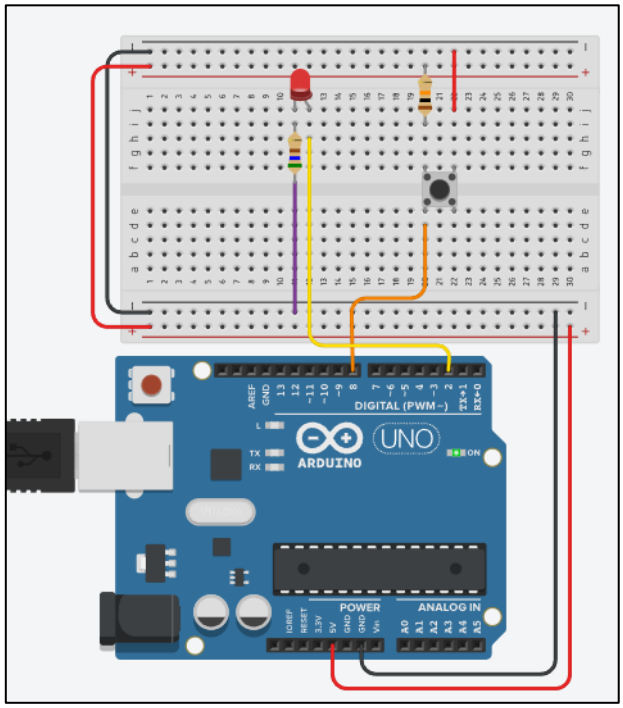


Figura 1.6 – Simulare microcontroler

Mai întâi setăm stările pentru automatul finit, care sunt Out, Time, next. Out este pentru starea ledului, unde 1 este ON și 0 este OFF. Time este timpul de întârziere în milisecunde pentru tranziții. Next este următoarea stare în funcție de intrare 0 sau 1.

```
struct State {
    unsigned long Out;
```

```
    unsigned long Time;
    unsigned long Next[2]; };
```

În continuare definim stările FSM, care are 2 stări. LED_ON, atunci când ieșirea este 1, iar următoarea stare depinde de intrare.

```
typedef const struct State STyp;
STyp FSM[2] = {
    {0, 100, {LED_OFF_STATE, LED_ON_STATE}},  // LED_OFF
    {1, 100, {LED_ON_STATE, LED_OFF_STATE}}   // LED_ON
};
```

Tabelul de tranziție a stărilor este afișat în figura 1.7.

Nr	Nume	Out	Delay	In = 0	In = 1
0	LED_OFF	0	100	LED_OFF	LED_ON
1	LED_ON	1	100	LED_ON	LED_OFF

Figura 1.7

Apoi setăm variabilele pentru starea automatului și debounce. Am folosit lastDebounceTime și debounceDelay sunt folosite pentru a evita citirea multiplă a unei singure apăsări de buton.

```
  int FSM_State = LED_OFF_STATE;
unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 50;
bool buttonPressed = false;
```

În funcția loop() verificăm starea butonului și debounce. Atunci când butonul este apăsat, comutăm la următoarea starea. Debounce-ul verifică dacă a trecut timpul minim pentru a considera apăsarea validă.

```
  int buttonState = digitalRead(BUTTON_PIN);
if (buttonState == LOW && !buttonPressed && (millis() - lastDebounceTime > debounceDelay)) {
    FSM_State = FSM[FSM_State].Next[1];
    buttonPressed = true;
    lastDebounceTime = millis();
}
```

```
if (buttonState == HIGH) {
    buttonPressed = false;
}
```

La fel în loop am realizat actualizarea ieșirii LED-ului și afișarea stării LED-ului în monitorul serial.

```
int output = FSM[FSM_State].Out;
digitalWrite(LED_PIN, output);
Serial.print("LED State: ");
if (output == LED_OFF_STATE) {
    Serial.println("OFF");
} else {
    Serial.println("ON");
}
```

La final facem o întârziere, intervalul căreia depinde de cel setat în automatul finit.

```
delay(FSM[FSM_State].Time);
```

6.2

Pentru a doua sarcină, am avut nevoie de mai multe LED-uri, 2 butoane și rezistoare. Sunt 2 butoane care controleaza schimbarea stării atunci când merge nord sau est. Afișarea este în figura 1.8.

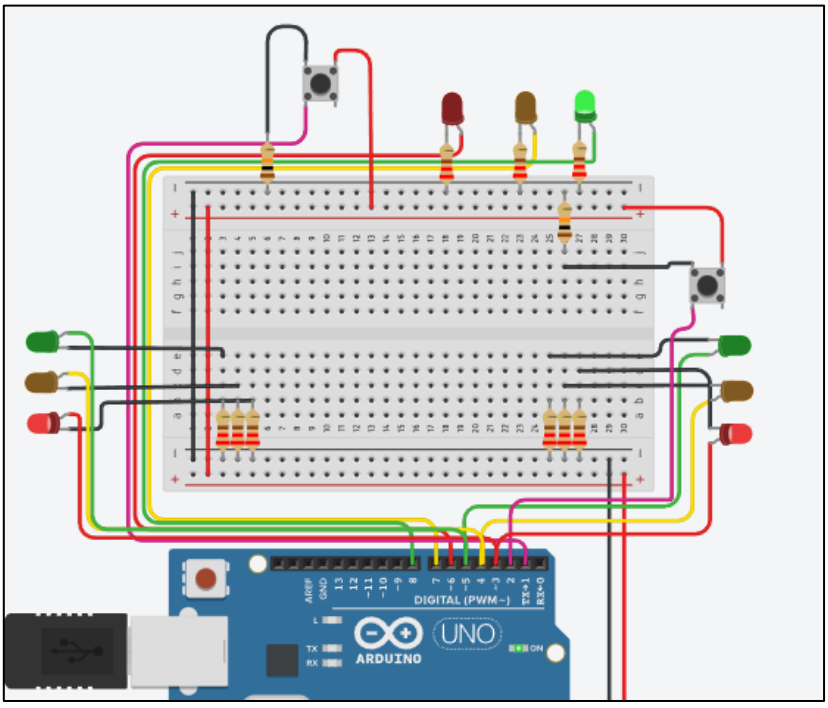


Figura 1.8 – Control motor de la potențiometru

Inițial am definit pinii și stările automatului.

```
#define NORTH_PIN 1
#define EAST_PIN 2
#define EAST_RED_PIN 3
#define EAST_YELLOW_PIN 4
#define EAST_GREEN_PIN 5
#define NORTH_RED_PIN 6
#define NORTH_YELLOW_PIN 7
#define NORTH_GREEN_PIN 8
#define goN 0
#define waitN 1
#define goE 2
#define waitE 3
```

Apoi am setat structura stării semaforului. Out este un model de 6 biți care setează starea LED-urilor (ON/OFF). Time este durata stării curente. Next reprezintă stările posibile în funcție de intrarea de la butoane (trafic de Nord sau Est).

```
struct State {
    byte Out;
    unsigned long Time;
    byte Next[4];
};
```

La fel setăm și tabelul de tranziții pentru automatul finit. Stările sunt atunci când merge Nord, merge Est, așteaptă Nord și așteaptă Est.

```
STyp FSM[4] = {
    {0b100001, 3000, {goN, waitN, goN, waitN}}, // Nord verde
    {0b100010, 1000, {goE, goE, goE, goE}},     // Nord galben
    {0b001100, 3000, {goE, goE, waitE, waitE}}, // Est verde
    {0b010100, 1000, {goN, goN, goN, goN}}      // Est galben
};
```

Tabelul de tranziție a stărilor este afișat în figura 1.9.

Nr	Nume	Out	Delay	In = 0	In = 1	In = 2	In = 3
0	goN	100 001	3000	goN	waitN	goN	waitN
1	waitN	100 010	1000	goE	goE	goE	goE
2	goE	001 100	3000	goE	goE	waitE	waitE
3	waitE	010 100	1000	goN	goN	goN	goN

Figura 1.9 Stările de tranziție a autoomatului finit

Apoi am creat o funcție pentru intrări și ieșiri. Inițial este verde pentru Nord. Funcția GetInput citește intrările de la butoane și le combină într-un singur număr binar care indică dacă există cerere de trecere din partea direcției Nord sau Est.

```
int FSM_State = goN;
int GetInput(void) {
    return digitalRead(NORTH_PIN) << 1 | digitalRead(EAST_PIN);
}
```

Starile butonului :

00 – 0 ; 01 – 1; 10 – 2; 11 - 3

Apoi am creat o funcție pentru intrări și ieșiri. Inițial este verde pentru Nord. Funcția GetInput citește intrările de la butoane și le combină într-un singur număr binar care indică dacă există cerere de trecere din partea direcției Nord sau Est.

```
int FSM_State = goN;
```

Următoarea funcție este pentru setarea ieșirilor, ledurile. SetOutput primește modelul binar de ieșire și setează starea fiecărui LED în funcție de acest model. Fiecare bit din out corespunde unui LED (1 pentru ON, 0 pentru OFF).

```
void SetOutput(byte out) {
    const int pins[] = {EAST_RED_PIN, EAST_YELLOW_PIN, EAST_GREEN_PIN, NORTH_RED_PIN, NORTH_YELLOW_PIN, NORTH_GREEN_PIN};
    for (int i = 0; i < 6; i++) {
        digitalWrite(pins[i], (out >> (5 - i)) & 1);
    }
}
```

Funcția setup() răspunde de setarea pinilor, setarea butoanelor ca intrări și LED-urile semaforului ca ieșiri.

```
void setup() {
    pinMode(NORTH_PIN, INPUT);
    pinMode(EAST_PIN, INPUT);
    int pins[] = {EAST_RED_PIN, EAST_YELLOW_PIN, EAST_GREEN_PIN, NORTH_RED_PIN, NORTH_YELLOW_PIN, NORTH_GREEN_PIN};
    for (int i = 0; i < 6; i++) pinMode(pins[i], OUTPUT);
}
```

În loop() setăm mai întâi ieșirile în funcție de starea curentă.

```
SetOutput(FSM[FSM_State].Out);
```

Setăm întârzierea.

```
delay(FSM[FSM_State].Time);
```

Citim intrările de la butoane.

```
int input = GetInput();
```

Facem tranziția către următoarea stare. Se actualizează starea curentă pe baza intrărilor de la butoane și a stării curente.

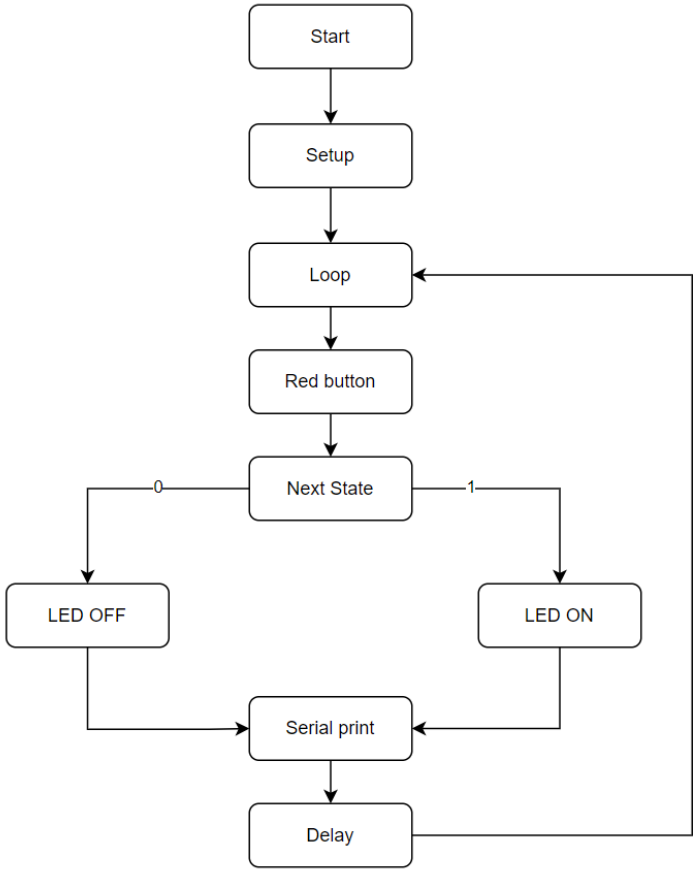
```
FSM_State = FSM[FSM_State].Next[input];
```

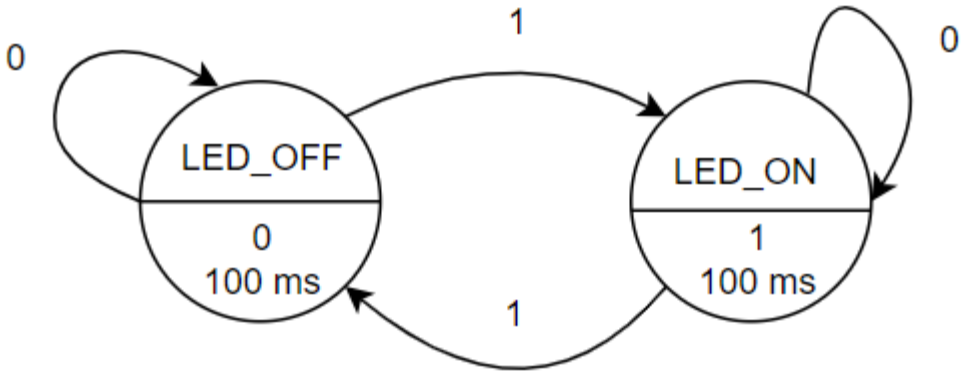
Concluzii

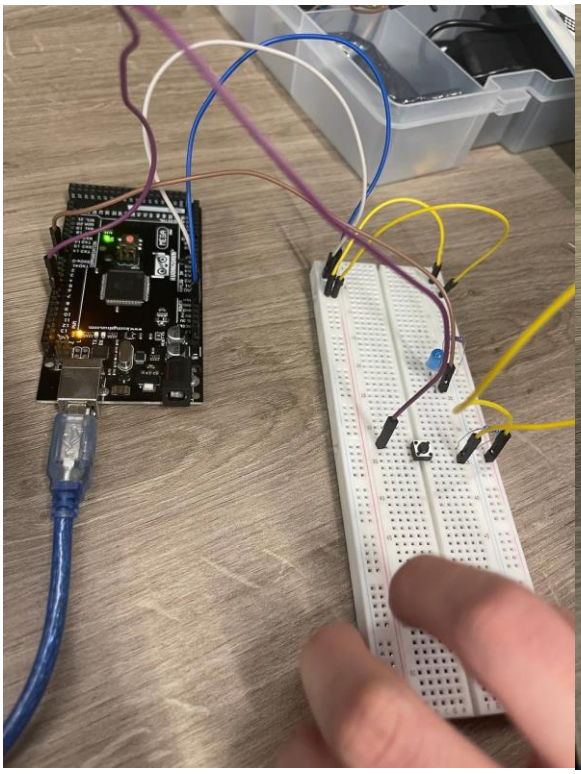
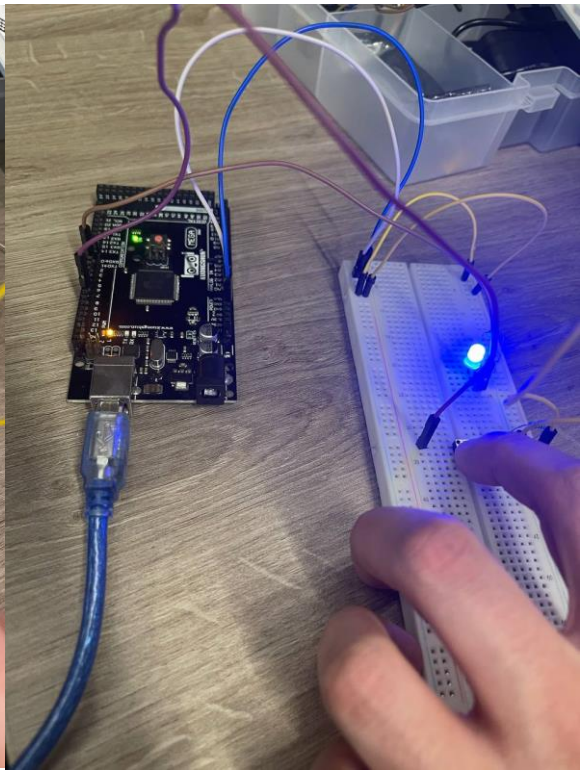
Lucrarea de laborator a presupus dezvoltarea unei aplicații IoT bazate pe Automate Finite, utilizând platforma Arduino UNO pentru controlul unui LED și simularea unui semafor. S-au proiectat două automate finite: unul pentru gestionarea unui LED prin intermediul unui buton și altul pentru semafor, demonstrând cum interacțiunea cu butoanele permite tranziții între stări. Aceste aplicații evidențiază avantajele microcontrolerelor în realizarea controlului simplu, fiind ideale pentru prototiparea rapidă în domeniul IoT. S-a utilizat o abordare structurată, de la configurarea pinilor și implementarea logicii FSM, până la testarea funcționalității în simulatoare și mediu real.

Anexă

6.1


Start
Setup
Loop
Red button
Next State
0
1
LED OFF
LED ON
Serial print
Delay


0
1
0
LED_OFF
0
100 ms
LED_ON
1
100 ms
1

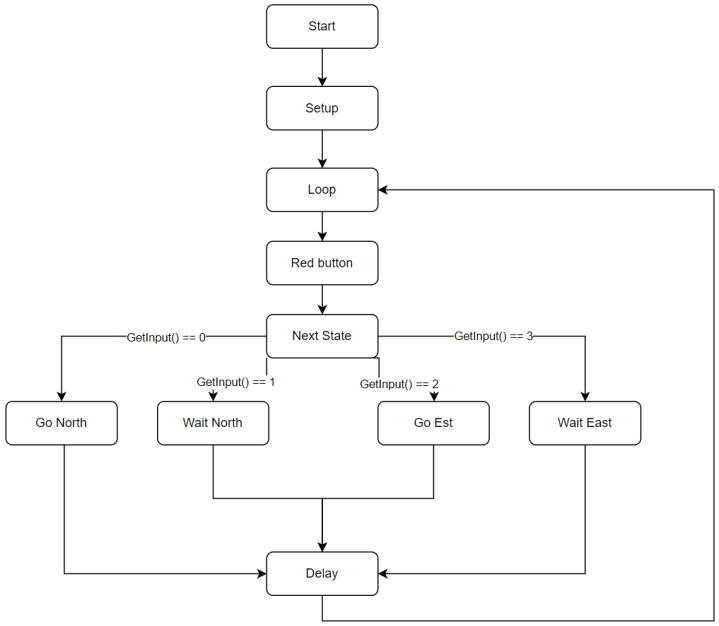



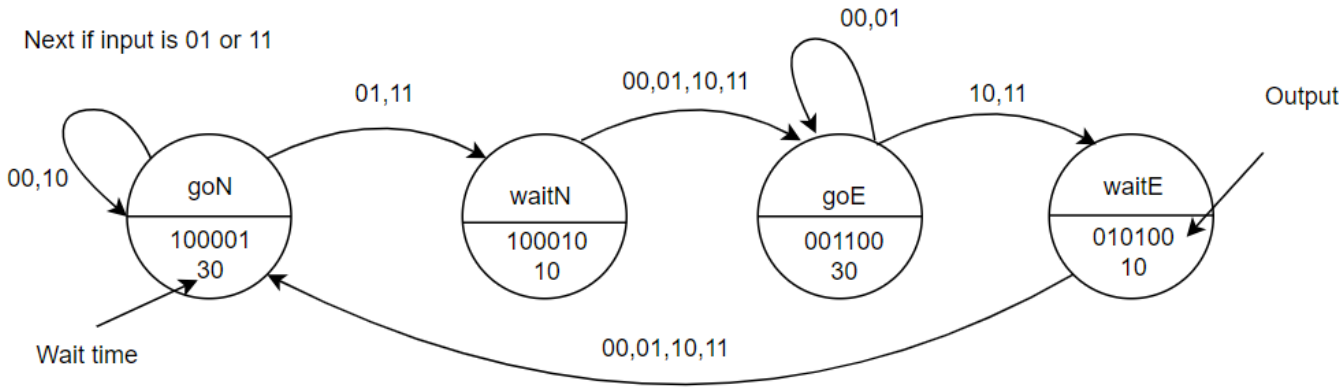
```
#define LED_PIN 2
#define BUTTON_PIN 8

#define LED_OFF_STATE 0
#define LED_ON_STATE 1

struct State {
    unsigned long Out;
    unsigned long Time;
    unsigned long Next[2];
};

typedef const struct State STyp;
STyp FSM[2] = {
    {0, 100, {LED_OFF_STATE, LED_ON_STATE}},
```

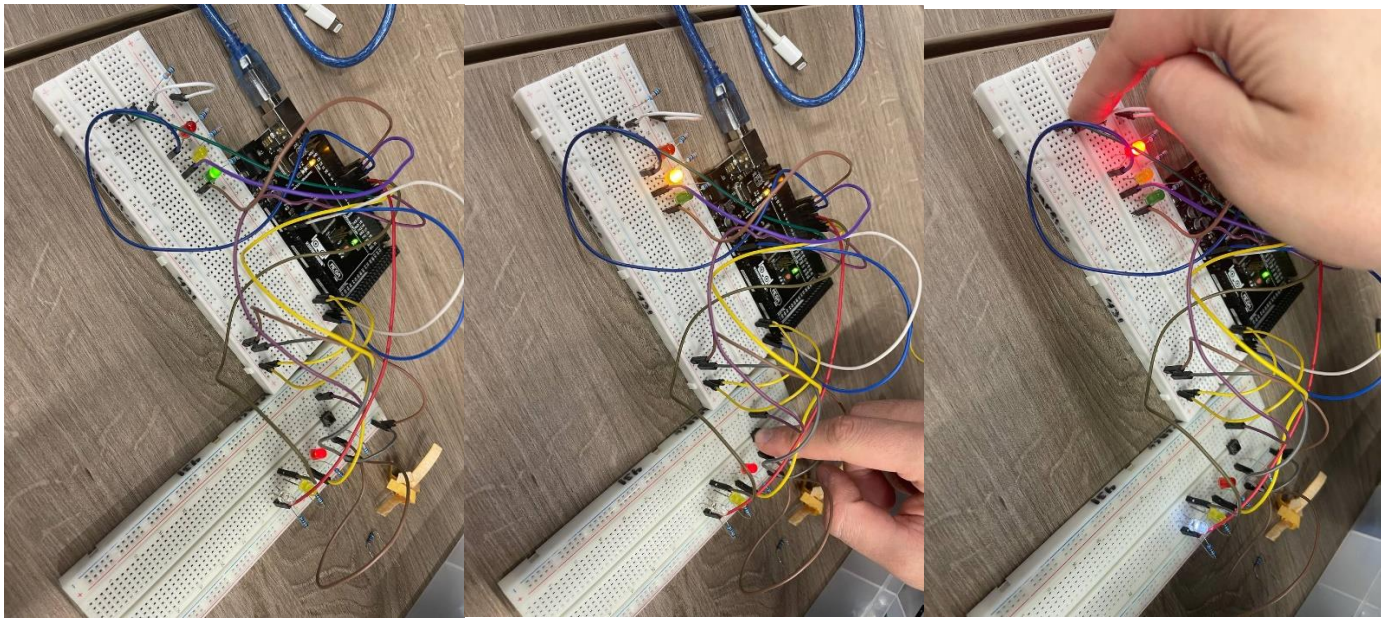
```
    {1, 100, {LED_ON_STATE, LED_OFF_STATE}}
};

int FSM_State = LED_OFF_STATE;
unsigned long lastDebounceTime = 0;
const unsigned long debounceDelay = 50;

bool buttonPressed = false;

void setup() {
    pinMode(BUTTON_PIN, INPUT_PULLUP);
    pinMode(LED_PIN, OUTPUT);
    Serial.begin(9600);
}

void loop() {
    int buttonState = digitalRead(BUTTON_PIN);
    if (buttonState == LOW && !buttonPressed && (millis() - lastDebounceTime > 
debounceDelay)) {
        FSM_State = FSM[FSM_State].Next[1];
        buttonPressed = true;
        lastDebounceTime = millis();
    }
       if (buttonState == HIGH) {
        buttonPressed = false;
    }
    int output = FSM[FSM_State].Out;
    digitalWrite(LED_PIN, output);
    Serial.print("LED State: ");
    if (output == LED_OFF_STATE) {
```

```
        Serial.println("OFF");
    } else {
        Serial.println("ON");
    }

    delay(FSM[FSM_State].Time);
}
```

6.2







```
#define NORTH_PIN 1

#define EAST_PIN 2

#define EAST_RED_PIN 3

#define EAST_YELLOW_PIN 4

#define EAST_GREEN_PIN 5

#define NORTH_RED_PIN 6

#define NORTH_YELLOW_PIN 7

#define NORTH_GREEN_PIN 8


#define goN 0

#define waitN 1

#define goE 2

#define waitE 3


struct State {

    byte Out;
```

```
    unsigned long Time;
    byte Next[4];
};

typedef const struct State STyp;

// Tabelul de tranzitii
STyp FSM[4] = {
    {0b100001, 3000, {goN, waitN, goN, waitN}}, // Go North
    {0b100010, 1000, {goE, goE, goE, goE}},     // Wait North
    {0b001100, 3000, {goE, goE, waitE, waitE}},  // Go East
    {0b010100, 1000, {goN, goN, goN, goN}}       // Wait East
};

int FSM_State = goN;
int GetInput(void) {
    return digitalRead(NORTH_PIN) << 1 | digitalRead(EAST_PIN);
}
void SetOutput(byte out) {
    const int pins[] = {EAST_RED_PIN, EAST_YELLOW_PIN, EAST_GREEN_PIN, NORTH_RED_PIN,
NORTH_YELLOW_PIN, NORTH_GREEN_PIN};
    for (int i = 0; i < 6; i++) {
        digitalWrite(pins[i], (out >> (5 - i)) & 1);
    }
}
```

```
void setup() {

    pinMode(NORTH_PIN, INPUT);

    pinMode(EAST_PIN, INPUT);

    int pins[] = {EAST_RED_PIN, EAST_YELLOW_PIN, EAST_GREEN_PIN, NORTH_RED_PIN, NORTH_YELLOW_PIN,
NORTH_GREEN_PIN};

    for (int i = 0; i < 6; i++) pinMode(pins[i], OUTPUT);

}

void loop() {

    SetOutput(FSM[FSM_State].Out);

    delay(FSM[FSM_State].Time);

    int input = GetInput()

    FSM_State = FSM[FSM_State].Next[input];

}
```