

XONIDAL

Universal Serial Bridge

Darian Alberto Camacho Salas
Backend / Integracion

Oscar Rodolfo Barragan Perez
Frontend / Interfaz Web

Facultad de Estudios Superiores Cuautitlan, UNAM
Ingenieria en Telecomunicaciones, Sistemas y Electronica

Asesor:
Dr. Raul Dali Cruz Morales

INTRODUCCION COMPLETA AL SISTEMA

¿Que es XONIDAL?

XONIDAL (Universal Serial Bridge) es una plataforma IoT educativa que permite establecer un puente de comunicacion universal entre una interfaz web y cualquier proyecto de Arduino a traves de un ESP32. El sistema esta disenado para enseñar los principios fundamentales de comunicacion serial, IoT y control remoto de dispositivos.

Arquitectura del Sistema



El sistema funciona en 4 capas:

Capa 1: Presentacion: Interfaz web con diseño cyberpunk

Capa 2: Aplicacion: Servidor Flask que maneja peticiones

Capa 3: Puente: ESP32 que conecta WiFi con comunicación serial

Capa 4: Control: Arduino que ejecuta los comandos en el hardware

REQUISITOS MINIMOS DEL SISTEMA

Hardware

- **Computadora** con Linux, Windows o MacOS
- **ESP32** (NodeMCU, ESP32 DevKit)
- **Arduino** (Uno, Nano, Mega)
- **LEDs integrados** en las placas
- **3 Cables Dupont** para conexión

Software

- Python 3.8 o superior
- Arduino IDE con soporte ESP32
- Navegador web moderno
- Conexión a Internet (inicial)

Conocimientos Previos

- Básico de terminal / línea de comandos
- Conceptos básicos de electrónica (opcional)
- Conocimiento básico de redes IP

INSTALACION EN DIFERENTES SISTEMAS OPERATIVOS

Instalacion en Arch Linux

Preparacion del Sistema

```
1 # Actualizar el sistema
2 sudo pacman -Syu
3
4 # Instalar herramientas basicas
5 sudo pacman -S base-devel git curl wget
```

Backend Flask

```
1 # Instalar Python y pip
2 sudo pacman -S python python-pip
3
4 # Instalar dependencias
5 pip install flask requests --break-system-packages
6
7 # Verificar instalacion
8 python --version
9 pip list | grep flask
```

Puerto Serial

```
1 # Agregar usuario al grupo dialout
2 sudo usermod -aG dialout $USER
3
4 # Verificar puertos disponibles
5 ls -la /dev/ttyUSB* /dev/ttyACM*
```

Instalacion en Ubuntu y Derivados

Preparacion del Sistema

```
1 # Actualizar repositorios
2 sudo apt update
3 sudo apt upgrade -y
4
5 # Instalar herramientas basicas
6 sudo apt install -y git curl wget build-essential
```

Backend Flask

```
1 # Instalar Python y pip
2 sudo apt install -y python3 python3-pip
3
4 # Instalar dependencias
5 pip3 install flask requests --break-system-packages
6
7 # Verificar instalacion
8 python3 --version
9 pip3 list | grep flask
```

Puerto Serial

```
1 # Agregar usuario al grupo dialout
2 sudo usermod -aG dialout $USER
3
4 # Verificar puertos
5 ls -la /dev/ttyUSB* /dev/ttyACM*
```

Instalacion en Windows

Preparacion del Sistema

1. Descargar Python de <https://www.python.org/downloads/>
2. **IMPORTANTE:** Marcar add Python to PATH”
3. Abrir Command Prompt como Administrador

Backend Flask

```
1 # Verificar Python
2 python --version
3
4 # Instalar dependencias
5 pip install flask requests
6
7 # Verificar instalacion
8 pip list | findstr flask
```

Instalacion en MacOS

Preparacion del Sistema

```
1 # Instalar Homebrew
2 /bin/bash -c "$(curl -fsSL https://raw.githubusercontent.com/
   Homebrew/install/HEAD/install.sh)"
3
4 # Actualizar Homebrew
5 brew update
```

Backend Flask

```
1 # Instalar Python
2 brew install python3
3
4 # Instalar dependencias
5 pip3 install flask requests
6
7 # Verificar instalacion
8 python3 --version
9 pip3 list | grep flask
```

CONFIGURACION DEL ARDUINO IDE

Instalacion por Sistema

Arch Linux:

```
1 sudo pacman -S arduino arduino-avr-core
```

Ubuntu/Debian:

```
1 sudo apt update  
2 sudo apt install arduino arduino-core
```

Windows/Mac: Descargar de <https://www.arduino.cc/en/software>

Agregar Soporte ESP32

1. Abrir Arduino IDE
2. Archivo → Preferencias
3. En ”Gestor de URLs Adicionales”, agregar:

```
1 https://raw.githubusercontent.com/espressif/arduino-esp32/gh-  
  pages/package_esp32_index.json
```

4. Herramientas → Placa → Gestor de tarjetas
5. Buscar e instalar ESP32 by Espressif Systems”

PROGRAMACION DE LOS MICRO- CONTROLADORES

Conexion Fisica ESP32 Arduino

ESP32	Arduino
<hr/>	
GPIO16 (RX)	↔ Pin 1 (TX)
GPIO17 (TX)	→ Pin 0 (RX)
GND	→ GND

IMPORTANTE

- Conexiones RX/TX CRUZADAS
- No conectar pines de alimentacion (5V/3.3V)

Codigo para ESP32 - esp32_xonidal.ino

```

1  /*
2   XONIDAL - ESP32 WiFi a Serial Bridge
3   Version Universal
4 */
5
6 #include <WiFi.h>
7
8 String SSID = "";
9 String PASSWORD = "";
10 bool configurado = false;
11 const int LED_PIN = 2; // LED integrado
12
13 WiFiServer server(80);
14
15 void configurarWiFi() {
16   Serial.println("\n=====");
17   Serial.println("XONIDAL - CONFIGURACION INICIAL");
18   Serial.println("=====");
19
20   Serial.print("SSID: ");
21   while (!Serial.available()) delay(100);
22   SSID = Serial.readStringUntil('\n');
23   SSID.trim();
24
25   Serial.print("PASSWORD: ");
26   while (!Serial.available()) delay(100);
27   PASSWORD = Serial.readStringUntil('\n');
28   PASSWORD.trim();
29
30   Serial.println("\nConectando...");
31   WiFi.begin(SSID.c_str(), PASSWORD.c_str());
32
33   int attempts = 0;

```

```
34     while (WiFi.status() != WL_CONNECTED && attempts < 30) {
35         delay(1000);
36         Serial.print(".");
37         attempts++;
38     }
39
40     if (WiFi.status() == WL_CONNECTED) {
41         Serial.println("\nCONECTADO!");
42         Serial.print("IP: ");
43         Serial.println(WiFi.localIP());
44         configurado = true;
45         server.begin();
46
47         for(int i = 0; i < 3; i++) {
48             digitalWrite(LED_PIN, HIGH);
49             delay(200);
50             digitalWrite(LED_PIN, LOW);
51             delay(200);
52         }
53     } else {
54         Serial.println("\nError. Reiniciando...");
55         delay(3000);
56         ESP.restart();
57     }
58 }
59
60 String base64Decode(String input) {
61     const char b64[] = "
62         ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789
63         +/";
64     String decoded = "";
65     int val = 0, valb = -8;
66
67     for (char c : input) {
68         if (c == '=') break;
69         const char *p = strchr(b64, c);
70         if (!p) break;
71         val = (val << 6) + (p - b64);
72         valb += 6;
73         if (valb >= 0) {
74             decoded += char((val >> valb) & 0xFF);
75             valb -= 8;
76         }
77     }
78     return decoded;
79 }
80
81 bool checkAuth(String authHeader) {
82     if (!authHeader.startsWith("Basic ")) return false;
83
84     String base64Credentials = authHeader.substring(6);
```

```
83     base64Credentials.trim();
84     String credentials = base64Decode(base64Credentials);
85
86     int separatorPos = credentials.indexOf(':');
87     if (separatorPos == -1) return false;
88
89     String user = credentials.substring(0, separatorPos);
90     String pass = credentials.substring(separatorPos + 1);
91
92     return (user == "admin" && pass == "1234");
93 }
94
95 void handleClient() {
96     WiFiClient client = server.available();
97
98     if (client) {
99         String header = "";
100        String authHeader = "";
101        String message = "";
102        bool authenticated = false;
103
104        while (client.connected()) {
105            if (client.available()) {
106                char c = client.read();
107                header += c;
108
109                if (c == '\n') {
110                    if (header.length() == 2) {
111
112                        int authIndex = header.indexOf("Authorization:");
113                        if (authIndex != -1) {
114                            int start = header.indexOf("Basic", authIndex);
115                            int end = header.indexOf("\r\n", authIndex);
116                            if (start != -1 && end != -1) {
117                                authHeader = header.substring(start, end);
118                                authHeader.trim();
119                                authenticated = checkAuth(authHeader);
120                            }
121                        }
122
123                        int bodyStart = header.indexOf("\r\n\r\n") + 4;
124                        if (bodyStart > 4) {
125                            String body = header.substring(bodyStart);
126                            if (body.startsWith("message=")) {
127                                message = body.substring(8);
128                                message.replace("%20", " ");
129                                message.replace("%2C", ", ");
130                                message.replace("%21", "! ");
131                                message.replace("%3F", "?");
132                                message.trim();
133                            }
134
135                        }
136
137                    }
138
139                }
140            }
141        }
142    }
143 }
```

```
134     }
135
136     if (!authenticated) {
137         client.println("HTTP/1.1 401 Unauthorized");
138         client.println("WWW-Authenticate: Basic realm=\""
139                         XONIDAL\"");
140         client.println("Content-Type: application/json");
141         client.println("Connection: close");
142         client.println();
143         client.println("{\"status\":\"error\",\"message\":"
144                         "\"Auth required\"}");
145     }
146     else if (message.length() > 0) {
147         digitalWrite(LED_PIN, HIGH);
148         delay(100);
149         digitalWrite(LED_PIN, LOW);
150
151         Serial.println(message);
152
153         client.println("HTTP/1.1 200 OK");
154         client.println("Content-Type: application/json");
155         client.println("Connection: close");
156         client.println();
157         client.println("{\"status\":\"ok\",\"message\":\""
158                         "Enviado: " + message + "\"}");
159     }
160     else {
161         client.println("HTTP/1.1 400 Bad Request");
162         client.println("Content-Type: application/json");
163         client.println("Connection: close");
164         client.println();
165         client.println("{\"status\":\"error\",\"message\":"
166                         "\"No message\"}");
167     }
168     break;
169 }
170 }
171 }
172 }
173
174     client.stop();
175 }
176 }
177
178 void setup() {
179     Serial.begin(9600);
180     pinMode(LED_PIN, OUTPUT);
```

```

181   digitalWrite(LED_PIN, LOW);
182
183   delay(1000);
184   Serial.println("\nXONIDAL - PUENTE UNIVERSAL");
185   configurarWiFi();
186 }
187
188 void loop() {
189   if (configurado) {
190     handleClient();
191   }
192
193   if (Serial.available()) {
194     String respuesta = Serial.readStringUntil('\n');
195     respuesta.trim();
196     if (respuesta.length() > 0) {
197       Serial.print("Arduino: ");
198       Serial.println(respuesta);
199     }
200   }
201
202   delay(10);
203 }
```

Instrucciones de Carga ESP32

1. Conectar ESP32 por USB
2. Seleccionar placa: "ESP32 Dev Module"
3. Seleccionar puerto correcto
4. Subir codigo (Ctrl+U)
5. Abrir Monitor Serial (9600 baud)
6. Ingresar SSID y password WiFi
7. ANOTAR LA IP QUE APARECE

Codigo para Arduino - arduino_xonidal.ino

```

1  /*
2   XONIDAL - Codigo BASE para Arduino
3   PERSONALIZA esta funcion segun tu proyecto
4 */
5
6 const int LED_PIN = 13; // LED integrado
7
8 String inputBuffer = "";
```

```
9  bool comandoCompleto = false;
10
11 void setup() {
12     Serial.begin(9600);
13     pinMode(LED_PIN, OUTPUT);
14     digitalWrite(LED_PIN, LOW);
15
16     delay(1000);
17     Serial.println("=====");
18     Serial.println("ARDUINO XONIDAL - LISTO");
19     Serial.println("=====");
20     Serial.println("Esperando comandos... ");
21 }
22
23 void loop() {
24     while (Serial.available()) {
25         char c = Serial.read();
26         if (c == '\n') {
27             comandoCompleto = true;
28         } else {
29             inputBuffer += c;
30         }
31     }
32
33     if (comandoCompleto) {
34         inputBuffer.trim();
35
36         if (inputBuffer.length() > 0) {
37             Serial.print("Recibido: ");
38             Serial.println(inputBuffer);
39
40             // Accion universal: LED parpadea
41             digitalWrite(LED_PIN, HIGH);
42             delay(200);
43             digitalWrite(LED_PIN, LOW);
44
45             // =====
46             // AREA DE PERSONALIZACION - MODIFICA AQUI
47             // =====
48             ejecutarComandoPersonalizado(inputBuffer);
49             // =====
50
51             Serial.println("Listo para siguiente comando");
52     }
53
54     inputBuffer = "";
55     comandoCompleto = false;
56 }
57
58     delay(10);
59 }
```

```
60 // =====
61 // FUNCION PERSONALIZABLE - MODIFICA SEGUN TU PROYECTO
62 // =====
63
64 void ejecutarComandoPersonalizado(String comando) {
65
66     // ----- EJEMPLOS -----
67
68     if (comando == "LED_ON") {
69         // digitalWrite(8, HIGH);
70         Serial.println("LED ENCENDIDO");
71     }
72     else if (comando == "LED_OFF") {
73         // digitalWrite(8, LOW);
74         Serial.println("LED APAGADO");
75     }
76     else if (comando.startsWith("SERVO_")) {
77         String valorStr = comando.substring(6);
78         int angulo = valorStr.toInt();
79         if (angulo >= 0 && angulo <= 180) {
80             // servo.write(angulo);
81             Serial.print("Servo a: ");
82             Serial.print(angulo);
83             Serial.println(" ");
84         }
85     }
86     else if (comando == "TEMP") {
87         Serial.println("Temperatura: 25.3 C (ejemplo)");
88     }
89     else {
90         Serial.print("Comando generico: ");
91         Serial.println(comando);
92     }
93 }
```

Instrucciones de Carga Arduino

1. Conectar Arduino por USB
2. Seleccionar placa (Uno, Nano, Mega)
3. Seleccionar puerto correcto
4. Subir codigo (Ctrl+U)
5. Abrir Monitor Serial para verificar

CONFIGURACION DEL SERVIDOR FLASK

Estructura de Directorios

```
XONIDAL/
    start.py                  # Servidor Flask
    requirements.txt          # Dependencias
    templates/
        index.html            # Interfaz web
    README.md                 # Documentacion
```

Archivo start.py

```
1  from flask import Flask, request, Response, jsonify,
2      render_template
3  import requests
4  import base64
5
6
7  USERNAME = "admin"
8  PWD = "1234"
9  ESP32_IP = None
10
11 def check_auth(auth_header):
12     if not auth_header:
13         return False
14     try:
15         auth_type, auth_info = auth_header.split(None, 1)
16         if auth_type.lower() != "basic":
17             return False
18         decoded = base64.b64decode(auth_info).decode("utf-8")
19         user, pwd = decoded.split(":", 1)
20         return user == USERNAME and pwd == PWD
21     except:
22         return False
23
24 def authenticate():
25     return Response(
26         "Autenticacion requerida", 401,
27         {"WWW-Authenticate": 'Basic realm="XONIDAL"'})
28
29
30 @app.route("/")
31 def index():
32     return render_template('index.html', ESP32_IP=ESP32_IP)
```

```
33
34 @app.route("/set_ip", methods=["POST"])
35 def set_ip():
36     global ESP32_IP
37     data = request.json
38     ESP32_IP = data.get("ip", "").strip()
39
40     if not ESP32_IP:
41         return jsonify({"status": "IP invalida"})
42
43     return jsonify({"status": f"IP guardada: {ESP32_IP}"})
44
45 @app.route("/send", methods=["POST"])
46 def send():
47     global ESP32_IP
48
49     if not ESP32_IP:
50         return jsonify({"status": "Configura IP primero"})
51
52     data = request.json
53     msg = data.get("message", "")
54
55     try:
56         response = requests.post(
57             f"http://{ESP32_IP}/print",
58             data={"message": msg},
59             timeout=2
60         )
61         if response.status_code == 200:
62             return {"status": f"OK: {msg}"}
63         else:
64             return {"status": f"Error {response.status_code}"}
65     except Exception as e:
66         return {"status": f"Error: {str(e)}"}
67
68 if __name__ == "__main__":
69     print("\n====")
70     print("XONIDAL SERVER")
71     print("====")
72     print("Local: http://localhost:5050")
73     print("Usuario: admin | Password: 1234")
74     print("====\n")
75     app.run(port=5050, host="0.0.0.0")
```

Archivo requirements.txt

```
1 # XONIDAL - Dependencias
2
3 # -----
4 # ARCH LINUX
5 # -----
6 sudo pacman -S python-pip
7 pip install flask requests --break-system-packages
8
9 # -----
10 # UBUNTU Y DERIVADOS
11 # -----
12 sudo apt update
13 sudo apt install python3 python3-pip -y
14 pip3 install flask requests --break-system-packages
15
16 # -----
17 # WINDOWS
18 # -----
19 pip install flask requests
20
21 # -----
22 # MAC OS
23 # -----
24 brew install python3
25 pip3 install flask requests
26
27 # -----
28 # EJECUCION
29 # -----
30 python start.py
```

EJECUCION DEL SISTEMA

Paso 1: Iniciar Servidor Flask

```
1 # Navegar al directorio del proyecto
2 cd XONIDAL
3
4 # Ejecutar servidor
5 python start.py
```

Paso 2: Acceso Local

- Abrir navegador: `http://localhost:5050`
- Usuario: **admin**
- Contrasena: **1234**
- Ingresar la IP del ESP32

Paso 3: Acceso Global (Opcional)

```
1 # En otra terminal  
2 cloudflared tunnel --url http://localhost:5050
```

Cloudflare generara una URL como: `https://random-name.trycloudflare.com`

OPERACION DEL SISTEMA

Flujo de Operacion Normal

1. **Encender hardware:** ESP32 y Arduino conectados por USB
2. **Iniciar servidor:** `python start.py`
3. **Abrir navegador** en la URL correspondiente
4. **Configurar IP** del ESP32 en la interfaz
5. **Enviar comandos** desde la web
6. **Verificar:** LEDs parpadean al recibir comandos

Comandos de Prueba

Comando	Funcion	Respuesta
HOLA	Cualquier texto	LED parpadea
TEST	Cualquier texto	LED parpadea
LED_ON	Personalizado	Accion especifica
TEMP	Personalizado	Lectura sensor

SOLUCION DE PROBLEMAS CO-MUNES

ESP32 no se conecta a WiFi

```
1 # Verificar credenciales en Monitor Serial  
2 # Comprobar que la red WiFi este disponible  
3 # Verificar que el ESP32 este en modo estacion
```

Arduino no recibe comandos

Verificar conexiones seriales:

- ESP32 TX (GPIO17) → Arduino RX (Pin 0)
- ESP32 RX (GPIO16) ← Arduino TX (Pin 1)
- GND conectados
- Velocidad serial: 9600 baudios

Puerto serial no detectado en Linux

```
1 # Verificar dispositivos  
2 ls -la /dev/ttyUSB* /dev/ttyACM*  
3  
4 # Agregar usuario al grupo dialout  
5 sudo usermod -aG dialout $USER  
6  
7 # Cerrar sesion y volver a entrar
```

Error .^ddress already in use

```
1 # Puerto 5050 ocupado  
2 sudo killall python  
3 # O cambiar puerto en start.py
```

PERSONALIZACION DEL PROYECTO

Modificar Arduino para tu proyecto

Edita la función ejecutarComandoPersonalizado():

```

1 void ejecutarComandoPersonalizado(String comando) {
2     // EJEMPLO: Control de motor DC
3     if (comando.startsWith("MOTOR_")) {
4         int velocidad = comando.substring(6).toInt();
5         analogWrite(9, velocidad);
6         Serial.print("Motor a velocidad: ");
7         Serial.println(velocidad);
8     }
9
10    // EJEMPLO: Lectura de sensor
11    else if (comando == "LUZ") {
12        int luz = analogRead(A0);
13        Serial.print("Nivel de luz: ");
14        Serial.println(luz);
15    }
16
17    // EJEMPLO: Control de relés
18    else if (comando == "RELE1_ON") {
19        digitalWrite(7, HIGH);
20        Serial.println("Rele 1 activado");
21    }
22 }
```

Cambiar credenciales de acceso

En start.py:

```

1 USERNAME = "admin"          # Cambiar por usuario deseado
2 PWD = "1234"                # Cambiar por contraseña segura
```

En esp32_xonidal.ino:

```

1 return (user == "admin" && pass == "1234"); // Actualizar aquí
```

CREDITOS

Desarrollado por:

Darian Alberto Camacho Salas - Backend, Integracion
Oscar Rodolfo Barragan Perez - Frontend, Interfaz Web

Agradecimientos:

Dr. Raul Dali Cruz Morales - Asesoria academica
FESC - UNAM - Facultad de Estudios Superiores Cuautitlan

Contacto:

Instagram: @xonidu

Facebook: xonidu

Email: xonidu@gmail.com