ESPE

UNIVERSIDAD DE LAS FUERZAS ARMADAS

INNOVACIÓN PARA LA EXCELENCIA

UNIDAD DE GESTIÓN DE TECNOLOGÍAS

DEPARTAMENTO DE ELECTRÓNICA Y COMPUTACIÓN

CARRERA DE ELECTRÓNICA MENCIÓN INSTRUMENTACIÓN & AVIÓNICA

PROYECTO DE TITULACIÓN PREVIO A LA OBTENCIÓN DEL TÍTULO DE TECNÓLOGO EN ELECTRÓNICA MENCIÓN INSTRUMENTACIÓN & AVIÓNICA

TEMA: “IMPLEMENTACIÓN DE TRES ARDUINO YUN, QUE PERMITE LA ADQUISICIÓN DE SEÑALES ANALÓGICAS Y DIGITALES EN FORMA INALÁMBRICA PARA PRÁCTICAS DE MICROCONTROLADORES”

AUTOR: VALLEJO CASTILLO WILLIAM MANUEL

DIRECTOR: ING. PABLO PILATASIG

LATACUNGA

2016

DEPARTAMENTO DE ELECTRÓNICA Y COMPUTACIÓN

CARRERA DE ELECTRÓNICA MENCIÓN INSTRUMENTACIÓN & AVIÓNICA

CERTIFICACIÓN

Certifico que el trabajo de titulación, **"IMPLEMENTACIÓN DE TRES ARDUINO YUN, QUE PERMITE LA ADQUISICIÓN DE SEÑALES ANALÓGICAS Y DIGITALES EN FORMA INALÁMBRICA PARA PRÁCTICAS DE MICROCONTROLADORES"** realizado por el señor **VALLEJO CASTILLO WILLIAM MANUEL**, ha sido revisado en su totalidad y analizado por el software anti-plagio, el mismo cumple con los requisitos teóricos, científicos, técnicos, metodológicos y legales establecidos por la Universidad de Fuerzas Armadas ESPE, por lo tanto me permito acreditarlo y autorizar al señor **VALLEJO CASTILLO WILLIAM MANUEL** para que lo sustente públicamente.

Latacunga, 07 de junio del 2016

_____

SR. ING. PABLO PILATASIG

DIRECTOR

DEPARTAMENTO DE ELECTRÓNICA Y COMPUTACIÓN

CARRERA DE ELECTRÓNICA MENCIÓN INSTRUMENTACIÓN & AVIÓNICA

AUTORÍA DE RESPONSABILIDAD

Yo, **VALLEJO CASTILLO WILLIAM MANUEL**, con cédula de identidad Nº 1711059202 declaro que este trabajo de titulación **"IMPLEMENTACIÓN DE TRES ARDUINO YUN, QUE PERMITE LA ADQUISICIÓN DE SEÑALES ANALÓGICAS Y DIGITALES EN FORMA INALÁMBRICA PARA PRÁCTICAS DE MICROCONTROLADORES"** ha sido desarrollado considerando los métodos de investigación existentes, así como también se ha respetado los derechos intelectuales de terceros considerándose en las citas bibliográficas.

Consecuentemente declaro que este trabajo es de mi autoría, en virtud de ello me declaro responsable del contenido, veracidad y alcance de la investigación mencionada.

En virtud de esta declaración, me responsabilizo del contenido, veracidad y alcance científico de trabajo de grado en mención.

Latacunga, 07 de junio del 2016

_____

Vallejo Castillo William Manuel

C.C: 1711059202

DEPARTAMENTO DE ELECTRÓNICA Y COMPUTACIÓN

CARRERA DE ELECTRÓNICA MENCIÓN INSTRUMENTACIÓN & AVIÓNICA

AUTORIZACIÓN

Yo, **VALLEJO CASTILLO WILLIAM MANUEL**, autorizo a la Universidad de las Fuerzas Armadas ESPE publicar en la biblioteca Virtual de la institución el presente trabajo de titulación **"IMPLEMENTACIÓN DE TRES ARDUINO YUN, QUE PERMITE LA ADQUISICIÓN DE SEÑALES ANALÓGICAS Y DIGITALES EN FORMA INALÁMBRICA PARA PRÁCTICAS DE MICROCONTROLADORES"** cuyo contenido, ideas y criterios son de mi autoría y responsabilidad

Latacunga, 07 de junio del 2016

_____

Vallejo Castillo William Manuel

C.C: 1711059202

DEDICATORIA

Tu afecto y tu cariño Camilita son los detonantes de mi felicidad, de mi esfuerzo, de mis ganas de buscar lo mejor para ti. Aun a tu corta edad, me has enseñado y me sigues enseñando muchas cosas de esta vida.

Te agradezco y te dedico a ti mi princesa porque tú fuiste mi motivación más grande para concluir con éxito este proyecto de tesis.

Se la dedico al forjador de mi camino, a mi padre celestial, el que me acompaña y siempre me levanta de mi continuo tropiezo al creador, de mis padres y de las personas que más amo, con mi más sincero amor.

AGRADECIMIENTO

Agradezco a Dios por protegerme durante todo mi camino y darme fuerzas para superar obstáculos y dificultades a lo largo de toda mi vida.

A mi esposa, que supo apoyarme, a no desfallecer ni rendirme ante nada y siempre perseverar. A mi hija Camilita que fue mi motor y mi inspiración para este arduo camino.

A la Institución y al Ing. Pablo Pilatasig, director de tesis, por su valiosa guía y asesoramiento a la realización de la misma.

Gracias a todas las personas que ayudaron directa e indirectamente en la realización de este proyecto.

ÍNDICE DE CONTENIDOS

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

RESUMEN

El presente Proyecto Técnico trata de la implementación de una interface gráfica de usuario (página web) desarrollada en código HTML para visualizar las señales analógicas y digitales enviadas por el Arduino Yun en forma inalámbrica. En la página web se muestra el valor de un sensor de temperatura LM35 conectado al pin A0 del Arduino Yun expresado en grados centígrados, la entrada digital que corresponde al pin D2 se representa mediante la imagen de un led, cuando la imagen es de fondo verde la entrada está en 1 lógica y cuando es rojo la entrada está en 0 lógico, la salida PWM que corresponde al pin D3 se modifica mediante una barra de desplazamiento horizontal, al final de la tabla creada están dos leds que muestran el estado de las salidas digitales D12 y D13. Para establecer la comunicación inalámbrica entre el Arduino Yun y el computador, es necesario habilitar el WiFi del Yun, esto crea una red inalámbrica a la misma que se debe conectar el computador. Cuando el computador está conectado a la red del Arduino Yun, está listo para acceder desde cualquier navegador web mediante la dirección http://arduino.local/ o mediante la dirección IP 192.168.240.1. También se explica otras formas de comunicarse con el Arduino Yun en forma inalámbrica, estas son a través de una consola y la librería puente. Mediante la consola se utilizó el software PuTTy para enviar peticiones al servidor en base a mensajes y con la librería puente se utiliza un navegador web, las peticiones al servidor se escriben en la barra de direcciones del navegador.

Palabras Claves

ARDUINO YUN

HTML

SEÑALES

WIFI

INTERFACE GRÁFICA DE USUARIO

ABSTRACT

This Technical Project is implementing a graphical user interface (website) developed in HTML code to display the analog and digital signals sent by the Arduino Yun wirelessly. On the website the value of a temperature sensor LM35 connected to pin A0 of Arduino Yun expressed in degrees Celsius, the digital input corresponding to pin D2 is represented by the image of a LED is displayed when the image is green background the entrance is on 1 logic and when it is red the input is logic 0, the PWM output corresponding to pin D3 is modified by a horizontal scroll bar at the bottom of the table created are two LEDs that show the status of the outputs D12 and D13 digital. To establish wireless communication between the Arduino Yun and the computer, you need to enable WiFi Yun, it creates a wireless network to it to be connected computer. When the computer is connected to the network Arduino Yun is ready to be accessed from any web browser using http: //arduino.local/ or by IP address 192.168.240.1. other ways of communicating with the Arduino Yun wirelessly, these through a library console and the bridge is also explained. the PuTTy software was used by the console to send requests to the server based on messages and the bridge library a web browser is used, the server requests are written in the browser address bar.

Keywords

ARDUINO YUN

HTML

SIGNALS

WIFI

GRAPHICAL USER INTERFACE

CAPÍTULO I

PLANTEAMIENTO DEL PROBLEMA

1.1 ANTECEDENTES

La Unidad de Gestión de Tecnologías de la Universidad de FF.AA-ESPE es un centro académico de formación tecnológica superior situado en la provincia de Cotopaxi, cantón Latacunga; en la calle Xavier Espinoza y Av. Amazonas, el mismo que cuenta con el Laboratorio de Instrumentación Virtual.

En la Universidad de Valladolid, en septiembre del 2015, se realizó un proyecto, cuyo tema es “Diseño de un programa en Android para el control de Arduino”, el autor fue Frades Estévez Jesús Alberto. En este trabajo se expone el desarrollo de una aplicación en Android para el control de la tarjeta Arduino Yún. La comunicación entre los dos dispositivos se realizó a través de una red WiFi (IEEE 802.11), permitiendo el control de las entradas/salidas digitales así como las entradas analógicas del módulo Arduino Yún. Posibilitando el control de dichas señales de forma remota.

En la Universidad Politécnica de Cataluña, en enero del 2015, se realizó una investigación de Tesis de Maestría, cuyo tema es “Despliegue de una red comunitaria de monitorización de calidad de aire”, el autor fue Omar Orlando Sosa Vaca. Este trabajo tiene como objetivo principal el monitoreo de los gases del aire, usando sensores de bajo costo y Arduino Yun que estarán instalados en varios balcones dentro de la ciudad donde residen personas interesadas en contribuir con el monitoreo del aire a través de lo cual se va a generar una red comunitaria de sensores. Este tema forma parte del “IOT” (Internet of things), donde se utiliza la tecnología para mejorar la vida de los seres humanos. “IOT” es un tema nuevo hoy en día y crece rápidamente porque aparecen nuevos retos y nuevas ideas donde la tecnología puede ayudar mucho. Ejemplo de esto son: Smart parking, monitoreo de parqueaderos libres. Noise Urban maps, monitoreo de ruido en las ciudades. Traffic congestion, monitoreo de congestión vehicular, etc.

1.2 PLANTEAMIENTO DEL PROBLEMA

En la actualidad la industria en general ha avanzado a pasos agigantados junto con el desarrollo de la tecnología, por tal motivo los dueños de empresas e industrias van actualizando constantemente sus máquinas de producción y monitoreo.

La adquisición de señales en forma inalámbrica a través de ARDUINO Yun, es parte fundamental en el aprendizaje de los alumnos y guía importante para los docentes de nuestra institución académica, siendo un perfil muy significativo en la industria.

Al no poseer este tipo de módulos o elementos actualizados en la Unidad de Gestión de Tecnologías ha dado origen a:

- Que los docentes no cuenten con el suficiente material didáctico para impartir conocimientos en adquisición de datos de redes industriales.
- Dificultad para los estudiantes en la manipulación de los equipos y dispositivos electrónicos actualizados.

Por lo expuesto es necesario que la Unidad de Gestión de Tecnologías cuente con los equipos actualizados, elementos, herramientas y técnicas que vayan a la par con la tecnología actual y que faciliten el desarrollo del proceso de aprendizaje, de manera particular el manejo y utilización del ARDUINO Yun en la adquisición en forma inalámbrica de señales.

1.3 JUSTIFICACIÓN

El desarrollo del presente proyecto pretende incentivar a docentes y estudiantes de la Unidad de Gestión de Tecnologías, donde la carrera de Electrónica y Aviónica está involucrada en el desarrollo y progreso del país a través de la industria y su objetivo principal es formar profesionales con conocimientos actualizados capaces de adaptarse a la tecnología avanzada, desempeñándose en cualquier función a la que esté a cargo.

La Unidad de Gestión de Tecnologías al ser un centro de educación Superior tiene la obligación en contar con laboratorios y talleres debidamente equipados, para que sus estudiantes puedan desarrollar Prácticas para adquirir habilidades

y destrezas qué les faciliten su inserción al ámbito laboral una vez culminada su carrera.

El proyecto estará enfocado en implementar módulos ARDUINO Yun de adquisición de señales, con entradas analógicas y digitales, los cuales sus señales serán adquiridas y monitoreadas en forma inalámbricas.

1.4 OBJETIVOS

1.4.1 Objetivo General

Implementar tres módulos ARDUINO Yun que permita la realización de prácticas en microcontroladores para de adquisición de señales analógicas y digitales.

1.4.2 Objetivos Específicos

- Indagar las características de los dispositivos que van a ser utilizados para la adquisición de señales.
- Analizar posibles alternativas de adquisición de señales digitales o analógicas
- Desarrollar prácticas, que ayuden a los estudiantes el proceso de aprendizaje.

1.5 ALCANCE

El proyecto está dirigido a la Carrera de Electrónica Mención Instrumentación y Aviónica de la Unidad de Gestión de Tecnologías como una herramienta muy importante para el proceso de aprendizaje teórico-práctico de los estudiantes y docentes de esta carrera. La finalidad de este proyecto es la configuración de los módulos de adquisición de señales con ARDUINO Yun y realizar aplicaciones con ejemplos prácticos.

CAPÍTULO II

MARCO TEÓRICO

2.1 Arduino

Arduino es una plataforma de hardware libre, basada en una placa con un microcontrolador y un entorno de desarrollo, diseñada para facilitar el uso de la electrónica en proyectos multidisciplinares

Existe una gran cantidad de productos oficiales de Arduino los cuales se muestran en la Figura 1.

Figura 1 Productos Oficiales de Arduino

Fuente: (Arduino, 2015)

2.2 Arduino Yun

El Arduino Yun cuenta con el mismo diseño del Arduino Uno solo que combina un micro controlador ATmega32U4 (igual que el Arduino Leonardo) y

un sistema Linux basado en el chipset Atheros AR9331. Ambos procesadores se pueden comunicar gracias a la librería Bridge que permite enviar comandos de Linux desde los programas (sketches) de Arduino.

El procesador Atheros AR9331, compatible con el servidor Linux, se encarga de la gestión de redes (Ethernet, WiFi), el USB Host (pendrive, teclados, ratones) y la microSD (almacenamiento de datos). (Frades Estévez, 2015)

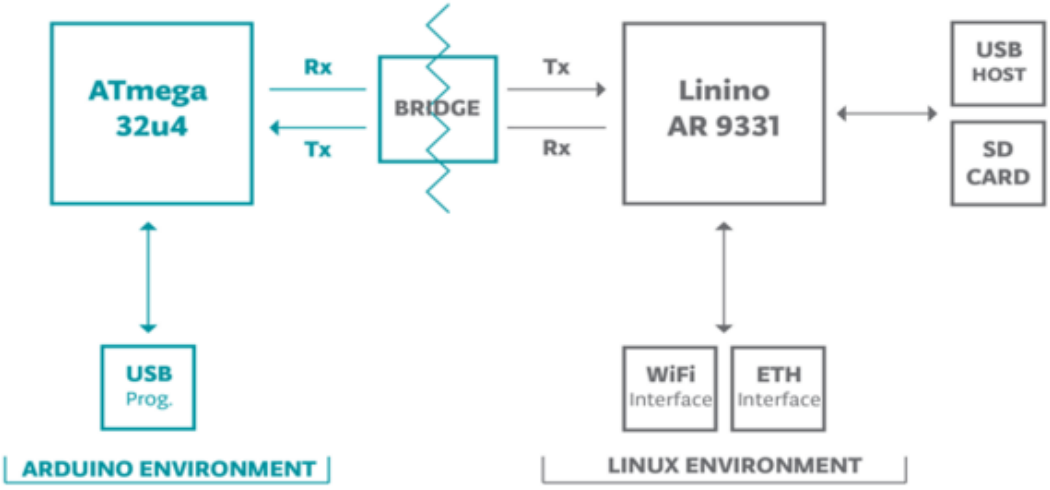

Figura 2 Diagrama de Bloques de Arduino Yun

Fuente: (Arduino, Arduino Yun, 2015)

Esta placa dispone de una combinación de 20 entradas y salidas digitales con 7 canales *PWM* y 12 canales de entrada analógicas. La tensión de funcionamiento es de 5 Voltios y no dispone de regulador de tensión como si cuentan otros modelos de Arduino. De esa forma, en ningún caso se debe superar dicha tensión, ya que produciría una sobrecarga y dañaría el microcontrolador.

Otras de las características de Arduino Yun se describen en las siguientes tablas.

Tabla 1.

Microcontrolador Arduino AVR

Microcontrolador	ATmega32U4
Tensión de funcionamiento	5V
Voltaje de entrada	5V
Pines E / S digitales	20
Canales PWM	7
Pines de entrada analógica	12
Corriente continua para Pin E / S	40 mA
Corriente CC para Pin 3.3V	50 mA
Memoria flash	32 KB (de los cuales 4 KB utilizado por el gestor de arranque)
SRAM	2.5 KB
EEPROM	1 KB
Velocidad de reloj	16 MHz

Tabla 2.

Microprocesador Linux

Procesador	Atheros AR9331
Arquitectura	@ 400 MHz MIPS
Tensión de funcionamiento	3.3V
Ethernet	IEEE 802.3 10 / 100Mbit / s
Wi-Fi	IEEE 802.11b / g / n

Continua

USB tipo A	2.0 Host
Lector de tarjetas	Sólo micro-SD
RAM	64 MB DDR2
Memoria flash	16 MB
SRAM	2.5 KB
EEPROM	1 KB
Velocidad de reloj	16 MHz

2.2.1 Alimentación

Como se detallan en las características anteriores, debe ser alimentado exclusivamente con 5V en corriente continua. Esta alimentación podemos obtenerla, bien a través de la conexión micro-USB que incorpora nuestra placa, o bien a través del pin "*Vin"*, con un regulador de tensión que ofrezca la tensión citada anteriormente.

2.2.2 Entradas y Salidas

Los pines digitales, tanto en las entradas como en las salidas, trabajan a 5 Voltios, pudiendo recibir un máximo de 40 mA (miliamperios). Tiene una resistencia pull-up de 20 a 50 kOhm desconectada por defecto. Las funciones asociados a estos pines son por ejemplo pinMode(), digitalWrite(), digitalRead() entre otras.

En la figura 3, se muestra el hardware del Arduino Yun

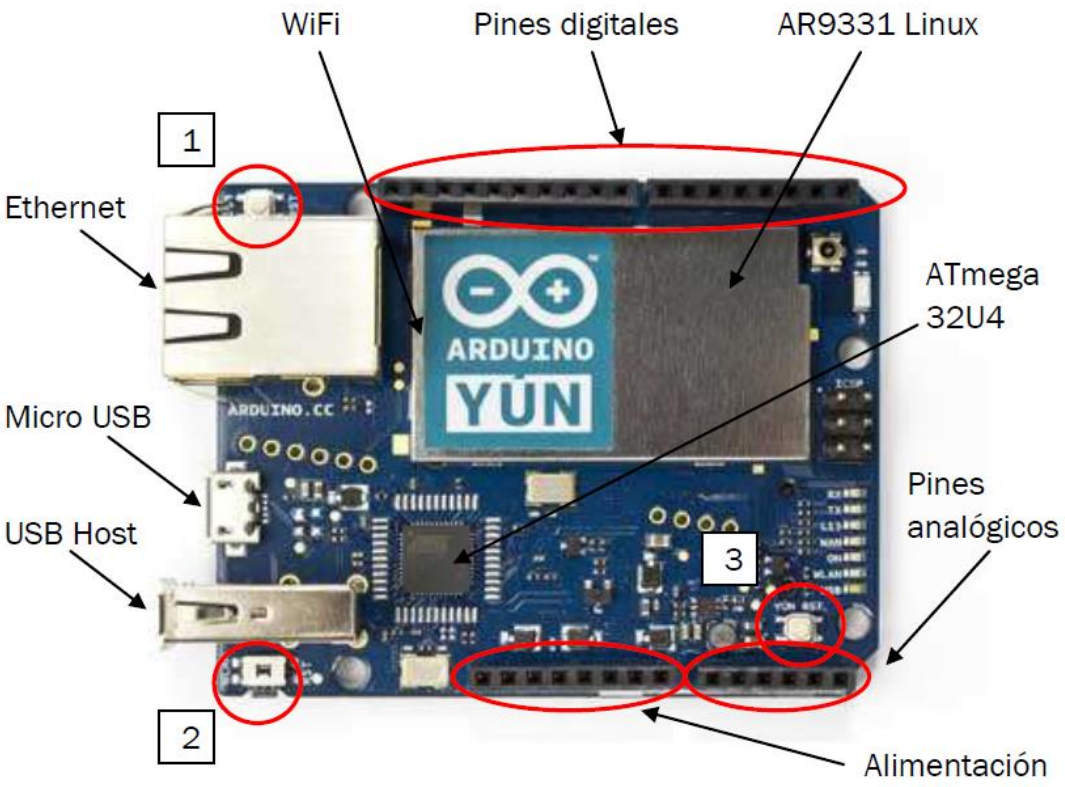

Figura 3 Placa Arduino Yun

Fuente: (Frades Estévez, 2015)

***Serial*: 0 (RX) y 1 (TX).** Se utiliza para recibir (RX) y transmitir (TX) datos serie TTL.

***TWI*: 2 (SDA) y 3 (SCL).** Soporta la comunicación TWI.

***Interrupciones externas*:** 3, 2, 0, 1 y 7. Estos pines se pueden configurar para provocar una interrupción.

***PWM*:** 3, 5, 6, 9, 10, 11 y 13: Proporciona una salida PWM de 8 bits con la función *analogWrite()*.

***SPI*:** En el conector ICSP. Estos terminales soportan la comunicación SPI a través de la librería SPI

***LED 13*:** El pin digital 13 lleva conectado un LED integrado en la propia placa (L13). Se encenderá cuando dicho pin se configura como salida y adopte un valor *HIGH*, mientras que con el valor *LOW* se apaga.

***Otros indicadores de estado*:** Como vemos en la figura 4, el Yun incorpora otros leds para ver el estado de la alimentación, la conexión WLAN, conexión WAN y USB.

***AREF*:** Voltaje de referencia para las entradas analógicas. Se utiliza *analogReference()*.

***Entradas analógicas*:** A0 - A5, A6 – A11 (en los pines digitales 4, 6, 8, 9, 10 y 12). El Arduino Yun tiene 12 entradas analógicas. Cada entrada proporciona 10 bits de resolución (1024 valores).

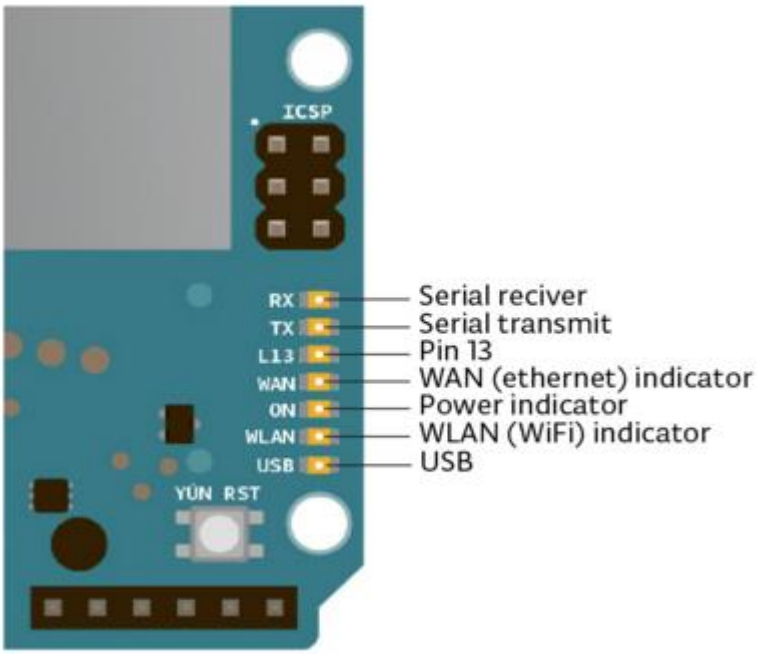

Figura 4 Leds indicadores de estado

Fuente: (Arduino, Arduino Yun, 2015)

Como se observa en la figura 3, existen tres botones en la placa Arduino con una función diferente cada uno:

1. *Botón RST 32U4*: Inicializa el ATmega 32U4 e inicializa todos los programas de Arduino. No se pierden los programas ya que se encuentran en la memoria flash.

2. *Botón RST WLAN*: Tiene una doble función. En primer lugar, pulsando 5 segundos sobre el botón, conseguimos restaurar el WiFi a la configuración de fábrica. La segunda función, presionando durante 30 segundos, conseguimos restaurar la imagen del sistema Linux, con la consiguiente pérdida de aplicaciones instaladas en Linux.

3. *Botón RST Yun*: Inicializa el AR9331 y el sistema Linux. Todos los datos almacenados en la memoria RAM se perderán y todos los programas que se estén ejecutando se terminarán.

La incorporación del AR9331 es la encargada de "administrar" una de las innovaciones del **Yún**: una distribución **Linux** basada en OpenWRT llamada **Linino**. La otra gran novedad la encontramos en una **interfaz WiFi** incorporada en la placa.

2.2.3 OpenWrt Yun

La Yun dirige una distribución de Linux llamada OpenWrt -Yun, basado en OpenWrt. Si bien es posible configurar el sistema desde la línea de comandos, hay una página web que permite configurar muchas de las diferentes opciones disponibles.

OpenWrt se describe como una distribución de Linux para dispositivos embebidos. En lugar de tratar de crear una única firmware estática, OpenWrt proporciona un sistema de archivos totalmente modificable con la gestión de paquetes. Esto le libera de la selección y aplicación de configuración proporcionada por el vendedor y le permite personalizar el dispositivo mediante el uso de paquetes para adaptarse a cualquier aplicación. (OpenWrt, 2015)

2.3 Entorno de Programación de Arduino

2.3.1 Estructura de un Sketch

La estructura básica del lenguaje de programación de Arduino es bastante simple y se compone de al menos dos partes. Estas dos partes necesarias, o funciones, encierran bloques que contienen declaraciones, estamentos o instrucciones. (Evans & Ruiz Gutierrez, 2011)

```
void setup( )
{
estamentos;
}
void loop( )
{
estamentos;
}
```

En donde setup( ) es la parte encargada de recoger la configuración y loop( ) es la que contienen el programa que se ejecutara cíclicamente (de ahí el termino loop ‑bucle-). Ambas funciones son necesarias para que el programa trabaje.

La función de configuración debe contener la declaración de las variables. Es la primera función a ejecutar en el programa, se ejecuta solo una vez, y se utiliza para configurar o inicializar pinMode (modo de trabajo de las E/S), configuración de la comunicación en serie y otras.

La función bucle (loop) contiene el código que se ejecutara continuamente (lectura de entradas, activación de salidas, etc) Esta función es el núcleo de todos los programas de Arduino y la que realiza la mayor parte del trabajo.

2.3.2 Estructura setup( )

La estructura setup( ) se invoca una sola vez cuando el programa empieza. Se utiliza para inicializar los modos de trabajo de los pins, o el puerto serie. Debe ser incluido en un programa aunque no haya declaración que ejecutar. Así mismo se puede utilizar para establecer el estado inicial de las salidas de la placa. (Evans & Ruiz Gutierrez, 2011)

```
void setup()
{
pinMode(pin, OUTPUT);             // configura el 'pin' como salida
}
```

2.3.3 Estructura loop( )

Después de llamar a setup( ), la estructura loop( ) hace precisamente lo que sugiere su nombre, se ejecuta de forma cíclica, lo que posibilita que el programa este respondiendo continuamente ante los eventos que se produzcan en la placa.

```
void loop()
{
digitalWrite(pin, HIGH);      // pone en uno (on, 5v) el´pin´
delay(1000);           // espera un segundo (1000 ms)
digitalWrite(pin, LOW);      // pone en cero (off, 0v.) el
delay(1000);           // ´pin´
}
```

2.3.4 Funciones

Una función es un bloque de código que tiene un nombre y un conjunto de instrucciones que son ejecutadas cuando se llama a la función. Son funciones setup( ) y loop( ) de las que ya se ha hablado. (Evans & Ruiz Gutierrez, 2011)

Las funciones de usuario pueden ser escritas para realizar tareas repetitivas y para reducir el tamaño de un programa. Las funciones se declaran asociadas a un tipo de valor. Este valor será el que devolverá la función, por ejemplo 'int' se utilizará cuando la función devuelva un dato numérico de tipo entero. Si la función no devuelve ningún valor entonces se colocará delante la palabra “void”, que significa “función vacía”. Después de declarar el tipo de dato que devuelve la función se debe escribir el nombre de la función y entre paréntesis se escribirán,

si es necesario, los parámetros que se deben pasar a la función para que se ejecute.

```
tipo nombreFunción(parámetros)
{
instrucciones;
}
```

La función siguiente devuelve un numero entero, delayVal( ) se utiliza para poner un valor de retraso en un programa que lee una variable analógica de un potenciómetro conectado a una entrada de Arduino.

Al principio se declara como una variable local, ´v´ recoge el valor leído del potenciómetro que estará comprendido entre 0 y 1023, luego se divide el valor por 4 para ajustarlo a un margen comprendido entre 0 y 255, finalmente se devuelve el valor ´v´ y se retornaría al programa principal.

```
int delayVal()
{
int v;              // crea una variable temporal 'v'
v= analogRead(pot);          // lee el valor del potenciómetro
v /= 4;              // convierte 0-1023 a 0-255
return v;         // devuelve el valor final
}
```

2.3.5 Variables

Una variable es una manera de nombrar y almacenar un valor numérico para su uso posterior por el programa. Como su nombre indica, las variables son números que se pueden variar continuamente en contra de lo que ocurre con las constantes cuyo valor nunca cambia. Una variable debe ser declarada y, opcionalmente, asignarle un valor. El siguiente código de ejemplo declara una variable llamada variableEntrada y luego le asigna el valor obtenido en la entrada analógica del PIN2:

```
int variableEntrada = 0;      // declara una variable y le asigna el valor 0
variableEntrada = analogRead(2);          // la variable recoge el valor analógico
                                          //  del PIN2
```

'variableEntrada' es la variable en sí. La primera línea declara que sera de tipo entero "int". La segunda línea fija a la variable el valor correspondiente a la entrada analógica PIN2. Esto hace que el valor de PIN2 sea accesible en otras partes del código.

Una vez que una variable ha sido asignada, o re-asignada, usted puede probar su valor para ver si cumple ciertas condiciones, o puede utilizar directamente su valor.

Como ejemplo ilustrativo veamos tres operaciones útiles con variables: el siguiente código prueba si la variable "entradaVariable" es inferior a 100, si es cierto se asigna el valor 100 a "entradaVariable" y, a continuación, establece un retardo (delay) utilizando como valor "entradaVariable" que ahora será como mínimo de valor 100:

```
if (entradaVariable < 100)      // pregunta si la variable es menor de 100
{
entradaVariable = 100;          // si es cierto asigna el valor 100
}
delay(entradaVariable);         // usa el valor como retardo
```

2.3.6 Declaración de variables

Todas las variables tienen que declararse antes de que puedan ser utilizadas. Para declarar una variable se comienza por definir su tipo como int (entero), long (largo), float (coma flotante), etc, asignándoles siempre un nombre, y, opcionalmente, un valor inicial.

Esto sólo debe hacerse una vez en un programa, pero el valor se puede cambiar en cualquier momento usando aritmética y reasignaciones diversas.

El siguiente ejemplo declara la variable entradaVariable como una variable de tipo entero “int”, y asignandole un valor inicial igual a cero. Esto se llama una asignación.

int entradaVariable = 0;

Una variable puede ser declarada al inicio del programa antes de la parte de configuración setup(), a nivel local dentro de las funciones, y, a veces, dentro de un bloque, como para los bucles del tipo if.. for.., etc. En función del lugar de declaración de la variable asi se determinara el ámbito de aplicación, o la capacidad de ciertas partes de un programa para hacer uso de ella.

Una variable global es aquella que puede ser vista y utilizada por cualquier función y estamento de un programa. Esta variable se declara al comienzo del programa, antes de setup().

Una variable local es aquella que se define dentro de una función o como parte de un bucle. Solo es visible y solo puede utilizarse dentro de la función en la que se declaró.

Por lo tanto, es posible tener dos o más variables del mismo nombre en diferentes partes del mismo programa que pueden contener valores diferentes. La garantía de que solo una función tiene acceso a sus variables dentro del programa simplifica y reduce el potencial de errores de programación.

El siguiente ejemplo muestra como declarar a unos tipos diferentes de variables y la visibilidad de cada variable:

```
int value;      // 'value' es visible para cualquier función
void setup()
{
// no es necesario configurar nada en este ejemplo
}
```

```
void loop()
{
for (int i=0; i<20;)     // 'i' solo es visible dentro del bucle for
{
i++
} /
float f;            // 'f' es visible solo dentro de loop()
}
```

2.3.7 Tipos de datos

byte

Byte almacena un valor numerico de 8 bits sin decimales. Tienen un rango entre 0 y 255.

```
byte unaVariable = 180; // declara 'unaVariable' como de tipo byte
```

int

Enteros son un tipo de datos primarios que almacenan valores numericos de 16 bits sin decimales comprendidos en el rango 32,767 to -32,768.

```
int x = 1500; // declara 'x' como una variable de tipo entero
```

Las variables de tipo entero "int" pueden sobrepasar su valor máximo o mínimo como consecuencia de una operación. Por ejemplo, si x = 32767 y una posterior declaración agrega 1 a x, x = x + 1 entonces el valor se x pasara a ser -32.768. (Algo así como que el valor da la vuelta).

long

El formato de variable numérica de tipo extendido "long" se refiere a números enteros (tipo 32 bits) sin decimales que se encuentran dentro del rango -2147483648 a 2147483647.

```
long unaVariable = 90000; // declara 'unaVariable' como de tipo long
```

float

El formato de dato del tipo "punto flotante" "float" se aplica a los numeros con decimales. Los numeros de punto flotante tienen una mayor resolucion que los de 32 bits con un rango comprendido 3.4028235E +38 a +38-3.4028235E.

```
float unaVariable = 3.14; // declara 'unaVariable' como de tipo flotante
```

arrays

Un array es un conjunto de valores a los que se accede con un número índice. Cualquier valor puede ser recogido haciendo uso del nombre de la matriz y el número del índice. El primer valor de la matriz es el que está indicado con el índice 0, es decir el primer valor del conjunto es el de la posición 0. Un array tiene que ser declarado y opcionalmente asignados valores a cada posición antes de ser utilizado.

```
int miArray[] = {valor0, valor1, valor2...}
```

Del mismo modo es posible declarar una matriz indicando el tipo de datos y el tamaño y posteriormente, asignar valores a una posición especifica:

```
int miArray[5];          // declara un array de enteros de 6 posiciones
miArray[3] = 10;         // asigna el valor 10 a la posición 4
```

Las matrices se utilizan a menudo para estamentos de tipo bucle, en los que la variable de incremento del contador del bucle se utiliza como índice o puntero del array. El siguiente ejemplo usa una matriz para el parpadeo de un LED.

Utilizando un bucle tipo for, el contador comienza en cero 0 y escribe el valor que figura en la posición de índice 0 en la serie que se escribió dentro del array parpadeo[ ], en este caso 180, que se envía a la salida analógica tipo PWM configurada en el PIN10, se hace una pausa de 200 ms y a continuación se pasa al siguiente valor que asigna el índice "i".

```
int ledPin = 10;        // LED en el PIN 10

byte parpadeo[ ] = {180, 30, 255, 200, 10, 90, 150, 60};  // array de 8 valores

void setup()

{

pinMode(ledPin, OUTPUT);          // configura la salida

}

void loop()

{

        for(int i=0; i<7; i++)

        {

        analogWrite(ledPin, parpadeo[ i ]);

        delay(200);              // espera 200ms

        }

}
```

2.3.8 Constantes

El lenguaje de programación de Arduino tiene unos valores predeterminados, que son llamados constantes. Se utilizan para hacer los programas más fáciles de leer. Las constantes se clasifican en grupos.

high/low

Estas constantes definen los niveles de salida altos o bajos y se utilizan para la lectura o la escritura digital para las patillas. ALTO se define como en la lógica de nivel 1, ON, o 5 voltios, mientras que BAJO es lógica nivel 0, OFF, o 0 voltios.

digitalWrite(13, HIGH); // activa la salida 13 con un nivel alto (5v.)

input/output

Estas constantes son utilizadas para definir, al comienzo del programa, el modo de funcionamiento de los pines mediante la instrucción pinMode de tal manera que el pin puede ser una entrada INPUT o una salida OUTPUT.

pinMode(13, OUTPUT); // designamos que el PIN 13 es una salida

2.3.9 Entradas/Salidas digitales

pinMode(pin, mode)

Esta instrucción es utilizada en la parte de configuración setup () y sirve para configurar el modo de trabajo de un PIN pudiendo ser INPUT (entrada) u OUTPUT (salida).

pinMode(pin, OUTPUT); // configura ‘pin’ como salida

Los terminales de Arduino, por defecto, estan configurados como entradas, por lo tanto no es necesario definirlos en el caso de que vayan a trabajar como entradas. Los pines configurados como entrada quedan, bajo el punto de vista electrico, como entradas en estado de alta impedancia.

Estos pines tienen a nivel interno una resistencia de 20 KΩ a las que se puede acceder mediante software. Estas resistencias se acceden de la siguiente manera:

pinMode(pin, INPUT); // configura el ‘pin’ como entrada

digitalWrite(pin, HIGH); // activa las resistencias internas

Las resistencias internas normalmente se utilizan para conectar las entradas a interruptores. En el ejemplo anterior no se trata de convertir un pin en salida, es simplemente un método para activar las resistencias interiores.

Los pines configurados como OUTPUT (salida) se dice que están en un estado de baja impedancia y pueden proporcionar 40 mA (miliamperios) de corriente a otros dispositivos y circuitos. Esta corriente es suficiente para alimentar un diodo LED (no olvidando poner una resistencia en serie), pero no es lo suficiente grande como para alimentar cargas de mayor consumo como relés, solenoides, o motores.

Un cortocircuito en las patillas Arduino provocara una corriente elevada que puede dañar o destruir el chip Atmega. A menudo es una buena idea conectar en la OUTUPT (salida) una resistencia externa de 470 o de 1000 Ω.

digitalRead(pin)

Lee el valor de un pin (definido como digital) dando un resultado HIGH (alto) o LOW (bajo). El pin se puede especificar ya sea como una variable o una constante (0-13).

valor = digitalRead(Pin); // hace que 'valor sea igual al estado leído en ´Pin´

digitalWrite(pin, value)

Envia al ´pin´ definido previamente como OUTPUT el valor HIGH o LOW (poniendo en 1 o 0 la salida). El pin se puede especificar ya sea como una variable o como una constante (0-13).

digitalWrite(pin, HIGH); // deposita en el 'pin' un valor HIGH (alto o 1)

2.3.10 Entradas/Salidas analógicas

analogRead(pin)

Lee el valor de un determinado pin definido como entrada analógica con una resolución de 10 bits. Esta instrucción sólo funciona en los pines (0-5). El rango de valor que podemos leer oscila de 0 a 1023.

valor = analogRead(pin); // asigna a valor lo que lee en la entrada ´pin'

analogWrite(pin, value)

Esta instrucción sirve para escribir un pseudo-valor analógico utilizando el procedimiento de modulación por ancho de pulso (PWM) a uno de los pines de Arduino marcados como “pin PWM”. El valor que se puede enviar a estos pines de salida analógica puede darse en forma de variable o constante, pero siempre con un margen de 0-255.

analogWrite(pin, valor); // escribe 'valor' en el 'pin' definido como analógico

Si se envía el valor 0 genera una salida de 0 voltios en el pin especificado; un valor de 255 genera una salida de 5 voltios de salida en el pin especificado.

2.4 Protocolo SSH

SSH (o Secure SHell) es un protocolo que facilita las comunicaciones seguras entre dos sistemas usando una arquitectura cliente/servidor y que permite a los usuarios conectarse a un host remotamente. A diferencia de otros protocolos de comunicación remota tales como FTP o Telnet, SSH encripta la sesión de conexión, haciendo imposible que alguien pueda obtener contraseñas no encriptadas. (Enterprise, 2009)

SSH está diseñado para reemplazar los métodos más viejos y menos seguros para registrarse remotamente en otro sistema a través de la shell de comando, tales como telnet o rsh. Un programa relacionado, el scp, reemplaza otros programas diseñados para copiar archivos entre hosts como rcp. Ya que estas aplicaciones antiguas no encriptan contraseñas entre el cliente y el

servidor, evite usarlas mientras le sea posible. El uso de métodos seguros para registrarse remotamente a otros sistemas reduce los riesgos de seguridad tanto para el sistema cliente como para el sistema remoto.

2.4.1 Características

El protocolo SSH proporciona los siguientes tipos de protección:

- Después de la conexión inicial, el cliente puede verificar que se está conectando al mismo servidor al que se conectó anteriormente.
- El cliente transmite su información de autenticación al servidor usando una encriptación robusta de 128 bits.
- Todos los datos enviados y recibidos durante la sesión se transfieren por medio de encriptación de 128 bits, lo cual los hacen extremamente difícil de descifrar y leer.
- El cliente tiene la posibilidad de reenviar aplicaciones X11 desde el servidor. Esta técnica, llamada reenvío por X11, proporciona un medio seguro para usar aplicaciones gráficas sobre una red.

Ya que el protocolo SSH encripta todo lo que envía y recibe, se puede usar para asegurar protocolos inseguros. El servidor SSH puede convertirse en un conducto para convertir en seguros los protocolos inseguros mediante el uso de una técnica llamada reenvío por puerto, como por ejemplo POP, incrementando la seguridad del sistema en general y de los datos.

2.5 PuTTY

PuTTY es un cliente Telnet y SSH gratuito para plataformas Windows que le permite conectarse a un servidor remoto desde un PC conectado a Internet.

PuTTY posee muchas opciones de configuración; es asi que podemos configurarlo de tal modo que nos permita una conexión a cualquier tipo de servidor. Además, no necesita instalarlo porque sólo se necesita el ejecutable.

A pesar de su aparente simplicidad, PuTTY es altamente configurable e incluye muchas opciones de ajuste de las conexiones, sesiones, características

SSH de seguridad y hasta apariencia de la ventana. Ello se debe a que el programa está particularmente dirigido a programadores y administradores de red, lo que significa que los novatos estarán obligados a trabajar un poco más. Pero, sin embargo, el programa cuenta con una interfaz sencilla y directa, que facilitara su tarea de inclusión.

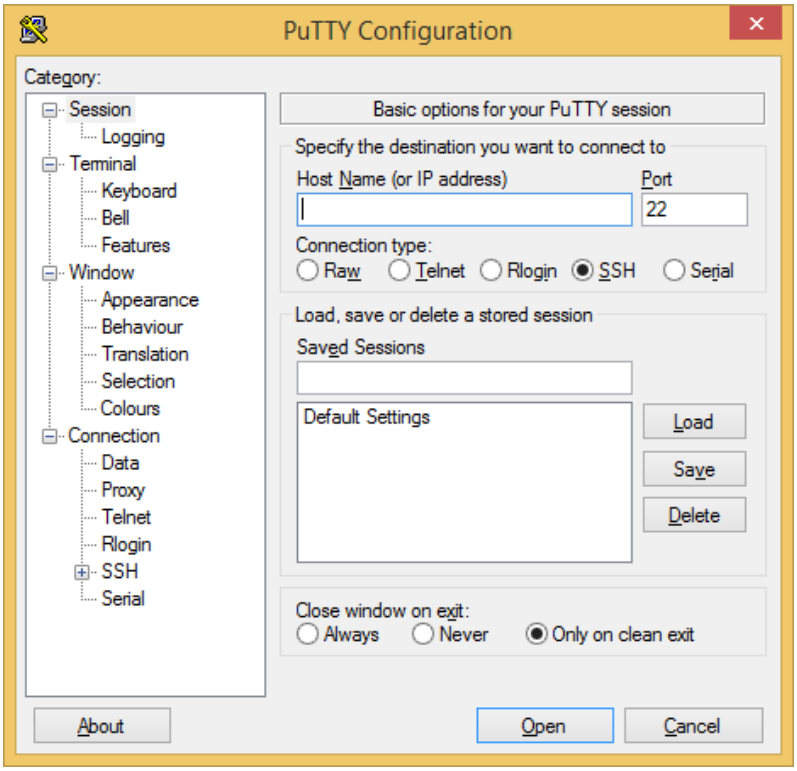

Figura 5 Ventana de configuración de PuTTY

2.6 Direccionamiento IP

Una dirección IP es un número de identificación de un ordenador o de una red (subred) - depende de la máscara que se utiliza. Dirección IP es una secuencia de unos y ceros de 32 bits expresada en cuatro octetos (4 byte) separados por puntos. Para hacer más comprensible se denomina en decimal como cuatro números separados por puntos. (Marbit, 2010)

En binario 10101100.00011000.00000111.00101011

En decimal 172.24.7.43

Dirección IP privada identifica el equipo dentro de una red LAN - Local Area Networks - dentro de una empresa o red doméstica.

Dirección IP pública identifica el equipo en internet. Es única - no se puede repetir.

Una dirección IP consta de dos partes. Primera parte identifica dirección de la red y la segunda sirve para identificar los equipos en la red. Para saber que rango de bits corresponde para cada parte se utiliza la máscara.

Máscara es combinación de 32 bits expresados en cuatro octetos (4 byte) separados por puntos. Es utilizada para describir cuál es la porción de una dirección IP que se refiere a la red o subred y cuál es la que se refiere al host. La máscara se utiliza para extraer información de red o subred de la dirección IP.

Dirección IP: 192.168.15.43

Máscara: 255.255.255.0

Tabla 3.

Explicación direccionamiento IP

IP	**192**	**168**	**15**	**43**
Máscara	255	255	255	0
	Dirección de Red			**Dirección de Host**

2.6.1 Clases de direccionamiento IP

Las direcciones IP se dividen en clases para definir las redes de tamaño grande (A), mediano (B), pequeño (C), de uso multicast (D) y de uso experimental (E). Dentro de cada rango de clases A, B, C existen direcciones privadas para uso interno y no las veremos en internet (Normativa RFC 1918).

Clase A

- Rango de direcciones IP: 1.0.0.0 a 126.0.0.0
- Máscara de red: 255.0.0.0
- Direcciones privadas: 10.0.0.0 a 10.255.255.255

Clase B

- Rango de direcciones IP: 128.0.0.0 a 191.255.0.0
- Máscara de red: 255.255.0.0
- Direcciones privadas: 172.16.0.0 a 172.31.255.255

Clase C

- Rango de direcciones IP: 192.0.0.0 a 223.255.255.0
- Máscara de red: 255.255.255.0
- Direcciones privadas: 192.168.0.0 a 192.168.255.255

Clase D

- Rango de direcciones IP: 224.0.0.0 a 239.255.255.255 uso multicast o multidifusión

Clase E

- Rango de direcciones IP: 240.0.0.0 a 254.255.255.255 uso experimental

La dirección **127.0.0.0/8** se denomina como - LoopBack Address - no se puede usar para direccionamiento privado o público.

La máscara **255.255.255.255 o /32** sirve para identificar un host específico.

Los métodos para expresar la máscara:

- Clase A **255.0.0.0** o **/8**
- Clase B **255.255.0.0** o **/16**
- Clase C **255.255.255.0** o **/24**

2.7 `Comunicación Inalámbrica WiFi

WIFI es una de las tecnologías de comunicación inalámbrica mediante ondas más utilizada hoy en día. WIFI, también llamada WLAN (wireless lan, red inalámbrica) o estándar IEEE 802.11.

El comité IEEE 802.11 es el encargado de desarrollar los estándares para las redes de área local inalámbricas. Este estándar, se basa en el mismo marco de estándares que Ethernet, garantizando un excelente nivel de interoperatividad y asegurando una implantación sencilla de las funciones y dispositivos de interconexión Ethernet/WLAN. (Kioskea, 2014)

El estándar 802.11 establece los niveles inferiores del modelo OSI para las conexiones inalámbricas que utilizan ondas electromagnéticas, por ejemplo:

- La capa física (a veces abreviada capa "PHY") ofrece tres tipos de codificación de información.
- La capa de enlace de datos compuesta por dos subcapas: control de enlace lógico (LLC) y control de acceso al medio (MAC).

La capa física define la modulación de las ondas de radio y las características de señalización para la transmisión de datos mientras que la capa de enlace de datos define la interfaz entre el bus del equipo y la capa física, en particular un método de acceso parecido al utilizado en el estándar Ethernet, y las reglas para la comunicación entre las estaciones de la red.

2.7.1 Estándares WiFi

802.11b

El estándar 802.11b permite un máximo de transferencia de datos de 11 Mbps en un rango de 100 metros aproximadamente en ambientes cerrados y de más de 200 metros al aire libre (o incluso más que eso con el uso de antenas direccionales).

802.11g

El estándar 802.11g permite un máximo de transferencia de datos de 54 Mbps en rangos comparables a los del estándar 802.11b. Además, y debido a que el estándar 802.11g utiliza el rango de frecuencia de 2.4 GHz con codificación OFDM, es compatible con los dispositivos 802.11b con excepción de algunos dispositivos más antiguos.

802.11n

Esta enmienda mejora los protocolos anteriores mediante la utilización de múltiples antenas de entrada y salida (MIMO). Puede funcionar en las bandas de 2.4 GHz y 5 GHz lo cual posibilita velocidades de 54 Mbit/s hasta 600 Mbit/s.

2.8 HTML

El lenguaje de marcas de hipertexto, HTML o (HyperText Markup Language) se basa en el metalenguaje SGML (*Standard Generalized Markup Language*) y es el formato de los documentos de la World Wide Web. El World Wide Web Consortium (W3C) es la organización que desarrolla los estándares para normalizar el desarrollo y la expansión de la Web y la que publica las especificaciones relativas al lenguaje HTML. (Lamarca Lapuente, 2013)

HTML fue concebido como un lenguaje para el intercambio de documentos científicos y técnicos adaptado para su uso por no especialistas en tratamiento de documentos. HTML resolvió el problema de la complejidad de SGML sirviéndose de un reducido conjunto de etiquetas estructurales y semánticas apropiadas para la realización de documentos relativamente simples. Pero, además de simplificar la estructura de los documentos, HTML soportaba el hipertexto.

Para crear documentos HTML sólo es necesario:

- **Un procesador de textos o un editor de documentos HTML**
- **Un navegador del WWW** o lo que se denomina "programa cliente" que permite el acceso a páginas WWW de Internet.

2.8.1 Estructura de un documento HTML

Un documento HTML comienza con la etiqueta <html>, y termina con </html>.

Dentro del documento hay dos zonas principales: el encabezamiento, delimitado por las marcas <HEAD> y </HEAD>, que sirve para definir algunos valores válidos para todo el documento, y el cuerpo, delimitado por las etiquetas <BODY> y </BODY>, donde reside la información del documento.

El elemento **<TITLE>** contenido dentro del encabezamiento permite especificar el título de un documento HTML. Este título no forma parte del documento en sí pues no se ve en la pantalla principal, sino que sirve como **título de la ventana** del programa que la muestra.

Existen muchos otros elementos que se engloban dentro del encabezamiento pero para la estructura básica del lenguaje HTML en su nivel básico no son necesarios.

El cuerpo de un documento HTML contiene el texto, imágenes, etc. que, con la presentación y los efectos que se decidan, se presentarán ante el usuario. Dentro del cuerpo se pueden aplicar una serie de efectos a través de diferentes marcas o etiquetas (también otros autores las denominan "directivas").

Así pues, la estructura de un documento HTML es la siguiente:

<HTML>
<HEAD>
<TITLE>Título de la página**</TITLE>**
</HEAD>
<BODY>
[Aquí se sitúan otras etiquetas que hacen posible visualizar la página]
</BODY>
</HTML>

Saltos de línea:

HTML no reconoce los finales de línea del editor de texto, pero la etiqueta
 desplaza el texto a la línea siguiente, y la etiqueta <P> también lo desplaza, dejando una línea de separación.

Tipos de letra:

Los tipos básicos son negrita, cursiva y teletipo o máquina de escribir, que utilizan los códigos **B**, **I**, **TT,** respectivamente.

<B>negrita</B>

<I>cursiva</I>

<TT>teletipo</TT>

Tabla 4.

Etiquetas para mostrar tablas en código HTML

Etiqueta de inicio	Etiqueta final	Descripción de la etiqueta
<TABLE>	</TABLE>	Contenedor para los bordes de la tabla
<TABLE BORDER>	</TABLE>	Etiqueta para tabla con bordes
<TR>	</TR>	Establece una fila dentro de una tabla
<TD>	</TD>	Define una celda dentro de un tabla
<TH>	</TH>	Centra una cabecera en la parte superior de la tabla o en un lado
<CAPTION>	</CAPTION>	Fija un título en la parte superior de la tabla.

CAPÍTULO III

DESARROLLO DEL PROYECTO

3.1 Preliminares

Para la implementación del proyecto de Titulación cuyo tema es: **IMPLEMENTACIÓN DE TRES ARDUINO YUN, QUE PERMITE LA ADQUISICIÓN DE SEÑALES ANALÓGICAS Y DIGITALES EN FORMA INALÁMBRICA PARA PRÁCTICAS DE MICROCONTROLADORES,** fue necesario lo siguiente:

- Arduino Yun
- Computador o laptop con tarjeta inalámbrica
- Software PuTTY
- Sensor de temperatura LM35
- Pulsadores
- Diodos led
- Resistencias
- Fuente de alimentación de 5 VDC

3.2 Activación de la red WiFi del Arduino YUN

Una vez que está instalado Arudino IDE, conecte la tarjeta Arduino Yun en cualquier puerto USB del computador, permanecerá encendido el led indicador ON.

Después de algunos segundos se encenderá el led indicador etiquetado como USB, el mismo que indicará que esta activada la red WiFi del Yun, en ese instante aparecerá en el computador una red llamada Arduino Yun-*XXXXXXXXXXX*.

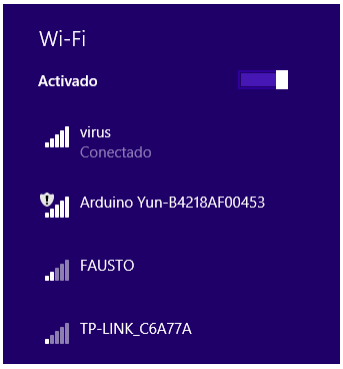

Figura 6 Red WiFi de Arduino Yun

En el caso de que aparezca la red del Arduino Yun en el computador, pulse el botón WLAN RST varias veces, hasta que se muestre la red inalámbrica.

En siguiente paso es conectarse a la red, para el caso del proyecto la red detectada se llama Arduino Yun-B4218AF00453.

Una vez conectada a la red, abra un explorador web e ingrese http://arduino.local o la dirección IP 192.168.240.1 en la barra de direcciones. Después de unos momentos, aparecerá una página web para pedir una contraseña, escriba **arduino** y de un clic en el botón **LOG IN.**

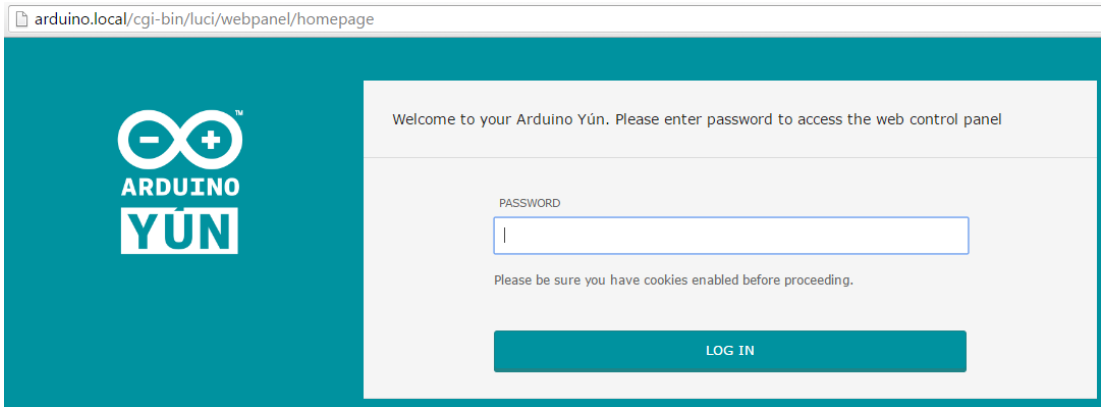

Figura 7 Mensaje de bienvenida al panel web

Al dar clic en el botón LOG IN, aparece una ventana donde esta especificada la dirección IP de la red inalámbrica del Arduino Yun.

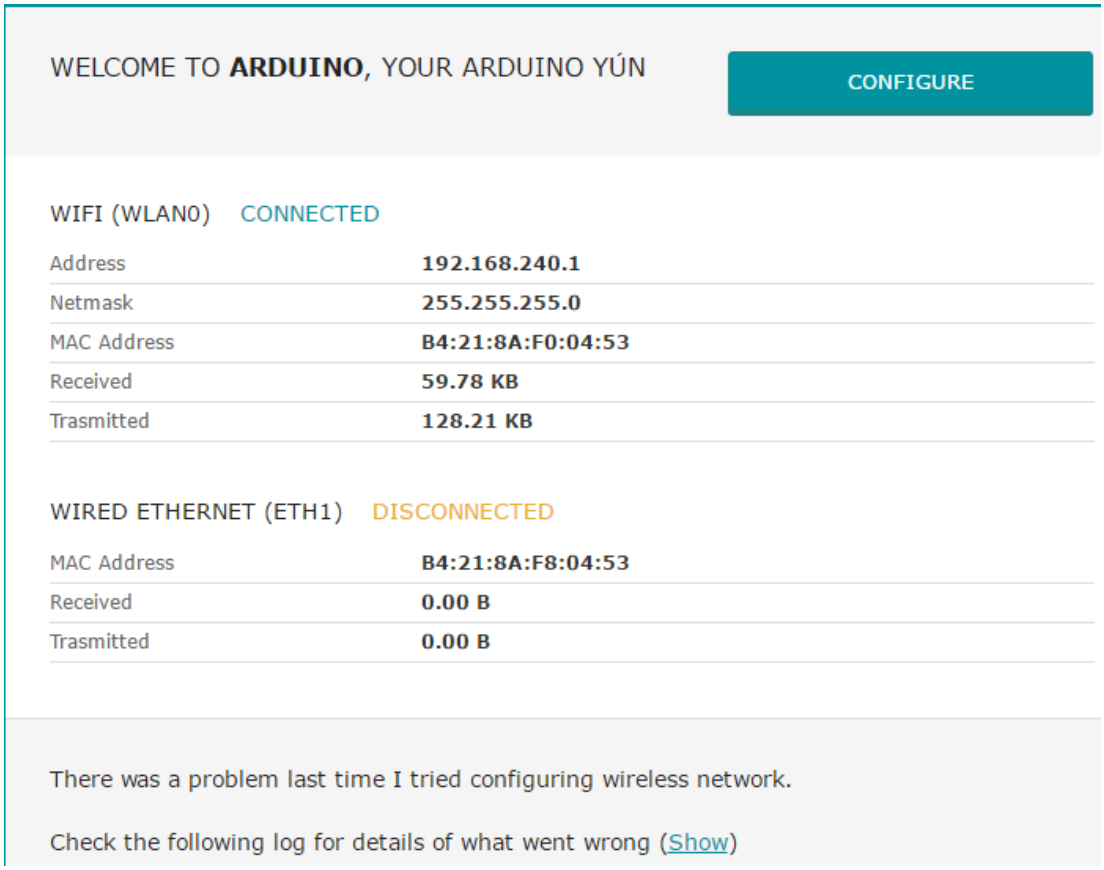

Figura 8 Visualización de la IP del Arduino Yun

Para configurar los parámetros del Yun se debe pulsar el botón CONFIGURE, caso contrario, cierre el navegador web.

3.3 Programación del Atmega 32 mediante WiFi

Cuando el Arduino Yun está en la misma red del computador, se puede descargar inalámbricamente cualquier programa al microcontrolador Atmega 32.

Para esto ejecute el IDE de arduino, abra el programa a descargar, en el menú Herramientas >Puerto seleccione de la lista el Arduino Yun, este viene junto con la dirección IP

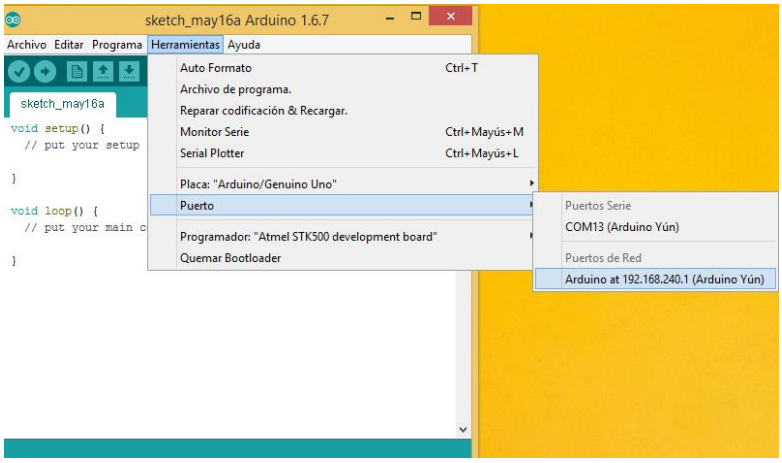

Figura 9 Puerto inalámbrico del Arduino Yun

De la misma manera seleccione la placa Arduino Yun.

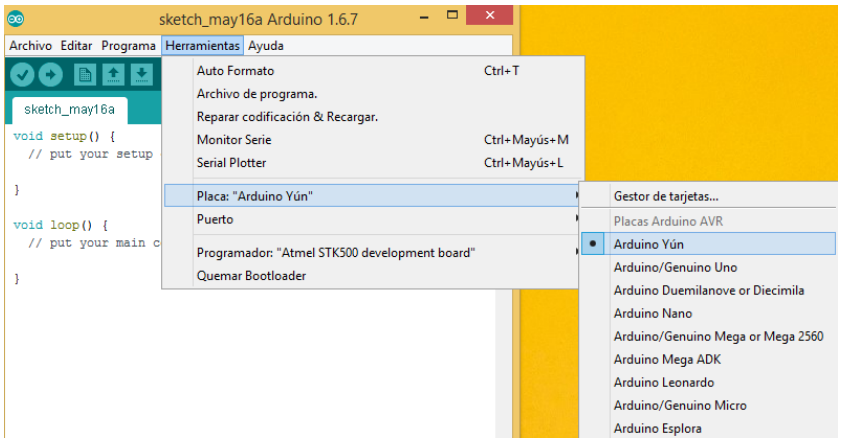

Figura 10 Selección de la tarjeta Arduino Yun

De clic en el botón subir, cuando aparezca la ventana de contraseña escriba **arduino** y el programa empezará a transferirse a la tarjeta Arduino Yun en forma inalámbrica.

3.4 Comunicación inalámbrica mediante consola

Este ejercicio se lo realiza mediante el protocolo de comunicación SSH que es un protocolo de comunicación entre equipos a distancia, conectados entre sí a través de una red. Arduino Yún es capaz de establecer comunicación inalámbrica por medio de SSH, con una computadora.

Para que exista la comunicación por medio de SSH será necesario utilizar un software que funcione como terminal para establecer la comunicación. El software que se usa es Putty

El programa que se carga al Arduino Yun es el siguiente:

```
#include <Console.h>

 const int ledPin = 13; // se declara el pin donde se conectará el LED

int incomingByte;      // la variable que contendrá la información de entrada


void setup() {

  // Se inicia la comunicación

  Bridge.begin();

  Console.begin();


  while (!Console){

    ; // Espera hasta que se conecte la consola

  }

  Console.println("Se ha conectado a la consola!!!!");

  // Se establece el pin donde se conecta el LED como salida

  pinMode(ledPin, OUTPUT);

}


void loop() {

  // Se verifica si hay información entrante

  if (Console.available() > 0) {
```

```
    // Se lee la información entrante
    incomingByte = Console.read();
    // Se hace la comparación. Si la entrada es una H (mayúscula) se enciende el LED
    if (incomingByte == 'H') {
      digitalWrite(ledPin, HIGH);
      Console.println("Se ha encendido el LED");
    }
    // Si la entrada es una L (mayúscula), se apaga el LED
    if (incomingByte == 'L') {
      digitalWrite(ledPin, LOW);
      Console.println("Se ha apagado el LED");
    }
  }
  delay(100);
  }
```

El código anterior hará que un LED conectado al pin 13 se encienda cuando, por medio de la consola en Putty, se envíe una **H**. Al enviar una **L**, el led se apagará. Cuando el Arduino Yun reciba la **H** enviará a la consola el mensaje "*Se ha encendido el LED"* y cuando reciba la **L** enviará a la consola el mensaje *"Se ha apagado el LED"*.

Abra el software Putty, escriba la dirección IP del Arduino Yun 192.168.240.1, el puerto de acceso 22 y el tipo de conexión SSH. Pulse el botón Open.

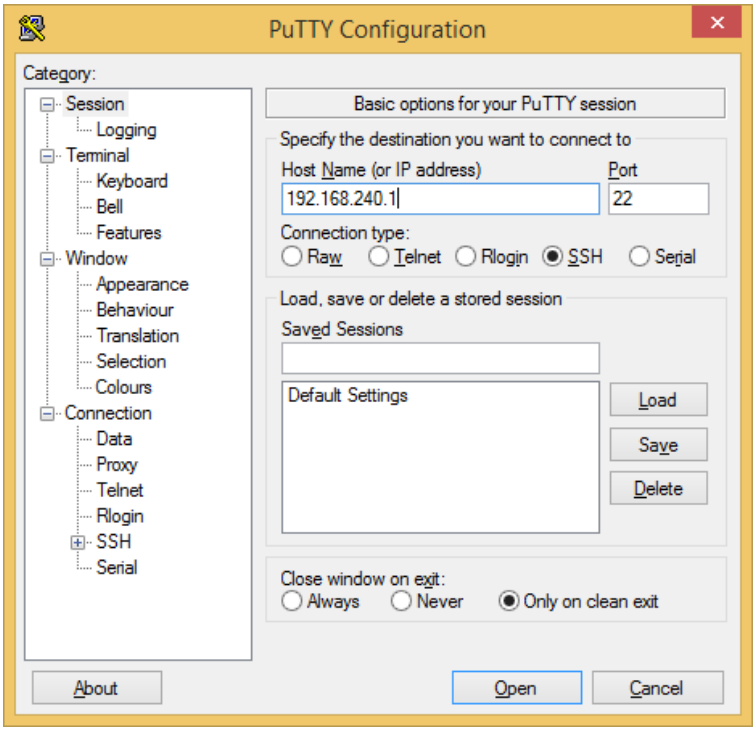

Figura 11 Ventana de configuración del software PuTTy

En **login as** escriba **root** y

En **password** escriba **arduino**, pulse enter

Figura 12 Venta de login y password de PuTTy

Cuando se tiene acceso, la ventana que aparecerá es la siguiente:

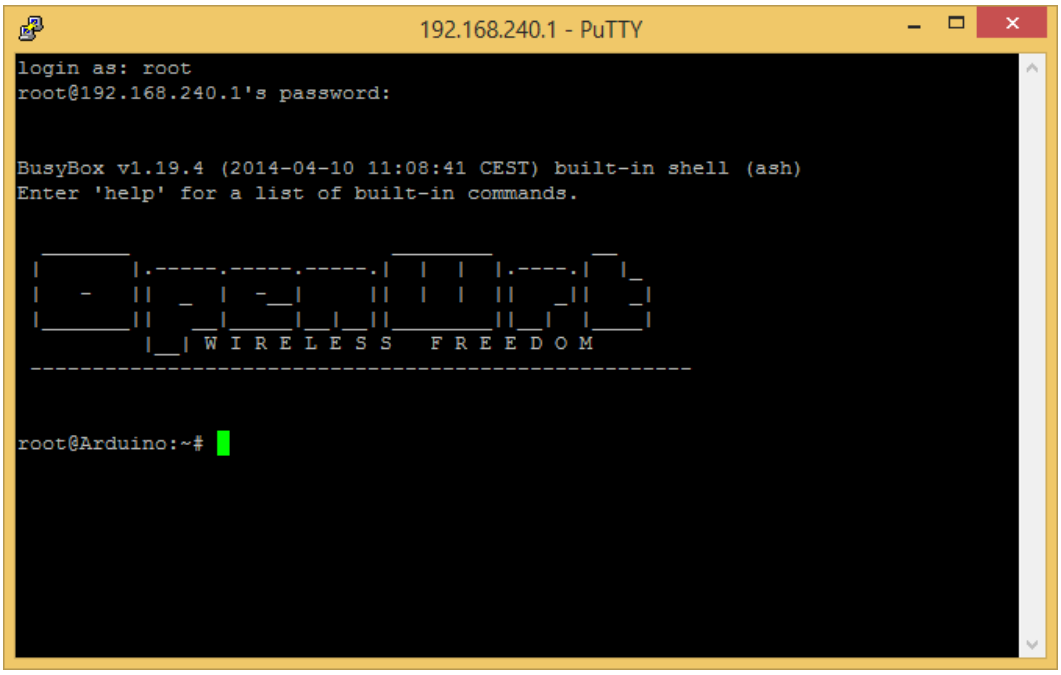

Figura 13 Ventana de consola de PuTTy

Ahora se debe enviar un comando al Arduino Yun para iniciar la consola. Teclee el comando ***telnet localhost*** **6571** en Putty y se obtiene el siguiente resultado.

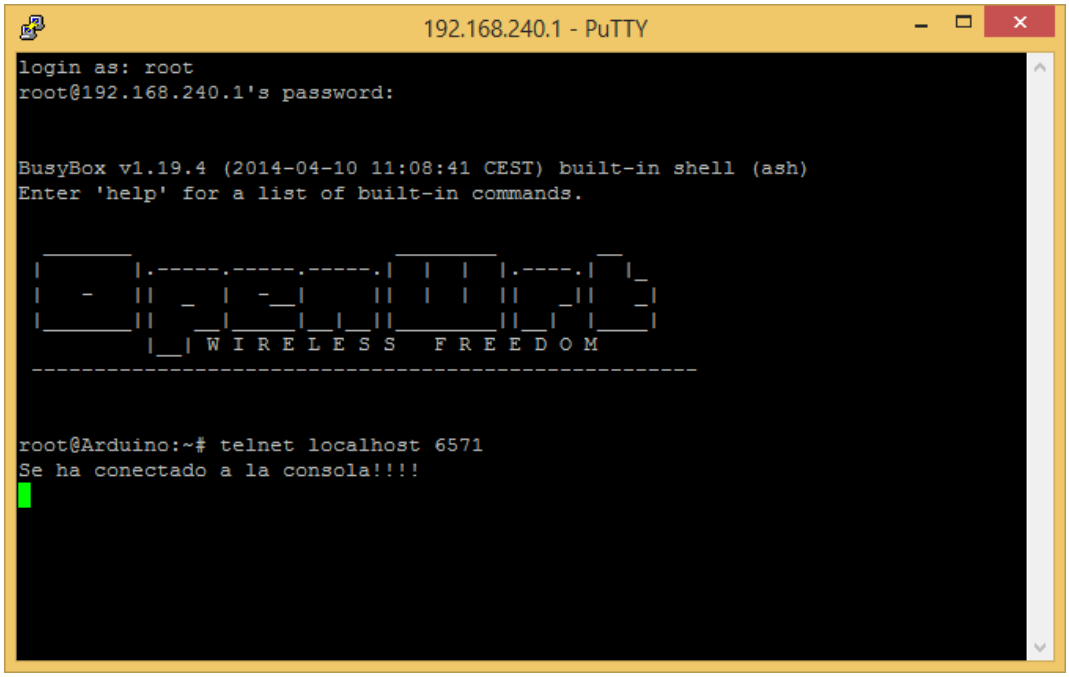

Figura 14 Consola de PuTTy conectada

Ahora que la consola está conectada, es fácil trabajar con el Arduino Yún. Si se envía la letra H, el LED encenderá y el Arduino Yun envía el mensaje "*Se ha encendido el LED"*. Con una L, el LED se apagará y el Arduino Yun envía el mensaje*"Se ha apagado el LED"*.

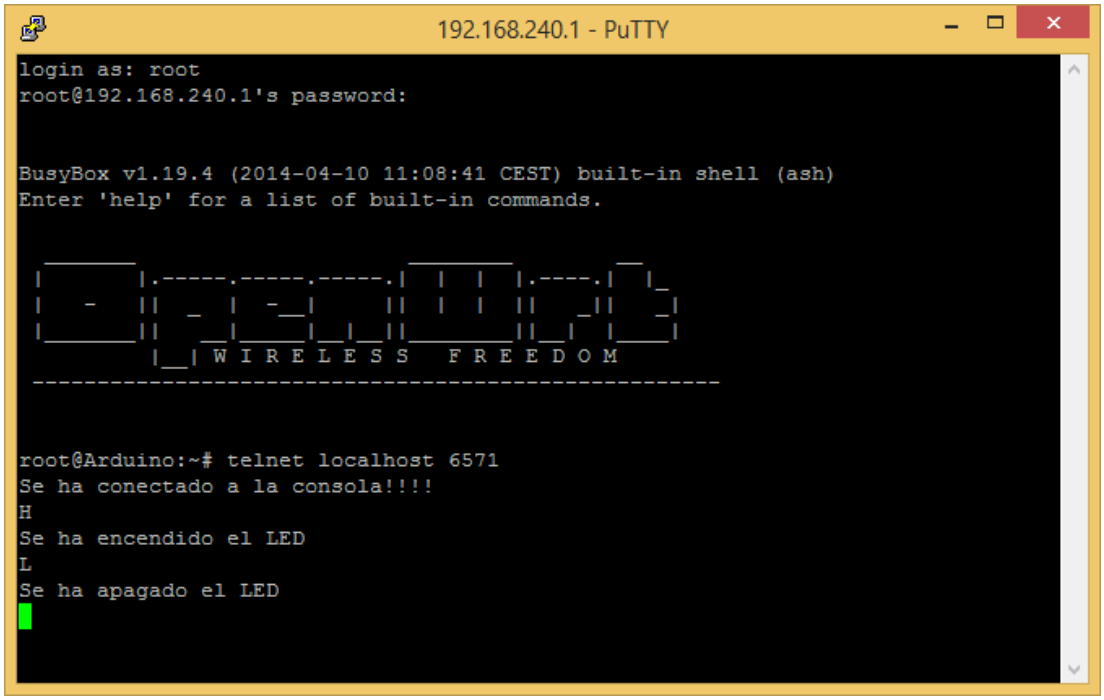

Figura 15 Prueba de resultados de la comunicación SSH

3.5 Comunicación inalámbrica mediante la librería Bridge

Se procederá a explicar cómo el Arduino Yun mediante la librería Bridge (puente) accede a los pines digitales y analógicas de la placa a través de llamadas REST (**RE**presentational **S**tate **T**ransfer) que es un tipo de arquitectura de desarrollo web que se apoya totalmente en el estándar HTTP. REST permite crear servicios y aplicaciones que pueden ser usadas por cualquier dispositivo o cliente que entienda HTTP.

Por lo tanto REST es el tipo de arquitectura más natural y estándar para crear APIs para servicios orientados a Internet.

El código que se emplea para demostrar la comunicación se estructura de la siguiente manera:

Primero se deben incluir las siguientes librerías:

```
#include <Bridge.h>
#include <BridgeClient.h>
#include <BridgeServer.h>
```

La siguiente línea es una instancia de un servidor que permite al Arduino Yun escuchar los clientes conectados.

```
BridgeServer servidor;
```

En la primera parte de la estructura setup( ), se configura el pin 13 como salida, se coloca a cero. La sentencia Bridge.begin() inicia el puente que toma aproximadamente 2 segundos, en ese instante se enciende el led conectado al pin 13.

```
void setup() {
pinMode(13, OUTPUT);
digitalWrite(13, LOW);
Bridge.begin();
digitalWrite(13, HIGH);
```

La segunda parte de la estructura setup( ), escucha las conexiones entrantes sólo procedentes de localhost. Las conexiones realizadas en Linux serán pasadas al procesador 32U4 para analizar y controlar los pines. Esto sucede en el puerto 5555. Finalmente inicia el servidor con server.begin ( ).

```
servidor.listenOnLocalhost();
servidor.begin();

}
```

En la estructura loop ( ), que va a crear una instancia llamada, Cliente para la gestión de la conexión. Si el cliente se conecta, procesa las solicitudes en forma personalizada (descrito más adelante) y luego cierra la conexión cuando haya terminado. El retardo de 50 ms, evita al procesador cargarse de trabajo.

```
void loop() {
BridgeClient cliente = servidor.accept();

if (cliente) {
    proceso(cliente);
    cliente.stop();
  }
delay(50);
}
```

Se crea una función llamada **proceso** que tiene como argumento la instancia Cliente creada anteriormente. En la variable **comando** se almacena la información entrante. Analiza los comandos REST por su funcionalidad (digital, analógico y modo) y pasa la información a la función creada para cada caso.

```
void proceso(BridgeClient cliente) {
  String comando = cliente.readStringUntil('/');

  // Si es comando digital
  if (comando == "digital") {
    digital(cliente);
  }

  // Si es comando analogico
  if (comando == "analog") {
    analogico(cliente);
  }

  // Si es modo comando
  if (comando == "mode") {
    modo(cliente);
  }
}
```

La función llamada **digital** analiza la solicitud del cliente para la forma de trabajo del pin digital, si el carácter que viene después es "/", significa que la URL va a tener un 1 o 0 a continuación, esta valor sirve para modificar el estado del pin; es decir, en alto y bajo respectivamente. Si no hay "/", lee el estado de un pin especificado.

```
void digital(BridgeClient cliente) {
  int pin, valor;

  // Lee numero del pin
  pin = cliente.parseInt();

  // Si el siguiente carácter es un "/" significa que tenemos una URL
  // con un valor como: "/digital/13/1"
  if (cliente.read() == '/') {
    valor = cliente.parseInt();
    digitalWrite(pin, valor);
  } else {
    valor = digitalRead(pin);
  }

  // Envia comentario al cliente
  cliente.print(F("Pin D"));
  cliente.print(pin);
  cliente.print(F(" en: "));
  cliente.println(valor);

  // Actualiza los datos almacenados
  String key = "D";
  key += pin;
  Bridge.put(key, String(valor));
}
```

De la misma forma que se creó una función para entradas y salidas digitales se crea otra para entradas y salidas analógicas. Cuando se trata de salidas analógicas se recibe el carácter "/" seguido por un número entre 0 y 255, se etiqueta al pin con la letra D, por ejemplo D3. Las salidas analógicas son del tipo PWM.

Cuando se trata de entradas analógicas se etiqueta el pin con la letra A, por ejemplo A0, la entradas analógicas leen un valor entre 0 y 1023 cuando en los terminales de la tarjeta existe un voltaje cuyo rango es de 0 a 5 VDC.

```
void analogico(BridgeClient cliente) {
  int pin, valor;

  // Lee el nuemro de pin
  pin = cliente.parseInt();

  // Si el siguiente carácter es un "/" significa que tenemos una URL
  // con un valor como: "/analog/5/120"
  if (cliente.read() == '/') {
    // Lee el valor y ejecuta el comando
    valor = cliente.parseInt();
    analogWrite(pin, valor);

    // Envia comentario al cliente
    cliente.print(F("Pin D"));
    cliente.print(pin);
    cliente.print(F(" fijado a un valor analogico de: "));
    cliente.println(valor);

    // Actualiza los datos almacenados
    String key = "D";
    key += pin;
    Bridge.put(key, String(valor));
  } else {
    // Lee el pin analogico
    valor = analogRead(pin);


    // Envia comentario al cliente
    cliente.print(F("Pin A"));
    cliente.print(pin);
    cliente.print(F(" lectura analogica: "));
    cliente.println(valor);

    // Actualiza los datos almacenados
    String key = "A";
    key += pin;
    Bridge.put(key, String(valor));
  }
}
```

La función **modo**, sirve para configurar si un pin digital es entrada o salida.

El código completo de la comunicación mediante la librería Brigde es:

#include <Bridge.h>

#include <BridgeClient.h>

#include <BridgeServer.h>

```
BridgeServer servidor;

void setup() {
pinMode(13, OUTPUT);
digitalWrite(13, LOW);
Bridge.begin();
digitalWrite(13, HIGH);

servidor.listenOnLocalhost();
servidor.begin();

}

void loop() {
BridgeClient cliente = servidor.accept();

if (cliente) {
    proceso(cliente);
    cliente.stop();
  }
delay(50);
}

void proceso(BridgeClient cliente) {
```

```
  String comando = cliente.readStringUntil('/');

  // Si es comando digital
  if (comando == "digital") {
    digital(cliente);
  }

  // Si es comando analogico
  if (comando == "analog") {
    analogico(cliente);
  }

  // Si es modo comando
  if (comando == "mode") {
    modo(cliente);
  }
}


void digital(BridgeClient cliente) {
  int pin, valor;

  // Lee numero del pin
  pin = cliente.parseInt();
```

```
  // Si el siguiente carácter es un "/" significa que tenemos una URL
  // con un valor como: "/digital/13/1"
  if (cliente.read() == '/') {
    valor = cliente.parseInt();
    digitalWrite(pin, valor);
  } else {
    valor = digitalRead(pin);
  }

  // Envia comentario al cliente
  cliente.print(F("Pin D"));
  cliente.print(pin);
  cliente.print(F(" en: "));
  cliente.println(valor);

  // Actualiza los datos almacenados
  String key = "D";
  key += pin;
  Bridge.put(key, String(valor));
}

void analogico(BridgeClient cliente) {
  int pin, valor;

  // Lee el nuemro de pin
```

```
pin = cliente.parseInt();

// Si el siguiente carácter es un "/" significa que tenemos una URL
// con un valor como: "/analog/5/120"
if (cliente.read() == '/') {
  // Lee el valor y ejecuta el comando
  valor = cliente.parseInt();
  analogWrite(pin, valor);

  // Envia comentario al cliente
  cliente.print(F("Pin D"));
  cliente.print(pin);
  cliente.print(F(" fijado a un valor analogico de: "));
  cliente.println(valor);

  // Actualiza los datos almacenados
  String key = "D";
  key += pin;
  Bridge.put(key, String(valor));
} else {
  // Lee el pin analogico
  valor = analogRead(pin);

  // Envia comentario al cliente
  cliente.print(F("Pin A"));
```

```
    cliente.print(pin);
    cliente.print(F(" lectura analogica: "));
    cliente.println(valor);

    // Actualiza los datos almacenados
    String key = "A";
    key += pin;
    Bridge.put(key, String(valor));
  }
}

void modo(BridgeClient cliente) {
  int pin;

  // Lee numero de pin
  pin = cliente.parseInt();

  // Si el siguiente cararcter no es un '/' se tiene una mal formada URL
  if (cliente.read() != '/') {
    cliente.println(F("error"));
    return;
  }

  String modo = cliente.readStringUntil('\r');
```

```
  if (modo == "input") {
    pinMode(pin, INPUT);
    // Envia comentario a cliente
    cliente.print(F("Pin D"));
    cliente.print(pin);
    cliente.print(F(" configurado como entrada!"));
    return;
  }

  if (modo == "output") {
    pinMode(pin, OUTPUT);
    // Envia comentario a cliente
    cliente.print(F("Pin D"));
    cliente.print(pin);
    cliente.print(F(" configurado como salida!"));
    return;
  }

  cliente.print(F("error: modo invalido "));
  cliente.print(modo);
}
```

Para comprobar el funcionamiento de la comunicación se debe utilizar los siguientes comandos, los mismos que deben escribirse en la barra de direcciones del navegador.

"arduino.local/arduino/digital/13" -> digitalRead(13)

"arduino.local /arduino/digital/13/1" -> digitalWrite(13, HIGH)

"arduino.local /arduino/analog/2/123" -> analogWrite(2, 123)

"arduino.local /arduino/analog/2" -> analogRead(2)

"arduino.local /arduino/mode/13/input" -> pinMode(13, INPUT)

"arduino.local /arduino/mode/13/output" -> pinMode(13, OUTPUT)

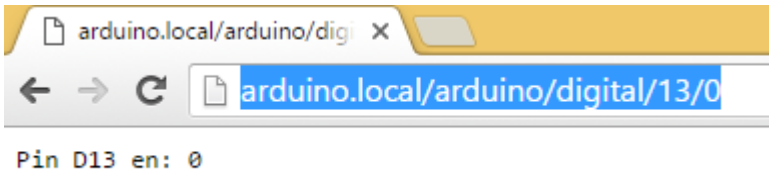

Figura 16 Comando para escribir valor en el pin 13

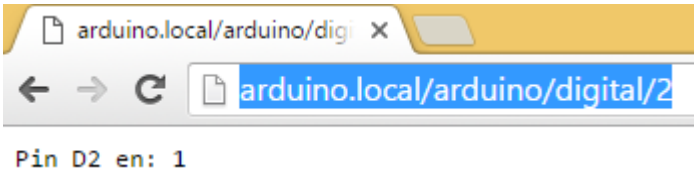

Figura 17 Comando para leer el estado del pin 2

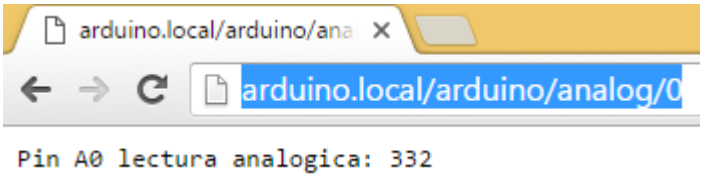

Figura 18 Comando para leer un valor analógico

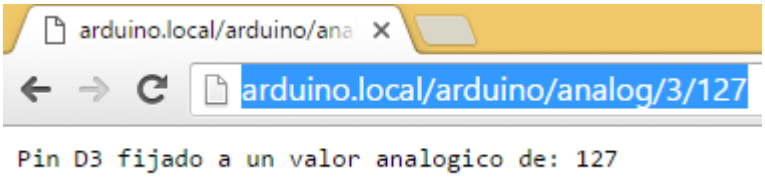

Figura 19 Comando para generar una señal PWM

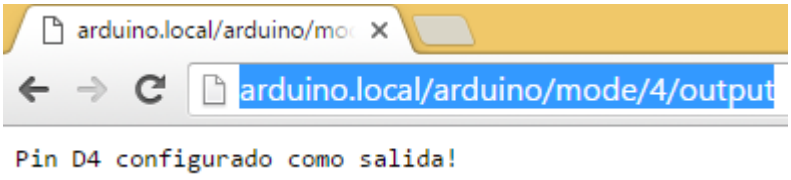

Figura 20 Comando para configurar dirección del pin

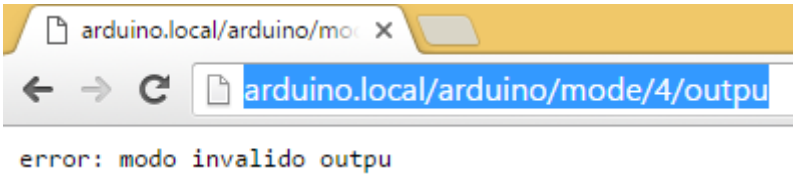

Figura 21 Error generado por comando mal escrito

3.6 Monitoreo de señales analógicas y digitales

La aplicación del monitoreo inalámbrico mediante Arduino Yun consiste en crear una interface (página web) mediante códigos HTML para monitorear y controlar señales analógicas y digitales. Se emplea la comunicación inalámbrica mediante la librería Bridge.

3.6.1 Código en Arduino

Se añade las librerías para la comunicación

```
#include <Bridge.h>
#include <BridgeClient.h>
#include <BridgeServer.h>
```

Se asigna los pines de entrada y salida digitales, el pin 2 se empleara como entrada, los pines 12 y 13 como salidas.

El pin 3 será la salida analógica PWM.

```
int dac = 0;
int DigitalPin[] = {2, 12, 13};
int DacPin = 3;
```

Se crea la instancia de servidor

```
BridgeServer servidor;
```

En la estructura setup ( ), se configura los pines digitales, se inicia el servidor y el puente.

```
void setup() {
pinMode(2,INPUT);
pinMode(12,OUTPUT);
pinMode(13,OUTPUT);
digitalWrite(13, LOW);
Bridge.begin();
digitalWrite(13, HIGH);
servidor.listenOnLocalhost();
servidor.begin();
}
```

En la estructura loop ( ), espera la solicitud de los clientes e inicia el proceso correspondiente.

```
void loop() {
BridgeClient cliente = servidor.accept();
if (cliente) {
proceso(cliente);
cliente.stop();
}
delay(50);
}
```

La función *proceso* lee el comando que envía el cliente y lo ejecuta, sea esta digital, analógico y el estado.

```
void proceso(BridgeClient cliente) {
String comando = cliente.readStringUntil('/');
if (comando == "digital") {
digitalComando(cliente);
}
if (comando == "dac") {
dacComando(cliente);
}
if (comando == "status") {
statusComando(cliente);
}
```

La función *digitalComando* cambia el estado de las salidas digitales (pin 12 y 13) y lee el estado de la entrada digital (pin 2).

```
void digitalComando(BridgeClient cliente) {
int pin, valor;
pin = cliente.parseInt();
if (cliente.read() == '/') {
valor = cliente.parseInt();
digitalWrite(pin, valor);
}
else {
valor = digitalRead(pin);
}
cliente.print(F("digital,"));
cliente.print(pin);
cliente.print(F(","));
cliente.println(valor);
}
```

La función *dacComando* genera una señal PWM por el pin digital 3.

```
void dacComando(BridgeClient cliente) {
int pin, valor;
pin = cliente.parseInt();
if (cliente.read() == '/') {
valor = cliente.parseInt();
dac = valor;
analogWrite(pin, valor);
}
else {
valor = dac;
}
cliente.print(F("dac,"));
cliente.print(pin);
cliente.print(F(","));
cliente.println(valor);
}
```

La función *statusComando* envía al cliente el estado de las señales y la temperatura que mide el sensor LM35 conectado al canal analógico 0.

```
void statusComando(BridgeClient cliente) {
int pin, valor;
cliente.print(F("Estado"));
for (int thisPin = 0; thisPin < 3; thisPin++) {
pin = DigitalPin[thisPin];
valor = digitalRead(pin);
cliente.print(F("#"));
cliente.print(pin);
cliente.print(F("="));
cliente.print(valor);
}
{
pin = DacPin;
valor = dac;
cliente.print(F("#"));
cliente.print(pin);
cliente.print(F("="));
cliente.print(valor);
}

{
valor = analogRead(0);
      // Convierte la lectura a voltaje
      float voltaje = (float)valor/1023*5;
      // Convierte el voltaje a temperatura
      byte temperatura = voltaje*100;
cliente.print(F("#A0"));
cliente.print(F("="));
cliente.print(temperatura);
}
cliente.println("");
}
```

El código completo es el siguiente:

```
#include <Bridge.h>
#include <BridgeClient.h>
#include <BridgeServer.h>

int dac = 0;
int DigitalPin[] = {2, 12, 13};
int DacPin = 3;
```

```
BridgeServer servidor;

void setup() {
pinMode(2,INPUT);
pinMode(12,OUTPUT);
pinMode(13,OUTPUT);
digitalWrite(13, LOW);
Bridge.begin();
digitalWrite(13, HIGH);
servidor.listenOnLocalhost();
servidor.begin();
}

void loop() {
BridgeClient cliente = servidor.accept();
if (cliente) {
proceso(cliente);
cliente.stop();
}
delay(50);
}

void proceso(BridgeClient cliente) {
String comando = cliente.readStringUntil('/');
if (comando == "digital") {
digitalComando(cliente);
}
if (comando == "dac") {
dacComando(cliente);
```

```
}
if (comando == "status") {
statusComando(cliente);
}
}

void digitalComando(BridgeClient cliente) {
int pin, valor;
pin = cliente.parseInt();
if (cliente.read() == '/') {
valor = cliente.parseInt();
digitalWrite(pin, valor);
}
else {
valor = digitalRead(pin);
}
cliente.print(F("digital,"));
cliente.print(pin);
cliente.print(F(","));
cliente.println(valor);
}

void dacComando(BridgeClient cliente) {
int pin, valor;
pin = cliente.parseInt();
if (cliente.read() == '/') {
valor = cliente.parseInt();
dac = valor;
analogWrite(pin, valor);
```

```
}
else {
valor = dac;
}
cliente.print(F("dac,"));
cliente.print(pin);
cliente.print(F(","));
cliente.println(valor);
}

void statusComando(BridgeClient cliente) {
int pin, valor;
cliente.print(F("Estado"));
for (int thisPin = 0; thisPin < 3; thisPin++) {
pin = DigitalPin[thisPin];
valor = digitalRead(pin);
cliente.print(F("#"));
cliente.print(pin);
cliente.print(F("="));
cliente.print(valor);
}
{
pin = DacPin;
valor = dac;
cliente.print(F("#"));
cliente.print(pin);
cliente.print(F("="));
cliente.print(valor);
}
```

```
{
valor = analogRead(0);
    // Convierte la lectura a voltaje
    float voltaje = (float)valor/1023*5;
    // Convierte el voltaje a temperatura
    byte temperatura = voltaje*100;
cliente.print(F("#A0"));
cliente.print(F("="));
cliente.print(temperatura);
}
cliente.println("");
}
```

3.6.2 Código HTML

La programación se ha realizado utilizando el Bloc de notas de Windows. La primera parte del código HTML es un script donde se envía las peticiones al servidor, en este caso al Arduino Yun.

```
<script type="text/javascript">


window.onload=Pinstatus;


function Pinstatus(){
morestatus();
}


function morestatus(){
setTimeout(morestatus, 1000);
document.getElementById("description").innerHTML = "Estado de
```

```
Procesamiento";
server = "/arduino/status/99";
request = new XMLHttpRequest();
request.onreadystatechange = updateasyncstatus;
request.open("GET", server, true);
request.send(null);
}

function updateasyncstatus(){
if ((request.readyState == 4) && (request.status == 200))
{

result = request.responseText;
document.getElementById("description").innerHTML = result;
fullset = result.split("#");
document.getElementById("description").innerHTML = fullset;

for(i = 1; i < fullset.length; i++){
PinPair = fullset[i];
singleset = PinPair.split("=");
PN = singleset[0];
Pinstatus = singleset[1];

if (PN > 11)
{
ActNum = "action" + PN;
ImgNum = "image" + PN;
if (Pinstatus == 0)
{
PinAct = "1";
image = "off.jpg";
```

```
}
else
{
PinAct = "0";
image = "on.jpg";
}

document.getElementById(ActNum).value = PinAct;
document.getElementById(ImgNum).src = image;
}

if (PN == 2)
{
ImgNum = "image" + PN;
if (Pinstatus == 1)

{
image = "led_on.jpg";
}

else
{
image = "led_off.jpg";
}

document.getElementById(ImgNum).src = image;
}

if (PN == 3 )
{
```

```
PinVal = parseInt(singleset[1]);
DacNum = "dac" + PN;
ValNum = "valueDac" + PN;

document.getElementById(DacNum).value = PinVal;
document.getElementById(ValNum).innerHTML = PinVal;
}

if (PN.substr(0,1) == "A")
{
PinVal = parseFloat(singleset[1]);
AnalogNum = "analog" + PN.substr(1,2);
document.getElementById(AnalogNum).value = PinVal;
}
}
}
}

function sendbutton(Pin,action){
document.getElementById("description").innerHTML = "Hizo Click en un Botón";
server = "/arduino/digital/" + Pin + "/" + action;
request = new XMLHttpRequest();
request.onreadystatechange = updateasyncbutton;
request.open("GET", server, true);

request.send(null);
}
```

```
function updateasyncbutton(){
if ((request.readyState == 4) && (request.status == 200))
{
result = request.responseText;
document.getElementById("description").innerHTML = result;
singleset = result.split(",");
PinType = singleset[0];
PinNum = singleset[1];
Pinstatus = singleset[2];
ActNum = "action" + PinNum;
ImgNum = "image" + PinNum;

if (Pinstatus == 0)
{
PinAct = "1";
image = "off.jpg";
}

else
{
PinAct = "0";
image = "on.jpg";
}

document.getElementById(ActNum).value = PinAct;
document.getElementById(ImgNum).src = image;
document.getElementById("description").innerHTML = result;
}

}
function sendDac(Pin,value){
```

```
ValNum = "valueDac" + Pin;
document.getElementById(ValNum).innerHTML=value;
document.getElementById("description").innerHTML = "Cambio de Slider";
server = "/arduino/dac/" + Pin + "/" + value;
request = new XMLHttpRequest();
request.onreadystatechange = updateasyncDac;
request.open("GET", server, true);
request.send(null);
}

function updateasyncDac(){
if ((request.readyState == 4) && (request.status == 200))
{
result = request.responseText;
singleset = result.split(",");
PinType = singleset[0];
PinNum = singleset[1];
PinVal = parseInt(singleset[2]);
DacNum = "dac" + PinNum;
ValNum = "valueDac" + PinNum;
document.getElementById(DacNum).value = PinVal;
document.getElementById(ValNum).innerHTML = PinVal;
document.getElementById("description").innerHTML = result;
}
}
</script>
```

La segunda parte del código HTML crea la interface para que el usuario interactúe con el Arduino Yun.

```
<body>
```

```
<DIV ALIGN=center><font face="Arial "size="4" color="red"> <B>Unidad de Gestión de Tecnologías</B></font></DIV>
<DIV ALIGN=center><font face="Arial "size="4" color="blue">Carrera de Electrónica Mención Instrumentación y Aviónica</font></DIV>

<DIV ALIGN=center><font face="Arial "size="4" color="green">Control y Monitoreo con Arduino Yun</font></DIV><br>

</body>

<font face="Arial">

<table style="margin: 0 auto" name="Table" border="1" cellpadding="6">

<tr> <th align="center" colspan="6" >Entrada Analogica</th></tr>


<tr>

<td align="center">

Temperatura en °C

<br>

<input type="text" style="text-align: center;" name="analogA0" id="analog0" value="0" size="6" readonly/>

</td>

</tr>

<tr> <th align="center" colspan="6" >Entrada Digital</th></tr>

<tr>

<td align="center">

Pin 2

<br>

<img src="led_off.jpg" width="50" id="image2" />
```

```
</td>

</tr>

<tr> <th align="center" colspan="6" >Salida Analogica</th></tr>

<tr>

<td align="center">

Pin 3

<br>

<input type="hidden" name="pin" value="3" id="pin3" />

<input type="range" style="width: 50px; height: 30px;" id="dac3" min="0" max="255" value="0" step="1" onchange="sendDac(document.getElementById('pin3').value,

this.value);"/>

<br>

<span id="valueDac3">0</span>

</td>

</tr>

<tr> <th align="center" colspan="6" >Salidas Digitales</th></tr>

<tr>

<td align="center">

Pin 12

<br>

<input type="hidden" name="pin" value="12" id="pin12" />

<input type="hidden" name="action" value="0" id="action12" />
```


</td>

</tr>

<tr>

<td align="center">

Pin 13

<input type="hidden" name="pin" value="13" id="pin13" />

<input type="hidden" name="action" value="0" id="action13" />

</td>

</tr>

</table>

 -

3.6.3 Pruebas de funcionamiento

Para comprobar el funcionamiento del monitoreo inalámbrico, siga los siguientes pasos.

Paso 1.

Energice el Arduino Yun y habilite la WiFi

Paso 2.

Una vez que aparezca la red inalámbrica del Arduino Yun en su computador conéctese a la red.

Paso 3.

Abra el navegador web Chrome u otro, en la barra de dirección escriba http://arduino.local/sd/, pulse enter.

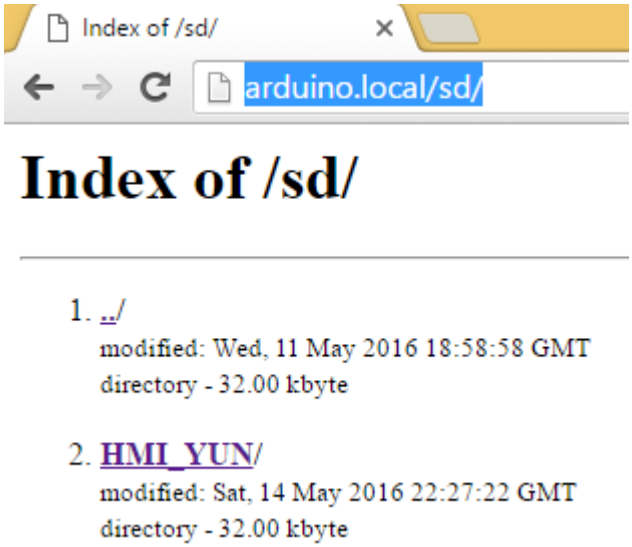

Figura 22 Acceso a la SD card del Arduino Yun

Paso 4.

De clic en HMI_YUN. Aparece una ventana donde pide usuario y contraseña. En usuario escriba ***root*** y en contraseña ***arduino***.

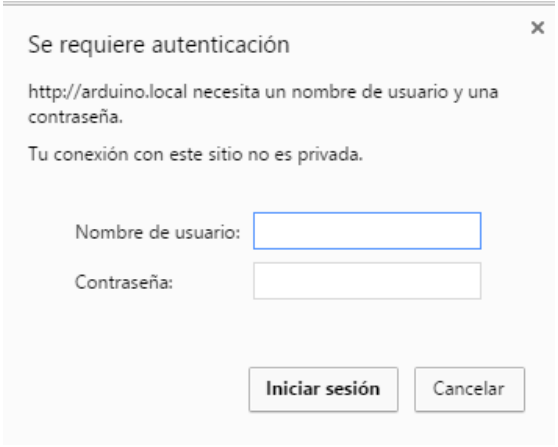

Figura 23 Ventana de autenticación

Pulse el botón **Iniciar sesión** y aparecerá la interface creada para el monitoreo y control de señales analógicas y digitales.

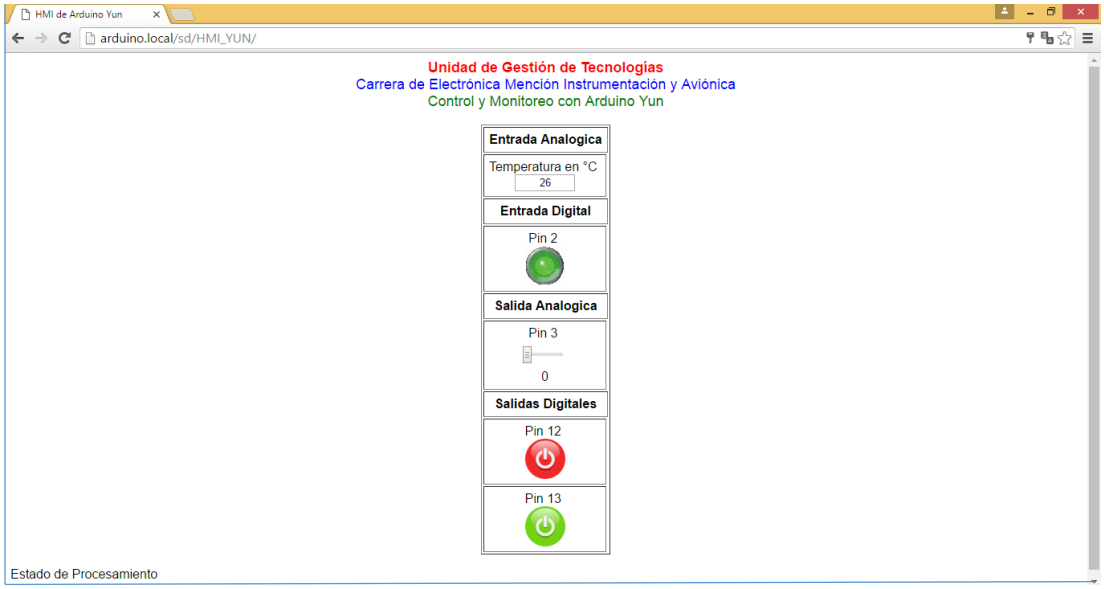

Figura 24 Interface para el monitoreo de señales

CAPÍTULO IV

CONCLUSIONES

4.1 Conclusiones

- El Arduino Yun es una tarjeta electrónica que posee un microcontrolador Atmel y un sistema Linux basado en el chipset Atheros AR9331. El sistema Linux maneja la red inalámbrica y se comunica con el microcontrolador mediante una librería llamada puente (Bridge) para que modifique el estado de las entradas y salidas.
- Para realizar la comunicación inalámbrica mediante la consola; es decir, enviando peticiones al servidor mediante mensajes, se empleó el software PuTTy, el mismo que utiliza el protocolo SSH.
- Cuando no se puede acceder al Arduino Yun mediante el navegador web al escribir http://arduino.local, se debe escribir la dirección IP 192.168.240.1 que es la dirección de fábrica que tiene el dispositivo.

4.2 Recomendaciones

- Pulsar varias veces el botón WLAN RST para habilitar la red inalámbrica del Arduino Yun.
- No alimentar con las de 5 DVC al Arduino Yun porque no posee regulador interno.
- Al adquirir señales analógicas mediante el Arduino Yun se debe considerar el rango de voltaje que es de 0 a 5 VDC. Si se necesita adquirir señales de mayor rango, es necesario acondicionarlas antes de conectar al Arduino Yun.
- Continuar con otras investigaciones referentes a este tema para que a futuro se puedan desarrollar otras aplicaciones.

GLOSARIO DE TÉRMINOS

LED	Diodo emisor de luz
HTML	Lenguaje de marcas de hipertexto
HTTP	Protocolo de transferencia de hipertexto
IEEE	Instituto de ingenieros eléctricos y electrónicos
IP	Protocolo de internet
REST	Transferencia de estado representacional
SSH	Intérprete de órdenes seguro
WiFi	Tecnología de comunicación inalámbrica

REFERENCIAS BIBLIOGRÁFICAS

Arduino. (2015). *Arduino Yun*. Obtenido de http://www.arduino.cc/en/Main/ArduinoBoardYun

Arduino. (2015). *Productos de Arduino*. Obtenido de http://www.arduino.cc/en/Main/Products

Enterprise, R. H. (2009). *Protocolo SSH*. Obtenido de http://web.mit.edu/rhel-doc/4/RH-DOCS/rhel-rg-es-4/ch-ssh.html

Evans, B., & Ruiz Gutierrez, J. (2011). *Arduino Programming Notebook.* San Francisco California.

Frades Estévez, J. A. (Septiembre de 2015). Diseño de un programa en Android para el control de Arduino. Valladolid, España.

Kioskea. (Junio de 2014). Introducción a Wifi. España.

Lamarca Lapuente, M. J. (13 de Diciembre de 2013). *HTML*. Obtenido de http://www.hipertexto.info/documentos/html.htm

Marbit. (2010). *Direccionamiento IP*. Obtenido de http://www.marbit.es/index_ip.html

OpenWrt. (Agosto de 2015). *OpenWrt Wireless Freedom*. Obtenido de https://openwrt.org/

ANEXOS

NEXO “A”

Programación HTML

```
<html>

<head>

<title>HMI de Arduino Yun</title>

<script type="text/javascript">

window.onload=Pinstatus;

function Pinstatus(){

morestatus();

}

function morestatus(){

setTimeout(morestatus, 1000);

document.getElementById("description").innerHTML = "Estado de
Procesamiento";

server = "/arduino/status/99";

request = new XMLHttpRequest();

request.onreadystatechange = updateasyncstatus;

request.open("GET", server, true);

request.send(null);
```

```
}

function updateasyncstatus(){

if ((request.readyState == 4) && (request.status == 200))

{

result = request.responseText;

document.getElementById("description").innerHTML = result;

fullset = result.split("#");

document.getElementById("description").innerHTML = fullset;

for(i = 1; i < fullset.length; i++){

PinPair = fullset[i];

singleset = PinPair.split("=");

PN = singleset[0];

Pinstatus = singleset[1];

if (PN > 11)

{

ActNum = "action" + PN;

ImgNum = "image" + PN;

if (Pinstatus == 0)
```

```
{
PinAct = "1";
image = "off.jpg";
}
else
{
PinAct = "0";
image = "on.jpg";
}
document.getElementById(ActNum).value = PinAct;
document.getElementById(ImgNum).src = image;
}
if (PN == 2)
{
ImgNum = "image" + PN;
if (Pinstatus == 1)
{
image = "led_on.jpg";
```

```
}

else

{

image = "led_off.jpg";

}

document.getElementById(ImgNum).src = image;

}

if (PN == 3 )

{

PinVal = parseInt(singleset[1]);

DacNum = "dac" + PN;

ValNum = "valueDac" + PN;

document.getElementById(DacNum).value = PinVal;

document.getElementById(ValNum).innerHTML = PinVal;

}

if (PN.substr(0,1) == "A")

{

PinVal = parseFloat(singleset[1]);
```

```
AnalogNum = "analog" + PN.substr(1,2);

document.getElementById(AnalogNum).value = PinVal;

}

}

}

}

function sendbutton(Pin,action){

document.getElementById("description").innerHTML = "Hizo Click en un Botón";

server = "/arduino/digital/" + Pin + "/" + action;

request = new XMLHttpRequest();

request.onreadystatechange = updateasyncbutton;

request.open("GET", server, true);

request.send(null);

}

function updateasyncbutton(){

if ((request.readyState == 4) && (request.status == 200))

{
```

```
result = request.responseText;

document.getElementById("description").innerHTML = result;

singleset = result.split(",");

PinType = singleset[0];

PinNum = singleset[1];

Pinstatus = singleset[2];

ActNum = "action" + PinNum;

ImgNum = "image" + PinNum;

if (Pinstatus == 0)

{

PinAct = "1";

image = "off.jpg";

}

else

{

PinAct = "0";

image = "on.jpg";

}
```

```
document.getElementById(ActNum).value = PinAct;

document.getElementById(ImgNum).src = image;

document.getElementById("description").innerHTML = result;

}

}

function sendDac(Pin,value){

ValNum = "valueDac" + Pin;

document.getElementById(ValNum).innerHTML=value;

document.getElementById("description").innerHTML = "Cambio de Slider";

server = "/arduino/dac/" + Pin + "/" + value;

request = new XMLHttpRequest();

request.onreadystatechange = updateasyncDac;

request.open("GET", server, true);

request.send(null);

}

function updateasyncDac(){

if ((request.readyState == 4) && (request.status == 200))

{
```

```
result = request.responseText;

singleset = result.split(",");

PinType = singleset[0];

PinNum = singleset[1];

PinVal = parseInt(singleset[2]);

DacNum = "dac" + PinNum;

ValNum = "valueDac" + PinNum;

document.getElementById(DacNum).value = PinVal;

document.getElementById(ValNum).innerHTML = PinVal;

document.getElementById("description").innerHTML = result;

}

}

</script>

</head>

<body>

<DIV ALIGN=center><font face="Arial "size="4" color="red"> <B>Unidad de Gestión de Tecnologías</B></font></DIV>

<DIV ALIGN=center><font face="Arial "size="4" color="blue">Carrera de Electrónica Mención Instrumentación y Aviónica</font></DIV>

<DIV ALIGN=center><font face="Arial "size="4" color="green">Control y Monitoreo con Arduino Yun</font></DIV><br>
```

```
</body>


<font face="Arial">

<table style="margin: 0 auto" name="Table" border="1" cellpadding="6">

<tr> <th align="center" colspan="6" >Entrada Analogica</th></tr>

<tr>

<td align="center">

Temperatura en °C

<br>

<input type="text" style="text-align: center;" name="analogA0" id="analog0"
value="0" size="6" readonly/>

</td>

</tr>

<tr> <th align="center" colspan="6" >Entrada Digital</th></tr>

<tr>

<td align="center">

Pin 2

<br>

<img src="led_off.jpg" width="50" id="image2" />
```

```
</td>

</tr>

<tr> <th align="center" colspan="6" >Salida Analogica</th></tr>

<tr>

<td align="center">

Pin 3

<br>

<input type="hidden" name="pin" value="3" id="pin3" />

<input type="range" style="width: 50px; height: 30px;" id="dac3" min="0"
max="255" value="0" step="1"
onchange="sendDac(document.getElementById('pin3').value,

this.value);"/>

<br>

<span id="valueDac3">0</span>

</td>

</tr>

<tr> <th align="center" colspan="6" >Salidas Digitales</th></tr>

<tr>

<td align="center">
```

Pin 12

<input type="hidden" name="pin" value="12" id="pin12" />

<input type="hidden" name="action" value="0" id="action12" />

<img src="off.jpg" width="50" id="image12"
onclick="sendbutton(document.getElementById('pin12').value,document.getElementById('action12').value);"/>

</td>

</tr>

<tr>

<td align="center">

Pin 13

<input type="hidden" name="pin" value="13" id="pin13" />

<input type="hidden" name="action" value="0" id="action13" />

<img src="off.jpg" width="50" id="image13"
onclick="sendbutton(document.getElementById('pin13').value,document.getElementById('action13').value);"/>

</td>

</tr>

```
</table>

<p id="description"> - </p>

</font>

</html>
```

DATOS PERSONALES

Nombre:	VALLEJO CASTILLO WILLIAM MANUEL
Nacionalidad:	Ecuatoriana
Fecha De Nacimiento:	13 DE JUNIO DE 1970
Cedula de Ciudadanía:	1711059202
Teléfonos:	0984997625/032266589
Correo Electrónico:	willyga1331@yahoo.com
Dirección Domicilio:	LATACUNGA BARRIO ILLUCHI (PANAMERICANA SUR)

ESTUDIOS REALIZADOS:

Primaria:

- Escuela FAE N°1 Quito

Secundaria:

- Colegio Técnico FAE N°1 Quito

Superior:

- Unidad de Gestión de Tecnologías “ESPE”

TITULOS OBTENIDOS:

- Bachiller técnico “Electromecánica”
- Tecnólogo en Electrónica Mención Instrumentación & Aviónica
- Suficiencia en el idioma ingles

EXPERIENCIAS LABORALES:

- Avionics Technical Centro de Mantenimiento Aeronáutico DIAF-CEMA

ACEPTACIÓN DEL USUARIO

Latacunga, 07 de junio del 2016

Yo, ING PABLO PILATÁSIG en calidad de encargado del Laboratorio de Instrumentación Virtual de la Unidad de Gestión de Tecnologías, me permito informar lo siguiente:

El proyecto técnico elaborado por el Sr. **VALLEJO CASTILLO WILLIAM MANUEL,** con el tema: **"IMPLEMENTACIÓN DE TRES ARDUINO YUN, QUE PERMITE LA ADQUISICIÓN DE SEÑALES ANALÓGICAS Y DIGITALES EN FORMA INALÁMBRICA PARA PRÁCTICAS DE MICROCONTROLADORES",** ha sido efectuado de forma satisfactoria en las dependencias de mi cargo y que la misma cuenta con todas las garantías de funcionamiento, por lo cual extiendo este aval que respalda el trabajo realizado por el mencionado estudiante.

Por tanto me hago cargo de todas las instalaciones realizadas por el Sr. estudiante.

Atentamente,

_____

ING. PABLO PILATÁSIG

ENCARGADO DEL LABORATORIO DE INSTRUMENTACIÓN VIRTUAL

Latacunga, 07 de junio del 2016

HOJA DE LEGALIZACIÓN DE FIRMAS

DEL CONTENIDO DE LA PRESENTE INVESTIGACIÓN SE RESPONSABILIZA EL AUTOR

_____

VALLEJO CASTILLO WILLIAM MANUEL

ID L00264971

DIRECTOR DE LA CARRERA DE ELECTRÓNICA MENCIÓN INSTRUMENTACIÓN & AVIÓNICA

_____

Ing. Pablo Pilatásig Director Carrera de Electrónica Mención Instrumentación & Aviónica