TECHNICAL UNIVERSITY OF CLUJ-NAPOCA
FACULTY OF AUTOMATION AND COMPUTER SCIENCE
COMPUTER SCIENCE SPECIALIZATION

LOCAL PARKING SYSTEM DESIGNED WITH OLIMEX ESP32-EVB DEVELOPMENT BOARD

Student: Ioan Simion Belbe
Group 30433, Semigroup I
Laboratory Coordinator: Zoltan Czako

Contents

I. Introduction

a. Motivation

Nowadays, with the common situation of globalization and its effects on superpopulation in the big cities, it becomes more and more demanding to have new parking lots. Our project comes into existing to fit the need of a parking system that satisfies both costs and performance. Since everything revolves around cheap and reliable products we chose to develop the first stage of a parking system which is the local management of the traffic and parking lots with the help of Olimex ESP32-EVB Development Board which gives to the engineer a great maneuverability and lots of features for handling and supporting continuous development of the local parking lot. Nevertheless, it is a system that once implemented it can offer almost the same flexibility to the administrator of the parking lot, having straight-forward features, and implementing easy-to-access functions and designs meant to extend easily the future implementing possibilities of the local parking system.

b. Objectives and Goal

Our main objectives for this this project are presented as a list of bullet-points sentences since it is much easier to follow them and see how they are coming to existence one-by-one:

- To have at least one functional Olimex ESP32-EVB implemented system.
- To create a communication link between the Olimex ESP32-EVB and a scanner device, i.e. Scanner Datalogic Gryphon GFS4150-9, and to implement the functions for handling this communication.
- To create a communication link between the Olimex ESP32-EVB and a RFID (radio frequency identification) device, i.e. Olimex RFID 1356-BOX, and to implement the functions for handling this communication.
- To create a receiving signals link between the Olimex ESP32-EVB and a car detector system, i.e. Car Detector GVD302 Dual Channels (Binary), and to implement the functions for handling the detection.
- To create a sending signals link between the Olimex ESP32-EVB and a car barrier, i.e. Parking Barrier Life SUPRA (RG1R DL S), and to implement the functions for handling the opening of the barrier.

- To create a receiving signals link between the Olimex ESP32-EVB and information button, and to implement the functions for handling the information display process logic.
- To create a receiving signals link between the Olimex ESP32-EVB and request ticket button, and to implement the functions for handling the requesting of a ticket process logic.
- To create an UDP Server on Olimex ESP32-EVB, and to implement the functions for handling the broadcast from an external server which manages the local parking systems.
- To create a HTTP Web Server on Olimex ESP32-EVB, and to implement the functions for handling FOTA (Firmware Over-The-Air), endpoints for local system commands from the external server, and a log system.
- To implement a physical conversion communication from RS-232 to UART, between the Scanner Datalogic Gryphon GFS4150-9 and Olimex ESP32-EVB.

Our goal for this project is to be having and be using a fully develop local parking system which can handle future developments upgrades, and it is fully functional within a bigger system which handles all the local systems. In this regard the local parking system is meant to be fully configurable for different purposes, i.e. entrance system or exit system.

c. Methodology

As for the methodology we'll be choosing to use different techniques which involves trials and errors, documentation-based systems, manual/automated testing frameworks, and administrator-system interaction interface.

d. Sources and Resources

As for the main sources used for this project, we will be relaying on the official's forum from Olimex and Espressif Systems companies, documentations for each device used, specialized programming helping platforms like www.stackoverflow.com, working with Arduino IDE developing environment, and making use of different tools like Terminal v1.93b found here.

Moreover, we will be making use of the laboratories resources in order to develop and test our project, and to finding solutions. The project will be host on a repository platform, i.e. www.github.com, under a private repository.

II. Theoretical Background

a. Developing Environments

There are three major developing environments/frameworks when considering programming on the Olimex ESP32-EVB. We would only use one for this project, but the other two are worth mentioning. Also, through this various range of options in regard to programming we will see that this board offers portability at its best.

First, and foremost it is recommended that developing on this board to be done on the same platform created by the same company who is responsible for the ESP32-EVB microcontroller, and that is Espressif Systems.

Their company offers a framework, Espressif IoT Development Framework, or ESP-IDF which comes packed with almost all the libraries you need to perform programming on the board (1). If the choice is getting maximum functionality and compatibility this framework can guarantee that for the ESP32 microcontroller, but not for the board as well.

In this regard the predecessor of ESP32, the ESP8266 microcontroller has a built-in processor. The issue with the former being that due to multitasking regarding the Wi-Fi stack, it needed to use a separate microcontroller for data processing, digital input and output, and for interfacing sensors. With the ESP32 microcontroller you get the commodity of not needing a second microcontroller for all the previous mentioned activities. And this is thanks to the microprocessors used by the ESP32 microcontroller which is an Xtensa® Dual-Core 32-bit LX6, which runs up to 600 DMIPS (2). The frequency on which ESP32 can perform depending on the board is between 160 MHz (3) or 240 MHz (4). That is very good speed for the system we are trying to develop which requires a microcontroller with connectivity options.

The processor's two cores are named Protocol CPU (PRO_CPU) and Application CPU (APP_CPU) (5). The Protocol CPU handles Wi-Fi, Bluetooth and other peripherals connections protocols like SPI, I2C, ADC etc. (5). And the Application CPU is only purpose is to be used for the application code. This differentiation is done and can be found in the Espressif Internet Development Framework. Arduino IDE and other frameworks/developing environments are based on ESP-IDF.

ESP-IDF manages the two core's processing and data exchange by using a real-time operating system kernel (freeRTOS). But the complexity at which it operates freeRTOS is high. Nevertheless, the following scheme (Figure 1 (6)), which is the block diagram of ESP32's with

the options in regards of communication and processing, gives hopes and trustiness for choosing this microcontroller for our project.

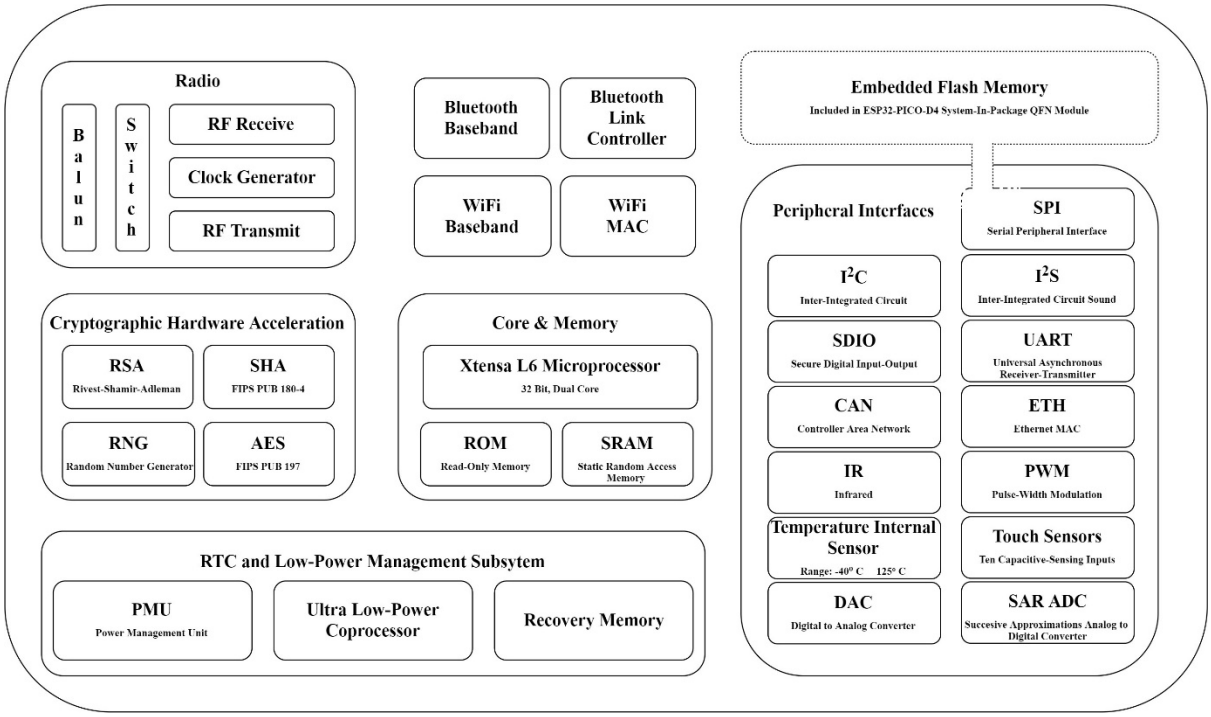


Figure 1. ESP32 Block Diagram

Along with the release of ESP32, Espressif Systems also offers a corresponding module ESP-WROOM-32. Despite his size (25.5 x 18.0 x 2.8mm) it is very easy to use the module due to integrated components such as antenna, oscillator and flash (7).

Also, one interesting thing to note is that the both processors are mapped symmetrically to this address space. It basically means, a register for example can be accessed from same address location from both the CPUs as shown in image (Figure 2 (7)) below.

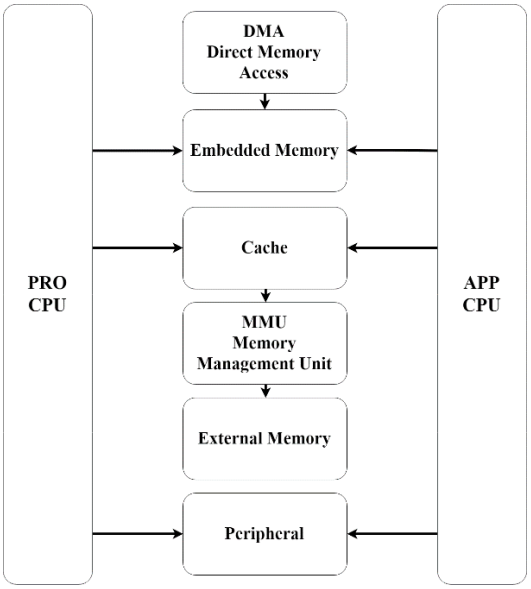


Figure 2. ESP-WROOM-32 Module

The second developing area worth mentioning is RIOT open source OS. RIOT is developed by a great community of developers and is based on a minimalistic kernel contained in a modular architecture. The goals on which RIOT was born are the following (8):

- minimized resource usage in terms of RAM, ROM;
- minimized power consumption;
- support for versatile configurations: 8-bit to 32-bit MCUs;
- wide range of boards and use-cases – in our case the Olimex ESP32-EVB board;
- minimized code duplication across configurations;
- portability of most of the code, across supported hardware;
- provide real-time capabilities;

The above goals lead to several principles which explain parts of the design of RIOT. These principles are the following (8):

1) Network Standards – RIOT focusses on open, standard network protocol specifications, e.g. IETF protocols;

2) System Standards – RIOT aims to comply to relevant standards, e.g. the ANSI C standard (C99), to take full advantage of the largest pool of 3rd party software. Extensive use of C language caters for both low resource requirements and easy programmability;

3) Unified APIs – RIOT aims to provide consistency of APIs across all supported hardware, even for hardware accessing APIs, to cater for both code portability and minimize code duplication;

Nonetheless, we won't be choosing this OS for developing our project since our project does not need the whole resources of an operating system, which comes to be dedicated for much larger scale projects. Also, portability of the programs implemented on board with similar specifications makes it easier to libraries needed outside the Espressif Systems libraries.

Our third and final option is Arduino IDE, which comes in hand for the project we are developing having all the functionalities of ESP-IDF implemented, being easy to use cross platform and requiring few resources in order to get it going. Not to say offering a much more reliability regarding install and get it up running.

The Arduino Integrated Development Environment (IDE) is a cross-platform application (for Windows, macOS, Linux) that is written in the programming language Java. It is being used for uploading and coding programs to embedded boards compatible with the

Arduino ones, but also, with the help of 3rd party cores, other vendor development boards, such as ESP32 (9).

There are lots of other frameworks that can be taken into consideration when programming on ESP32, but they are programming language specific, i. e. MicroPython, Espruino, Platform etc.

b. Components Description

While implementing this project we will refer especially to the components that make our system work. We will start by presenting the most important component, which is the Olimex ESP32-EVB developing board.

Olimex ESP32-EVB is an open source hardware boards which use the ESP32-WROOM-32 module. The key features of the boards are (10):

- Ethernet LAN interface
- MicroSD card interface
- IR interface
- CAN interface
- two relays
- UEXT connector with I2C, SPI and UART interfaces

As from this list using the UEXT connector, a lot of off-board hardware modules can be connected to Olimex ESP32-EVB to extend the hardware without the need for soldering iron or breadboards.

More about this we can on the following image (Figure 3 (10)) about the board pinout connections.

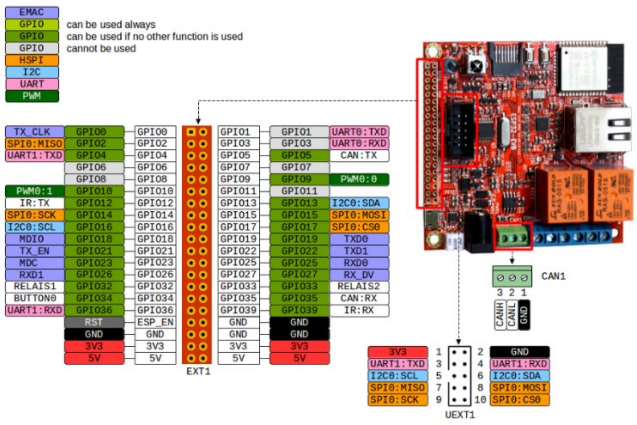


Figure 3. Olimex ESP32-EVB Pinout

Another component we will present is the scanner which we choose to work with, Scanner Datalogic Gryphon GFS4150-9. The Gryphon GFS4XXX is a fully self-contained standard range 1D bar code scanning module for use in self-service kiosks or other semi-automated equipment requiring the ability to read a bar code. For linear CCD Scanner for OEM applications we use GFS4150-9 Gryphon Fixed Scanner imager communicates through an RS-232 9 pin connectors (11).

Another important component is the RFID tag reader, Olimex RFID 1356-BOX. MOD-RFID1356 is an RFID station inside the Olimex RFID 1356-BOX, able to interact with ISO15693 compliant VICC transponders. All the complexity of RFID tag detection, verification and decoding are handled by MOD-RFID1356. Custom command requests can be initiated to facilitate VICC EEPROM read and write operations (12).

MOD-RFID1356 supports two distinctive modes of operation, easily switched by a single button press. It can emulate USB HID keyboard and input the read TAG ID directly to any Windows application, including the venerable Notepad. Or it can emulate USB CDC serial port for easy access from custom user applications using standard code for COM port access (12).

Last component to be presented is the Car Detector GVD302 Dual Channels (Binary). GVD series loop detector (sensor) is designed on microprocessor basis, by the change from vibration frequency of loop which is buried under the road, the loop sensor detects the passing or existing of vehicles. Main functionalities it provides are the following (13):

- Dialer On-Off switch detects the sensitivity and bootstrap function of sensitivity
- Dialer On-Off switch select action mode and output mode of the relay
- Tunable working frequency
- Power-on automatic reset and manual reset functions

III. Implementation and Design

a. Diagrams

For implementing the system proposed we thought to work in the system of a Gantt Chart. Gantt charts for project management display tasks over time, creating a visual roadmap of the project's progress. Gantt charts make it easy to:

- Visualize the entire project from start to finish.
- Improve time management.
- Clearly communicate with your team.

The diagram describing best our current project and its functionalities, i.e. architectural diagram is the one in the following image (Figure 4).

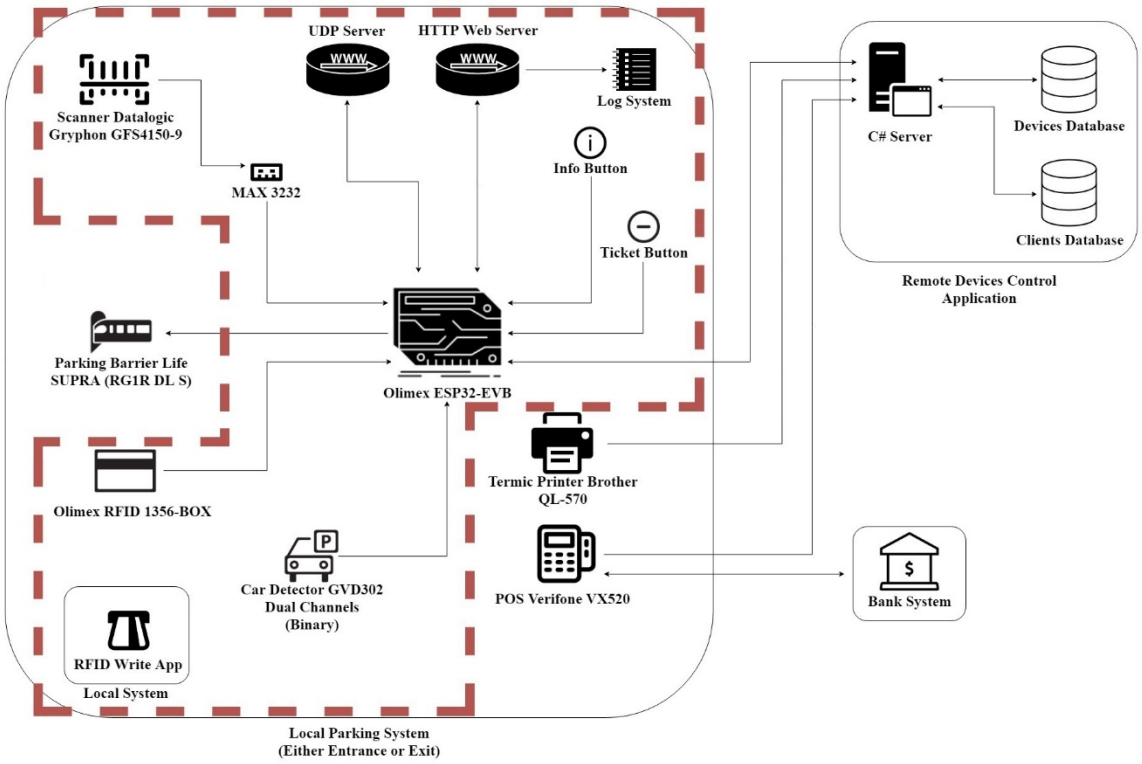


Figure 4. Communication System Implemented

As well this is the way we choose the track our progress developing this type of connections that meet in the previous diagram.

In the subchapter Components Connections, we will continue to present each development stage provided by the following diagram:

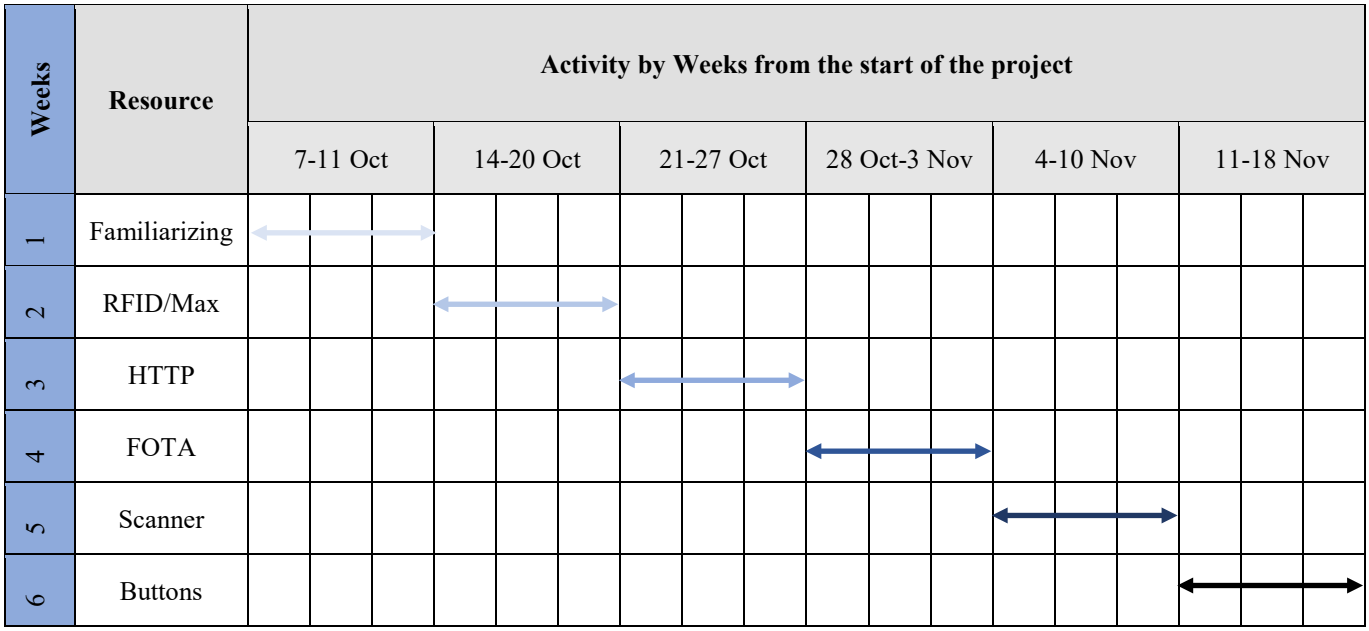


Figure 5. Gantt Chart

b. Components Connections

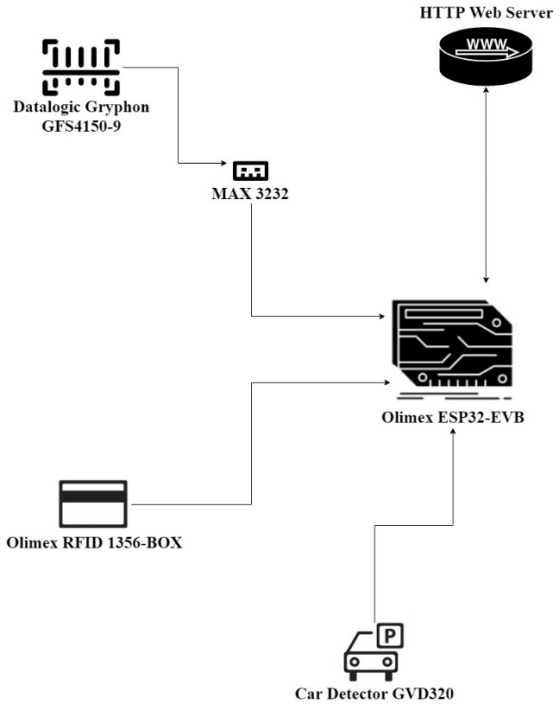
Week 1:

In my first Week I got myself acquaintance how the project of parking was designed and how it was thought, being a modular project, in which different hardware communicates locally with a Olimex ESP32-EVB plate. The company also has a maintenance contract for a parking lot developed by the company Equinsa, and with the help of that parking system was physically presented to me the operation of an induction loop for machine detection, OCR recognition of numbers License plate.

Week 2:

During the second week, steps were continued to understand the operation of the new parking system and the devices that will communicate with the Olimex ESP32-EVB plate were consulted and chosen. Once established which products will enter the composition of the parking system and how they will communicate with each other, I have had to read the documentation of those devices and understand from these documentations how to use and schedule. The devices on which it was established to communicate with the parking system are: A Olimex RFID module 1356-BOX used to read and write access cards in the subscription-based parking, a Datalogic Gryphon GFS4150-9 scanner used For scanning parking vouchers, a MAX 3232 module used for switching from RS232 communication to UART communication of the scanner and an induction loop-based GVD320 detector used for car detection before and after a barrier (used two Induction loops). Towards the end of the week I started actually

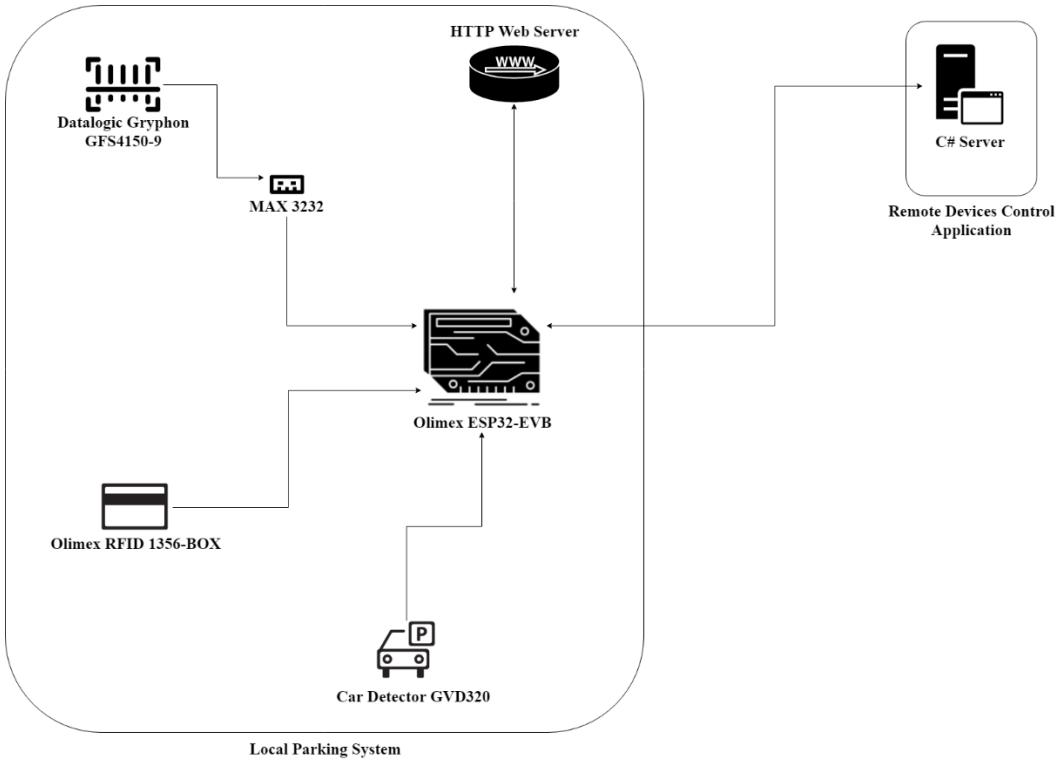
working small tasks in the Arduino IDE (development program and language used for the whole project on the embedded side). These tasks consist mainly of small tasks that resulted in the ultimate construction of an HTTP server running on the Olimex ESP32-EVB plate. In this respect I had the help of the Guardian to explain the unknown notions, such as controlling the devices with the help of Endpointuri, etc. For a better understanding of the project it was recommended that I make a weekly chart with the functionalities and connections of the devices understood and studied that week. So this week's diagram is as follows:



Week 3:

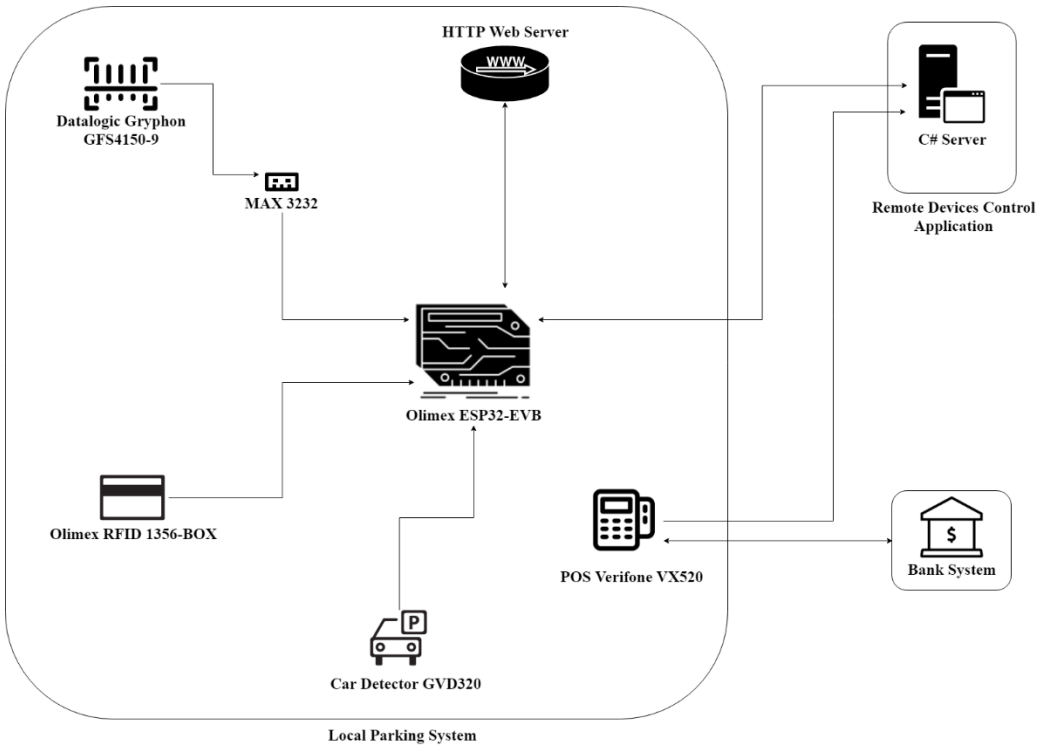
In the third week it was brought to my attention at the beginning of the week the Book of Neil Kolban, *Kolban's book on ESP32*, the book I had to study to better understand the architecture of the microcontroller ESP32 developed by Espressif Systems used on the Olimex plate. As for the tasks offered to be resolved, they continued organically the development of the server last week, namely the implementation of FOTA technology for the plaque in question. Thus, the project coordinator defined the principles of Over The Air Technologies and through the GET and POST methods for HTTP servers I had to write a little program that would do exactly that. I was also guided by the specific bookstores I should use. At the end of the week I was informed that the implementation of this server serves as a purpose of remote communication with another server written in C# using .net Core on which a software is running local device management, and the implementation of last week's endpoints was aimed at facilitating this

communication based on a local IP. This week the diagram has not undergone large changes, but only a small addition to the remote server:



Week 4:

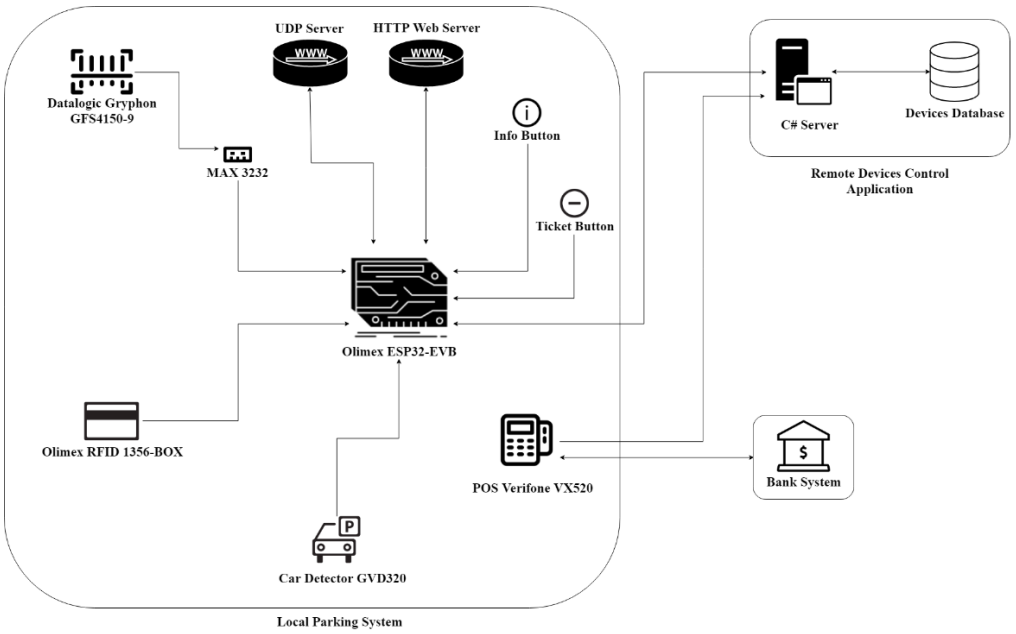
At week four with the understanding of the functionalities offered by this microcontroller we solved various tasks involving the devices of previous weeks, so we wrote code for communication between the scanner and the plate, and we succeeded with the Guardian to connect two induction loops to the machine detector and establish communication between the detector and the plate. At the same time this week I had the opportunity to work something other than embedded. So I was assigned to one of my colleagues working on the part of the app that manages the devices. So we discovered that this project will implement different payment systems: by SMS, through POS. A. I had to help create a C# wrapper over a DLL written in C++, because those at the bank with which the company has contract for the project had only bookstores written in C++, and the application developed by our company is in C#. I was very excited for this week because although I knew the concepts of OOP, I never programmed in C#, but the training made by the new colleague was consistent and I managed to contribute to the new implementation, the communication being established based on IP between the bank's POS and the administrative server. The diagram for this week is as follows:



Week 5:

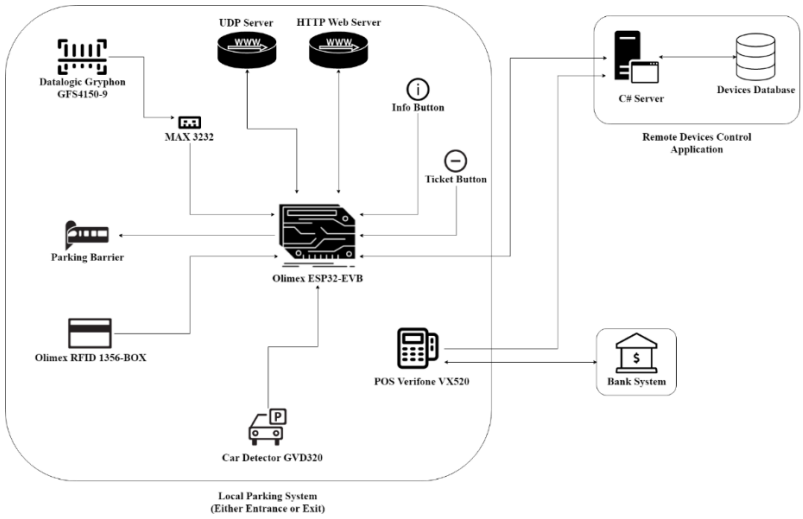
In the fifth week we continued with the implementation of needs in the communication between Olimex ESP32-EVB and the administrator server, such as: sending the MAC address of the microcontroller to be registered in a database of devices and sending Necessary signals when a car is present and a driver requests entry into the parking lot. The first issue demanded that sending the MAC address be made only on demand via a UDP connection, so the administrator server initializing a broadcast with a key known only to the company's devices. At the same time, the tutor gained more confidence in me and was entrusted with the implementation of two buttons with two different functionalities, one that issued parking tickets and one later used to provide information to users.

The diagram following the discoveries made this week for the parking system is as follows:



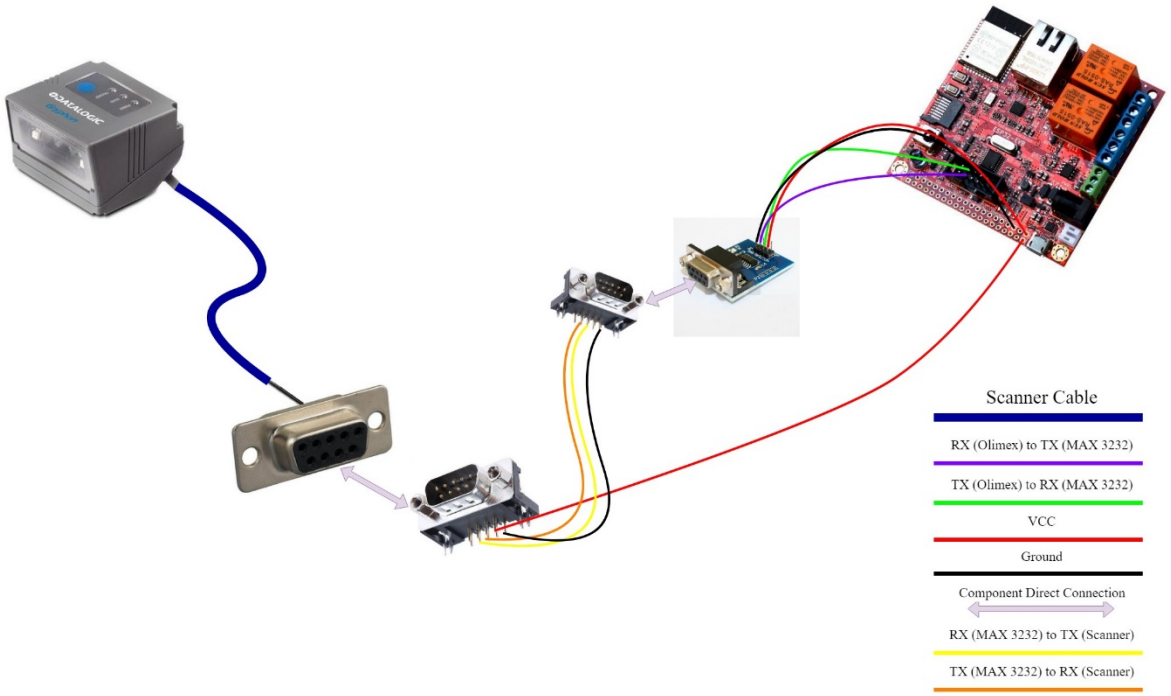
Week 6:

In the sixth week we continued with the tasks of implementing the buttons taken at the end of last week. Then on Wednesdays I was asked to help with the programming and use of the relays on the Olimex plate to control the parking barrier, each tile controlling a single barrier, the local system can be defined both as a system for controlling activities at and a system for controlling exit activities, choice and coordination being made by the administrative system. The diagram for this week is as follows:

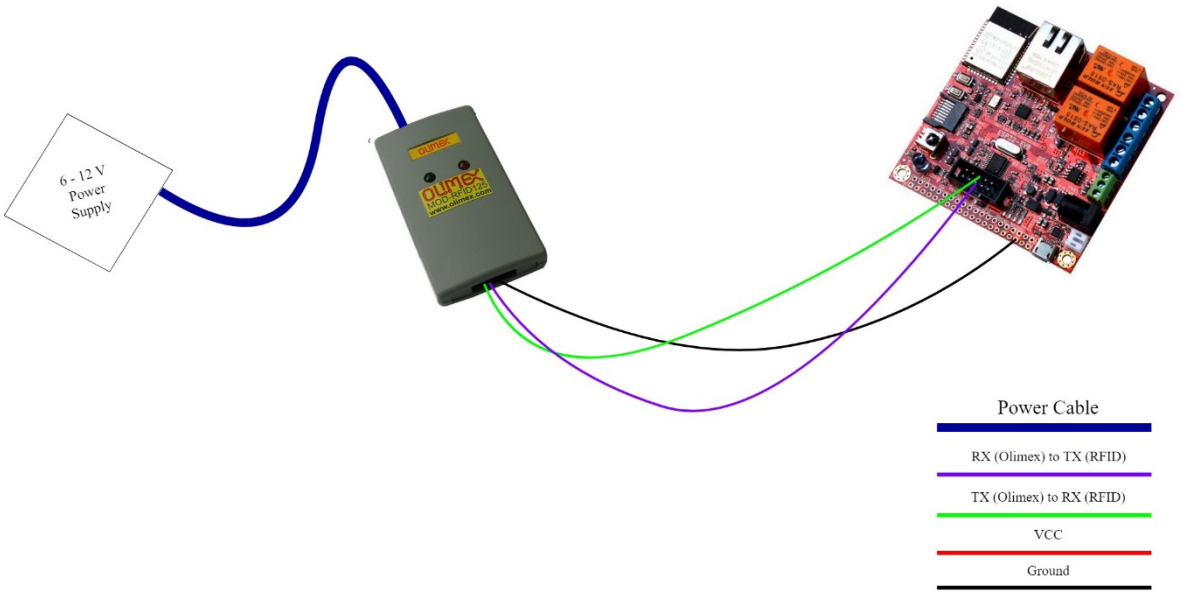


c. Control Systems

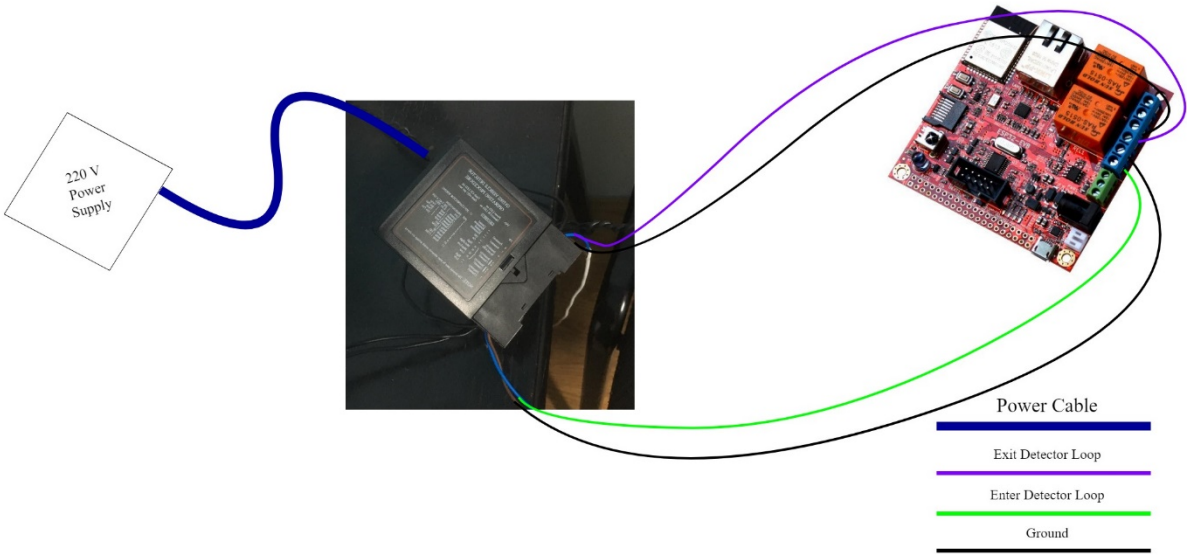
For the control system we will present each of the components and their connections in diagrams. We will start with the connection between the Olimex Board and the Scanner.



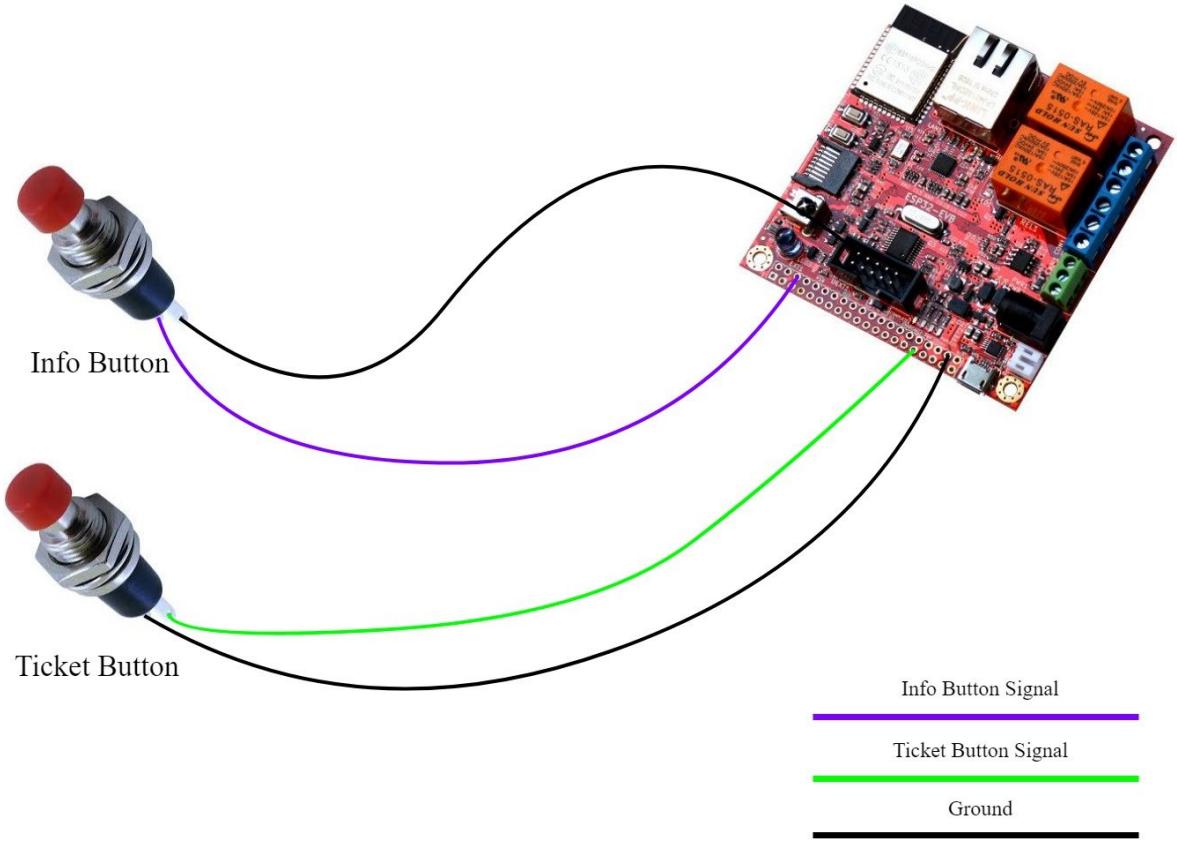
The control system between the Olimex 1356-RFID and the board is provided by the following image.



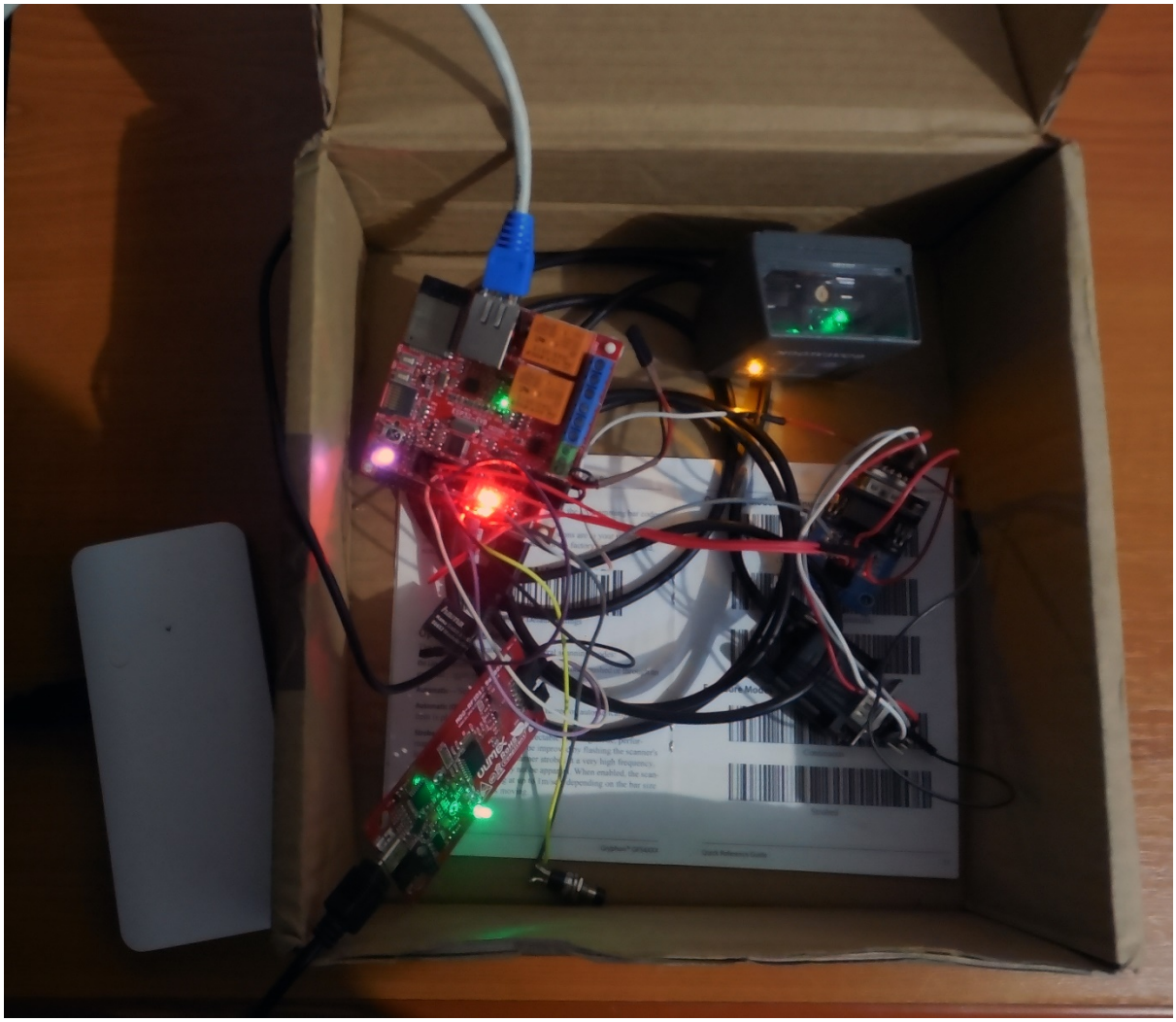
The control between the car detector which we won't be presenting in our project, but only the functionality principle is presented as follows.



Finally the buttons for request ticket and info are presented below.



In the end everything connected together will look like in the following picture:



d. Algorithms

Regarding the algorithms we implemented we find in the Annex 1 the source code for our project.

IV. Testing

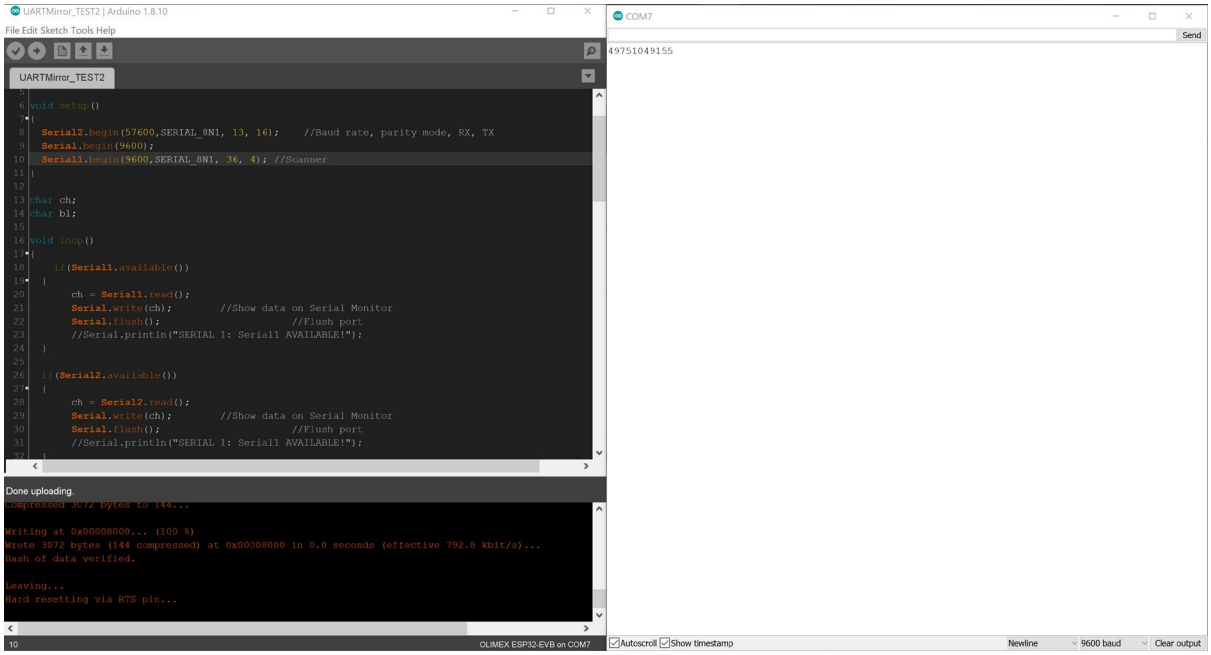
As for testing we provide a few screen shots showing us every an each of one of the functionalities we engage in developing.

While this we tested our system component to evaluate one or more properties of interest. In general, these properties indicate the extent to which the component or system under test:

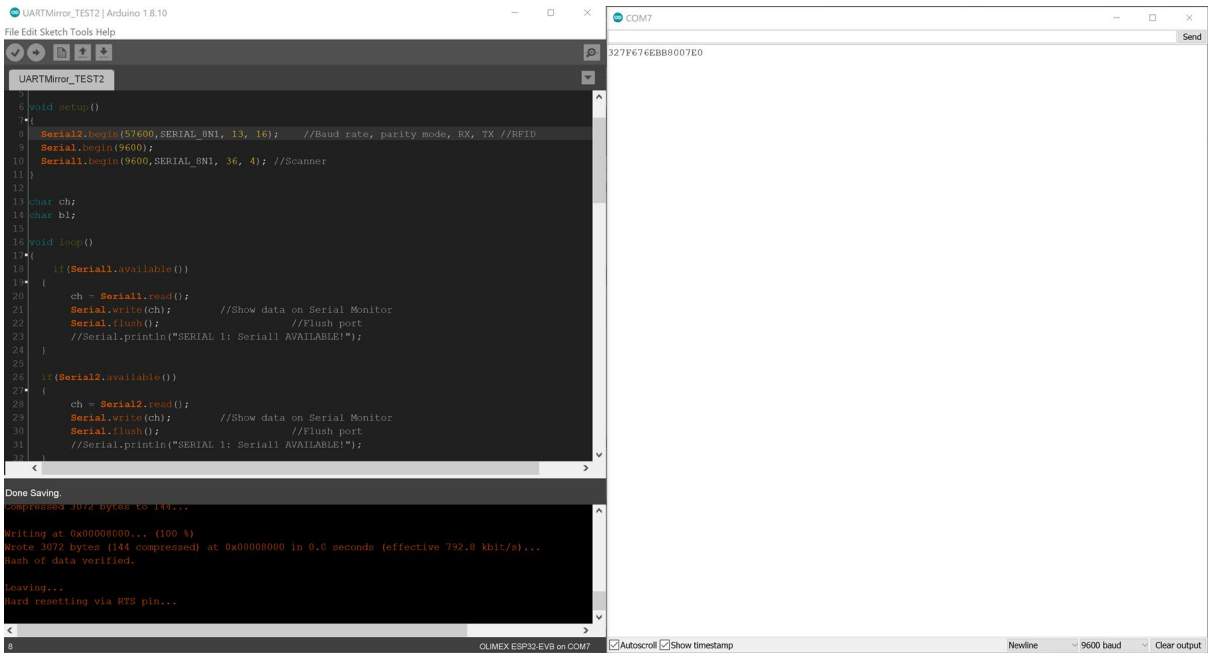
- meets the requirements that guided its design and development,
- responds correctly to all kinds of inputs,

- performs its functions within an acceptable time,
- it is sufficiently usable,
- can be installed and run in its intended environments, and
- achieves the general result its stakeholders desire.

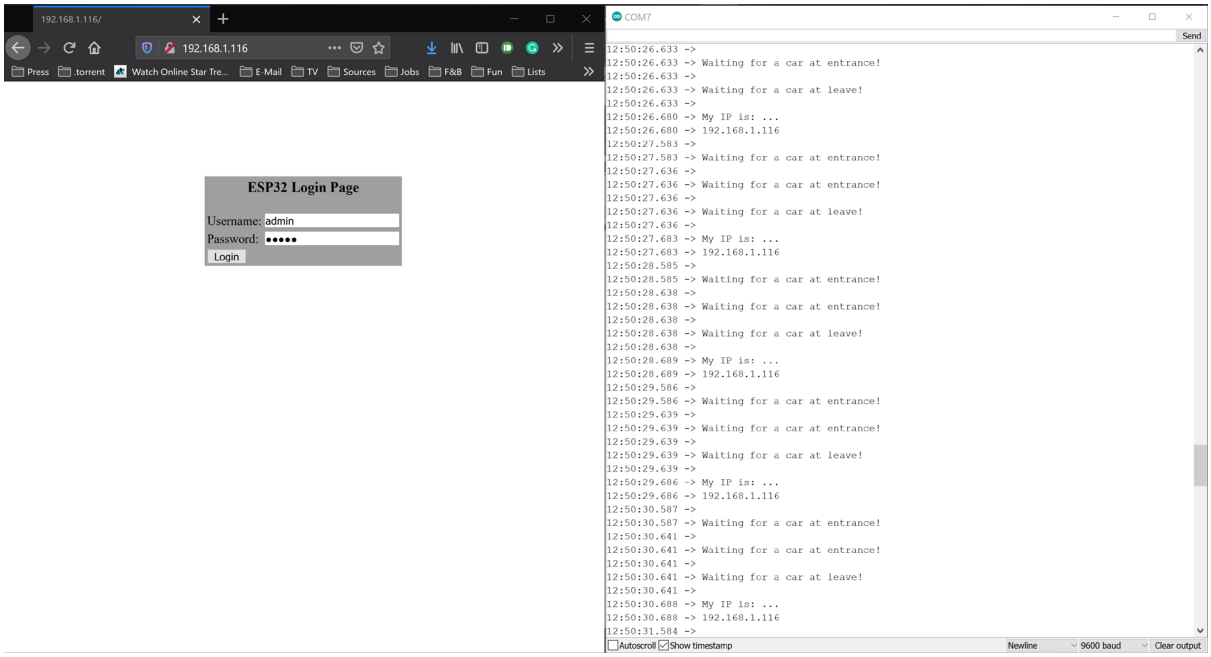
Scanner Test:

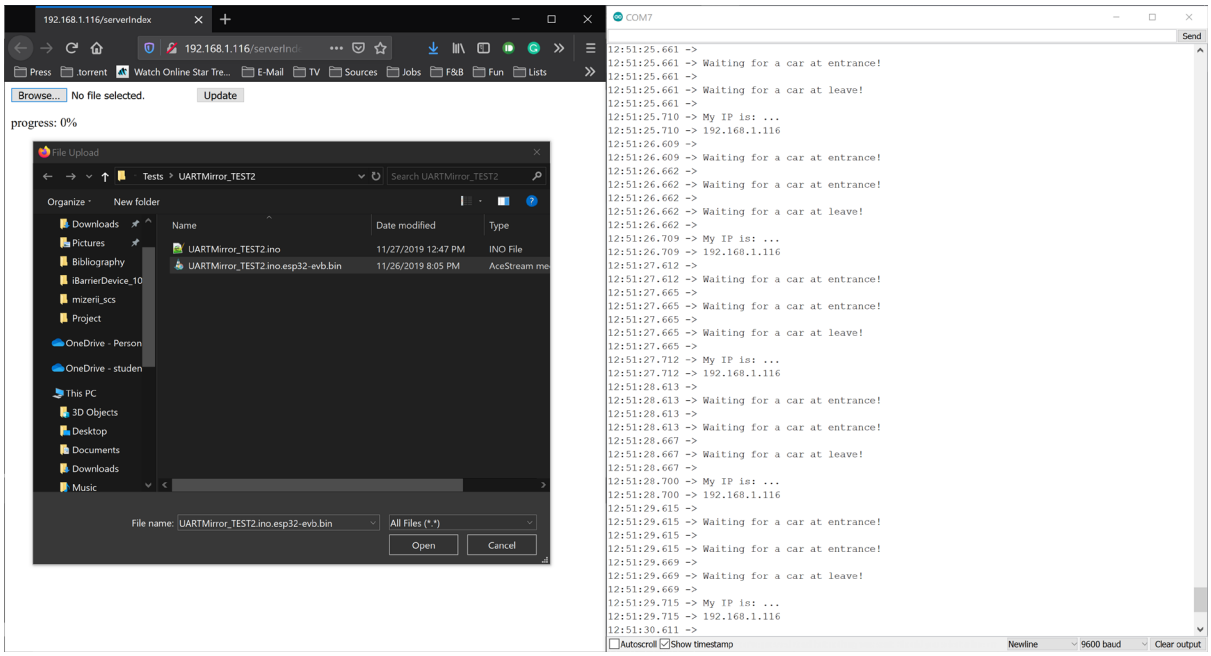


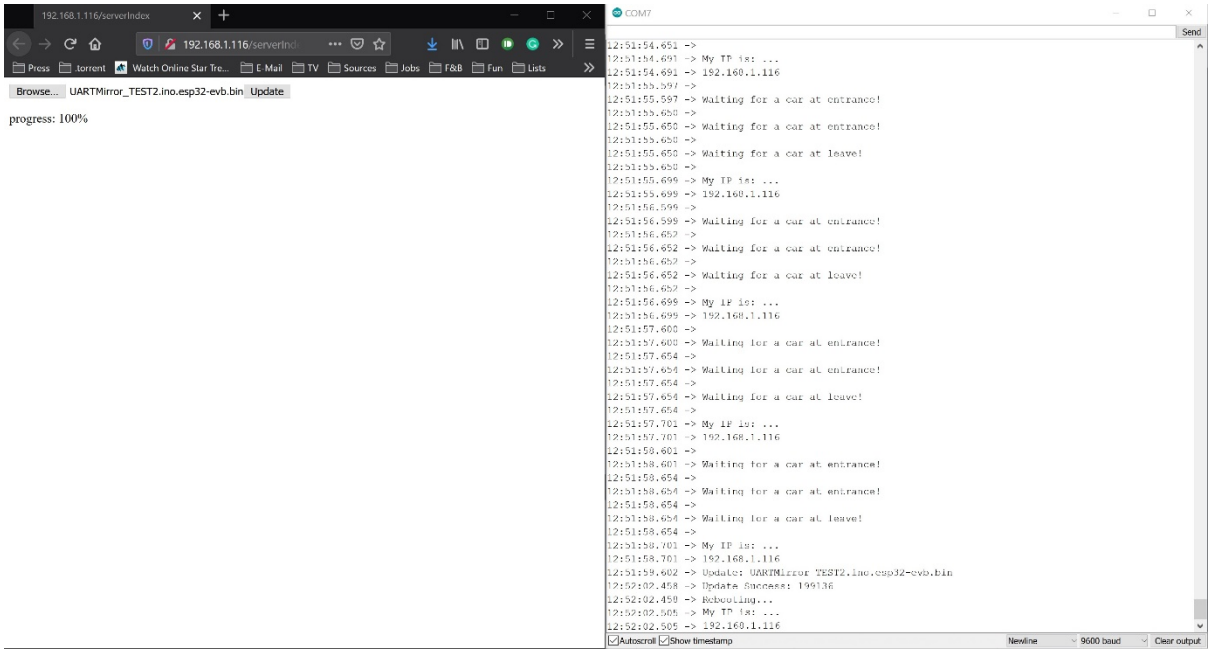
RFID Test:



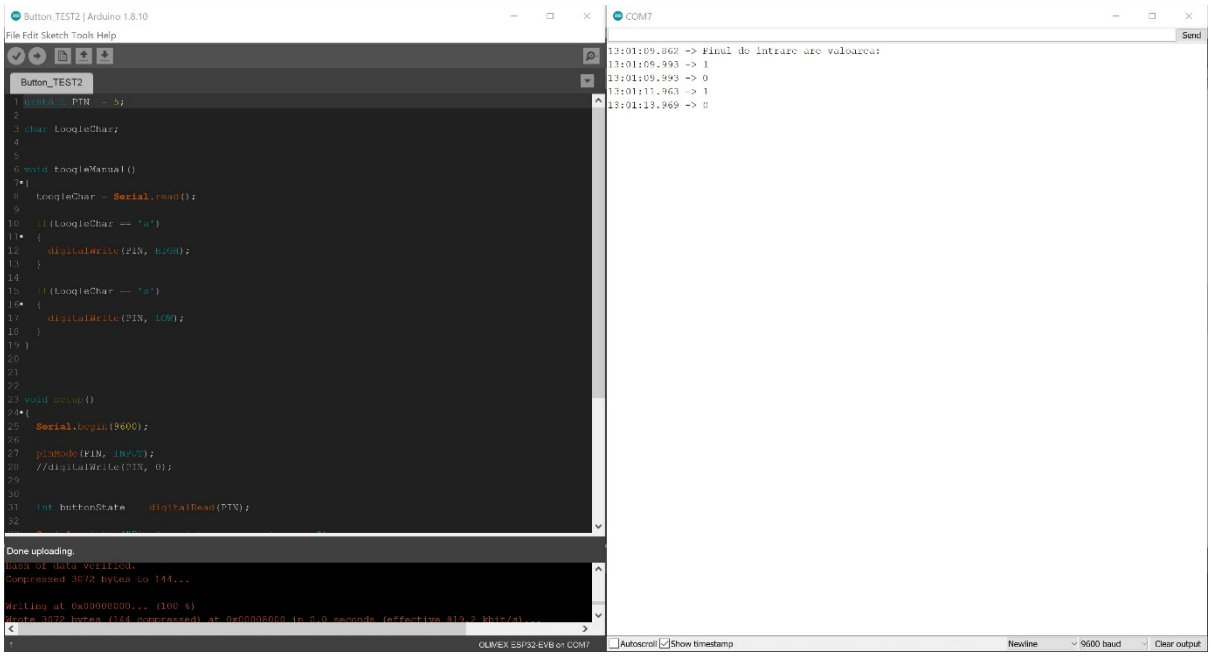
FOTA Test:


192.168.1.116/
192.168.1.116
ESP32 Login Page
Username: admin
Password:
Login
COM7
Send
12:50:26.633 ->
12:50:26.633 -> Waiting for a car at entrance!
12:50:26.633 ->
12:50:26.633 -> Waiting for a car at leave!
12:50:26.633 ->
12:50:26.680 -> My IP is: ...
12:50:26.680 -> 192.168.1.116
12:50:27.583 ->
12:50:27.583 -> Waiting for a car at entrance!
12:50:27.636 ->
12:50:27.636 -> Waiting for a car at entrance!
12:50:27.636 ->
12:50:27.636 -> Waiting for a car at leave!
12:50:27.636 ->
12:50:27.683 -> My IP is: ...
12:50:27.683 -> 192.168.1.116
12:50:28.585 ->
12:50:28.585 -> Waiting for a car at entrance!
12:50:28.638 ->
12:50:28.638 -> Waiting for a car at entrance!
12:50:28.638 ->
12:50:28.638 -> Waiting for a car at leave!
12:50:28.638 ->
12:50:28.689 -> My IP is: ...
12:50:28.689 -> 192.168.1.116
12:50:29.586 ->
12:50:29.586 -> Waiting for a car at entrance!
12:50:29.639 ->
12:50:29.639 -> Waiting for a car at entrance!
12:50:29.639 ->
12:50:29.639 -> Waiting for a car at leave!
12:50:29.639 ->
12:50:29.686 -> My IP is: ...
12:50:29.686 -> 192.168.1.116
12:50:30.587 ->
12:50:30.587 -> Waiting for a car at entrance!
12:50:30.641 ->
12:50:30.641 -> Waiting for a car at entrance!
12:50:30.641 ->
12:50:30.641 -> Waiting for a car at leave!
12:50:30.641 ->
12:50:30.688 -> My IP is: ...
12:50:30.688 -> 192.168.1.116
12:50:31.584 ->
Autoscroll
Show timestamp
Newline
9600 baud
Clear output


192.168.1.116/serverIndex
192.168.1.116/serverIndex
Browse...
No file selected.
Update
progress: 0%
File Upload
Tests › UARTMirror_TEST2
Search UARTMirror_TEST2
Organize
New folder
Name
Date modified
Type
UARTMirror_TEST2.ino
11/27/2019 12:47 PM
INO File
UARTMirror_TEST2.ino.esp32-evb.bin
11/26/2019 8:05 PM
File name: UARTMirror_TEST2.ino.esp32-evb.bin
All Files (*.*)
Open
Cancel
COM7
Send
12:51:25.661 ->
12:51:25.661 -> Waiting for a car at entrance!
12:51:25.661 ->
12:51:25.661 -> Waiting for a car at leave!
12:51:25.661 ->
12:51:25.710 -> My IP is: ...
12:51:25.710 -> 192.168.1.116
12:51:26.609 ->
12:51:26.609 -> Waiting for a car at entrance!
12:51:26.662 ->
12:51:26.662 -> Waiting for a car at entrance!
12:51:26.662 ->
12:51:26.662 -> Waiting for a car at leave!
12:51:26.662 ->
12:51:26.709 -> My IP is: ...
12:51:26.709 -> 192.168.1.116
12:51:27.612 ->
12:51:27.612 -> Waiting for a car at entrance!
12:51:27.665 ->
12:51:27.665 -> Waiting for a car at entrance!
12:51:27.665 ->
12:51:27.665 -> Waiting for a car at leave!
12:51:27.665 ->
12:51:27.712 -> My IP is: ...
12:51:27.712 -> 192.168.1.116
12:51:28.613 ->
12:51:28.613 -> Waiting for a car at entrance!
12:51:28.613 ->
12:51:28.613 -> Waiting for a car at entrance!
12:51:28.667 ->
12:51:28.667 -> Waiting for a car at leave!
12:51:28.667 ->
12:51:28.700 -> My IP is: ...
12:51:28.700 -> 192.168.1.116
12:51:29.615 ->
12:51:29.615 -> Waiting for a car at entrance!
12:51:29.615 ->
12:51:29.615 -> Waiting for a car at entrance!
12:51:29.669 ->
12:51:29.669 -> Waiting for a car at leave!
12:51:29.669 ->
12:51:29.715 -> My IP is: ...
12:51:29.715 -> 192.168.1.116
12:51:30.611 ->
Autoscroll
Show timestamp
Newline
9600 baud
Clear output



Buttons Test:



The code for all our test is to be found in Annex 2 as well.

V. Conclusions

In conclusion we implemented the system we proposed in the manner we projected our objectives, and those that are implemented are the following, which are coming as a direct match to our objectives:

- One functional Olimex ESP32-EVB implemented system.
- A communication link between the Olimex ESP32-EVB and a scanner device, i.e. Scanner Datalogic Gryphon GFS4150-9, and to implement the functions for handling this communication.
- A communication link between the Olimex ESP32-EVB and a RFID (radio frequency identification) device, i.e. Olimex RFID 1356-BOX, and to implement the functions for handling this communication.
- A sending signals link between the Olimex ESP32-EVB and a car barrier, i.e. Parking Barrier Life SUPRA (RG1R DL S), and to implement the functions for handling the opening of the barrier.
- A receiving signals link between the Olimex ESP32-EVB and information button, and to implement the functions for handling the information display process logic.
- A receiving signals link between the Olimex ESP32-EVB and request ticket button, and to implement the functions for handling the requesting of a ticket process logic.
- An UDP Server on Olimex ESP32-EVB, and to implement the functions for handling the broadcast from an external server which manages the local parking systems.
- A HTTP Web Server on Olimex ESP32-EVB, and to implement the functions for handling FOTA (Firmware Over-The-Air), endpoints for local system commands from the external server, and a log system.
- A physical conversion communication from RS-232 to UART, between the Scanner Datalogic Gryphon GFS4150-9 and Olimex ESP32-EVB.

Nevertheless the is home for improving, as we showed in our General Diagram there is to be implemented an individual application for writing the RFID tags, there's more to work on attaching a barrier, and differentiating the Olimex Board to act as a double agent, both as an entrance unit in the parking lot and an exit unit on the request of the admin server. We also

need to improve our project by adding a POS and establish a communication link through local IP with it.

VI. Annexes

Annex 1

```
#include <ETH.h>
#include <ESPmDNS.h>

#include <WiFi.h>
#include <WiFiClient.h>
#include <WiFiUdp.h>

#include <WebServer.h>

#include <HTTPClient.h>
#include <HTTPUpdate.h>
#include <Update.h>


#define DEVICE_CONNECTED     1
#define DEVICE_NOT_CONNECTED 0

#define AVAILABLE            1
#define BLOCKED              0

WiFiUDP UDP;

//Communication port for the running local server is 80
WebServer server(80);

unsigned int UDPPortListen     = 44398; // local port to listen on
unsigned int UDPPortSend       = 44399; // local port to send HTTP
MAC response
const    int pulseInSec        = 2000;  // was 5000
unsigned int presenceDetection = 0;

const String keyUDP           = "iPark_TNT_Server";
char         packetBuffer[255];                      // buffer to
hold incoming packet
uint8_t      replyBuffer[9]   = "Hi Luci!";          // a string to
send back

static bool deviceStatus = DEVICE_NOT_CONNECTED;
static bool ETHConnected = false;

String versio = "| iBariera ver. 1.01 | TNT Computers SRL |
www.tntcomputers.ro | office@tntcomputers.ro |";

//WiFi Requirements
//const char* ssid     = "BT_HQ1";
//const char* password = "Synapticlab01";

//Unimplemented server -> still testing...
```

```
const String serverName   = "http://192.168.1.177/cmd";
const String serverUpdate =
"http://tntcomputers.ro/iot/update/iBariere.ino.generic.bin";

//The two Olimex relays are on the following pins 32 and 33
const uint8_t PIN_OPEN_GATE  = 32;
const uint8_t PIN_CLOSE_GATE = 33;

//Input pins for detecting a car
const uint8_t PIN_INPUT_DETECTOR_ENTER = 12;
const uint8_t PIN_INPUT_DETECTOR_LEAVE = 27;

//Pins for the buttons
const uint8_t BUTTON_TICKET = 34;
const uint8_t BUTTON_INFO   = 5;

uint8_t ticketButtonStatus = 0;
uint8_t infoButtonStatus   = 0;

volatile bool validEntrance = false;
volatile bool validLeave    = false;

volatile bool ticketState = AVAILABLE;
volatile bool infoState   = AVAILABLE;

//Pins for the scanner
const uint8_t PIN_SCANNER_RX = 36;
const uint8_t PIN_SCANNER_TX =  4;

//Pins for the RFID
const uint8_t PIN_RFID_RX = 13;
const uint8_t PIN_RFID_TX = 16;

char   incomingByte;
String readBarCode, barCode;

/*
 * Login page
 */

const char* loginIndex =
 "<form name='loginForm'>"
    "<table width='20%' bgcolor='A09F9F' align='center'>"
        "<tr>"
            "<td colspan=2>"
                "<center><font size=4><b>ESP32 Login
Page</b></font></center>"
                "<br>"
            "</td>"
            "<br>"
            "<br>"
        "</tr>"
```

```
        "<td>Username:</td>"
        "<td><input type='text' size=25 name='userid'><br></td>"
        "</tr>"
        "<br>"
        "<br>"
        "<tr>"
            "<td>Password:</td>"
            "<td><input type='Password' size=25
name='pwd'><br></td>"
            "<br>"
            "<br>"
        "</tr>"
        "<tr>"
            "<td><input type='submit' onclick='check(this.form)'
value='Login'></td>"
        "</tr>"
    "</table>"
"</form>"
"<script>"
    "function check(form)"
    "{"
      "if(form.userid.value=='admin' && form.pwd.value=='admin')"
      "{"
        "window.open('/serverIndex')"
      "}"
      "else"
      "{"
        " alert('Error Password or Username')/*displays error
message*/"
      "}"
    "}"
"</script>";

/*
 * Server Index Page
 */

const char* serverIndex =
"<script
src='https://ajax.googleapis.com/ajax/libs/jquery/3.2.1/jquery.min.j
s'></script>"
"<form method='POST' action='#' enctype='multipart/form-data'
id='upload_form'>"
   "<input type='file' name='update'>"
        "<input type='submit' value='Update'>"
    "</form>"
 "<div id='prg'>progress: 0%</div>"
 "<script>"
    "$('form').submit(function(e){"
      "e.preventDefault();"
      "var form = $('#upload_form')[0];"
      "var data = new FormData(form);"
```

```
      " $.ajax({"
        "url: '/update',"
        "type: 'POST',"
        "data: data,"
        "contentType: false,"
        "processData:false,"
        "xhr: function() {"
          "var xhr = new window.XMLHttpRequest();"
          "xhr.upload.addEventListener('progress', function(evt) {"
            "if (evt.lengthComputable) {"
              "var per = evt.loaded / evt.total;"
              "$('#prg').html('progress: ' + Math.round(per*100) +
'%');"
            "}"
          "}, false);"
          "return xhr;"
        "},"
        "success:function(d, s) {"
          "console.log('success!')"
        "},"
        "error: function (a, b, c) {"
        "}"
      "});"
    "});"
 "</script>";

/*
 * giveBarCode...
 */

void giveBarCode()
{
  server.send(200, "text/plain", barCode);
  Serial.print("\n");
}

/*
 * giveClientIP...
 */

void giveClientIP()
{
  Serial.print(server.client().remoteIP());
  Serial.print("\n");
}

/*
 * getClientIp...
 */

void getClientIp()
{
```

```
  Serial.print("\nServer: I got a request from client with IP: ");

  Serial.print(server.client().remoteIP());
  Serial.print("\n");

  server.send(200, "text/plain", "Server: I got the client's IP.");
}

/*
 * blockButtons...
 */

void blockButtons()
{
  pinMode(BUTTON_TICKET, OUTPUT);
  digitalWrite(BUTTON_TICKET, HIGH);

  pinMode(BUTTON_INFO, OUTPUT);
  digitalWrite(BUTTON_INFO, HIGH);

  ticketState = BLOCKED;
  infoState   = BLOCKED;

  Serial.println("\nButtons DISABLED!");
  Serial.println("");
}

/*
 * enableButtons...
 */

void enableButtons()
{
  pinMode(BUTTON_TICKET, INPUT);
  pinMode(BUTTON_INFO, INPUT);

  ticketState = AVAILABLE;
  infoState   = AVAILABLE;

  Serial.println("\nButtons ENABLED!");
  Serial.println("");
}

/*
 * openGate...
 */

void openGate()
{
  digitalWrite(PIN_OPEN_GATE, 1);

  Serial.print("\nEndpoint: /open by: ");
```

```
  giveClientIP();

  server.send(200, "text/plain", "I opened the gate.");

  delay(pulseInSec);

  digitalWrite(PIN_OPEN_GATE, 0);

  validEntrance = true;

  blockButtons();
}

/*
 * closeGate...
 */

void closeGate()
{
    digitalWrite(PIN_CLOSE_GATE, 1);

    Serial.print("\nEndpoint: /close by:");
    giveClientIP();

    server.send(200, "text/plain", "I closed the gate.");

    delay(pulseInSec);

    digitalWrite(PIN_CLOSE_GATE, 0);

    validLeave = true;

    enableButtons();
}

/*
 * handleRoot...
 */

void handleRoot()
{
  //server.send(200, "text/plain", versio);

  server.send(200, "text/plain", ETH.localIP().toString());

  server.send(200, "text/plain", "\nServer: Resolved by my mDNS");
}

/*
 * getID...
 */
```

```
void getID()
{
  digitalWrite(PIN_OPEN_GATE, 1);

  Serial.print("\nEndpoint: /id accesed by:");
  giveClientIP();

  server.send(200, "text/plain", "Hello, you received my ID!");

  delay(pulseInSec);

  digitalWrite(PIN_OPEN_GATE, 0);
}

/*
 * handleNotFound...
 */

void handleNotFound()
{
  String message = "Command Not Found\n\n";

  message += "URI: ";
  message += server.uri();

  message += "\nMethod: ";
  message += (server.method() == HTTP_GET) ? "GET" : "POST";

  message += "\nArguments: ";
  message += server.args();
  message += "\n";

  for (uint8_t i = 0; i < server.args(); i++)
  {
    message += " " + server.argName(i) + ": " + server.arg(i) +
"\n";
  }

  server.send(404, "text/plain", message);
}

/*
 * runningUpdateFirmware...
 */

void runningUpdateFirmware()
{
   /*return index page which is stored in serverIndex */

  server.on("/", HTTP_GET,
  []()
  {
```

```
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", loginIndex);
  });

  server.on("/serverIndex", HTTP_GET,
  []()
  {
    server.sendHeader("Connection", "close");
    server.send(200, "text/html", serverIndex);
  });

  /*handling uploading firmware file */

  server.on("/update", HTTP_POST,
    []()
    {
        server.sendHeader("Connection", "close");
        server.send(200, "text/plain", (Update.hasError()) ? "FAIL"
: "OK");

        ESP.restart();
    },
    []()
    {
        HTTPUpload& upload = server.upload();

        if (upload.status == UPLOAD_FILE_START)
        {
            Serial.printf("Update: %s\n", upload.filename.c_str());

            if (!Update.begin(UPDATE_SIZE_UNKNOWN))
            {
                //start with max available size
                Update.printError(Serial);
            }
        }
        else if (upload.status == UPLOAD_FILE_WRITE)
        {
            /* flashing firmware to ESP*/
            if (Update.write(upload.buf, upload.currentSize) !=
upload.currentSize)
            {
                Update.printError(Serial);
            }
        }
        else if (upload.status == UPLOAD_FILE_END)
        {
            if (Update.end(true))
            {
                //true to set the size to the current progress
                Serial.printf("Update Success: %u\nRebooting...\n",
upload.totalSize);
```

```
            }
            else
            {
                Update.printError(Serial);
            }
        }
  });
}

/*
 * startHTTPServer...
 */

void startHTTPServer()
{
  server.on("/", handleRoot);

  //server.send(200, "text/plain", version);

  server.on("/open"    , openGate);
  server.on("/close"   , closeGate);
  server.on("/scan"    , closeGate);
  server.on("/show"    , closeGate);
  server.on("/id"      , getID);
  server.on("/clientip", giveClientIP);
  server.on("/barcode" , giveBarCode);

  server.onNotFound(handleNotFound);

  server.begin();

  Serial.println("HTTP server opened!");
}

/*
 * WiFiEvent...
 */

void WiFiEvent(WiFiEvent_t event)
{
  switch (event)
  {
    case SYSTEM_EVENT_ETH_START:
      Serial.println("ETH Started");
      //set eth hostname here
      ETH.setHostname("ESP32-Ethernet");
      break;

    case SYSTEM_EVENT_ETH_CONNECTED:
      Serial.println("ETH Connected");
      break;
```

```
    case SYSTEM_EVENT_ETH_GOT_IP:
      Serial.print("ETH MAC: ");
      Serial.print(ETH.macAddress());

      Serial.print(", IPv4: ");
      Serial.print(ETH.localIP());

      if (ETH.fullDuplex())
      {
        Serial.print(", FULL_DUPLEX");
      }
      Serial.print(", ");

      Serial.print(ETH.linkSpeed());
      Serial.println("Mbps");

      ETHConnected = true;
      break;

    case SYSTEM_EVENT_ETH_DISCONNECTED:
      Serial.println("ETH Disconnected");

      ETHConnected = false;

      deviceStatus = DEVICE_NOT_CONNECTED;
      break;

    case SYSTEM_EVENT_ETH_STOP:
      Serial.println("ETH Stopped");

      ETHConnected = false;

      deviceStatus = DEVICE_NOT_CONNECTED;
      break;

    default:
      break;
  }
}

/*
 * UDPHandling...
 */

void UDPHandling() //checks if there is an UDP broadcast and handles
it
{
  // if there's data available, read a packet
  int packetSize = UDP.parsePacket();

  if(packetSize)
  {
```

```
    Serial.print("Received packet of size ");
    Serial.println(packetSize);

    Serial.print("From ");
    IPAddress remoteIp = UDP.remoteIP();
    Serial.print(remoteIp);

    Serial.print(", port ");
    Serial.println(UDP.remotePort());

    // read the packet into packetBufffer
    int len = UDP.read(packetBuffer, 255);

    if (len > 0)
    {
      packetBuffer[len] = 0;
    }

    Serial.println("Contents:");
    Serial.println(packetBuffer);

    String packetBuff(packetBuffer); //converting to string to check
with the key of the UDP payload
    String returnAddress = UDPResponseBuild();

    if(packetBuff == keyUDP) //checks if the UDP payload is the one
expected
    {
      HTTPSendMAC(returnAddress); //sending http response with
device MAC
      deviceStatus = DEVICE_CONNECTED;
    }
  }
}

/*
 * HTTPSendMAC...
 * function that sends the MAC to the server via HTTP
 */

void HTTPSendMAC(String hostName)
{
  HTTPClient HTTP;

  Serial.print("[HTTP] begin... \n");
  HTTP.begin(hostName); //HTTP  hostName

  Serial.print("[HTTP] GET... \n");
  int HTTPCode = HTTP.GET();
  Serial.println(HTTPCode);

  if(HTTPCode > 0)
```

```
  {
    Serial.printf("[HTTP] GET... code %d\n", HTTPCode);
    if(HTTPCode == 302) //HTTP_CODE_OK
    {
      String payload = "TRĂIASCĂ POPORUL ROMÂN!";
//HTTP.getString();
      Serial.println(payload);
    }
    if(HTTPCode == HTTP_CODE_OK)
    {
      String payload = HTTP.getString(); //will be used later
      Serial.println(payload);
    }
    else
    {
      Serial.printf("[HTTP] GET... failed, error: %s\n",
HTTP.errorToString(HTTPCode).c_str()); //for 404 error
    }
    HTTP.end();
  }
  Serial.println("Here I should respond to the server/Luci!");
//debug
}

/*
 * UDPResponseBuild...
 * function that prepares the string for the HTTP response after UDP
broadcast
 */

String UDPResponseBuild()
{
  String IPServer = UDP.remoteIP().toString(); //Constructing return
address for response

  String returnAddress = "http://" + IPServer +":" + UDPPortSend +
"/Barriers/Register/?MAC=" + ETH.macAddress();

  Serial.println("The return address with endpoint as STRING is:" +
returnAddress); //debug

  return returnAddress;
}

/*
 * handleDetectionEnter...
 */

bool handleDetectionEnter()
{
  uint8_t buttonStateForEnter =
digitalRead(PIN_INPUT_DETECTOR_ENTER);
```

```
  if(buttonStateForEnter == 1)
  {
    Serial.println("\nA car is present at entrance!");
    return true;
  }
  else
  {
    Serial.println("\nWaiting for a car at entrance!");
    return false;
  }
}

/*
 * handleDetectionLeave...
 */

bool handleDetectionLeave()
{
  uint8_t buttonStateForLeave =
digitalRead(PIN_INPUT_DETECTOR_LEAVE);

  if(buttonStateForLeave == 1)
  {
    Serial.println("\nA car is present at leave!");
    return true;
  }
  else
  {
    Serial.println("\nWaiting for a car at leave!\n");
    return false;
  }
}

/*
 * handleButtonTicket...
 *
 * This button is normally closed, in this code it means that the
button
 * was pushed when his state is active low.
 *
 * His state is affected by the delay set in loop(), meaning that if
 * the button is not hold pushed down at least the delay period the
 * button changed state won't be recognized by Olimex.
 *
 * Once can change the delay period in order to make more
responsive.
 *
 * The button is responsive only if Olimex knows that a car is
waiting at entrance.
 */
```

```
void handleButtonTicket()
{
  while(handleDetectionEnter() == true)
  {
    if(ticketState == AVAILABLE)
    {
      Serial.println("Button for ticket is allowed to be pressed.");

      ticketButtonStatus = digitalRead(BUTTON_TICKET); //takes value
of the button

      if (ticketButtonStatus == LOW) // check if the input is LOW
(button pressed)
      {
        Serial.print("\nButton for ticket PRESSED!\n");
        //need to send request to server for a ticket
      }
      else
      {
        Serial.println("Button for ticket was NOT PRESSED!");
        Serial.println("");
      }
    }
    else
    {
      Serial.println("Button for ticket is NOT allowed to be
pressed.");
      Serial.println("");
    }

    break;
  }
}

/*
 * handleButtonInfo...
 *
 * This button is normally open, in this code it means that the
button
 * was pushed when his state is active high.
 *
 * His state is affected by the delay set in loop(), meaning that if
 * the button is not hold pushed down at least the delay period the
 * button changed state won't be recognized by Olimex.
 *
 * Once can change the delay period in order to make more
responsive.
 *
 * The button is responsive only if Olimex knows that a car is
waiting at entrance.
 */
```

```
void handleButtonInfo()
{
  while(handleDetectionEnter() == true) //try with if!!!!
  {
    if(infoState == AVAILABLE)
    {
      Serial.println("Button for info is allowed to be pressed.");

      infoButtonStatus = digitalRead(BUTTON_INFO); //takes value of
the button

      if (infoButtonStatus == HIGH)// check if the input is LOW
(button pressed)
      {
        Serial.print("\nButton for info PRESSED, info REQUIRED!\n");

        server.send(200, "text/plain", "Request for information!");
      }
      else
      {
        Serial.println("Button for info was NOT PRESSED!");
        Serial.println("");
      }
    }
    else
    {
      Serial.println("Button for info is NOT allowed to be
pressed.");
      Serial.println("");
    }

    break;
  }
}

/*
 * handleScanner...
 */

void handleScanner()
{
  while(Serial1.available())
  {
    incomingByte = Serial1.read();

    readBarCode += incomingByte;

    if(incomingByte == '\r')
    {
      Serial.println("\nOlimex: I got the Scanner Bar Code --> ");
      Serial.print(readBarCode + "\n\n");
      Serial.flush();
```

```
      barCode = readBarCode;

      Serial.print(barCode + "\n\n");
      readBarCode = "";
    }
  }

}


void handleRFID()
{
  while(Serial2.available())
  {
    incomingByte = Serial2.read();

    readBarCode += incomingByte;

    if(incomingByte == '\r')
    {
      Serial.println("\nOlimex: I got the RFID Bar Code --> ");
      Serial.print(readBarCode + "\n\n");
      Serial.flush();

      barCode = readBarCode;

      Serial.print(barCode + "\n\n");
      readBarCode = "";
    }
  }

}


void setup()
{
  Serial.begin(9600);

  //Serial communication through Olimex and the Scanner
  Serial1.begin(2400, SERIAL_8N1, PIN_SCANNER_RX, PIN_SCANNER_TX);

  //Serial communication through Olimex and the RFID
  Serial2.begin(57600, SERIAL_8N1, PIN_RFID_RX, PIN_RFID_TX);

  pinMode(PIN_CLOSE_GATE , OUTPUT);
  digitalWrite(PIN_CLOSE_GATE , 0);

  pinMode(PIN_OPEN_GATE  , OUTPUT);
  digitalWrite(PIN_OPEN_GATE  , 0);

  pinMode(PIN_INPUT_DETECTOR_ENTER , INPUT);
```

```
  pinMode(PIN_INPUT_DETECTOR_LEAVE , INPUT);

  pinMode(BUTTON_TICKET , INPUT);
  pinMode(BUTTON_INFO   , INPUT);

  runningUpdateFirmware();

  Serial.print("IP Address: ");
  Serial.println(ETH.localIP());

  Serial.println("MAC Address: ");
  Serial.println(ETH.macAddress());

  WiFi.onEvent(WiFiEvent);

  ETH.begin();

  startHTTPServer();

  UDP.begin(UDPPortListen);

  Serial.println("Finished the setup!");
}


void loop()
{
  server.handleClient();

  handleButtonTicket();
  handleButtonInfo();

  handleScanner();

  handleRFID();

  handleDetectionLeave();

  if (ETHConnected)
  {
    if(deviceStatus == DEVICE_NOT_CONNECTED)
    {
      UDPHandling();
    }
  }

  Serial.println("My IP is: ...");
  Serial.println(ETH.localIP());

  delay(1000);
}
```

Annex 2

```
uint8_t PIN  = 5;

char toogleChar;


void toogleManual()
{
  toogleChar = Serial.read();

  if(toogleChar == 'a')
  {
    digitalWrite(PIN, HIGH);
  }

  if(toogleChar == 's')
  {
    digitalWrite(PIN, LOW);
  }
}



void setup()
{
  Serial.begin(9600);

  pinMode(PIN, INPUT);
  //digitalWrite(PIN, 0);


  int buttonState  = digitalRead(PIN);

  Serial.println("Pinul de intrare are valoarea: ");
  Serial.println(buttonState);
}


void loop()
{
  //for testing
  toogleManual();

  Serial.println(digitalRead(PIN));

  delay(2000);
}
```

```
#include <HardwareSerial.h>

//HardwareSerial Serial1(1);

void setup()
{
  Serial2.begin(57600,SERIAL_8N1, 13, 16);    //Baud rate, parity
mode, RX, TX //RFID
  Serial.begin(9600);
  Serial1.begin(9600,SERIAL_8N1, 36, 4); //Scanner
}

char ch;
char bl;

void loop()
{
    if(Serial1.available())
  {
       ch = Serial1.read();
       Serial.write(ch);        //Show data on Serial Monitor
       Serial.flush();                        //Flush port
       //Serial.println("SERIAL 1: Serial1 AVAILABLE!");
  }

  if(Serial2.available())
  {
       ch = Serial2.read();
       Serial.write(ch);        //Show data on Serial Monitor
       Serial.flush();                        //Flush port
       //Serial.println("SERIAL 1: Serial1 AVAILABLE!");
  }
}
```

VII. Bibliography

1. **Explore Embedded.** Overview of ESP32 features. What do they practically mean? . *Exploreembedded.com.* [Online] Explore Embedded. [Cited: 11 5, 2019.] https://www.exploreembedded.com/wiki/Overview_of_ESP32_features._What_do_they_practically_mean%3F.

2. **Espressif Systems.** Espressif Announces the Launch of ESP32 Cloud on Chip and Funding by Fosun Group. *Espressif.com.* [Online] Espressif Systems, 09 07, 2016.

[Cited: 11 05, 2019.] https://www.espressif.com/en/media_overview/news/espressif-announces-launch-esp32-cloud-chip-and-funding-fosun-group.

3. **Harizanov, Martin.** ESP32. *Harizanov.com.* [Online] 12 18, 2015. [Cited: 11 05, 2019.] https://harizanov.com/2015/12/esp32/.

4. **Espressif Systems.** ESP32 Series Datasheet. *Espressif.com.* [Online] 3.2, 2019. [Cited: 11 05, 2019.] https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

5. —. ESP32 Technical Reference Manual. *Espressif.com.* [Online] 4, 2018. [Cited: 11 05, 2019.] https://www.espressif.com/sites/default/files/documentation/esp32_technical_reference_manual_en.pdf.

6. **Jimblom.** Enginursday: First Impressions of the ESP32. An overview of Espressif's sequel to the ESP8266 - a new WiFi/Bluetooth-enabled system-on-chip, with a massive GPIO count. [Online] Sparkfun.com, 01 21, 2016. [Cited: 11 05, 2019.] https://www.sparkfun.com/news/2017.

7. *Comparative Analysis and Practical Implementation of the ESP32 Microcontroller Module for the Internet of Things.* **Alexander Maier, Andrew Sharp, Yuriy Vagapov.** Wrexham : IEEE, 2017. 2017 Internet Technologies and Applications (ITA). pp. 143-4.

8. *RIOT: an Open Source Operating System for Low-end Embedded Devices in the IoT.* **Emmanuel Baccelli, Cenk Gündoğan, Oliver Hahm, Peter Kietzmann, Martine S. Lenders, Hauke Petersen, Kaspar Schleiser, Thomas C. Schmidt, and Matthias Wählisch.** 6, s.l. : IEEE, December 2018, IEEE Internet of Things Journal, Vol. 5, p. 4431.

9. **Bush, Steve.** Updated: Arduino announces FPGA board, ATmega4809 in Uno Wi-Fi mk2, cloud-based IDE and IoT hardware. [Online] Electronicsweekly.com, 05 18, 2018. [Cited: 11 05, 2019.] https://www.electronicsweekly.com/news/products/bus-systems-sbcs/arduino-announced-fpga-board-new-atmega-uno-wi-fi-2018-05/.

10. **Schorcht, Gunar.** Olimex ESP32-EVB. [Online] Riot-os.org. [Cited: 11 05, 2019.] https://riot-os.org/api/group__boards__esp32__olimex-esp32-evb.html.

11. **Owner of Site.** Datalogic Gryphon GFS41XX Quick Reference Manual Page 9. [Online] Manualslib.com. [Cited: 11 05, 2019.] https://www.manualslib.com/manual/908500/Datalogic-Gryphon-Gfs41xx.html?page=9#manual.

12. **Olimex Ltd.** Olimex MOD-RFID1356 User's Manual. *Olimex.com.* [Online] Revision C, 06 2015. [Cited: 11 05, 2019.] https://www.olimex.com/Products/Modules/RFID/MOD-RFID1356-BOX/resources/MOD-RFID1356-BOX.pdf.

13. **Tianjin G-TEK Automation Technology Co., Ltd.** Binary- Channel Vehicle Detector. [Online] Tianjin G-TEK Automation Technology Co., Ltd, 2013. [Cited: 11 05, 2019.] http://www.gtekautomation.com/eshoplist_jc.asp?id=6&cid=58.