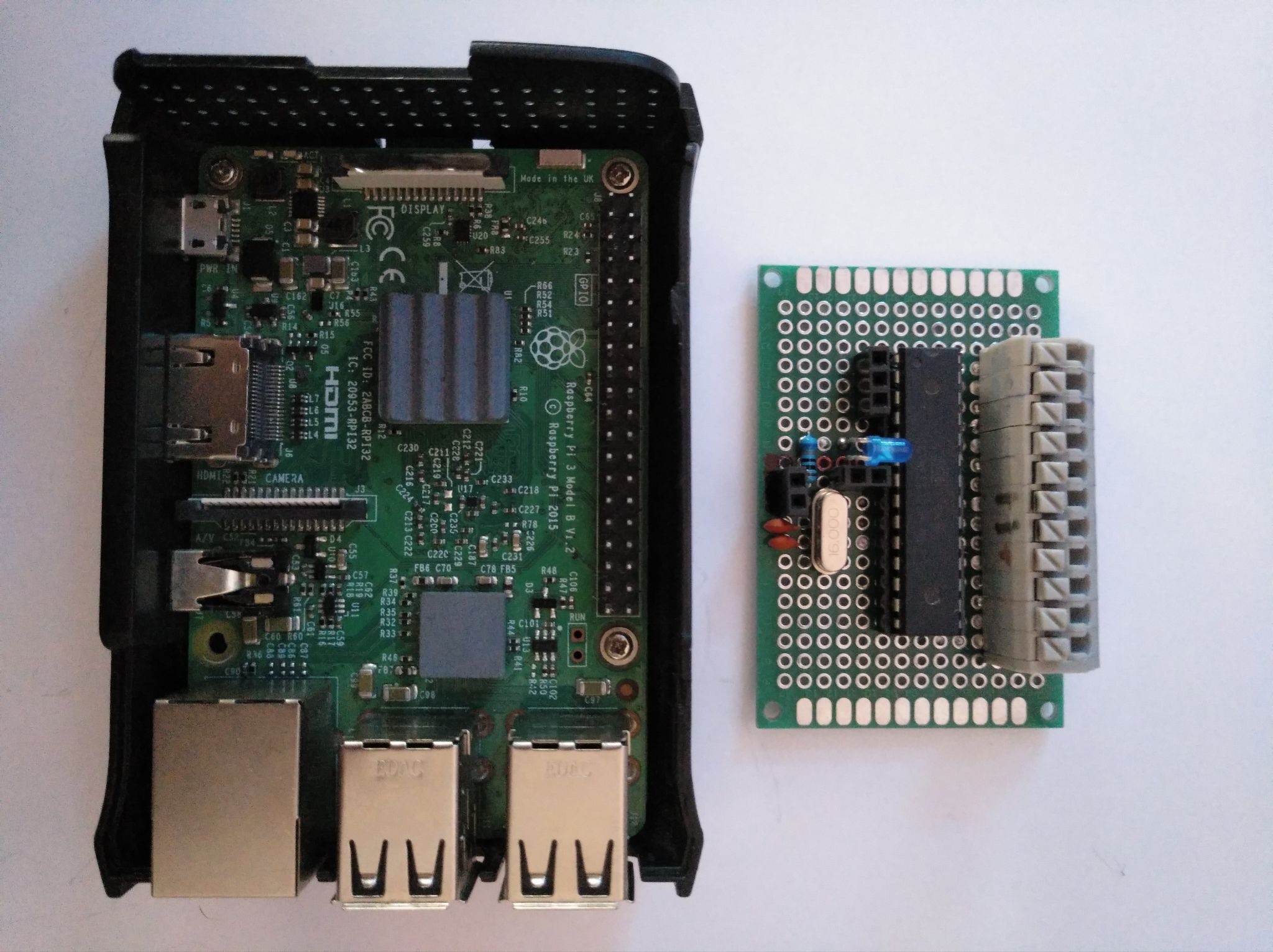
# 

# **DOCUMENTACIÓN - BLUECITY - IES EL RINCÓN\_VERSION**

**--Parking With RaspberryPi / ATMEGA328P BOX--**



Fecha: 06/09/2021

Email: [azan3t@gmail.com](mailto:azan3t@gmail.com)

*ÍNDICE:*

[**DOCUMENTACIÓN - BLUECITY - IES EL RINCÓN\_VERSION**](#_u924swc0j1ts) **1**

[**ABSTRACT**](#_5rl5iot1xnj7) **5**

[**1-.DESCRIPCIÓN DEL PROYECTO**](#_x1zhykn1n1wu) **6**

[1.1-. Representación del HARDWARE en el Proyecto](#_hao9u21ndvbe) 6

[**2-.COMPOSICIÓN DEL SOFTWARE BLUECITY-EL RINCÓN VERSIÓN**](#_1nbdsrt9et1s) **7**

[2.1-. Información de “README.md”](#_dc1z9wp1z37) 7

[2.2-. Información de “backend”](#_pr08htbaxpc7) 7

[2.3-. Información de “frontend”](#_rctgyps04bij) 7

[2.4-. Información de “box\_raspberry\_atmega”](#_ts7ngqe57xa6) 8

[2.4.1-. Información de “sketch\_Atmega\_BOX.ino”](#_yywf3cq6wv65) 8

[2.4.2-. Información de “NodeRed\_AtmegaBOX\_BLUECITY.json”](#_11eg5qmzyql6) 8

[2.5-. Información de “box\_simulator\_backend”](#_k0stthhnhkoo) 9

[2.6-. Representación de las partes SOFTWARE en el HARDWARE](#_uadqnvjgyx5n) 9

[**3-. PROTOCOLOS EMPLEADOS PARA LA COMUNICACIÓN**](#_dnv0p87zmbqw) **10**

[3.1-. Explicación e Implementación de I2C](#_zbm8c9ui2tor) 10

[3.1.1-. Representación de Conexión I2C en nuestro proyecto](#_743n0cj8nrzm) 10

[3.2-.Explicación e Implementación de WebSockets](#_qihv3lt3hxup) 11

[3.2.1-. Representación de los WebSocket sobre el proyecto](#_ynvxwq3p6na7) 11

[**4-.COMUNICACIÓN ENTRE LAS APLICACIONES DEL PROYECTO**](#_w6gryupej00y) **12**

[4.1-. Diagrama de Comunicación entre las Aplicaciones](#_gqg5jq9p6tz6) 13

[4.2-. Comunicación del Frontend con el Backend](#_f6wv1oboczbj) 14

[4.3-. Comunicación del Backend con Box\_Simulator\_Backend](#_lxdtsjol5j4z) 15

[4.4-. Comunicación de Box\_Simulator\_Backend con FLOW-NodeRed](#_j57ro8ijibsy) 16

[4.5-. Comunicación de FLOW-NodeRed con ATMEGA328P](#_31n2hio94vj3) 17

[4.5.1-. Representación del FLOW-NodeRed (Comunicación WebSocket e I2C)](#_682i7nqm7t3e) 18

[4.6-. Funcionamiento del ATMEGA328P](#_xmtpz7emz49v) 19

[4.6.1-. Listado de “COMMANDS” para ATMEGA328P](#_bes3ef6iyk8v) 21

[4.6.2-. Listado de “StatusCode” de ATMEGA328P](#_iifcq3teaq8y) 22

[**5-. DESPLIEGUE E INSTALACIÓN DEL SOFTWARE DEL PROYECTO**](#_91pr82nntjn7) **24**

[5.1-. Instalando “frontend” y “backend” en el Servidor](#_71bkoqokiybv) 24

[5.2-.Instalando Entorno y “FLOW-NodeRed” en RaspberryPi](#_ydrwft7jaayx) 25

[5.2.1-.En caso de NO TENER Sistema Operativo en RaspberryPi](#_hjozzj3tuc11) 25

[5.2.2-.En caso de SI TENER Sistema Operativo en RaspberryPi](#_oyiupqb4jf1k) 25

[5.3-. Instalando “Box\_Simulator\_Backend” en RaspberryPi](#_to7gj5ls4spv) 26

[5.4-. Cargar “sketch” a ATMEGA328P desde RaspberryPi](#_rg4m2yc94mc6) 27

[**6-. MATERIALES NECESARIOS DE LA PARTE FÍSICA DEL PROYECTO**](#_l12wr4n0ptw6) **28**

[6.1-. Componentes para “la base” del Parking](#_x9xhwfl0d67u) 28

[6.2-. Componentes para “Parte Física” de UN “BOX”](#_1755uvb2qasj) 28

[6.3-. Componentes para “el Cuadro Eléctrico” de UN “BOX”](#_t27mxji7599h) 29

[6.4-. Cálculo de consumo de la instalación para UN BOX](#_koy3cxs223qh) 30

[6.4.1-. Cálculo de consumo para componentes de 5V de UN BOX](#_8c1d6zc3i32e) 30

[6.4.2-. Cálculo de consumo para componentes de 24V de UN BOX](#_cupfn8hlrxck) 30

[6.4.3-. Cálculo de consumo Máximo “Total” de la instalación](#_ee91ajsoi4f8) 31

[6.4.3.1-. Cálculo de consumo Máximo CON Patineta en carga](#_ruh6pj886sy9) 31

[6.4.3.2-. Cálculo de consumo Máximo SIN Patineta en carga](#_cteih7fxyqtz) 31

[6.5-. Determinando “Convertidores CC” según Consumo Calculado](#_g3cl5msvi45p) 32

[6.6-. Componentes para Instalación Fotovoltaica de 12V](#_x4wnykweafta) 32

[**7-.PLANOS DE MONTAJE COMPLETO DEL PARKING**](#_f6kcx147e0bf) **33**

[7.1-. Montaje de Instalación Fotovoltaica a Cuadro eléctrico](#_35ecb7r90kfj) 34

[7.2-. Montaje de la PCB con el ATMEGA328P](#_38zwsi379jig) 35

[7.3-. Conexiones de Raspberry, ATMEGA328P y sus módulos](#_ryrmhxopqnke) 36

[7.4-. Conexiones de “BOX Físico” a Bornes del Cuadro](#_k12z6xw6phws) 37

[7.5-. Plano de Conexión Completo de Cuadro y Box](#_l5xn0qsy242k) 38

# ABSTRACT

El proyecto consiste en una red de “Parkings sostenibles y autosuficientes” para patinetas eléctricas, que son gestionados por un Servidor Web y ofrecen su servicio vía Aplicación Web.

Cada “Parking” cuenta con una instalación fotovoltaica de la cual se nutren los distintos componentes de la instalación y almacena la energía sobrante para utilizarla cuando sea necesario.

Cada uno de estos “Parkings” cuenta con uno o varios “Boxes” que permitirán el estacionamiento y carga de las patinetas eléctricas.

Este proyecto, **es una rama** del proyecto original “Bluecity” en el cuál colaboramos con varios institutos de la Península.

Nuestro instituto (IES El Rincón) nos encargamos de crear la APP con NodeJS y ReactJS(Backend y Frontend) para la gestión y uso de esta Red de Parkings.

Los demás institutos realizaron el diseño y montaje del “Box Físico”,y el diseño y montaje de la parte eléctrica y electrónica de este, en definitiva, todo lo perteneciente al cuadro eléctrico y programación del PLC.

El “Box” fue realizado para trabajar con un **PLC**(Controlador Lógico Programable) y con una **instalación fotovoltaica a 24V**.

El proyecto original fue presentado y posteriormente expuesto en el Museo Elder.

Después de esto, **el equipo del IES El Rincón** nos propusimos mejorar el proyecto para hacerlo más económico, escalable y fácil de mantener.

# 1-.DESCRIPCIÓN DEL PROYECTO

Para nuestra versión del proyecto, únicamente hemos reutilizado el “Box físico” del proyecto original y hemos rediseñado tanto la instalación fotovoltaica como todo el cuadro eléctrico, dotándolo de componentes más económicos y que se ajustan al trabajo real que necesitamos realizar.

**En la RAMA Rediseñada y realizada por el IES El Rincón:**

* Hemos reutilizado el “Box físico” del proyecto original.
* Se ha rediseñado la instalación fotovoltaica para trabajar a 12V.
* Se ha rediseñado la parte electrónica del parking, en concreto todo el cuadro eléctrico:

-- **Suprimimos el PLC** que cuenta con un coste inicial de 1500€ para 3 “Boxes” y con un coste de 1000€ más por cada 4 “Boxes” que se deseen agregar. Además de precisar de un ordenador como intermediario para la comunicación con el Servidor Central, que también queda suprimido.

-- **Agregamos una RaspberryPi-3B** un ordenador de bajo coste apenas 50€, capaz de controlar hasta 127 “Boxes” y un chip microcontrolador ATMEGA328P junto con un módulo de 6 relés, y 2 optoacopladores, estos componentes suponen un costo de menor de 20€ de coste para cada “Box” que necesitemos, abaratando el coste de forma exponencial y el consumo eléctrico.

* Se ha automatizado el añadido de “Boxes”, solo teniendo que dar de alta en la Base de Datos y conectar el ATMEGA328P programado con su Dirección correcta a tomar en el “Parking” correspondiente.

Por lo que mejora el mantenimiento y la escalabilidad de la Red.

## 1.1-. Representación del HARDWARE en el Proyecto



# 2-.COMPOSICIÓN DEL SOFTWARE BLUECITY-EL RINCÓN VERSIÓN

El proyecto con proceso de instalación se encuentra en el repositorio:

<https://github.com/azanet/bluecity-test/tree/feature/atmega>

Para esta versión necesitaremos 6 de las partes de las que existen en el repositorio de Github y que explicaremos posteriormente.

Estas son:

- *“README.md” (Instrucciones de instalación)*

* “**backend**”

- “**frontend**”

+ “box\_raspberry\_atmega”

|- **NodeRed\_AtmegaBOX\_BLUECITY.json**

**|**- **sketch\_Atmega\_BOX.ino**

- “**box\_simulator\_backend**”

## 2.1-. Información de “README.md”

El “**README.md”** contiene las instrucciones a seguir para el proceso de instalación y despliegue del “backend”, “frontend” y “box\_simulator\_backend”.

## 2.2-. Información de “backend”

Para la programación del “backend” se ha utilizado Node JS.

El backend es “el cerebro” de nuestra aplicación web, está a la escucha de órdenes por parte de los usuarios, envía órdenes hacia los “Parkings”, escucha las respuestas de estos y determina la escritura en la Base de Datos.

El Backend cuenta con un Server Websocket a través de la cual se realizan las distintas comunicaciones (entre el “frontend” y el “box\_simulator\_backend”).

Esta parte de la aplicación, se encontrará en el Servidor expuesto a Internet.

## 2.3-. Información de “frontend”

Para la programación del “frontend” se ha utilizado ReactJS.

El frontend es “la parte gráfica” de nuestra aplicación web, es decir, es la parte que el usuario vé y a través de la cual interactúa con la Red de Parkings.

El frontend se comunica a través de un Cliente Websocket con el Backend.

Esta parte de la aplicación, se encontrará en el Servidor expuesto a Internet.

## 2.4-. Información de “box\_raspberry\_atmega”

Esta parte tiene todo lo necesario para hacer uso de la versión de “Parking” con la RaspberryPi y los ATMEGA328P.

Contiene el código con el que debe ser programado el ATMEGA328P y una aplicación (o flujo de trabajo) para NodeRed al cual llamaremos **FLOW-NodeRed** para referirnos a este a partir de ahora, esta aplicación se encarga de la comunicarse y operar con el ATMEGA328P.

Como se comentó en esta carpeta nos encontraremos con dos archivos:

* “**sketch\_Atmega\_BOX.ino**”
* “**NodeRed\_AtmegaBOX\_BLUECITY.json**”

Los cuales explicaremos a continuación.

## 2.4.1-. Información de “sketch\_Atmega\_BOX.ino”

Este es el código de nuestro chip microcontrolador ATMEGA328P.

Se ha desarrollado empleando el lenguaje de programación de Arduino.

Este lenguaje no es un C++ puro sino que es una adaptación proveniente de avr-libc que provee de una librería de C de alta calidad para usar con GCC (compilador de C y C++) en los microcontroladores AVR de Atmel.

Este dispositivo ATMEGA328P una vez programado, contará con una serie de comandos que esperará recibir y contestará con un código de estado, ambos (command y StatusCode) detallados más adelante en este documento.

Este programa se encontrará alojado únicamente en los chips ATMEGA328P.

## 2.4.2-. Información de “NodeRed\_AtmegaBOX\_BLUECITY.json”

Esta parte del proyecto, se desarrolló utilizando NodeRed.

NodeRed es un motor de flujos con enfoque IoT, que permite definir gráficamente flujos de servicios, a través de protocolos estándares.

El NodeRed\_AtmegaBOX\_BLUECITY.json es un FLOW o flujo de trabajo que debemos importar a NodeRed para que funcione.

Esta aplicación, es la que conecta directamente con el ATMEGA328P.

También cuenta con un Servidor Websocket, para permitir la comunicación y poder ser operado fácilmente desde cualquier otra aplicación.

Esta aplicación esperará a que un Cliente WebSocket se conecte y envíe una serie de comandos dirigidos a un “Box” determinado.

La aplicación enviará la orden al ATMEGA328P correspondiente al “Box” indicado y esperará el “StatusCode” que responderá el ATMEGA328P.

En base a este “StatusCode” El FLOW-NodeRed determinará si el ATMEGA debe seguir con algún proceso, o finalmente responde con el “StatusCode” al Cliente-WS.

Esta parte de la aplicación, se encontrará en la RaspberryPi de cada “Parking”.

## 2.5-. Información de “box\_simulator\_backend”

Para desarrollar el “box\_simulator\_backend” se ha utilizado Node JS.

El “box\_simulator\_backend” es una aplicación que hace que la Raspberry se convierta en un “Parking”.

Esta aplicación dotará al “Parking” con una ID que debe ser única para cada uno.

También actúa como intermediario **entre** el **Backend del Server Central** de nuestra aplicación web **y** **el FLOW-NodeRed** encargado de comunicarse y operar con los ATMEGA328P (Los “Boxes” del “Parking”).

Realiza la comunicación hacia el Backend a través de un Cliente Websocket que se conecta al Servidor Websocket que ejecuta el Backend.

Y Realiza la comunicación hacia el FLOW-NodeRed a través de un Cliente Websocket que se conecta al Servidor Websocket que ejecuta nuestro FLOW-NodeRed.

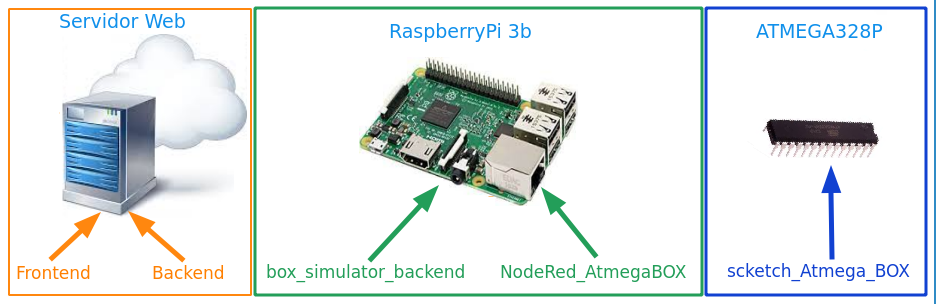
Esta aplicación determina si la dirección del “Parking” se corresponde con el “Parking” con el que el Servidor Central quiere comunicarse.

En caso de ser el Parking deseado, prepara el mensaje para enviarle al FLOW-NodeRed, que se comunica directamente con los ATMEGA328P(los “Boxes”).

Esta parte de la aplicación, se encontrará en la RaspberryPi de cada “Parking”.

## 2.6-. Representación de las partes SOFTWARE en el HARDWARE

Se mostrará el Hardware sobre el que instalaremos las distintas partes de nuestra aplicación. especificando dónde va cada parte del software.



# 3-. PROTOCOLOS EMPLEADOS PARA LA COMUNICACIÓN

Para la comunicación de las distintas partes de nuestro proyecto hemos utilizado los protocolos **I2C** y **WebSocket**.

Se explicará:

**- El protocolo I2C empleado para comunicar RaspberryPi con el ATMEGA328P.**

**- El protocolo WebSocket empleado para comunicar el frontend, backend, box\_simulator\_backend y el FLOW-NodeRed.**

## 3.1-. Explicación e Implementación de I2C

Para la comunicación entre RaspberryPi y ATMEGA328P se ha utilizado el **protocolo I2C,** ya que estos dos dispositivos disponen de este puerto/protocolo**.**

**I2C** es un puerto y protocolo de comunicación serial, define la trama de datos y las conexiones físicas para transferir bits entre 2 dispositivos digitales.

El puerto incluye dos cables de comunicación, SDA y SCL.

Además el protocolo permite conectar hasta 127 dispositivos esclavos con esas dos líneas. También es conocido como IIC ó TWI – Two Wire Interface.

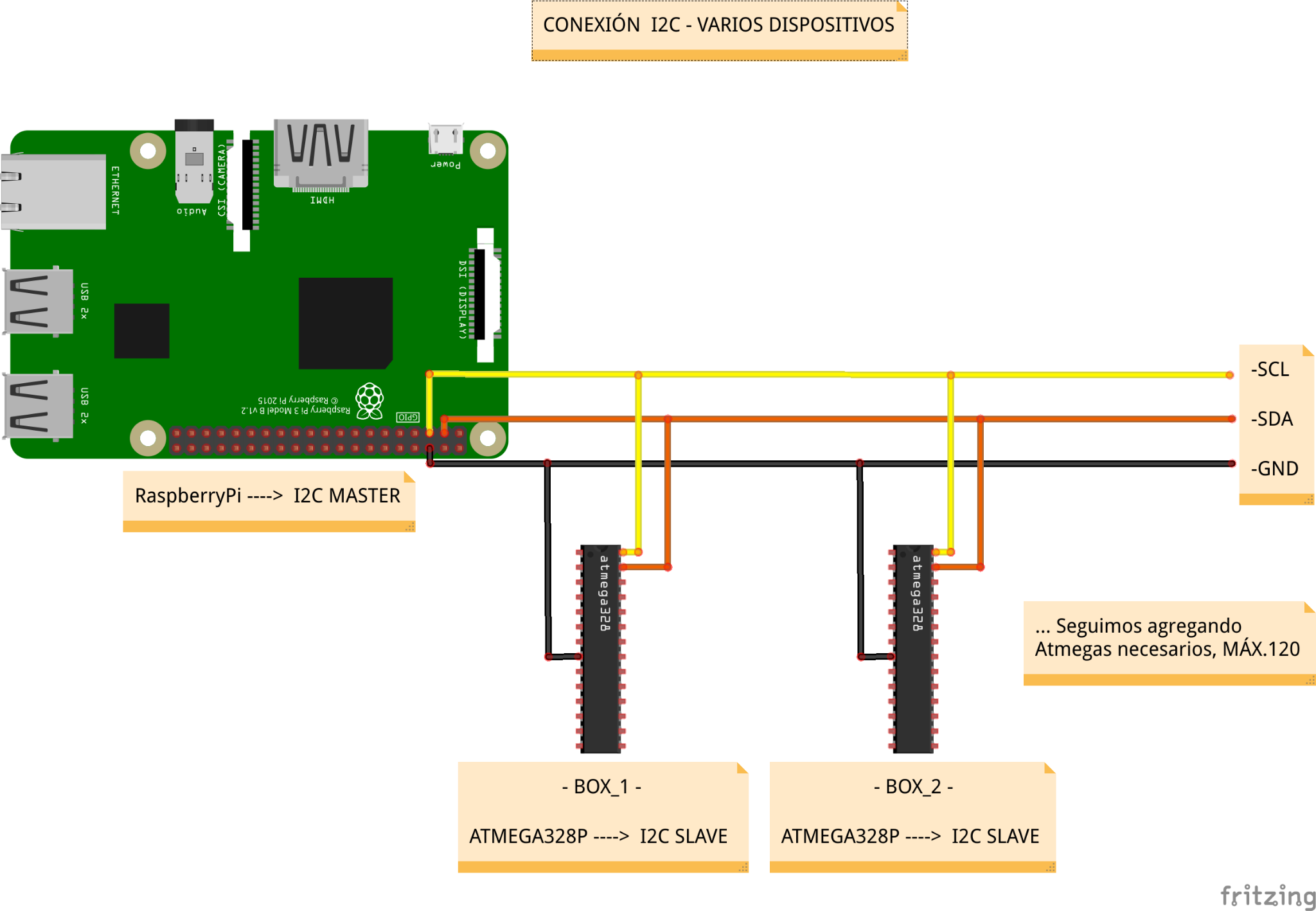
En una comunicación digital, como en todas las comunicaciones, existen distintos dispositivos o elementos.

En el caso de **I2C** se diferencian dos elementos básicos, un **MAESTRO** y un **ESCLAVO**.

*Más información sobre I2C:* <https://hetpro-store.com/TUTORIALES/i2c/>

En nuestro caso, el **MAESTRO será la RaspberryPi** y **los ESCLAVOS los ATMEGA328P.**

## 3.1.1-. Representación de Conexión I2C en nuestro proyecto



## 3.2-.Explicación e Implementación de WebSockets

Para comunicar las diferentes partes de nuestro proyecto, frontend, backend, box\_simulator\_backend y el FLOW-NodeRed. Se ha utilizado el protocolo WebSocket, ya que facilita y agiliza la comunicación entre las partes.

**WebSocket** es un protocolo de red basado en TCP que establece cómo deben intercambiarse datos entre redes. Puesto que es un protocolo fiable y eficiente, es utilizado por prácticamente todos los clientes.

El protocolo TCP establece conexiones entre dos **puntos finales de comunicación**, llamados ***sockets***. De esta manera, el intercambio de datos puede producirse en las dos direcciones al mismo tiempo(**LA COMUNICACIÓN ES BIDIRECCIONAL**).

La **ventaja** de este intercambio es que **se accede de forma más rápida a los datos** y la información que se solicita se muestra *en tiempo real*.

WebSocket permite así una comunicación directa entre un cliente Websocket y un servidor WebSocket.

*Más información sobre WebSocket:*

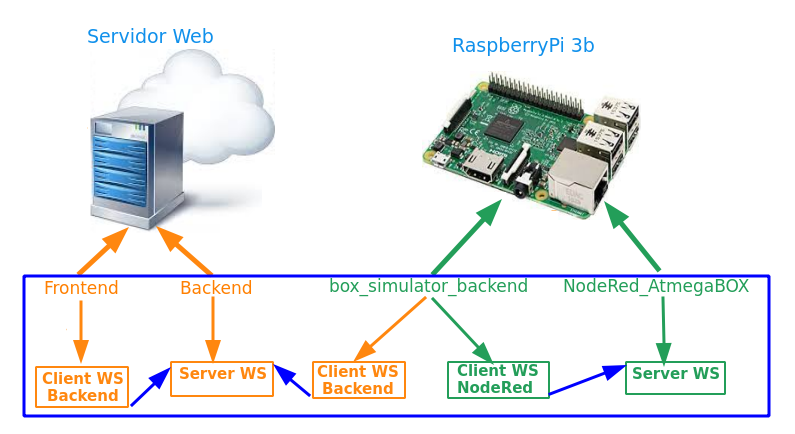
<https://www.ionos.es/digitalguide/paginas-web/desarrollo-web/que-es-websocket/>

Nuestro proyecto cuenta con un total de 2 Servidores WS(Websocket) y varios Clientes que se conectan a estos.

Se indican los Servidores o Clientes WS qué implementa cada parte del proyecto:

* **Backend** ⇒ **Servidor-WS**.
* **Frontend** ⇒ **Client-WS a Backend**.
* **Box\_simulator\_backend** ⇒ **Client-WS a Backend y Client-WS al FLOW-NodeRed**.
* **FLOW-NodeRed** ⇒ **Servidor-WS**.

## 3.2.1-. Representación de los WebSocket sobre el proyecto



# 4-.COMUNICACIÓN ENTRE LAS APLICACIONES DEL PROYECTO

Comenzaremos explicando la comunicación que realiza el proyecto partiendo desde el frontend, la parte que ve y maneja el usuario.

Iniciaremos con la explicación de la comunicación entre el “Frontend” y “Backend” (“la vista” y “el cerebro” de la APP), pasando por los “intermediarios” hasta llegar al “ATMEGA328P”, que es el encargado de operar directamente con los distintos elementos del “Box Físico”.

Se detalla:

**- Cómo el “frontend” se comunica con el “backend”.**

**- Cómo “box\_simulator\_backend” se comunica con el “backend” del Server Central.**

**- Cómo el “box\_simulator\_backend” se comunica con el FLOW-NodeRed.**

**- Cómo opera el FLOW-NodeRed con los ATMEGA328P.**

**- Cómo trabaja el ATMEGA328P.**

**- Ejemplo en conjunto de Usuario reservando el “BOX” de un “Parking”.**

## 

En la siguiente página, se adjunta un Diagrama con el flujo de la comunicación entre las distintas Aplicaciones.

El Diagrama complementa las explicaciones de los apartados que se encuentran dentro de este punto.

## 

## 

## 4.1-. Diagrama de Comunicación entre las Aplicaciones

## 4.2-. Comunicación del Frontend con el Backend

La comunicación entre “Frontend” y ”Backend”, como indicamos en el punto 3.2 se realiza a través de WebSocket(WS).

El “Backend” utiliza su Servidor-WS y el “Frontend” implementa un Cliente-WS que se conecta al Server-Ws del “Backend”.

**- El Backend cuenta con un Servidor-WS**, este Servidor-WS cuenta con unos “sockets” que fueron definidos por nosotros y a través de los cuales RECIBE o EMITE información a los distintos Clientes-WS que se conecten a este Servidor-WS y estén ESCUCHANDO o EMITIENDO a través de esos “sockets” que están definidos.

- **El Frontend cuenta con un Cliente-WS que conecta al Servidor-WS del Backend,** este Cliente-WS está a la escucha por unos “sockets” concretos que permiten que la página reaccione y muestre información en tiempo real sobre el estado de los “Parkings” y sus “Boxes”.

Cualquier cambio que se produzca en la BBDD o por alguna función definida en el Servidor, el Servidor-WS emitirá a través de un “socket” al Cliente-WS la orden de refrescar la página junto con la información que debe ser actualizada.

Los cambios en la BBDD normalmente vienen producidos por “un Usuario” que está operando desde su DISPOSITIVO con la Aplicación Web(el Frontend).

El usuario al realizar una reserva, cancelar, etc.., desde el Frontend el Cliente-WS de este envía a través del “socket” que le corresponda información hacia el Servidor-WS del Backend.

Cuando el Servidor WS del Backend recibe el mensaje del Cliente WS en uno de sus “sockets” se realizará en el Backend las operaciones que implemente dicho “socket”, si es escribir en la BBDD escribirá y probablemente el Server-WS emita otro mensaje a través de un “socket” concreto a los Clientes-WS para informarles del cambio.

## 4.3-. Comunicación del Backend con Box\_Simulator\_Backend

La comunicación entre “Backend” y ”Box\_Simulator\_Backend”,también se realiza a través de WebSocket(WS).

El “Backend” utiliza su Servidor-WS y el “Box\_Simulator\_Backend” implementa un Cliente-WS que se conecta al Server-Ws del “Backend”.

**- El Backend cuenta con un Servidor-WS**, este Servidor-WS cuenta con unos “sockets” que fueron definidos por nosotros y a través de los cuales RECIBE o EMITE información a los distintos Clientes-WS que se conecten a este Servidor-WS y estén ESCUCHANDO o EMITIENDO a través de esos “sockets” que están definidos.

- **El ”Box\_Simulator\_Backend” cuenta con un Cliente-WS que conecta al Servidor-WS del Backend.**

Este Cliente-WS está a la escucha por unos “sockets” concretos que reciben la información desde el Backend, a consecuencia en mayor parte de un Usuario que ha operado desde el Frontend y ha provocado un cambio en la BBDD o por alguna función definida en el Servidor.

En ese caso, el Backend emitirá a través de un “socket” al Cliente-WS la información que necesita para que actúe sobre el “BOX” de un “Parking” concreto.

El ”Box\_Simulator\_Backend” Recibe por su Cliente-WS desde el “Backend” la información que necesita para trabajar con un “BOX”.

Analiza la información recibida para determinar si es él “Parking Solicitado”.

Si este ”Box\_Simulator\_Backend” es el “Parking Solicitado”, seguirá analizando la información para determinar que “BOX” es demandado y que “COMMAND” debe mandar (a través de Websocket) al FLOW-NodeRed, para que este a su vez se lo haga llegar al ATMEGA.

El “FLOW-NodeRed” Responderá (a través de WebSockets) al ”Box\_Simulator\_Backend” con un mensaje indicando la “Address del BOX” y un “StatusCode”.

El ”Box\_Simulator\_Backend” analizará el “StatusCode” recibido para determinar qué información tiene que devolver al “Backend” a través del Cliente-WS que apunta a este, y envía a través del “socket” que le corresponda información hacia el Servidor-WS del Backend.

El Servidor-WS del Backend recibe por “un socket” el mensaje enviado desde el ”Box\_Simulator\_Backend” y realiza las operaciones que implementa dicho “socket”, si es escribir en la BBDD escribirá y probablemente el Server-WS emita otro mensaje a través de un “socket” concreto a los Clientes-WS para informarles del cambio.

## 4.4-. Comunicación de Box\_Simulator\_Backend con FLOW-NodeRed

La comunicación entre ”Box\_Simulator\_Backend” y “FLOW-NodeRed”, también se realiza a través de WebSocket(WS).

El “FLOW-NodeRed” implementa un Servidor-WS y el “Box\_Simulator\_Backend” implementa un Cliente-WS que se conecta al Server-Ws del “FLOW\_NodeRed”.

**- El FLOW-NodeRed cuenta con un Servidor-WS**, este Servidor-WS cuenta con un solo “socket” a través del cual RECIBE o EMITE información a los Clientes-WS que se conecten a este Servidor-WS y estén ESCUCHANDO o EMITIENDO a través de este “socket”.

- **El ”Box\_Simulator\_Backend” cuenta con DOS Cliente-WS, uno que conecta al Servidor-WS del “FLOW-NodeRed” y conecta con el Servidor-WS del “Backend”.**

El Cliente-WS del “Box\_Simulator\_Backend” que está conectado al “Backend”, Recibe por algún “socket” información enviada desde el “Backend”.

El “Box\_Simulator\_Backend” Analiza la información llegada desde el “Backend” para determinar si su “ID del Parking” es con la que el Backend quiere operar.

Si el “ID del Parking” es el correcto, continúa analizando la información que le ha llegado del “Backend” para determinar qué COMANDO debe enviar y hacia qué “BOX” va a ir dirigido.

Con estos datos prepara un Mensaje en JSON que enviará al “FLOW-NodeRed”.

El “Box\_Simulator\_Backend” utilizando el Cliente-WS que está conectado al Servidor-WS del “FLOW-NodeRed”.

Envía el Mensaje en JSON a través del “socket” hacia el “FLOW-NodeRed”.

El Mensaje contiene un “COMMAND” y una “ADDRESS” que el “FLOW-NodeRed” se encargará de hacérselo llegar al “ATMEGA328P” que corresponda y el “ATMEGA” Devolverá una Respuesta al “FLOW-NodeRed”.

El “FLOW-NodeRed” Recibe el Mensaje con el “StatusCode” y “ADDRESS” del Box.

Analiza el “StatusCode” y determina si debe enviar un mensaje para informar de algún cambio al “Box\_Simulator\_Backend”, Si este el el caso…

El “FLOW-NodeRed” Envía a través del “socket” la Respuesta obtenida del “ATMEGA” hacia el Cliente-WS del “Box\_Simulator\_Backend”.

Esta Respuesta contiene un “StatusCode” y una “ADDRESS” que informan de las acciones y estados del “BOX”.

El “Box\_Simulator\_Backend” Recibe la Respuesta desde el “FLOW-NodeRed” y ANALIZA el “StatusCode” que determina la acción o estado se encuentra el “BOX”.

El “Box\_Simulator\_Backend” haciendo uso de su Cliente-WS que está conectado al Server-WS del “Backend”, dependiendo del valor del “StatusCode” analizado, se decanta por uno de los ”sockets” implementados del Server-WS del “Backend” (cada “socket” se encarga de informar de un suceso determinado en el “Box”).

Finalmente el “Box\_Simulator\_Backend” envía a través de ese “socket” al “Backend” la información del ”Parking” y “Box” que dieron lugar a esa acción.

## 4.5-. Comunicación de FLOW-NodeRed con ATMEGA328P

El FLOW-NodeRed(que se ejecuta en una RaspberryPi), como indicamos en el punto 3 se comunica con los ATMEGA328P utilizando el puerto y protocolo I2C en la que RaspberryPi desempeña el papel de MASTER.

La comunicación entre FLOW-NodeRed y ATMEGA:

* **El FLOW-NodeRed recibe desde “el Cliente-WS que ejecuta el Box\_Simulator\_Backend”** (a través del único “socket” que implementa)la información necesaria para operar con algún “BOX”(ATMEGA).
* **El FLOW-NodeRed monta el mensaje a enviar al ATMEGA** con la información que le ha llegado por el “socket”.

**El mensaje a enviar** a través del NODO I2C **consta de un JSON** que contiene:

-- Un “COMMAND” (que será frente al que reaccione el ATMEGA).

-- Una “ADDRESS” (que indicará a qué “BOX” o ATMEGA va dirigido).

* **La “ADDRESS” será “corregida” automáticamente por el FLOW-NodeRed.**

-- El motivo de esto es porque la primera dirección que puede asignarse en

I2C es la “3” y para que todo esté en “armonía” es mejor aplicarle esa

insignificante corrección.

-- Al Recibir la información por el “socket” se suma +2.

-- Al Enviar información de vuelta sobre esa misma “ADDRESS” por el

“socket” de salida le resta -2.

* **El “FLOW-NodeRed” Envía por I2C el “COMMAND” al ATMEGA328P** **de la “ADDRESS”** **indicada**(y corregida).
* **El “FLOW-NodeRed” Solicita una respuesta al ATMEGA328P de dicha “ADDRESS”** a la cual el ATMEGA responderá con un “StatusCode”.
* **El “FLOW-NodeRed” Recibe el “StatusCode” que responde el ATMEGA y determinará** **cómo seguir:**

-- Si debe seguir solicitando o no Respuestas del ATMEGA.

-- Si es necesario o no Enviar el “StatusCode” recibido, a través del

“socket” al Cliente-WS que ejecuta el “Box\_Simulator\_Backend”.

## 

## 

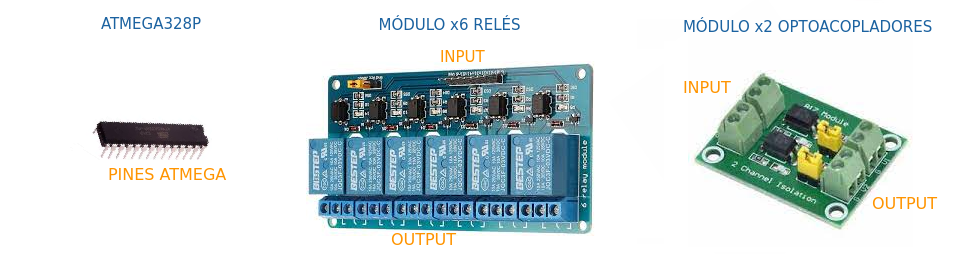
## 4.5.1-. Representación del FLOW-NodeRed (Comunicación WebSocket e I2C)

## 4.6-. Funcionamiento del ATMEGA328P

El **ATMEGA328P** es un chip microcontrolador que dispone de puertos, módulos y librerías que permiten implementar una gran variedad de protocolos de comunicación y que tiene la capacidad de trabajar con sensores y actuadores a través de sus PINES los cuales serán controlados con la programación que le implementemos.

El dispositivo ideado por nosotros para poder controlar el “BOX Físico” utilizando este chip, también necesita de un módulo de RELÉS y OPTOACOPLADORES.

**Se adjunta una imagen con los elementos nombrados anteriormente y sus I/O**:



La conexión entre estos elementos se disponen de la siguiente manera:

- **La Entrada de los Relés están conectadas a un Pin concreto del ATMEGA cada uno**.

Desde los Pines del ATMEGA controlaremos la activación de los Relés.

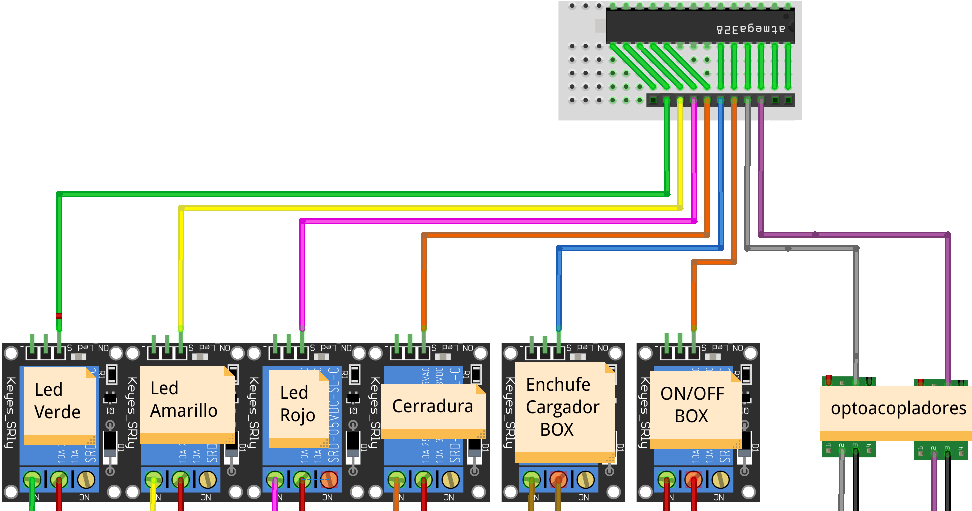
- **La Salida de los RELÉS se encuentran conectados a los componentes ACTUADORES** del “BOX Físico”(leds, cerradura, toma de corriente).

A su vez y con parecido funcionamiento al de los RELÉS pero esta vez de forma inversa.

- **La SALIDA de los OPTOACOPLADORES están conectados a un Pin concreto del ATMEGA cada uno**, a los PINES les llegará la información a través del OPTOACOPLADOR.

- **La ENTRADA de los OPTOACOPLADORES se encuentran conectados con los SENSORES** del “BOX Físico”(Sensor de puerta y de patineta) que emiten la información.

**Los Relés y Optoacopladores son necesarios para poder trabajar con elementos que funcionan a un Voltaje diferente.**

(Atmega y sus pines funcionan a 5V y los componentes del BOX Físico a 24V).

Dado que el ATMEGA328P implementa su propio programa, se definen unas normas de comunicación y estructura de trabajo (creadas por quién la programó y por los protocolos que implementa) las cuales deben seguirse para poder operar con este dispositivo de forma correcta.

Adaptándonos a “sus Normas”, podemos utilizar el ATMEGA328P en conjunto a otras aplicaciones (eso es lo que hace el FLOW-NodeRed).

Las normas a tener en cuenta son las siguientes:

- El **ATMEGA328P se encuentra utilizando el puerto y protocolo I2C en modo SLAVE**, Por lo tanto **se dedicará a operar o responder cuando se lo indique el MASTER**.

**- Cada ATMEGA328P debe tener una “address” ÚNICA dentro del PARKING en el que se encuentre**(comienza en 3 la primera dirección válida).

**- El ATMEGA328P está a la escucha esperando a la orden del MASTER.**

**Puede recibir un comando** “Command” **o una Solicitud para responder** “StatusCode”**.**

**- Si al ATMEGA328P le llega un comando** “Command”, este desarrolla las tareas según **su programación**, las cuales **consistirán en enviar y recibir señales** a través de sus **PINES operando con el BOX Físico**.

**- Si ATMEGA328P recibe una “solicitud de Respuesta” responderá con un “StatusCode”** determinado por el estado de un procedimiento que está realizandose o que ya ha finalizado.

El “**StatusCode**” retornado por ATMEGA, **lo controlará “el FLOW-NodeRed**” que como ya expusimos anteriormente, se encargará de seguir con el flujo del programa dependiendo el “StatusCode” recibido.

A continuación **se adjuntan los listados de “Commands” y “StatusCode”** con los que trabaja o puede trabajar nuestro ATMEGA328P.

## 4.6.1-. Listado de “COMMANDS” para ATMEGA328P

Ejemplo de mensaje a enviar al ATMEGA desde el FLOW-NodeRed:

* {"box\_id":"15","command":"A"}.
* ATMEGA respondera con un “StatusCode” de 3 digitos "XXX".

LISTA DE COMANDOS QUE TIENE PROGRAMADO NUESTRO ATMEGA.

EL COMANDO CONSTA DE **UN solo 'Char'**, ES LO ÚNICO QUE HAY QUE MANDAR AL ATMEGA.

| **COMANDO** | **FUNCIÓN A LA QUE LLAMA** | **DEFINICION DE LA FUNCION** |
| --- | --- | --- |
| A | freeBox() | Establece el parking como LIBRE. |
| B | reservedBox() | Establece el parking como RESERVADO. |
| C | occupiedBox() | Establece el parking como OCUPADO. |
| F |  | Se utiliza cuando el cambio de estado se realizó FORZOSAMENTE. |
| L | returnResponseCode | Retorna el ÚLTIMO “StatusCode”. |
| M | setCloseDoorMillis() | Setear MILISEGUNDOS para DELAY que determina que la puerta SE CONSIDERA CERRADA. |
| N | setOpenDoorMillis() | Setear MILISEGUNDOS para DELAY que determina que la puerta SE CONSIDERA ABIERTA. |
| P | activateCharger() | ACTIVAR CORRIENTE "DEL ENCHUFE PARA el CARGADOR de la Patineta DEL BOX". |
| Q | deactivateCharger() | DESACTIVAR CORRIENTE "DEL ENCHUFE PARA el CARGADOR de la Patineta DEL BOX". |
| R | activateBox() | ACTIVAR CORRIENTE COMPLETA DEL BOX. |
| S | deactivateBox() | CORTAR CORRIENTE COMPLETA DEL BOX. |
| T | forceOccupiedBox() | Forzar a ESTADO OccupiedBOX. |
| U | forceReservedBox() | Forzar a ESTADO ReservedBOX. |
| V | forceFreeBox() | Forzar a ESTADO freeBOX. |
| W | openDeadlock() | Abrir CERRADURA. |
| X | closeDeadlock() | Cerrar CERRADURA. |
| Y | onLEDS() | Encender LOS LEDS DE ESTADO. |
| Z | offLEDS() | Apagar LOS LEDS DE ESTADO. |

## 4.6.2-. Listado de “StatusCode” de ATMEGA328P

ESTA LISTA (**en formato JSON),** son los “StatusCode” QUE IRÁ DEVOLVIENDO EL ATMEGA

Y con los cuales SABREMOS EL ESTADO o TAREA que esté realizando EL BOX.

--------------------------------------------------------------------------------

{

"000":"Atmega RECIÉN ENCENDIDO",

**////####ESTADOS de "FREE BOX" ////**

"100":"BOX-FREE=> PUERTA ABIERTA, NO PATINETA",

"101":"BOX-FREE=> PUERTA ABIERTA, SI PATINETA",

"102":"BOX-FREE=> PUERTA ABIERTA",

"110":"BOX-FREE=> PUERTA CERRADA, NO PATINETA",

"111":"BOX-FREE=> PUERTA CERRADA, SI PATINETA, CERRADURA-close AL ABRIR PUERTA",

"130":"BOX-FREE=> SE HA ABIERTO LA PUERTA, PATINETA NO DETECTADA",

"120":"BOX-FREE=> SENSOR INDICA QUE LA PUERTA NUNCA SE HA ABIERTO",

"150":"BOX-FREE=> PUERTA ABIERTA, NO PATINETA",

"151":"BOX-FREE=> PUERTA ABIERTA, SI PATINETA",

"152":"BOX-FREE=> PUERTA CERRADA, NO PATINETA",

"153":"BOX-FREE=> PUERTA CERRADA, SI PATINETA",

"154":"BOX FORZADO A ESTADO 'Free'",

"155":"BOX A ESTADO FREE CERRADURA CERRADA CON PUERTA ABIERTA, INTENTARON CERRAR

CON PATINETA DENTRO",

"156":"BOX ESTABLECIDO COMO 'Free', SIN USO DEL CIERRE",

**////####ESTADOS de "RESERVED BOX" ////**

"200":"BOX-RESERVED=> PUERTA ABIERTA, NO PATINETA",

"201":"BOX-RESERVED=> PUERTA ABIERTA, SI PATINETA",

"210":"BOX-RESERVED=> PUERTA CERRADA, NO PATINETA",

"211":"BOX-RESERVED=> PUERTA CERRADA, SI PATINETA",

"230":"BOX ESTABLECIDO 'Reserved' CORRECTAMENTE, POR PRIMERA VEZ",

"254":"BOX FORZADO A ESTADO 'Reserved'",

"255":"BOX ESTABLECIDO 'Reserved', CERRADURA CERRADA CON PUERTA ABIERTA,

INTENTARON CERRAR CON PATINETA DENTRO",

"256":"BOX ESTABLECIDO 'Reserved', CON CERRADURA ABIERTA, USUARIO INTENTA CERRAR

EN ESTE ESTADO CON LA PATINETA DENTRO", //(cuando se abra la puerta, se

// cerrará la cerradura )

**////####ESTADOS de "OCCUPIED BOX" ////**

"300":"BOX-OCCUPIED=> PUERTA ABIERTA, NO PATINETA ",

"301":"BOX-OCCUPIED=> PUERTA ABIERTA, SI PATINETA",

"302":"BOX-OCCUPIED=> PUERTA ABIERTA,

"310":"BOX-OCCUPIED=> PUERTA CERRADA, NO PATINETA", //Se FORZARÁ a 'FreeBox'

"311":"BOX ESTABLECIDO COMO 'Occupied', PUERTA CERRADA, SI PATINETA",

"330":"BOX-OCCUPIED=> SE HA ABIERTO LA PUERTA, PATINETA NO DETECTADA",

"320":"BOX-OCCUPIED=> SENSOR INDICA QUE LA PUERTA NUNCA SE HA ABIERTO",

"350":"BOX-OCCUPIED=> PUERTA ABIERTA, NO PATINETA",

"351":"BOX-OCCUPIED=> PUERTA ABIERTA, SI PATINETA",

"352":"BOX-OCCUPIED=> PUERTA CERRADA, NO PATINETA",

"353":"BOX-OCCUPIED=> PUERTA CERRADA, SI PATINETA", //IGUAL que el 311

"354":"BOX FORZADO A ESTADO 'Occupied'",

**////####ESTADOS de ACCIONES DE MANTENIMIENTO////**

"500":"Apagados LOS LEDS DE ESTADO",

"501":"Encendidos LOS LEDS DE ESTADO",

"510":"Forzado Cerrar Cerradura",

"511":"Forzado Abrir Cerradura",

"400":"CORTAR CORRIENTE COMPLETA DEL BOX",

"401":"ACTIVAR CORRIENTE COMPLETA DEL BOX",

"410":"CORTADA CORRIENTE DEL ENCHUFE PARA el CARGADOR de la Patineta DEL BOX",

"411":"ACTIVAR CORRIENTE DEL ENCHUFE PARA el CARGADOR de la Patineta DEL BOX",

"600":"Seteados MILISEGUNDOS para DELAY que determina si la puerta es ABIERTA",

"610":"Seteados MILISEGUNDOS para DELAY que determina si la puerta es CERRADA",

"900":"Cambio de estado 'forzado'",

"999":"Se ha introducido UN Comando NO EXISTENTE"

}

**Aquí termina el JSON con los “StatusCode” y sus correspondientes definiciones.**

## 

# 5-. DESPLIEGUE E INSTALACIÓN DEL SOFTWARE DEL PROYECTO

En este punto se indica como proceder a instalar las distintas aplicaciones de las que se compone el proyecto y en el Hardware donde deben ser instaladas.

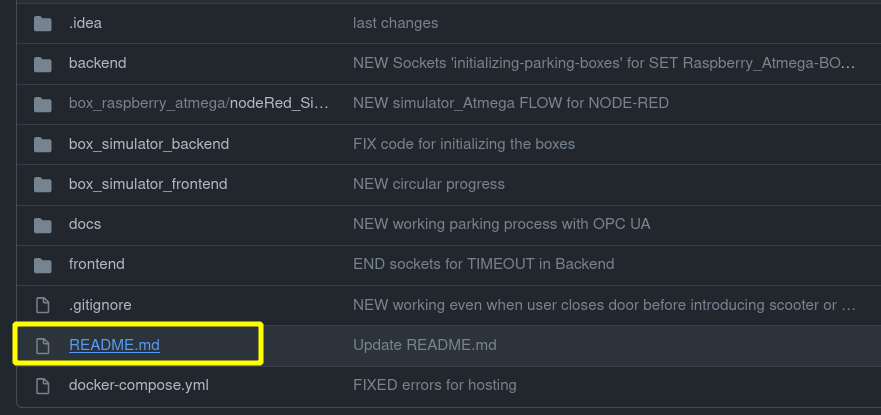
En este punto encontraremos:

* **Cómo instalar “frontend” y “backend” en el Servidor.**
* **Cómo instalar “Box\_Simulator\_Backend” en RaspberryPi.**
* **Cómo instalar NodeRed y el “FLOW-NodeRed” en RaspberryPi.**
* **Cómo cargar “sketch” a ATMEGA328P desde RaspberryPi.**

## 5.1-. Instalando “frontend” y “backend” en el Servidor

**Estas “dos” aplicaciones tienen que ser instaladas en la parte del Servidor que está Expuesto a Internet**.

Para proceder con la instalación de estas dos, **se deben seguir las instrucciones que vienen detalladas en el Archivo “README.md”** contenido en el repositorio del proyecto indicado en el [punto 2](#_1nbdsrt9et1s).



## 5.2-.Instalando Entorno y “FLOW-NodeRed” en RaspberryPi

**La instalación de esta aplicación debe realizarse en RaspberryPi.**

Para realizar la instalación del entorno (NodeJS, npm y NodeRed) es necesario contar con el Sistema Operativo de RaspberryPi ya preparado.

## 5.2.1-.En caso de NO TENER Sistema Operativo en RaspberryPi

**En caso de empezar de cero y necesitar realizar la instalación del SO**,

Seguir las instrucciones de su página oficial donde informan como proceder a la instalación del Sistema Operativo, y se adjunta URL oficial de las Imágenes.

Pag. Oficial Install RaspberryPi OS: <https://www.raspberrypi.org/software/>

Imágenes disponibles:<https://www.raspberrypi.org/software/operating-systems/>

## 5.2.2-.En caso de SI TENER Sistema Operativo en RaspberryPi

La instalación expuesta a continuación, se realizó con apoyo de la documentación oficial de “NodeRed”:<https://nodered.org/docs/getting-started/raspberrypi>

**1-** Abrimos una “Consola”.

**2-** Instalamos requerimientos necesarios,

EJECUTANDO EN LA CONSOLA:

***sudo apt install build-essential git curl***

**3-**Ordenamos Descargar y Ejecutar el “Script” que instalará NodeJS,npm y NodeRed.

EJECUTANDO EN LA CONSOLA:

***bash <(curl -sL*** [***https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered***](https://raw.githubusercontent.com/node-red/linux-installers/master/deb/update-nodejs-and-nodered)***)***

**Ya estará el entorno instalado.**

**4-** Ejecutamos el siguiente comando para deshabilitar el inicio de NodeRed al arranque del sistema, ya que debemos iniciar el servicio de forma distinta:

- EJECUTAR DESDE LA CONSOLA:

***sudo systemctl disable nodered.service***

**\*\* Para ejecutar NodeRed**:

Se realiza a través de la “consola” y debido a que vamos a ejecutarlo en un

entorno “reducido” lo ejecutaremos con el siguiente comando que hará que la

memoria se libere cuando llegue al tamaño que le indiquemos.

- EJECUTAR DESDE LA CONSOLA:

***node-red-pi --max-old-space-size=128***

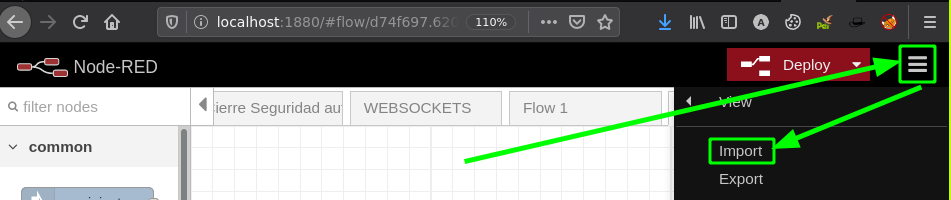
**\* NodeRed se ejecuta y ofrece su servicio por el puerto “1880”.**

**5-** Accedemos a la Aplicación NodeRed a través del Navegador con la URL:

[***http://localhost:1880***](http://localhost:1880)

**6- Procederemos a importar el “FLOW” del proyecto a “NodeRed”**:

- En la esquina superior derecha, accedemos al “Menú-Hamburguesa”> “Import”.



- Se nos desplegará una ventana,y le damos a **“Select a file to import”**.

- Buscamos el archivo [“NodeRed\_AtmegaBOX\_BLUECITY.json”](#_11eg5qmzyql6), lo Seleccionamos y lo Importamos.

**- Nuestro FLOW se habrá importado y ya estará ejecutándose**.

**7-** Para finalizar, **crearemos una “tarea Cron” para lanzar el NodeRed** con el comando modificado **al inicio del sistema** y comience a funcionar.

EJECUTAMOS EN UNA CONSOLA:

***crontab -e***

**- Se nos abrirá en la Consola el archivo que maneja las tareas Cron.**

Nos desplazamos hasta el final del archivo y agregamos la siguiente línea:

***@reboot ( sleep 50 ; node-red-pi --max-old-space-size=128)***

**Para guardar y salir**, pulsamos CTRL+X,y le indicamos que SI queremos guardar los cambios.

**- Listo! ya podemos olvidarnos del “FLOW-NodeRed”.**

Se iniciará automáticamente al Inicio del Sistema.

**\*\* Recordar que de ser necesario acceder a la aplicación de NodeRed:**

Accedemos a la Aplicación NodeRed a través del Navegador con la URL:

<http://localhost:1880>

## 5.3-. Instalando “Box\_Simulator\_Backend” en RaspberryPi

**La instalación de esta aplicación debe realizarse en RaspberryPi.**

Los requerimientos necesarios para poder instalar ésta aplicación (NodeJS y npm) ya están instalados(se instalaron en el [punto anterior](#_ydrwft7jaayx)), así que solo falta proceder a la instalación.

Para proceder con la instalación de esta aplicación, **se deben seguir las instrucciones que vienen detalladas en el Archivo “README.md”** contenido en el repositorio del proyecto indicado en el [punto 2](#_1nbdsrt9et1s).

**Dirigirse directamente al apartado que habla sobre la instalación de “Box\_Simulator\_Backend”.**

## 5.4-. Cargar “sketch” a ATMEGA328P desde RaspberryPi

**La instalación de esta aplicación se realiza en el ATMEGA328P.**

Para poder cargar el “sketch” al ATMEGA, se debe tener instalado el Software Arduino IDE(<https://www.arduino.cc/en/software>).

y contar con un Arduino o un programador USB to TTL para poder comunicarse con el ATMEGA.

**Suponiendo que vamos a realizar la carga del “sketch” desde la RaspberryPi:**

- **Procedemos a instalar el software de Arduino**.

EJECUTAMOS EN UNA CONSOLA:

***sudo apt install arduino arduino-core-avr -y***

Esperamos.. y Arduino se instalará.

- **Abrimos Arduino como superusuario (para evitar problemas con los permisos)**.

EJECUTAMOS EN UNA CONSOLA:

***sudo arduino***

**- Damos a “Archivo” > “Abrir”, Buscamos el “sketch\_Atmega\_BOX.ino” contenido en el repositorio del proyecto, lo Seleccionamos y lo Abrimos.**

**- IMPORTANTE ANTES DE CARGARLO, tenemos que ASIGNAR LA “ID” Correcta al ATMEGA.**

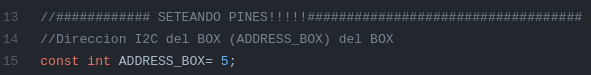
**Recordar que la primera dirección posible que puede asignar el protocolo I2C a un dispositivo ES LA “3”.**

Así que debemos sumar +2 a la que sea la “ID real del Box en el Parking”**.**

**Ej.***Si el Box del Parking es el 1, en el ATMEGA debe asignarse la dirección “3”.*

*Si el Box del Parking es el 3, en el ATMEGA debe asignarse la dirección “5”.*

- **El valor a modificar es “ADDRESS\_BOX” actualmente en la línea 15 del scketch.**



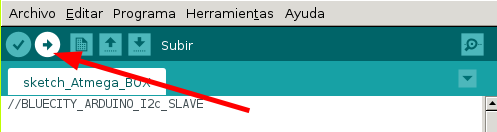
- **Una vez está correcta nuestra “ADDRESS\_BOX”.**

**Procedemos a conectar el ATMEGA a la RaspberryPi**(u otro dispositivo) ya sea

con una placa Arduino o con el USB to TTL (convertidor USB a Serial).

- **En el apartado “Herramientas”> Seleccionamos el Puerto y la Placa Arduino** Uno**.**

**Le damos a “Subir”**, esperamos a que compile, lo cargue y LISTO!!



**\* ATMEGA328P ya estará programado.**

# 6-. MATERIALES NECESARIOS DE LA PARTE FÍSICA DEL PROYECTO

## 6.1-. Componentes para “la base” del Parking

-Para **cada** “**Parking**” (puede **albergar hasta 120 BOXES)**:

| NOMBRE | IMAGEN | CANTIDAD | Corriente de Trabajo |
| --- | --- | --- | --- |
| Raspberry Pi 3b+ |  | 1 | 5V - 3000mA |
| MicroSD >= 32Gb  *(La VELOCIDAD*  *de la tarjeta es IMPORTANTE)* |  | 1 | 0 |
| **CONSUMO TOTAL DEL CONJUNTO / POTENCIA:** | | | 5V-3000mA = 15Watts |

## 6.2-. Componentes para “Parte Física” de UN “BOX”

| NOMBRE | IMAGEN | CANTIDAD | Corriente de Trabajo |
| --- | --- | --- | --- |
| BOX Físico |  | 1 | 0 |
| Detector Fotoeléctrico  (Proximidad) |  | 1 | 24V - <=100mA |
| Sensor Inductivo |  | 1 | 24V - <=200mA |
| Solenoide  Cerradura |  | 1 | 24V - 1000mA |
| Led Rojo |  | 1 | 24V - 20mA |
| Led Amarillo |  | 1 | 24V - 20mA |
| Led Verde |  | 1 | 24V - 20mA |
| **CONSUMO TOTAL DEL CONJUNTO / POTENCIA:** | | | 24V-1360mA = 32,64Watts |

## 6.3-. Componentes para “el Cuadro Eléctrico” de UN “BOX”

-Para **cada** “**BOX**” que se necesite instalar.

**Esta es la parte electrónica correspondiente al cuadro eléctrico:**

| NOMBRE | IMAGEN | CANTIDAD | Corriente de Trabajo |
| --- | --- | --- | --- |
| Componentes para PCB de ATMEGA328P | | | |
| ATMEGA328P |  | 1 | 5V - 20mA |
| Cristal 16Mhz |  | 1 | 0 |
| Condensadores cerámicos 22pF |  | 2 | 0 |
| Resistencia 1/4w  220-Ohm |  | 1 | 0 |
| Led |  | 1 | 5V - 20mA |
| Placa PCB |  | 1 | 0 |
| COMPONENTES PARA REALIZAR LA CONEXIÓN CON EL BOX | | | |
| Cables **Dupont** (**Male-Female**) 30cm |  | 30 | 0 |
| Módulo Arduino **x6** Relés 5v to 220v |  | 1 | 6-Relés x 60 mA = 360mA  5V - 360mA |
| Módulo **x2** Optoacopladores  24v/12 to 5v |  | 1 | 2-Opto. x 20 mA = 40mA  5V - 40mA |
| **CONSUMO TOTAL DEL CONJUNTO / POTENCIA:** | | | 5V-440mA = 2,2Watts |

## 6.4-. Cálculo de consumo de la instalación para UN BOX

- Se realizará el Cálculo de la potencia necesaria de los componentes que trabajan a la misma Tensión(V), para poder determinar los Convertidores CC (de Corriente Continua) que necesitamos para alimentar los distintos componentes del Parking y Box.

- También se realiza el cálculo del consumo de TODO el conjunto, para poder saber cual es la Potencia mínima necesaria que nuestra instalación Fotovoltaica debe tener para poder alimentar el Parking, cargar la patineta y poder seguir cargando la batería.

## 6.4.1-. Cálculo de consumo para componentes de 5V de UN BOX

| NOMBRE | CANTIDAD | Corriente de Trabajo | |
| --- | --- | --- | --- |
| [Componentes “base” del Parking](#_x9xhwfl0d67u) | 1 | 5V - 3000mA | |
| [Componentes para](#_t27mxji7599h)  [“Cuadro eléctrico”de Un BOX”](#_t27mxji7599h) | 1 | 5V - 440mA | |
| **CONSUMO TOTAL DEL CONJUNTO / POTENCIA:** | | | 5V-3440mA = 17,2Watts  Redondeando hacia arriba:  **5V-4000mA = 20W** |

## 

## 6.4.2-. Cálculo de consumo para componentes de 24V de UN BOX

| NOMBRE | CANTIDAD | Corriente de Trabajo | |
| --- | --- | --- | --- |
| [Componentes para “parte física” del Box](#_1755uvb2qasj) | 1 | 24V - 1360mA | |
| **CONSUMO TOTAL DEL CONJUNTO / POTENCIA:** | | | 24V-1360mA = 32,64Watts  Redondeando hacia arriba:  **24V-2000mA = 48W** |

## 6.4.3-. Cálculo de consumo Máximo “Total” de la instalación

Para calcular el consumo total de la instalación, sumaremos el consumo de los componentes calculados en los posteriores.

## 6.4.3.1-. Cálculo de consumo Máximo CON Patineta en carga

| **COMPONENTES** | **POTENCIA (Watts)** |
| --- | --- |
| [Componentes a 5V](#_8c1d6zc3i32e) **+** [Componentes a 24V](#_cupfn8hlrxck) + Inversor | 20W + 48W + 100W = 168W |
| **Consumo Máximo estimado del Parking:** | **168 Watts = 12V-14000mA** |

## **6.4.3.2-. Cálculo de consumo Máximo SIN Patineta en carga**

| **COMPONENTES** | **POTENCIA (Watts)** |
| --- | --- |
| [Componentes a 5V](#_8c1d6zc3i32e) **+** [Componentes a 24V](#_cupfn8hlrxck) | 20W + 48W = 68W |
| **Consumo Máximo estimado del Parking:** | **68 Watts = 12V-5667mA** |

## 6.5-. Determinando “Convertidores CC” según Consumo Calculado

Estos son los “Convertidores de Corriente Continua” calculados y “el Inversor” NECESARIOS para trabajar con una Instalación Fotovoltaica de 12V.

-**Convertidores de Corriente Continua para un “Parking” con 1 “BOX”:**

| NOMBRE | IMAGEN | CANTIDAD | Corriente de Trabajo |
| --- | --- | --- | --- |
| [**12v** to **5v-4A (20W)**](#_8c1d6zc3i32e)  *(Electrónica cuadro)* |  | 1 | [Convertidor CC de 12V a 5V-5A](https://es.aliexpress.com/item/32676859440.html?spm=a2g0o.productlist.0.0.7e594e660hZmrP&algo_pvid=d6bd0c0c-2770-4db1-8241-adced38386e8&algo_exp_id=d6bd0c0c-2770-4db1-8241-adced38386e8-1&pdp_ext_f=%7B%22sku_id%22%3A%2267143820051%22%7D)  (20W) |
| [**12v** to **24v-2A (48W)**](#_cupfn8hlrxck)  *(Electrónica BOX Físico)* |  | 1 | [Convertidor CC de 12V a 24V-2A](https://es.aliexpress.com/item/4001187627223.html?spm=a2g0o.productlist.0.0.24707b2bmcBMVn&algo_pvid=9b0b445b-7a7e-456b-9920-3c210c69a611&algo_exp_id=9b0b445b-7a7e-456b-9920-3c210c69a611-26&pdp_ext_f=%7B%22sku_id%22%3A%2210000015213293587%22%7D)  (48W) |
| **Inversor**-**12V DC**  to **220V AC (100W)**  *(Hacia toma del cargador)* |  | 1 | **Inversor**-**12V DC**  to **220V AC**  **(100W)** |

## 6.6-. Componentes para Instalación Fotovoltaica de 12V

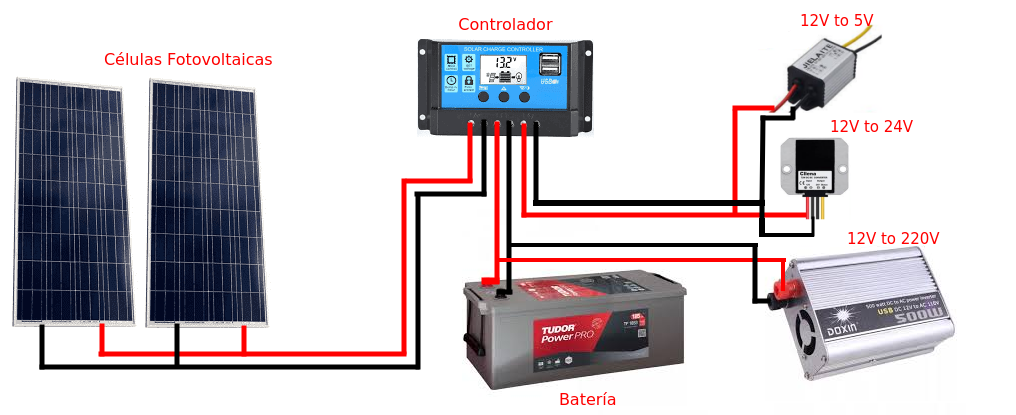
| NOMBRE | IMAGEN | CARACTERÍSTICAS | CANTIDAD | |
| --- | --- | --- | --- | --- |
| **Células Fotovoltaicas** |  | [**P.mínima NECESARIA -- 200W**](#_ee91ajsoi4f8)  **Actualmente=2x160W**  **(18 Volts - 8.5Amp)**  **P.actual= 320W** | 2x(160W) | |
| **Controlador**  **Batería** |  | **MÍNIMO**  **12 Volts - 20 Amp** | 1 | |
| **Batería** |  | **MÍNIMO**  **12 Volts - 80 Amp** | 1 | |

# 7-.PLANOS DE MONTAJE COMPLETO DEL PARKING

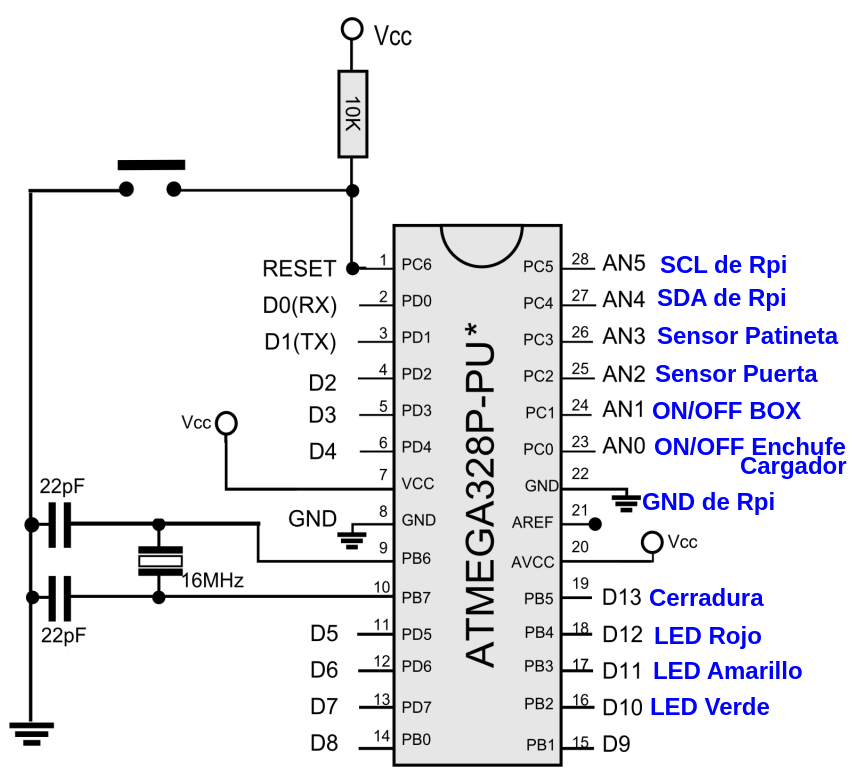
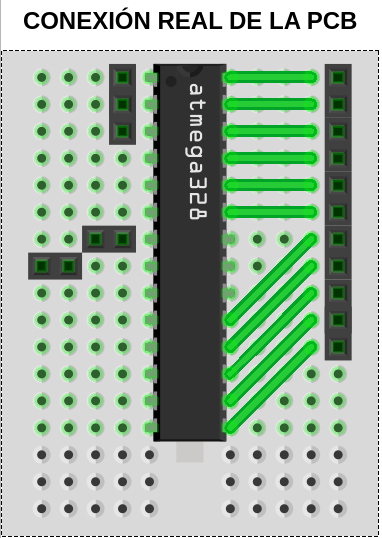
Se adjuntan a continuación los diferentes planos para realizar el montaje de las distintas partes del Parking Físico.

## 

## 7.1-. Montaje de Instalación Fotovoltaica a Cuadro eléctrico



## 7.2-. Montaje de la PCB con el ATMEGA328P

**ATMEGA\_BOX - PIN REFERENCE**

| **ATMEGA\_BOX PIN REFERENCE** | | |
| --- | --- | --- |
| **DE: PIN ATMEGA** | **A: SENSOR** | **A: ACTUADOR** |
| **A5** |  | SCL de Raspberry |
| **A4** |  | SDA de Raspberry |
| **A3** | Sensor Patineta |  |
| **A2** | Sensor Puerta |  |
| **A1** |  | Relé ON/OFF BOX |
| **A0** |  | Relé Cargador del BOX |
| **13** |  | Relé Cerradura |
| **12** |  | Relé LED Rojo |
| **11** |  | Relé LED Amarillo |
| **10** |  | Relé LED Verde |
| **9** |  | Servo Cerradura(test) |

## 7.3-. Conexiones de Raspberry, ATMEGA328P y sus módulos

## 7.4-. Conexiones de “BOX Físico” a Bornes del Cuadro

## 7.5-. Plano de Conexión Completo de Cuadro y Box

