Report 7

Subject: Embedded Systems
Topic: Finite State Machines

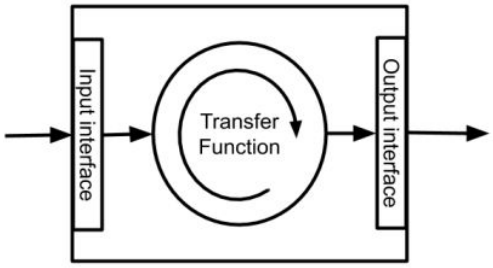
Student: Marusic Diana, FAF-171
Teacher: Bragarenco Andrei

Content

1. Domain
2. Component description
3. Implementation
4. Proof
5. Annex

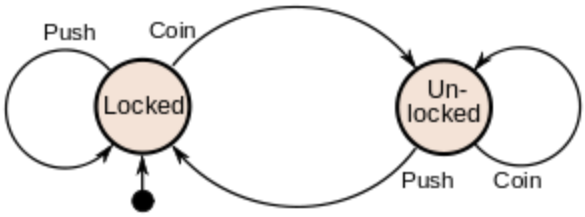
Domain

Finite state machines



A finite-state machine or finite-state automaton, finite automaton, or simply a state machine, is a mathematical model of computation. It is an abstract machine that can be in exactly one of a finite number of states at any given time. It can be implemented with hardware or software and can be used to simulate sequential logic and some computer programs. Finite state automata generate regular languages. Finite state machines can be used to model problems in many fields including mathematics, artificial intelligence, games, and linguistics.

Example of Finite State Machine (FSM):



A system where particular inputs cause particular changes in state can be represented using finite state machines. This example describes the various states of a turnstile. Inserting a coin into a turnstile will unlock it, and after the turnstile has been pushed, it locks again. Inserting a coin into an unlocked turnstile, or pushing against a locked turnstile will not change its state.

Types of FSM:

There are two types of finite state machines (FSMs):

- **deterministic finite state machines**, often called deterministic finite automata,
- **non-deterministic finite state machines**, often called non-deterministic finite automata

Deterministic finite state machines

A deterministic finite automaton (DFA) is described by a five-element tuple

$(Q, \Sigma, \delta, q_0, F)$

Q = a finite set of states

Σ = a finite, nonempty input alphabet

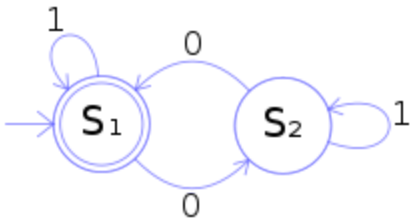
δ = a series of transition functions

q_0 = the starting state

F = the set of accepting states

There must be exactly one transition function for every input symbol in Σ from each state.

DFAs can be represented by diagrams of this form:



Non-deterministic finite state machines

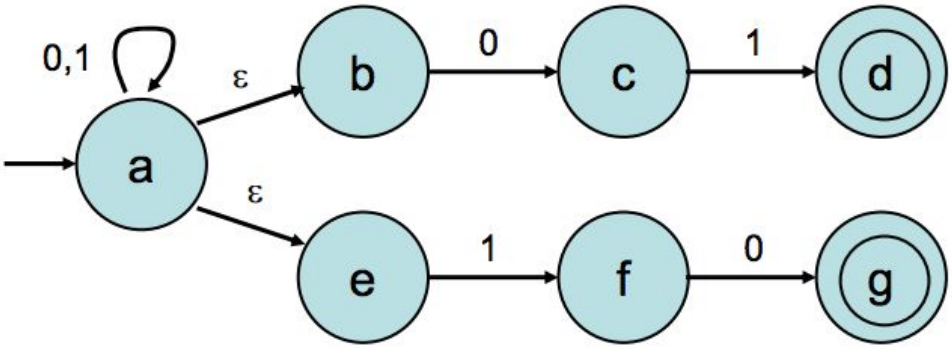
Similar to a DFA, a nondeterministic finite automaton (NDFA or NFA) is described by a five-element tuple

$(Q, \Sigma, \delta, q_0, F)$

Unlike DFAs, NDFAs are *not* required to have transition functions for every symbol in Σ, and there can be multiple transition functions in the same state for the same symbol. Additionally, NDFAs can use null transitions, which are indicated by ϵ. Null transitions allow the machine to jump from one state to another without having to read a symbol.

An NDFA accepts a string x if there exists a path that is compatible with that string that ends in an accept state.

NDFAs can be represented by diagrams of this form:



State

A **state** is a description of the status of a system that is waiting to execute a transition.

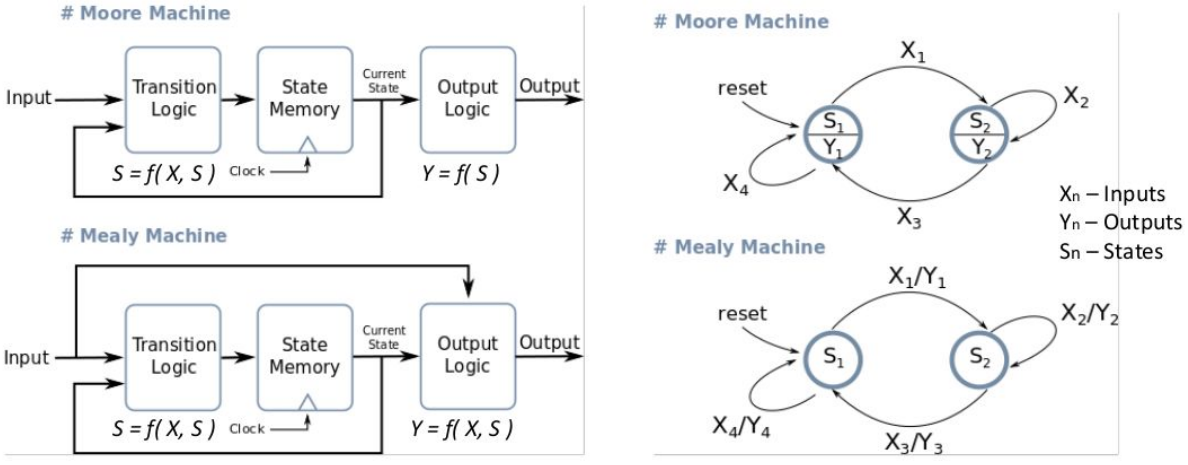
Transition

A transition defines how a machine would react to the event, by exiting one state and entering another state.

Action

An action represents the description of an activity to be executed at a certain time.

Finite State Machines - Moore vs Mealy



http://calangorobo.blogspot.com/2018/

Evaluation of FInite State Automata - Moore vs Mealy

Moore Automata

NextState = f(Input, CurrentState)

Output = g(CurrentState)

TaskMooreFSM() {

1. Evaluate output, that depends only on **current state**
2. Save defined period of state
3. Collect input
4. Evaluate next state that depends on **Input** and **current state**

}

Mealy Automata

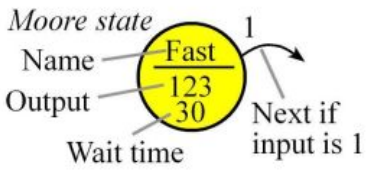
NextState = f(Input, CurrentState)

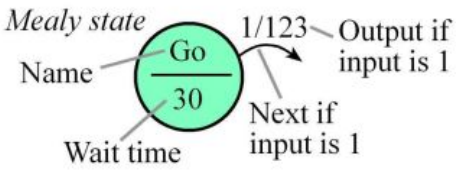
Output = h(Input, CurrentState)

TaskMealyFSM() {

1. Save defined period of state
2. Collect input
3. Evaluate output, that depends only on **current state**
4. Evaluate next state that depends on **Input** and **current state**

}

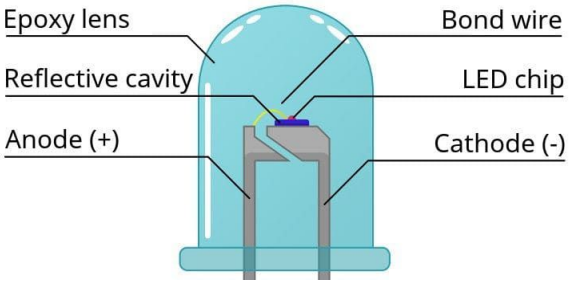




https://courses.edx.org/courses/course-v1:UTAustinX+UT.6.03x+1T2016

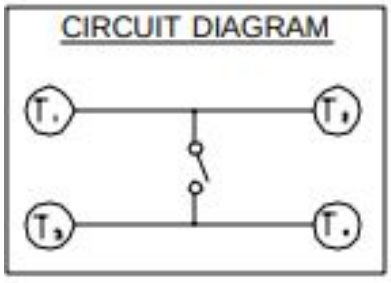
Component description

LED (Light-Emitting Diode)



LED is a semiconductor light source that emits light when current flows through it. Electrons in the semiconductor recombine with electron holes, releasing energy in the form of photons. The color of the light (corresponding to the energy of the photons) is determined by the energy required for electrons to cross the band gap of the semiconductor. White light is obtained by using multiple semiconductors or a layer of light-emitting phosphor on the semiconductor device.
There are some advantages of LEDs over incandescent light bulbs. Unlike incandescent bulbs, LEDs don't have filaments that burn out, they use less electricity, and they don't get very hot. LEDs last just as long as a standard transistor. The lifespan of an LED surpasses the short life of an incandescent bulb by thousands of hours.

Button (push button)



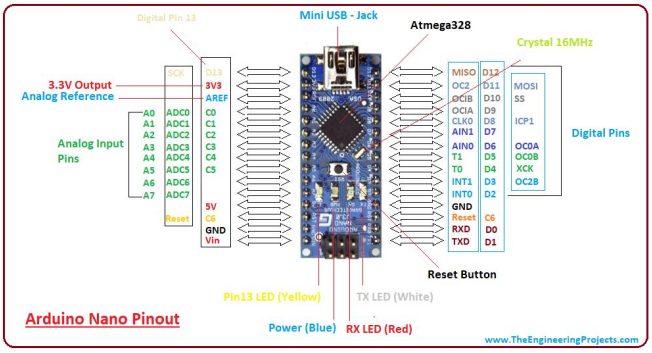
A push-button or simply button is a simple switch mechanism to control some aspect of a machine or a process. Buttons are typically made out of hard material, usually plastic or metal. These buttons usually contain a spring to return to their un-pushed state

Proximity sensor FC-51

This sensor sends infrared waves and if there's something they reflect back and inform us that something is near, and it depends on how you set the potentiometer to adjust the detection range.

Infrared Obstacle Avoidance Proximity Sensors Module has built in IR transmitter and IR receiver that sends out IR energy and looks for reflected IR energy to detect presence of any obstacle in front of the sensor module. The module has on board potentiometer that lets user adjust detection range. The sensor has very good and stable response even in ambient light or in complete darkness.

Arduino Nano

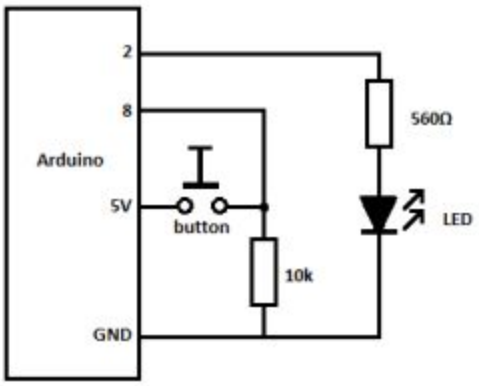


The **Arduino Nano** is a small, complete, and breadboard-friendly board based on the ATmega328P (Arduino Nano 3. x). It has more or less the same functionality of the Arduino Duemilanove, but in a different package. It lacks only a DC power jack, and works with a Mini-B USB cable instead of a standard one. It has 14 digital pins (D0-D13) and 8 analog input pins (D0-D7). Its small dimensions make it a good board for compact projects.

Implementation

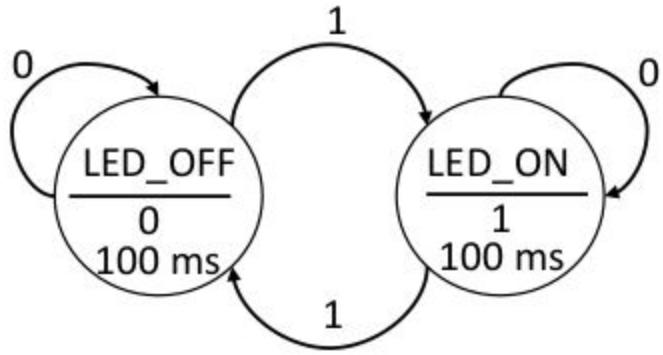
Button-LED circuit

For implementing the FSM project with button - LED, the following circuit was used with Arduino nano, button, white LED and resistors.



FSM button-LED

The finite state machine for the Button-LED is presented below. There are 2 states - LED_OFF and LED_ON with the transitions having values of 0 or 1. Each transition represents the input value from the button. When the LED is OFF, if the input from the button is 0, it will remain in the same state, but if the button is 1, it will change to the next state - LED_ONN. When the LED is on, while the button is 0, it will maintain the same state, and if the button is 1, it will change to the next state - LED_OFF.



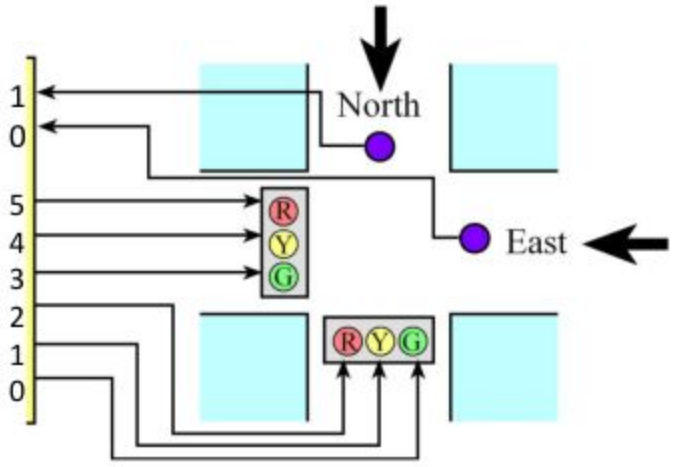
Transition table button-LED

Num	Name	Out	Delay	In=0	In=1
0	**LED_OFF**	**0**	**500ms**	**LED_OFF**	**LED_ON**
1	**LED_ON**	**1**	**500ms**	**LED_ON**	**LED_OFF**

The transition table for the button-LED application is presented above. It describes the automata presented above. It shows that when the LED is off (the state is LED_OFF), and the In from the button is 0, it will remain off, when it is 1, it will be ON (transition to state LED_ON). The same for the second transition - LED_ON. If the input is 0, it will remain in the state LED_ON and if it is 1, it will change to LED_OFF.

Semaphore

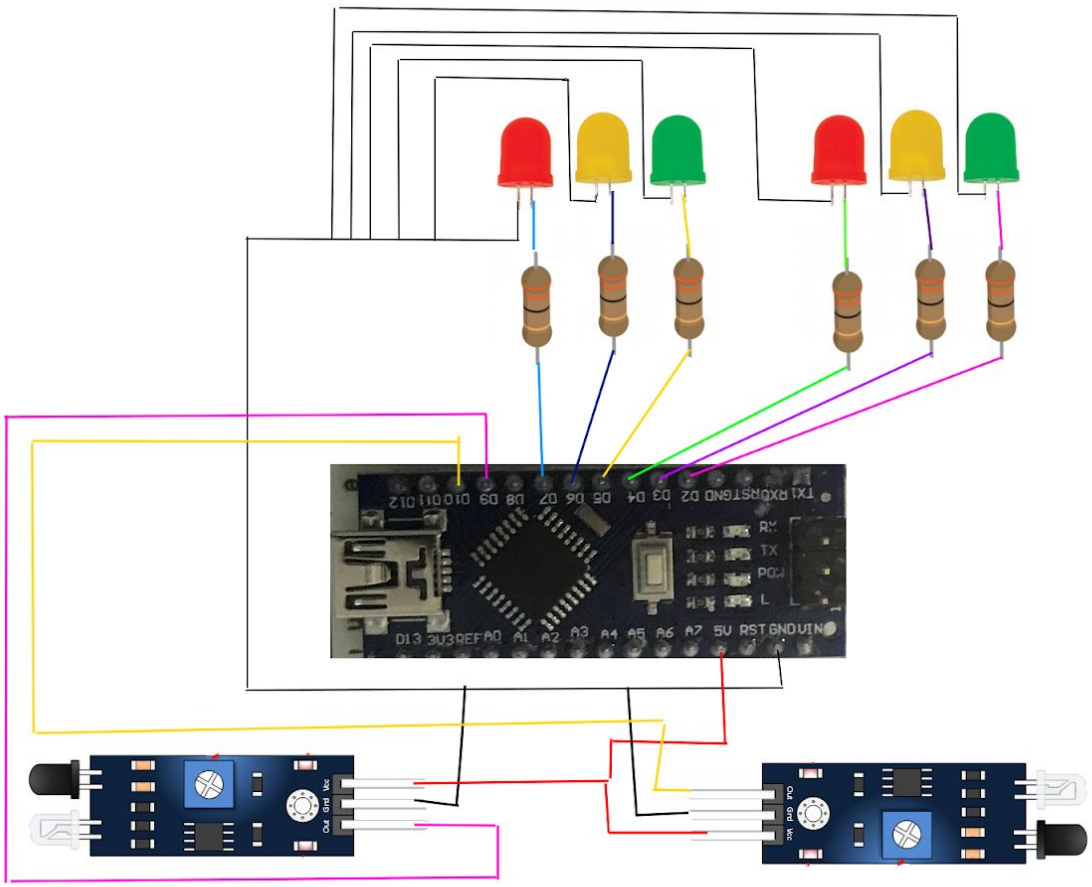
The general scheme for the semafore FSM application is presented below. The task is to use 2 semaphores, with 2 buttons (sensors) to direct the traffic from North and East directions.



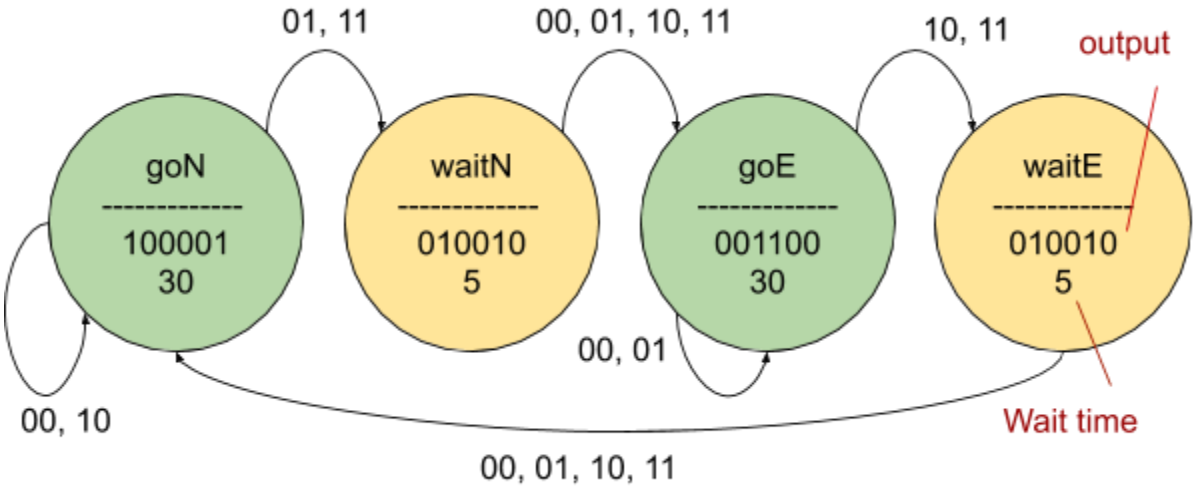
Circuit

For each semaphore, it will be used 3 LEDs- red, yellow, green. For detecting traffic coming from each direction, proximity Infra-red sensors will be used. Each LED is connected with a resistor to one digital pin of Arduino nano and the FC-51 proximity sensors are connected to the pins D9 and D10 of the Arduino.

The electronic circuit is presented in the scheme below:



FSM Semaphore



Transition table Semaphore

Num	Name	Out	Delay	In=0	In=1	In=2	In=3
0	goN	100001	3000	goN	waitN	goN	waitN
1	waitN	010010	500	goE	goE	goE	goE
2	goE	001100	3000	goE	goE	waitE	waitE
3	waitE	010010	500	goN	goN	goN	goN

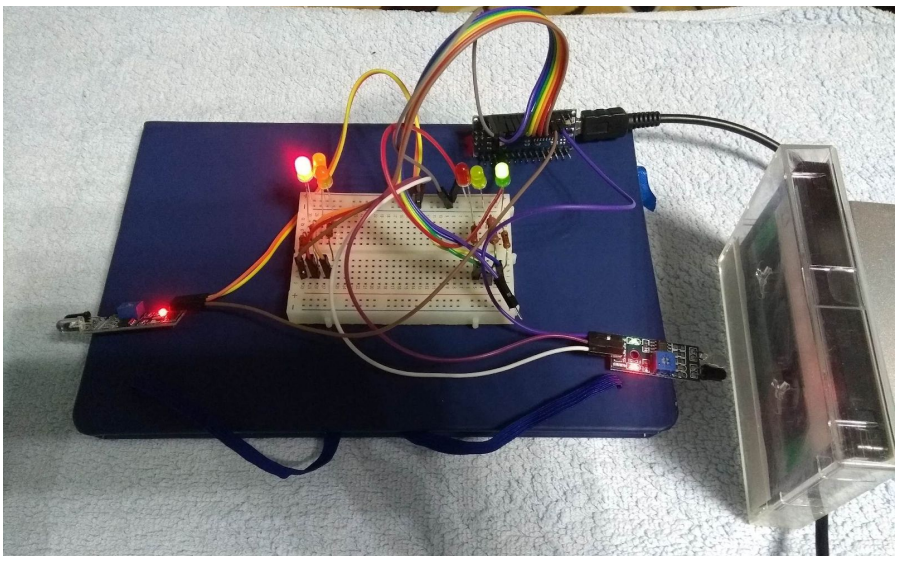
The Finite State Machine representation and the transition table are presented above. For the semaphore FSM there are 4 states. There are 2 states - goN and goE, that are marked green, that represent the states when traffic from respective directions - North and East are allowed to pass (the semaphore in the respective direction is green and from the other direction is red). These 2 states have the duration of 3000 ms (3 seconds). The other 2 states - waitN (1) and waitE (3) are marked yellow in the FSM scheme and they represent the situation when the semaphore is yellow.

One possible scenario according to this table and scheme is:

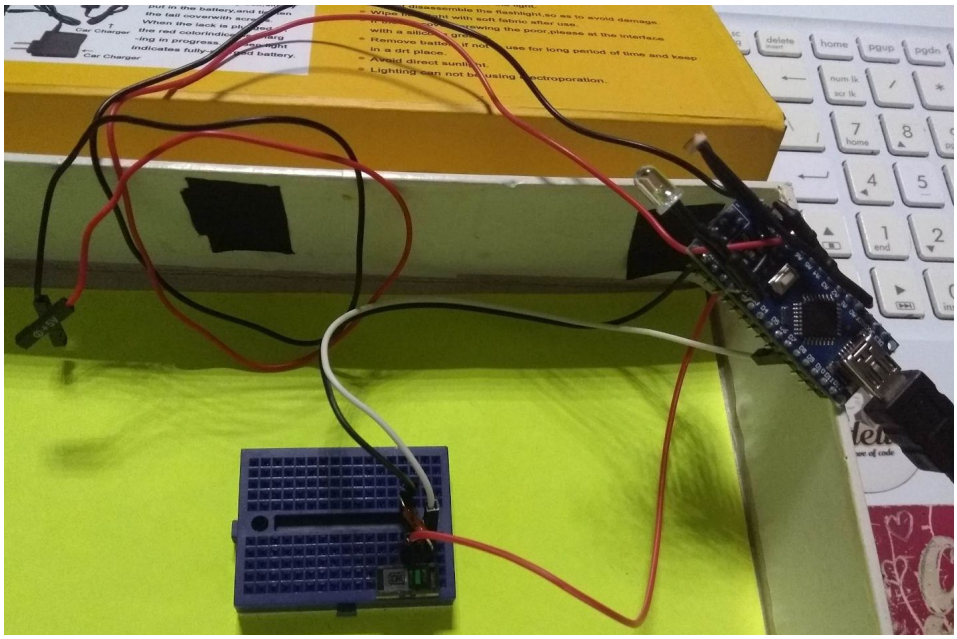
1. The state is 0. goN. The semaphore is green in the North direction and Red in the East direction.
2. There are no cars coming in the North direction. One car approaches the East direction. The proximity sensor changes from 0 to 1 in the East direction and from 1 to 0 in the North direction.
3. Transition to state 1. waitN - both semaphores turn yellow.
4. Transition to state 2.goE. The semaphore in the East direction turns green and semaphore in the North direction turns red.
5. Car passes from the East direction. There are no cars in the East direction.
6. A car approaches in the North direction. Transition to state 3. waitE. The semaphores from both directions turn yellow.
7. Transition to state 0.goN. The semaphore from the North direction turns green and the semaphore from the East direction turns red.

Proof

Semaphore proof link:
https://www.youtube.com/watch?v=zEQ6Ekoo1u8



Button-LED proof link:
https://www.youtube.com/watch?v=UMnaCX7RYN0



Annex 1 - source code for Button-LED

```
------------
Main file
------------
#include "led.h"
#include "button.h"

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

typedef const struct State STyp;

STyp FSM[2] = {
        {0,50, {LED_OFF_STATE, LED_ON_STATE}},
        {1,50, {LED_ON_STATE, LED_OFF_STATE}}
};

int FSM_State = LED_OFF_STATE;


void InitialStateInit() {
        FSM_State = LED_OFF_STATE;
}


void setup() {
        InitialStateInit();
  LEDInit();
  ButtonInit();
}


void loop() {
        int output = FSM[FSM_State].Out;
```

```
	LEDSetState(output);

	delay(FSM[FSM_State].Time *10);
	int input = ReadButton();

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

---
button.h

```
#ifndef _BUTTON_H_
#define _BUTTON_H_

#include <Arduino.h>

#define BUTTON_PIN 8

void ButtonInit();
int ReadButton();

#endif
```

---
button.cpp

```
#include "button.h"


void ButtonInit() {
	pinMode(BUTTON_PIN, INPUT);
}


int ReadButton() {
 return digitalRead(BUTTON_PIN);
}
```

---
led.h

```
#ifndef _LED_H_
#define _LED_H_


#define LED_OFF_STATE 0
#define LED_ON_STATE 1

#define LED_PIN 2

#include <Arduino.h>

void LEDInit();
void LEDSetState(int);

#endif
```

---
led.cpp

```
#include "led.h"


void LEDInit() {
	pinMode(LED_PIN, OUTPUT);
}


void LEDSetState(int state) {
	digitalWrite(LED_PIN, state);
}
```

Annex 2 - source code for Semaphore application

```
#define NORTH_PIN        9
#define EAST_PIN        10

#define EAST_RED_PIN     2
#define EAST_YELLOW_PIN  3
#define EAST_GREEN_PIN   4
#define NORTH_RED_PIN    5
#define NORTH_YELLOW_PIN 6
#define NORTH_GREEN_PIN  7

#define goN   0  // 0b00
#define waitN 1  // 0b01
#define goE   2  // 0b00
#define waitE 3  // 0b01



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


typedef const struct State STyp;

STyp FSM[6] = {
        {0b100001, 3000, {goN, waitN, goN,  waitN}},
        {0b010010, 500,  {goE, goE,   goE,  goE}},
 {0b001100, 3000, {goE, goE,   waitE, waitE}},
        {0b010010, 500,  {goN, goN,   goN,  goN}},
};

int FSM_State = goN;

void setup() {
        pinMode(NORTH_PIN, INPUT);
        pinMode(EAST_PIN, INPUT);
```

```
	pinMode(EAST_RED_PIN, OUTPUT);
	pinMode(EAST_YELLOW_PIN, OUTPUT);
	pinMode(EAST_GREEN_PIN, OUTPUT);

	pinMode(NORTH_RED_PIN, OUTPUT);
	pinMode(NORTH_YELLOW_PIN, OUTPUT);
	pinMode(NORTH_GREEN_PIN, OUTPUT);

	FSM_State = goN;
}



int GetInput(void) {
	int northSensor = digitalRead(NORTH_PIN);
	int eastSensor = digitalRead(EAST_PIN);
	if(northSensor && eastSensor) {
		return 0b11;
	}
	//else
	if(northSensor) {
		return 0b10;
	}
	// else
	if(eastSensor) {
		return 0b01;
	}
	// else
	return 0b00;
}

void SetOutput(int out) {
	int ledState;

	ledState = (out & (1 << 5)) ? HIGH : LOW;
	digitalWrite(EAST_RED_PIN, ledState);

	ledState = (out & (1 << 4)) ? HIGH : LOW;
	digitalWrite(EAST_YELLOW_PIN, ledState);

	ledState = (out & (1 << 3)) ? HIGH : LOW;
	digitalWrite(EAST_GREEN_PIN, ledState);
```

```
	ledState = (out & (1 << 2)) ? HIGH : LOW;
	digitalWrite(NORTH_RED_PIN, ledState);

	ledState = (out & (1 << 1)) ? HIGH : LOW;
	digitalWrite(NORTH_YELLOW_PIN, ledState);

	ledState = (out & (1 << 0)) ? HIGH : LOW;
	digitalWrite(NORTH_GREEN_PIN, ledState);
}

void loop() {
	// 1. Output based on current state
	int output = FSM[FSM_State].Out;
	SetOutput(output);

	// 2. Wait for time relevant to state
	delay(FSM[FSM_State].Time * 2);

	// 3. Get input
	int input = GetInput();

	// 4. Change state based on input and current state
	FSM_State = FSM[FSM_State].Next[input];

}
```