Instituto Superior Técnico

Applications and Computation for the Internet of Things

3rd Lab work – Project: Control of Traffic Lights at a Roundabout

Group:		
Student 1:		
Student 2:		
Student 3:		

Goal:

The goal of this project is to implement a modular and automatic traffic lights system for a roundabout. The overall system must have some degree of fault tolerance and fail-safe behaviour.

In the laboratory, traffic lights will be replaced by LEDs, otherwise the system to be implemented exhibits a realistic behaviour. All traffic lights are connected to a central controller over a wireless link based on the Enhanced ShockBurst (ESB) protocol.

Description:

Build an embedded system to simultaneously control a traffic light system of 4 accesses of the roundabout.

Each access must support the following movements of vehicles:

- For vehicles entering the roundabout: turn right
- For vehicles in the roundabout:
    - Continue in the roundabout;
    - Get off the roundabout turning right.

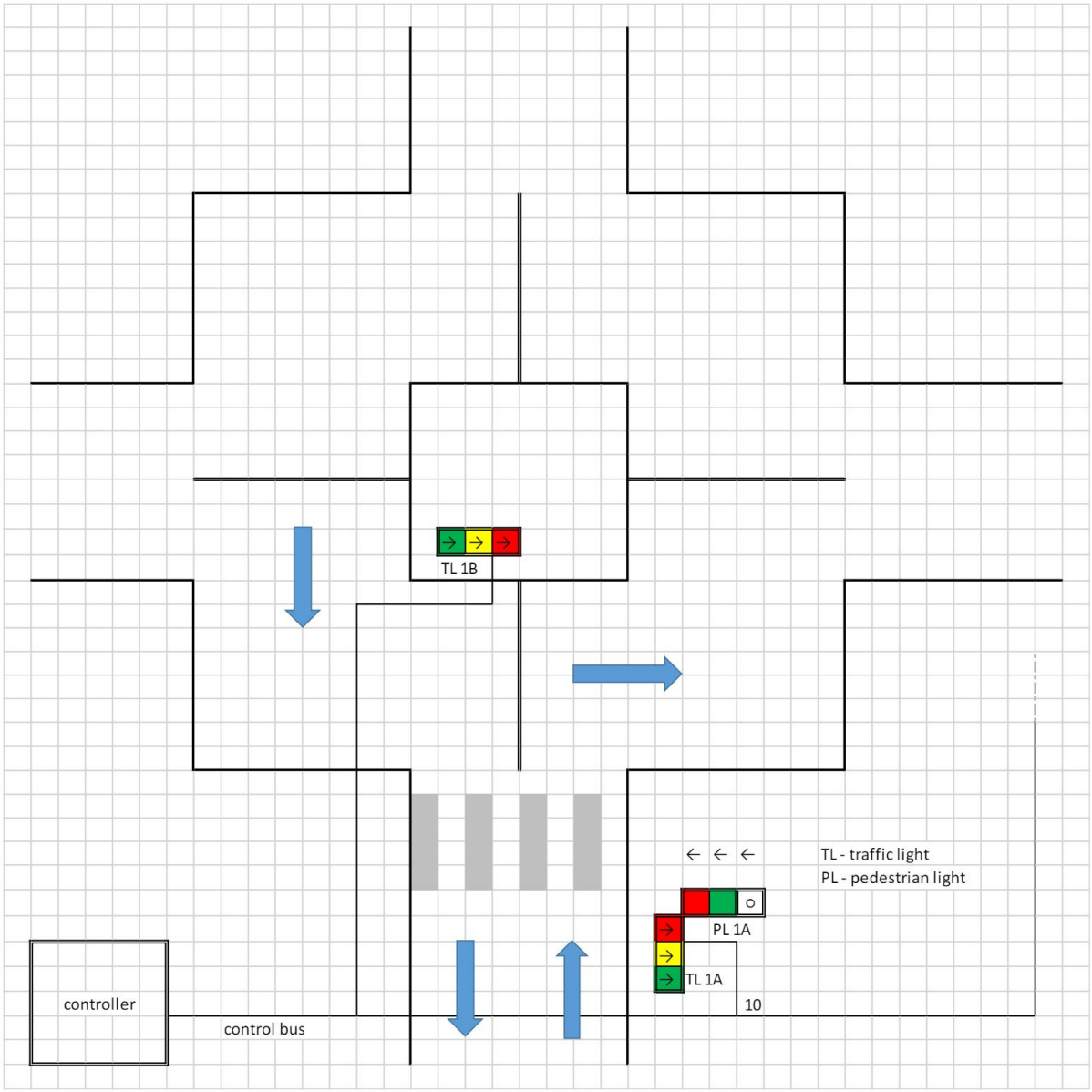
Each access – k – has two traffic lights (TL kA, TL kB):

- TL kA controls accesses of vehicles that want to enter the roundabout and has lights and a press button for pedestrians to signal their intent to cross the street

(PL kA). To simplify the assembly of the circuit only one set of pedestrian lights and button will be implemented on one side of the street.

- TL kB controls the flow of the vehicles inside the roundabout, preventing the passage of vehicles once traffic in the nearby street is allowed in the roundabout. Only vehicles turning right, to get off the roundabout, are allowed to proceed.

All traffic lights must be connected to a central controller which orchestrates all traffic lights. The functionalities of the controller and the traffic lights must be implemented according to a specified protocol so that any traffic light can work with any implementation of the controller, meaning, the implementations must be interoperable between every group of students.



References:

1. https://www.arduino.cc/en/Reference/digitalWrite
2. https://www.arduino.cc/en/Reference/AnalogRead
3. https://www.arduino.cc/en/Reference/Serial
4. https://www.arduino.cc/en/Tutorial/Calibration
5. https://www.arduino.cc/en/Tutorial/PWM
6. https://www.arduino.cc/en/Reference/Delay (or https://www.arduino.cc/reference/en/language/functions/time/millis/)

Recommendations:

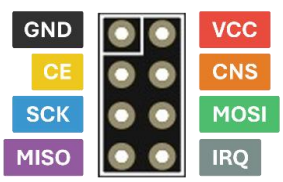
For a safe working environment and to prevent any damage to the hardware, please adhere to the following recommendations. As you progress with your work, check off each box to ensure all safety measures are followed.

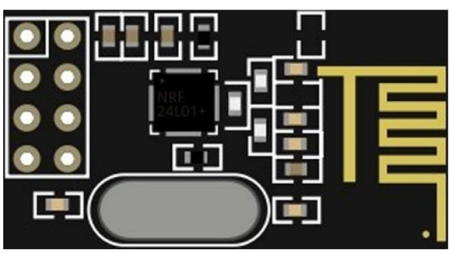
Always ensure that the circuit is disconnected from the power source (either the power supply or the PC) when you are working on it.	
Before connecting the circuit to the power source and turning it on, please consult with the teacher or the person in charge of the laboratory.	
Verify that all components of the circuit (such as resistors, capacitors) are properly connected to prevent short circuits or hardware damage.	

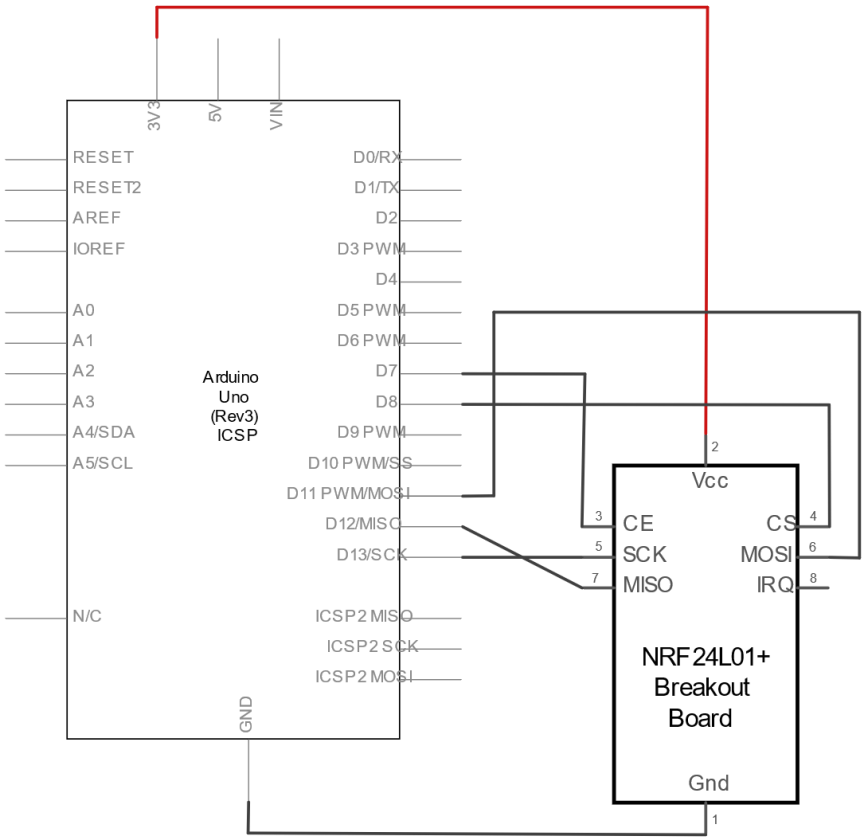
Message Transmission:

To put the system to work it is necessary to implement the communication link connecting two controllers. For this we will use a custom library built on top of the radio frequency modules given on the Arduino Starter Kit (nRF24L01). These modules will allow communication between every Arduino on your classroom or at home.

The nRF24L01 modules need to be connected to your Arduino UNO boards in the following way:







On campus your two controllers will need to be connected to the local network, more precisely, need to be connected to the controller pre-installed in your laboratory classroom. The code below is a template of all the required libraries and functions that need to be initialised to allow a correct behaviour of your devices:

defaultAtCampus.ino

```ino
#include <SPI.h>
#include <RF24.h>
#include <RF24Network.h>
#include "CampusStudentNode.h"

uint16_t sensorNode = 01; // change to your classroom sensorNode
char this_node_name[NAME_LENGTH] = "G01I01"; // change to your group num
char other_node_name[NAME_LENGTH] = "G01I02"; // change to your group num
int channel = 90;

CampusStudentNode studentNode(sensorNode, name, channel);

void setup()
{
  Serial.begin(115200);
  while (!Serial)
  {
    // some boards need this because of native USB capability
  }
  studentNode.init();
  // put your setup code here, to run once
}

void loop()
{
  studentNode.performEssentialOperations();
  // put your main code here, to run repeatedly
}
```

At home you can continue your work but in a simplified manner. Since you are outside the campus you can not reach the local network; you need to create a network of you own. In this case you should use the template below for your work.

Keep in mind that the library you will be using is now *HomeStudentNode*, instead of *CampusStudentNode*. Be aware that the initialisation of the HomeStudentNode do not receive the *SensorNode* (installed on campus) but your node ID (*this_node*).

defaultAtHome.ino

```
#include <SPI.h>
#include <RF24.h>
#include <RF24Network.h>
#include "HomeStudentNode.h"

uint16_t this_node = 00;
uint16_t other_node = 01;
char name[NAME_LENGTH] = "G01I01";
int channel = 90;

HomeStudentNode studentNode(this_node, name, channel);

void setup()
{
  Serial.begin(115200);
  while (!Serial)
  {
    // some boards need this because of native USB capability
  }
  studentNode.init();
  // put your setup code here, to run once
}

void loop()
{
  // put your main code here, to run repeatedly
}
```

To send messages to other controllers you can run two different functions, sendMessage() or sendPayload(). These two functions accept a name/ID, a type of message and a value of any data type.

```
studentNode.sendMessage(other_node_name, 'T', temperature);

studentNode.sendPayload(02, 'T', temperature);
```

The sendPayload() function only works if you know the node ID you want to send the message to. This function will only work at home environment since you specify the node IDs of the controllers. At campus is better to use the sendMessage() function since you typically only know the names of the nodes. At home is impossible to use the sendMessage() function.

To receive messages from other controllers you can run the receiveSimpleMessage() function. It accepts any kind of data types and returns a Message with a specific format.

```
struct Message { char type = '\0'; T content; };
```

```
Message<int> msg = studentNode.receiveSimpleMessage<int>();
```

To call the Message arguments just type `msg.type` and `msg.content`.

Traffic Light code:

The traffic light is implemented with an Arduino UNO with multiple I/O devices. Every traffic light must have up to 5 LEDs: 3 of them represent the traffic control colour signals (TL kA) – green, yellow, and red – and 2 represent the pedestrians control signals (PL kA) – green, and red. Every traffic light may also have a pedestrian button to reduce the waiting time for pedestrians to cross the street. Each traffic control light (green, yellow, and red) must have fault detection capability to detect that it is always turned OFF and does not react to its control.

To simplify the implementation of the prototype each Arduino board may control both traffic lights corresponding to an entry in the roundabout (TL kA, and TL kB). However, the control tasks associated with each traffic light must be implemented with self-contained software modules, exchanging information through a strict message passing discipline, without any variables in shared memory. The coordination between TL kA and TL kB must be self-sufficient, implemented locally and not by the roundabout controller.

Controller/Orchestrator code:

The controller must be designed as an Arduino UNO with I/O devices attached.

- 2 LEDs: a red LED to show the controller status (ON or OFF), and a blue LED to indicate activity on the communication bus;
- An ON/OFF button (turn on/turn off);
- A potentiometer to select the period of traffic control (entre street #1 → enter street #2 → enter street #3 → enter street #4 → enter street #1 → ...)

To reduce the hardware requirements the controller function may be implemented on an Arduino UNO board associated with an entry of the roundabout (i.e., it may share the same board with the traffic light function). However, it must be implemented by a

self-contained software module, exchanging information with the traffic lights function through a strict message passing discipline, without any variables in shared memory.

Requirements:

1. Initial state of the system must – controller turned OFF (red LED OFF); all traffic lights blinking yellow with a 1 second period (ON + OFF cycle time).

2. It must be possible to turn the controller ON and OFF, pressing the button:

    a. When turned ON, the controller must start a cyclic sequence of control of the roundabout. Starting with entry 1:

        i. Block entries 2, 3 and 4 - Command entries 2, 3 and 4 to go RED, wait for the acknowledgements, command entry 1 to go GREEN.

        ii. Wait a control period [2, 15] seconds (controlled by potentiometer).

        Iii. Block entry 1 – command entry 1 to go RED, wait for the acknowledgement, command entry 2 to go GREEN, and so on (then 3, 4, 1, …).

    b. When turned OFF, the controller must signal all traffic lights to start blinking yellow, going back to the initial state.

3. While receiving or sending data the controller's blue LED must blink.

4. The traffic light color transitions must be:

    a. Red → Yellow (immediately before: pedestrian Green → Red).

    b. Yellow → Green.

    c. Green → Yellow.

    d. Yellow → Red (immediately after: pedestrian Red → Green).

    e. Each passage through Yellow will take 0,5 seconds (constant, independent of the control period).

5. It must be possible to shorten the cycle time by half by pressing the pedestrian button - when the button is pressed the remaining of the cycle time is halved. The reduction affects a single cycle after which the system reverts to its normal operation cycle.

6. While one traffic light is performing a red-yellow-green-yellow-red cycle, the other must have its red light always on (and, in part, pedestrian green). In other words, two traffic lights must not be simultaneously green.

7. All communications between the traffic lights must be performed via controller (see section Modular Programming).

8. It must be possible for the controller and any traffic lights, to detect faults of the communications link, or in other traffic lights:

a. After 2 control periods – [2×2, 2×15] seconds – with persistent faults:

i. The traffic light that detected the missing or faulty communication must start blinking yellow (both lights which control traffic at the entry – TL kA and TL kB).

ii. The controller must communicate to other traffic lights to start blinking yellow, too. Then it must turn itself off, going back to the initial state of the system.

iii. Faults can be detected by the controller (e. g. lack of response to commands from a traffic light), or by a traffic light (e. g. lack of command from the controller to change the accesses to the roundabout).

9. All communications in the system must be performed using the ESB protocol.

Modular Programming:

Communications between the modules of the system (traffic lights and controller) must respect a specific API to build a modular and interoperable system. This approach:

- Will enable two teams of programmers, working in parallel, to build both controller and traffic light separately;
- Will allow controllers and traffic lights from different teams to work together properly.

Controller – Traffic light communication API:

Controller – The controller must implement the following messages to be sent via ESB:

- RED (x) – where X is the identifier of the entry to be closed.
- GREEN (x) – where X is the identifier of the entry to be opened.
- OFF (x) – signals a traffic light to start blinking yellow.
- PING (x) – where X is the identifier of the entry to be checked. It requests the traffic light of the entry to send back its status. (In this case the STATUS replaces the ACK message.)
- ACK (x) – It is the acknowledge message returned in response to x. The request/command was correctly received.

Traffic lights – traffic lights must implement the following functions to be sent via ESB:

- TIME (x) – signals the controller (x = 0) to shorten the cycle period due to a pedestrian request.

- PING (x) – where X is the identifier of the entry to be checked. It requests the traffic light of the entry to send back its status. (In this case the STATUS replaces the ACK message.)
- STATUS (x, state) – Sends to x the status of the traffic lights at the entry.
- ACK (x) – It is the acknowledge message returned in response to x. The request/command was correctly received.

All the messages except the Status(x) should have the following format: *Sender* | *OperationNumber* | *Destination* | *Integrity Byte*.

The Status(x) should be: *Sender* | *OperationNumber* | *Destination* | *Information* | *Integrity*.

Each one of these members is 1 byte long so, they must be converted to char.

1. The send will always be the controller (entry 0) but it is maintained for possible future needs, for the same reason the Destination is kept.
2. The operation numbers are:
    a. RED(x) - 0
    b. GREEN(x) - 1
    c. OFF(x) - 2
    d. PING(x) - 3
    e. ACK(x) - 4
    f. STATUS(x) - 5
3. The Integrity byte is the char of the sum of the previous message bytes: char (Sender + OperationNumber + Destination + Message)
4. The Information byte is [pedestRedFailing, pedestYellowFailing, pedestGreenFailing, redFailing, yellowFailing, greenFailing, timerActivated, 0]. The first 3 bits should be 1 if the lights near pedestrian are broken, the following 3 should be 1 if the other traffic lights are broken, the seventh bit should be 1 if the pedestrians pressed the button, and the last bit has no meaning.

RED(x)

The message sent from the controller to traffic light #x will be:

0 | 0 | x | x and the response which is an ACK message will be x | 4 | 0 | x + 4

GREEN(x)

The message sent from the controller to traffic light #1 will be:

0 | 1 | x | 1 + x and the response, which is an ACK message, will be x | 4 | 0 | x + 4

OFF(x)

The message sent from the controller to traffic light #1 will be:

0 | 2 | x | 2 + x and the response, which is an ACK message, will be x | 4 | 0 | x + 4

PING(x)

The message sent from the controller to traffic light #1 will be:

0 | 3 | x | 3 + x and the response, which is a STATUS message, will be x | 5 | 0 | Status | x + Status + 5

Simplified Input/Output

When merging several modules on the same Arduino controller the number of interface ports required to implement the specifications may exceed the number of input/output pins available. Therefore, the following simplifications are recommended.

Requirements	Simplification
Controller: two LEDs to show status and communication activity.	Use only one LED to show both status (ON/OFF) and communications activity, eventually pulsing with different patterns.
Identification of the Arduino: • 0 – controller; • 1, 2, 3, 4 – pairs of traffic lights at each entry of the roundabout.	Code the identifier of each Arduino with just two bits: • 1, 2, 3, 4 – pairs of traffic lights at each entry of the roundabout; • The main controller of the roundabout is loaded at 1
Traffic Light: check the operation condition of the 3 lights (red, yellow, and green).	Check only the operation condition of the red LED since it is the most critical to enforce safety.
Traffic Light with pedestrian signal.	Implement only the green pedestrian signal: green ON means “sage to cross the street”; green OFF “do not cross.