TECHNICAL UNIVERSITY OF MOLDOVA
FACULTY OF COMPUTERS, INFORMATICS
AND MICROELECTRONICS
DEPARTMENT OF SOFTWARE ENGINEERING AND
AUTOMATICS

Laboratory work nr. 6

User interaction - Automates Finite – Semafor(Trafic light)

A elaborat: **Brinza Cristian**
st. gr. FAF-212

A verificat: **Moraru Dumitru**
lect. univ

Chișinău 2024

THE TASK OF THE LABORATORY WORK:

To create an MCU-based application that will implement the Finite Automata as follows: Designing the Automatic Finite traffic light application

- use the serial interface for automata operation reports
- reuse as much as possible the solutions presented in the previous labs
- review the resources taught in the course clocking

Marking:

- 5 - simple device activation application
- +1 - for modular implementation of the project
- +1 - for the presentation and explanation of the Semaphore transition tables
- +1 - for the presentation and explanation of the Semafor Finite Automata diagrams
- +1 - for demonstrating evidence of physical implementation

NOTE: maximum time limit possible only when presenting physical performance!!

Penalties:

- 1 - penalty for NOT using STDIO
- 1 - penalty for each week late from the deadline
- 1 - penalty for non-compliance with the report format

PROGRESS OF THE WORK

1 Main functions/methods used to execute the task

Explication about this chapterIn this chapter I will explain the functionality of diferent parts of the executed task

In the Main Sketch File:

- *setup():*This function initializes the Arduino's GPIO pins for controlling the LEDs, setting them as OUTPUT. It also sets the initial state of the traffic lights by calling **SetOutput()**. This setup ensures that the traffic light system is ready to function correctly from the start.
- *loop():* Acting as the core of the Arduino sketch, this function continuously runs and handles the state transitions and blinking logic. It checks the elapsed time to determine when to transition to the next state and handles the blinking of the green lights during the yellow light states. This function is crucial for maintaining the correct sequence of the traffic light system.

In Device Control and Status Reporting:

- *SetOutput(unsigned long out):* Controls the output state of the LEDs based on the given bit pattern. It directly sets the HIGH or LOW state of each traffic light LED (red, yellow, green for both north and south). This method ensures that the traffic lights display the correct signal based on the current state of the finite state machine (FSM).

State Management:

- *FSM State Transitions:* The FSM is defined as an array of State structs, each containing the output pattern for the LEDs, the time duration for the state, and the next states. The FSM manages the sequence of the traffic lights, ensuring that they follow the correct timing and transition patterns for go, wait, and stop signals.

Blinking Logic:

- *Blinking Control:* The sketch handles the blinking of the green lights during the yellow light states (waitN and waitE). It uses the BLINK_INTERVAL to toggle the green lights on and off, providing a visual indication of the transition state.

Explanation of Chapters

This section outlines the implementation and functionality of the main sections of our code. Through the setup and loop functions, the sketch establishes a responsive system capable of managing a traffic light sequence for two directions (north and south). The integration of direct control methods, along with the FSM for state transitions, exemplifies a practical approach to managing traffic lights in a controlled environment.

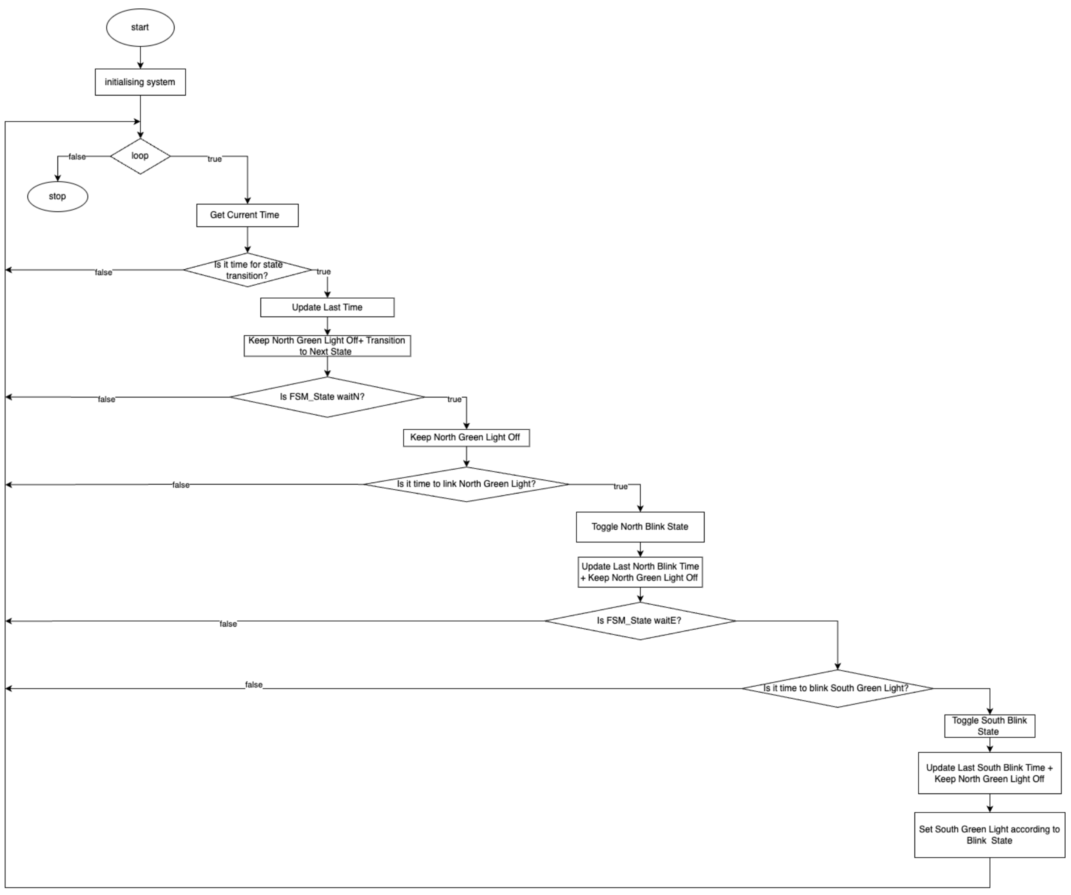
Device control functions such as SetOutput highlight the application's ability to affect physical changes through digital outputs, a core aspect of automation and control systems. The blinking control ensures that the user remains informed about the transition states, fostering an intuitive interaction between the system and its observers.

2 Block Diagram

The diagram is consisting of the following main blocks:

- **Main Program (sketch.ino):** The central point that initializes and controls other components (LEDs) and manages the program flow.
- **FSM Definition:** Represents the logic for state transitions, including output control, timing, and next state determination.
- **Blinking Control:** Manages the blinking of the green lights during the yellow light states, ensuring visual feedback for transition states.
- **Arduino Board:** The physical layer where the LEDs are connected, with GPIO pins used to control the traffic lights.

The Figure 1 depicted the UML program flow.



start
initialising system
loop
false
true
stop
Get Current Time
Is it time for state transition?
false
true
Update Last Time
Keep North Green Light Off+ Transition to Next State
Is FSM_State waitN?
false
true
Keep North Green Light Off
Is it time to link North Green Light?
false
true
Toggle North Blink State
Update Last North Blink Time + Keep North Green Light Off
Is FSM_State waitE?
false
Is it time to blink South Green Light?
false
Toggle South Blink State
Update Last South Blink Time + Keep North Green Light Off
Set South Green Light according to Blink State

Figure 1 Program schema.

3 Simulated or real assembled electrical schematic diagram :

The Figure 2 depicted the phisical board schema.

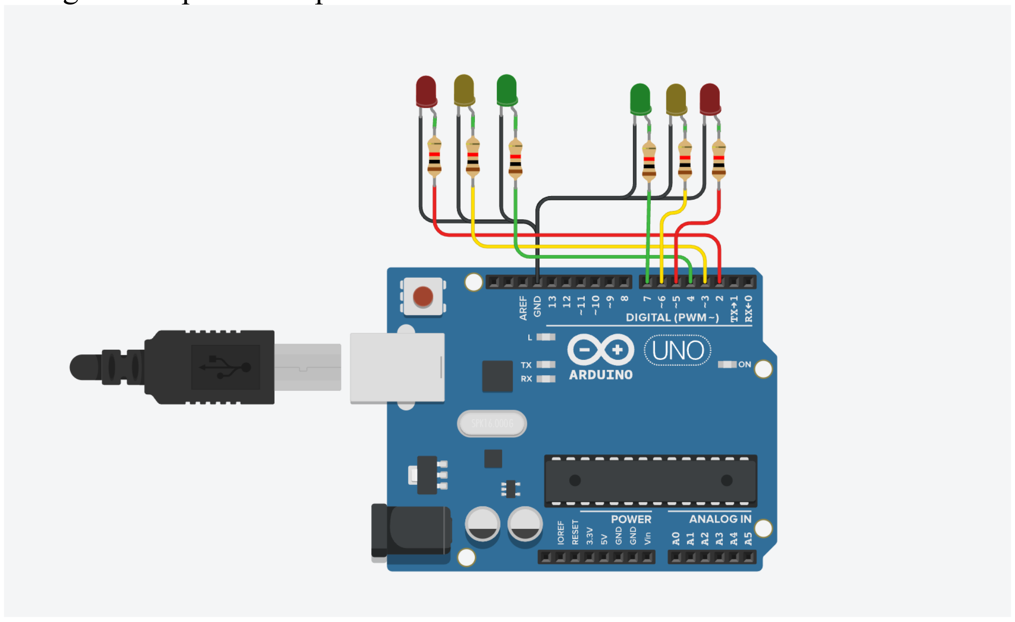
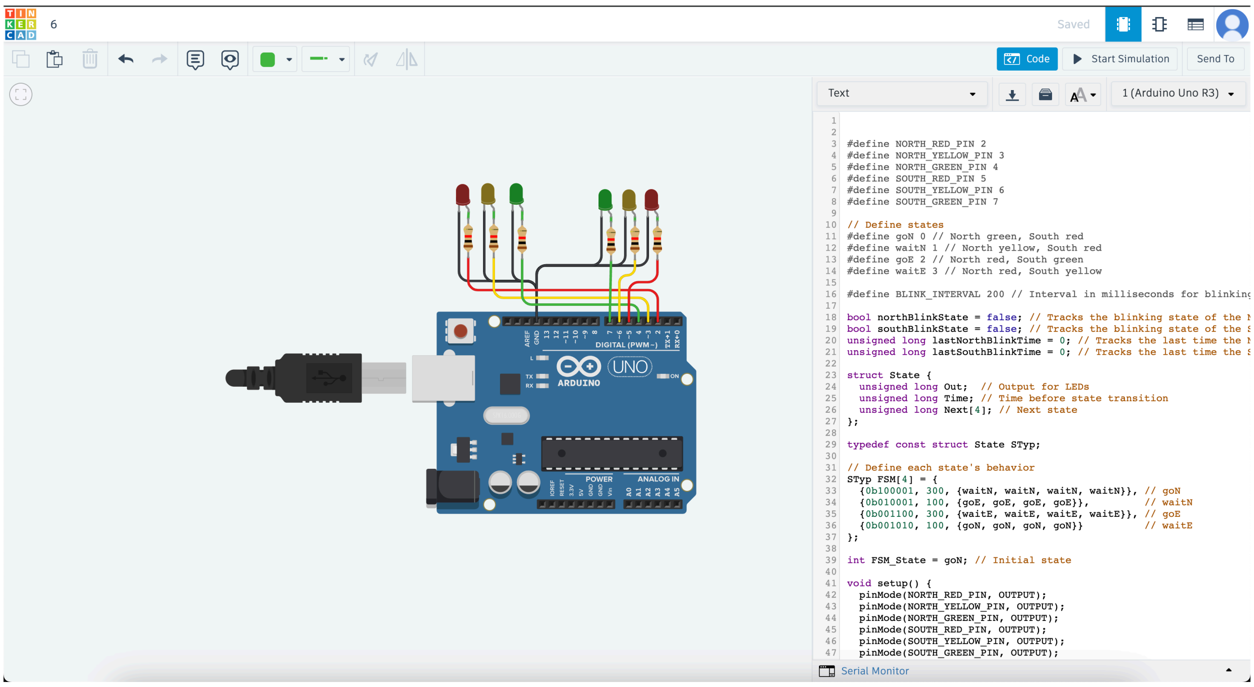


Figure 2 Phisical board schema.

4 Screenshots of the simulation execution:

The Figure 3 depicted a sceenshot of the TINKERCAD simulation :



```

#define NORTH_RED_PIN 2
#define NORTH_YELLOW_PIN 3
#define NORTH_GREEN_PIN 4
#define SOUTH_RED_PIN 5
#define SOUTH_YELLOW_PIN 6
#define SOUTH_GREEN_PIN 7

// Define states
#define goN 0 // North green, South red
#define waitN 1 // North yellow, South red
#define goE 2 // North red, South green
#define waitE 3 // North red, South yellow

#define BLINK_INTERVAL 200 // Interval in milliseconds for blinkin

bool northBlinkState = false; // Tracks the blinking state of the
bool southBlinkState = false; // Tracks the blinking state of the
unsigned long lastNorthBlinkTime = 0; // Tracks the last time the
unsigned long lastSouthBlinkTime = 0; // Tracks the last time the

struct State {
  unsigned long Out;  // Output for LEDs
  unsigned long Time; // Time before state transition
  unsigned long Next[4]; // Next state
};

typedef const struct State STyp;

// Define each state's behavior
STyp FSM[4] = {
  {0b100001, 300, {waitN, waitN, waitN, waitN}}, // goN
  {0b010001, 100, {goE, goE, goE, goE}},         // waitN
  {0b001100, 300, {waitE, waitE, waitE, waitE}}, // goE
  {0b001010, 100, {goN, goN, goN, goN}}          // waitE
};

int FSM_State = goN; // Initial state

void setup() {
  pinMode(NORTH_RED_PIN, OUTPUT);
  pinMode(NORTH_YELLOW_PIN, OUTPUT);
  pinMode(NORTH_GREEN_PIN, OUTPUT);
  pinMode(SOUTH_RED_PIN, OUTPUT);
  pinMode(SOUTH_YELLOW_PIN, OUTPUT);
  pinMode(SOUTH_GREEN_PIN, OUTPUT);
```

Figure 3 Screenshot of the simulation.

CONCLUSION

This project aimed to create an efficient and reliable traffic light control system using the Arduino platform. The implementation of a finite state machine (FSM) to manage the sequence of traffic lights for a two-direction intersection has proven to be both an enlightening and fulfilling experience. This journey from design to execution has provided valuable insights into the intricate interplay between hardware and software components.

Throughout the development process, we adhered to best practices in coding, focusing on readability, maintainability, and modularity. Clear variable names, structured functions, and a well-organized finite state machine ensured that the system was both robust and scalable. This careful planning and execution facilitated a seamless debugging process and enabled us to quickly adapt to any necessary changes or improvements.

One of the most significant accomplishments of this project was the effective handling of state transitions and the blinking logic for the traffic lights. This was achieved through precise timing and control mechanisms, demonstrating the capability of the Arduino platform to manage real-time tasks efficiently. The use of the SetOutput function to directly manipulate the GPIO pins showcased the power of digital outputs in controlling physical devices, a core aspect of automation systems.

This project was not just a technical achievement but a comprehensive learning experience that spanned the full spectrum of embedded system design. The successful implementation of this traffic light control system stands as a testament to our ability to apply theoretical concepts to practical applications. The knowledge and skills gained from this project lay a solid foundation for future endeavors in the field of automation and control systems. This experience has sparked a deeper interest in exploring the vast possibilities within the realm of embedded systems and has equipped us with the confidence to tackle more complex challenges in the future.

BIBLIOGRAPHY

1 EDUCBA: *Introduction to Embedded Systems.* Cursuri electronice online ©2020 [quote 10.02.2024] Available: https://www.educba.com/what-is-embedded-systems/

2 ARDUINO: *Arduino UNO.* The official website of Arduino modules, ©2020 [quote 10.02.2024]. Available: https://www.arduino.cc/.

3 TUTORIALSPOINT: *Embedded Systems Tutorial*. Comprehensive guide for beginners and professionals to learn Embedded System basics to advanced concepts, including microcontrollers, processors, and real-time operating systems. ©2023 [quote 10.02.2024] Available: https://www.tutorialspoint.com/embedded_systems/index.htm

4 GURU99: *Embedded Systems Tutorial: What is, History & Characteristics*. A detailed introduction to embedded systems, covering microcontrollers, microprocessors, and the architecture of embedded systems, as well as their applications and advantages. ©2023 [quote 10.02.2024] Available: https://www.guru99.com/embedded-systems-tutorial.html

5 JAVATPOINT: *Embedded Systems Tutorial.* Offers basic and advanced concepts of Embedded System, designed for beginners and professionals. ©2023 [quote 10.02.2024] Available: https://www.javatpoint.com/embedded-systems-tutorial

6 DEEPBLUE*: Embedded Systems Tutorials Introduction* | Embedded Systems Online Course. ©2023 [quote 10.02.2024] Available: https://deepbluembedded.com/embedded-systems-tutorials/

APPENDIX 1

Code of main.ino:

```
#define NORTH_RED_PIN 2
#define NORTH_YELLOW_PIN 3
#define NORTH_GREEN_PIN 4
#define SOUTH_RED_PIN 5
#define SOUTH_YELLOW_PIN 6
#define SOUTH_GREEN_PIN 7

// Define states
#define goN 0 // North green, South red
#define waitN 1 // North yellow, South red
#define goE 2 // North red, South green
#define waitE 3 // North red, South yellow

#define BLINK_INTERVAL 200 // Interval in milliseconds for blinking

bool northBlinkState = false; // Tracks the blinking state of the North green
light
bool southBlinkState = false; // Tracks the blinking state of the South green
light
unsigned long lastNorthBlinkTime = 0; // Tracks the last time the North green
light toggled
unsigned long lastSouthBlinkTime = 0; // Tracks the last time the South green
light toggled

struct State {
  unsigned long Out;  // Output for LEDs
  unsigned long Time; // Time before state transition
  unsigned long Next[4]; // Next state
};

typedef const struct State STyp;

// Define each state's behavior
STyp FSM[4] = {
  {0b100001, 300, {waitN, waitN, waitN, waitN}}, // goN
  {0b010001, 100, {goE, goE, goE, goE}},         // waitN
  {0b001100, 300, {waitE, waitE, waitE, waitE}}, // goE
  {0b001010, 100, {goN, goN, goN, goN}}          // waitE
};

int FSM_State = goN; // Initial state

void setup() {
  pinMode(NORTH_RED_PIN, OUTPUT);
  pinMode(NORTH_YELLOW_PIN, OUTPUT);
  pinMode(NORTH_GREEN_PIN, OUTPUT);
  pinMode(SOUTH_RED_PIN, OUTPUT);
  pinMode(SOUTH_YELLOW_PIN, OUTPUT);
  pinMode(SOUTH_GREEN_PIN, OUTPUT);

  SetOutput(FSM[FSM_State].Out); // Set the initial output
}

void loop() {
  static unsigned long lastTime = millis();
  unsigned long currentMillis = millis();

  // Handle state transition based on time
  if (currentMillis - lastTime >= FSM[FSM_State].Time * 10) {
    lastTime = currentMillis;
    FSM_State = FSM[FSM_State].Next[0];
    SetOutput(FSM[FSM_State].Out);
  }

  // Handle blinking of the North green light during North yellow state
  if (FSM_State == waitN) {
    if (currentMillis - lastNorthBlinkTime >= BLINK_INTERVAL) {
      lastNorthBlinkTime = currentMillis;
      northBlinkState = !northBlinkState;
      digitalWrite(NORTH_GREEN_PIN, northBlinkState ? HIGH : LOW); // Toggle the
North green light
    }
  } else {
    if (northBlinkState) {
      northBlinkState = false;
      digitalWrite(NORTH_GREEN_PIN, LOW);
    }
  }

  // Handle blinking of the South green light during South yellow state
  if (FSM_State == waitE) {
    if (currentMillis - lastSouthBlinkTime >= BLINK_INTERVAL) {
      lastSouthBlinkTime = currentMillis;
      southBlinkState = !southBlinkState;
      digitalWrite(SOUTH_GREEN_PIN, southBlinkState ? HIGH : LOW); // Toggle the
South green light
    }
  } else {
    if (southBlinkState) {
      southBlinkState = false;
      digitalWrite(SOUTH_GREEN_PIN, LOW);
    }
  }
}

void SetOutput(unsigned long out) {
  digitalWrite(NORTH_GREEN_PIN, (out & (1 << 0)) && !northBlinkState ? HIGH :
LOW);
  digitalWrite(NORTH_YELLOW_PIN, out & (1 << 1) ? HIGH : LOW);
  digitalWrite(NORTH_RED_PIN, out & (1 << 2) ? HIGH : LOW);
  digitalWrite(SOUTH_GREEN_PIN, (out & (1 << 3)) && !southBlinkState ? HIGH :
LOW);
  digitalWrite(SOUTH_YELLOW_PIN, out & (1 << 4) ? HIGH : LOW);
  digitalWrite(SOUTH_RED_PIN, out & (1 << 5) ? HIGH : LOW);
}
```