ESCUELA SUPERIOR POLITECNICA DEL LITORAL

INFORME FINAL DE PRACTICAS COMUNITARIAS

PROYECTO:

SISTEMA DE MONITOREO DE DATOS AMBIENTALES EN CULTIVOS DE ARROZ

INTEGRANTES:

DEBBIE DANIELLA DONOSO CALLE

ERICK GABRIEL GARCÍA NARVÁEZ

WILLIAM JOSEPH AYALA QUINDE

GABRIEL STÉFANO VILLANUEVA ROSERO

PERIODO:

PAE 2022

ING. CARLOS ENRIQUE CEDENO ZAMORA

Tutor

Contenido

Descripción del problema

Posterior al desarrollo de la primera versión del sistema de monitoreo de datos ambientales en cultivos de arroz, se detectaron distintos problemas durante la instalación de la red de sensores encargada de comunicar y sensar los datos ambientales, entre los problemas encontrados tenemos:

Poca duración de batería: Ya que los nodos permanecían compartiendo y sensando los datos ambientales durante todo el día de manera ininterrumpida, la energía suministrada por la batería de litio no era suficiente para asegurar jornadas largas de funcionamiento.

Incorrecta transmisión de datos: Este problema se presenta en el caso de que un nodo hijo deje de trasmitir los datos que está midiendo a su nodo padre. El nodo padre sigue enviando a su nodo padre superior los últimos datos recibidos del sensor hijo, dando la impresión de que el nodo hijo sigue midiendo datos cuando en realidad está fuera de línea.

Organización de equipos

Para solucionar los problemas mencionados se optó por formar dos grupos, que se enfoquen en solucionar los problemas de ahorro de energía y comunicación de los nodos, de esta manera dividimos los esfuerzos y reducimos el tiempo en encontrar una solución, los grupos y sus integrantes se detallan a continuación:

- Grupo de Sincronización:
    - Erick Gabriel García Narváez
    - Debbie Daniella Donoso Calle
- Grupo de Comunicación:
    - William Joseph Ayala Quinde
    - Gabriel Stéfano Villanueva Rosero

Documentación inicial

Para el desarrollo del proyecto presentado se usó como guía la tesis titulada como:

Proyecto Integrador - Monitoreo de Cultivos.

En esta tesis se presenta el problema inicial y su solución, la cual busca dar acceso a pequeños y medianos productores a soluciones tecnológicas que permitan ayudarlos a tomar decisiones basado en datos ambientales capturados en tiempo real.

Como solución principal se propuso una red de sensores inalámbricos de 10 nodos utilizando como microcontrolador un Arduino Nano y un módulo de comunicación nrf24l01+. La consta nodos sensores que miden datos de temperatura, humedad del suelo, radiación solar, y precipitación, y 1 nodo Gateway que receptó los datos de la red y las envió a una Raspberry que los subió a un sistema de visualización desarrollado con Grafana y Django.

Si bien el plan propuesto cumple con los objetivos establecidos en su desarrollo de tesis como equipo encargado notamos que no se consideraron variables importantes a la hora de una implementación en campo, variables como durabilidad de las baterías y consistencia de los datos enviados al servidor que se encarga de mostrar los datos obtenidos.

Esto hace que el proyecto se vuelva poco escalable y confiable ante la vista de productores experimentados en su área que conocen de manera empírica y artesanal las condiciones del ambiente.

Se procedió a realizar una búsqueda/ investigación para encontrar información relevante sobre cómo se podría configurar el reloj RTC DS3231 en un Arduino Nano, para lo cual se determinó que la mejor manera de realizar esta configuración y sincronización de todos los nodos era a través de unas líneas de código adicionales, así también de un script .bat para enviar la hora de la PC a través del serial.

Se procedió a realizar una búsqueda/ investigación para encontrar información relevante sobre cómo se podría configurar el reloj RTC DS3231 en un Arduino Nano, para lo cual se determinó que la mejor manera de realizar esta configuración y sincronización de todos los nodos era a través de unas líneas de código adicionales, así también de un script .bat para enviar la hora de la PC a través del serial.

Para lograr apagar o encender los dispositivos a una hora determinada, se agregaron nuevas funciones en el código, de esta manera se puede colocar a qué hora se requiere que los dispositivos se apaguen o en su caso luego de que tiempo se deben encender, es así como se logra el objetivo de ahorro de energía.

Propuesta solución

Comunicación

Para que el nodo padre pueda armar la trama correcta y mostrar los datos actualizados de los nodos hijos, se realizaron modificaciones en la función que valida la trama formada por los datos recibidos de los nodos hijos.

La función que valida la trama actualmente cuenta el número de veces que los datos de uno de los nodos hijos llega vacío. Este contador luego es comparado contra un límite superior y si sobrepasa el límite configurado, se presenta los datos del nodo hijo que no está enviado datos como vacíos.

Debido a que el límite superior para el cual se compara el contador es muy grande, esto provoca que la trama se siga enviando con los últimos datos recibidos del nodo hijo durante un extenso periodo de tiempo. Para que el nodo padre construya una trama con los valores apropiados, se bajó el valor del límite superior con el que se compara el contador en la función que valida la

trama. De esta manera, el usuario puede percibir de forma inmediata cuando existe un nodo hijo que no está enviando información.

Debido a la topología de la red, un nodo hijo también puede recibir datos, por lo que la misma modificación se realizó en la función que recibe los datos. En esta función existe una validación con la misma lógica de contador de error y comparación.

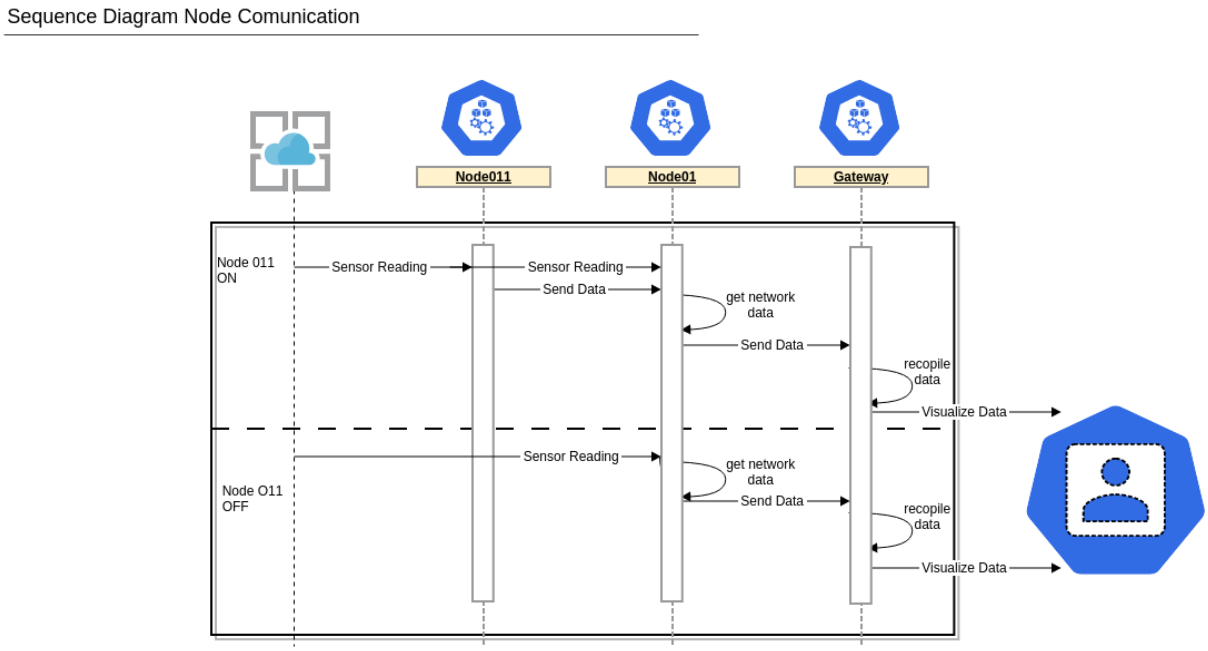


Ilustración 1. Diagrama de Secuencia de la mejora implementada en la comunicación de Datos.

Una vez implementada las mejoras en los nodos, el nodo padre (Gateway) envía una trama correcta mostrando los datos que ha recibido de los nodos. En la Ilustración 1, se nota que cuando se tiene uno de los nodos fuera de línea, el usuario visualizará la data enviada solo por el nodo hijo que está enviando la información por la red. En cambio, cuando ambos nodos hijos estén en línea y enviando data, el usuario visualizará la trama con ambos datos recibidos de los nodos hijos.

Sincronización

Para llevar un control de los eventos que requieran que sean ejecutados por todos los nodos de manera síncrona, se añadió al circuito un módulo DS3231.

El DS3231 es un reloj de tiempo real (RTC) I2C de bajo coste y gran precisión extremadamente preciso (RTC) con un oscilador de cristal temperatura (TCXO) y cristales integrados. El dispositivo incorpora una entrada de batería, y mantiene la precisión de la hora cuando se interrumpe la

alimentación principal del dispositivo, lo que permite mantener la sincronización aun después de resolver fallos o realizar mantenimientos que desconecten los sensores.

Para sincronizar los relojes de cada sensor se puede seguir los pasos mostrados en el siguiente diagrama:

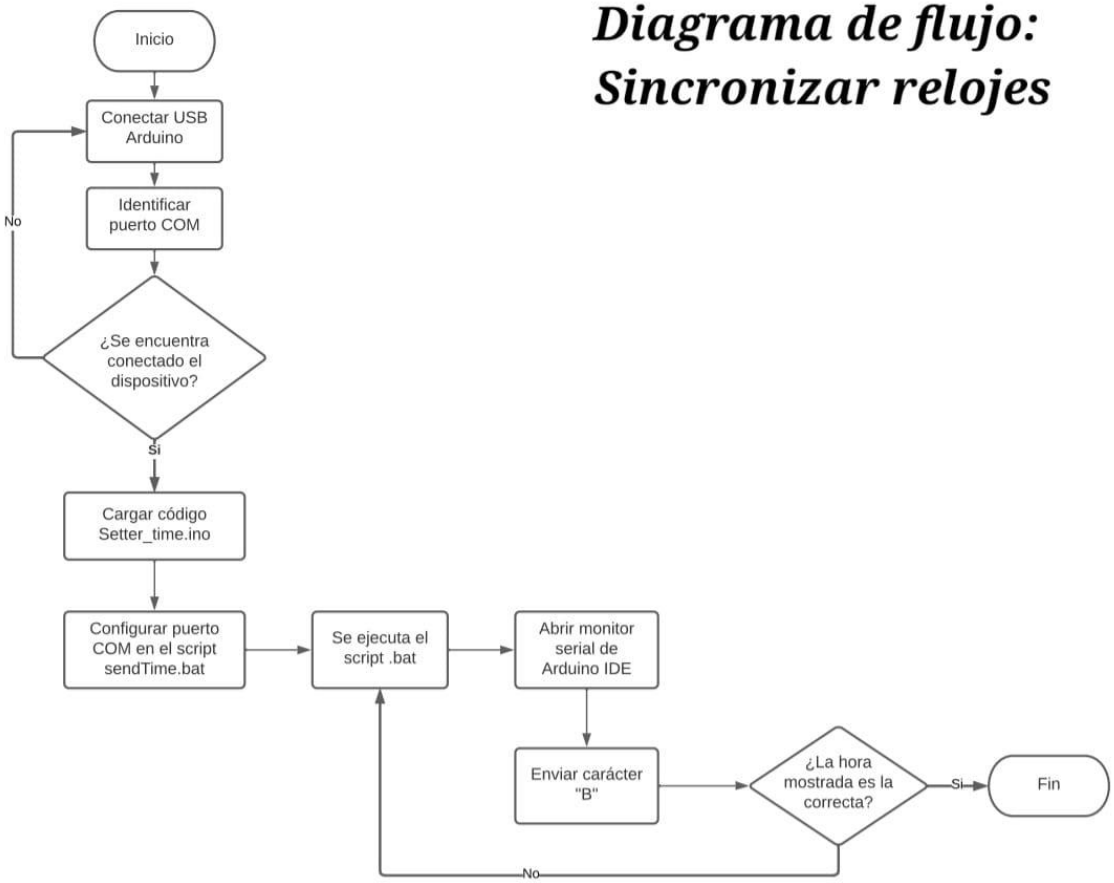


Ilustración 2. Se observa el diagrama de flujo que muestra cómo se sincronizarán los relojes, en el mismo se puede apreciar que como primer paso se debe conectar el nodo hijo y nodo padre al PC (en donde se realizaron las respectivas pruebas), luego se deben identificar los correspondientes puertos COM al que cada nodo se encuentra conectado, si el dispositivo no se encuentra conectado se deberá conectar el cable USB del Arduino nano, caso contario si se encuentra conectado se deberá cargar el código setter_time.ino a los nodos respectivos, como consiguiente se deberá configurar el puerto COM respectivo en el script senTime.bat (se encargará de enviar la hora de la PC por serial), como siguiente paso se ejecuta el script .bat y se abre el monitor serial de Arduino IDE para poder enviar el carácter 'B' y de esta manera observar que se tiene la misma hora del PC, si la hora mostrada no es la correcta se deberá ejecutar de nuevo el script .bat , caso contrario se realiza la sincronización de buena manera.

Ahorro energía

Para poder entender de una mejor manera el funcionamiento del despertar de los equipos (modo de ahorro de energía) se puede observar el siguiente diagrama de secuencia:

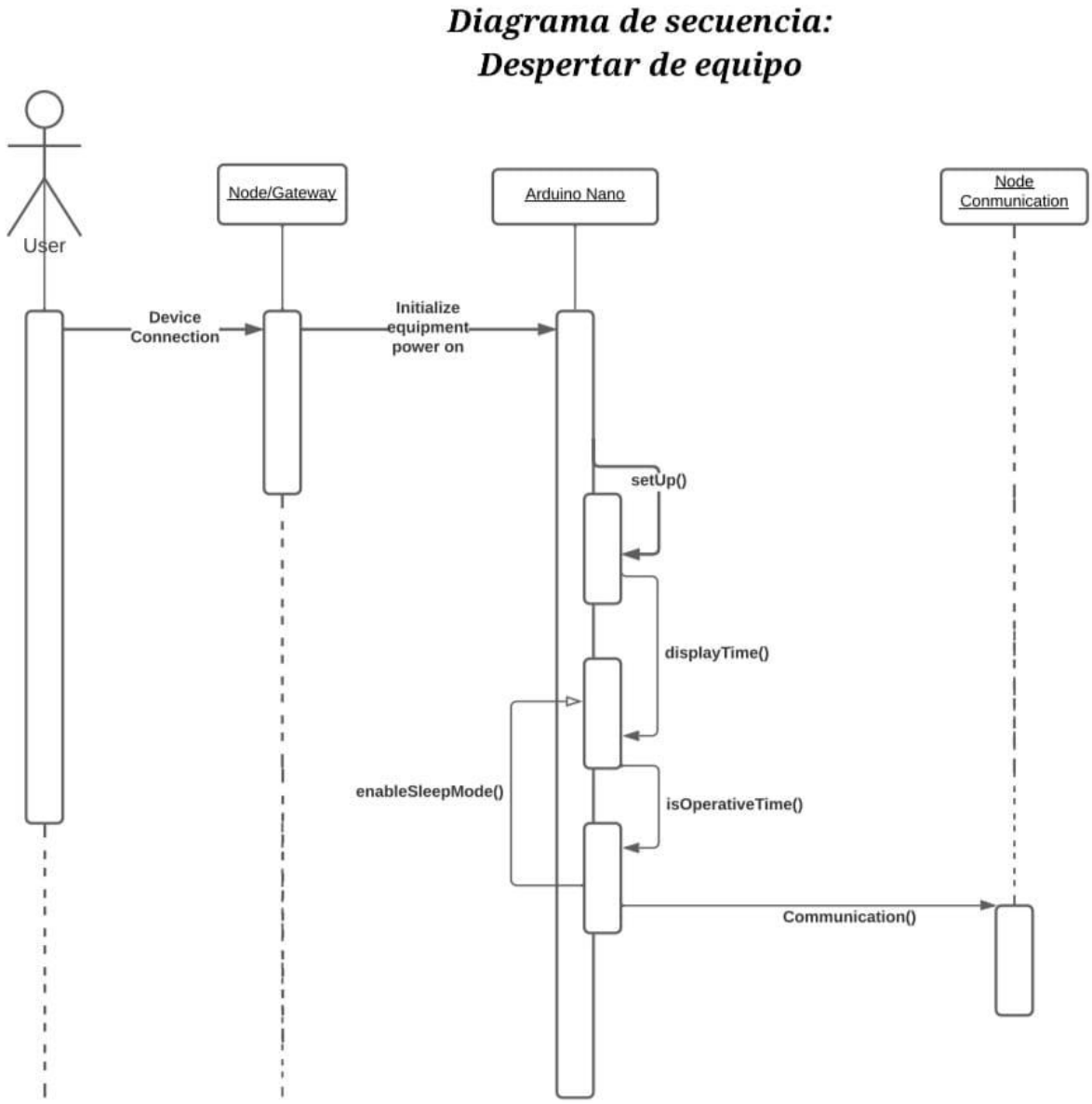


Ilustración 3. Se observa el diagrama de secuencia que muestra el despertar del equipo; es decir, como funcionarían los dispositivos con el modo de bajo consumo de energía, el proceso para que los nodos entren en el modo de ahorro de energía comienza cuando el usuario realiza la conexión física de los dispositivos tanto de los nodos hijo como del Gateway, luego de esto se encienden los

equipos pero internamente sucede lo descrito a continuación en el Arduino nano (código cargado en mismo/nodo01): Al encender el dispositivo se ejecuta internamente las líneas de código dentro del setUp(), en donde se encuentra el método displayTime() y dentro de éste se llama al método isOperativeTime(), cuando se ejecute este método se envía una notificación de ejecución correcta al método displayTime() y por consiguiente se habilita el modo sleep, finalmente se realiza la comunicación entre los nodos.

Repositorio

Mencionar una tarea y el nombre del archivo/script junto con el enlace del código

Para sincronizar los relojes de cada nodo Sensor y Gateway se usaron los siguientes códigos:

- Código Receptor Arduino: Setter_Timer_Time_Code.ino
- Código Transmisor Batch: SendTime.bat

Para habilitar el modo reposo se agregó la función enableSleepMode() dentro de ambos nodos de transmisión:

- Código Receptor Arduino: Nodo_01.ino
- Código Transmisor Batch: Nodo_Gateway.ino

Para la mejora de la comunicación entre nodos, se usaron los siguientes códigos:

- Código Nodo Gateway: Nodo_Gateway.ino
- Código Nodo Hijo: Nodo01.ino

Fotos y evidencia

- Demostración de Sincronización y activación de estado Reposo de los Nodos: Video

Ilustración 4. Desarmando Nodo para conectar modulo DS3231

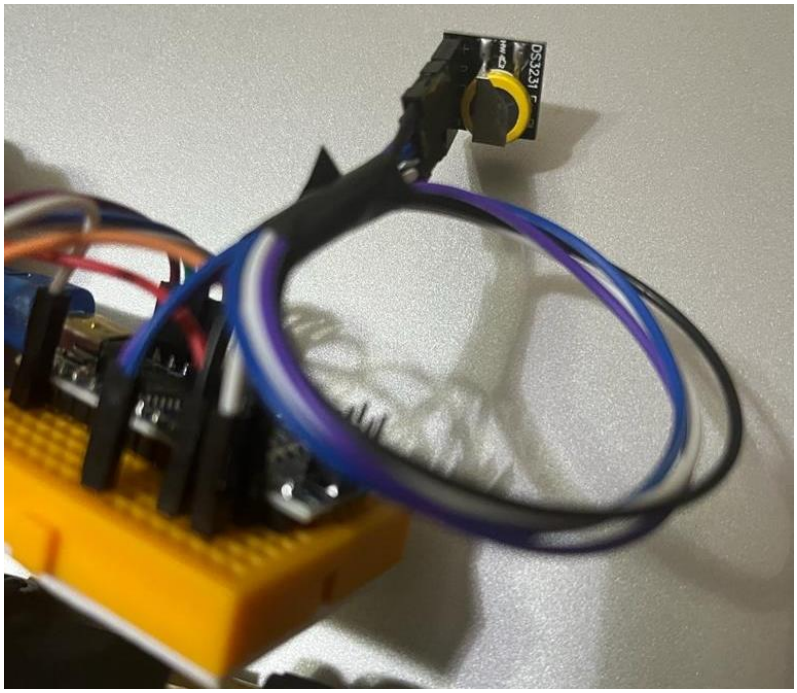


Ilustración 5. Conectando Módulo DS3231

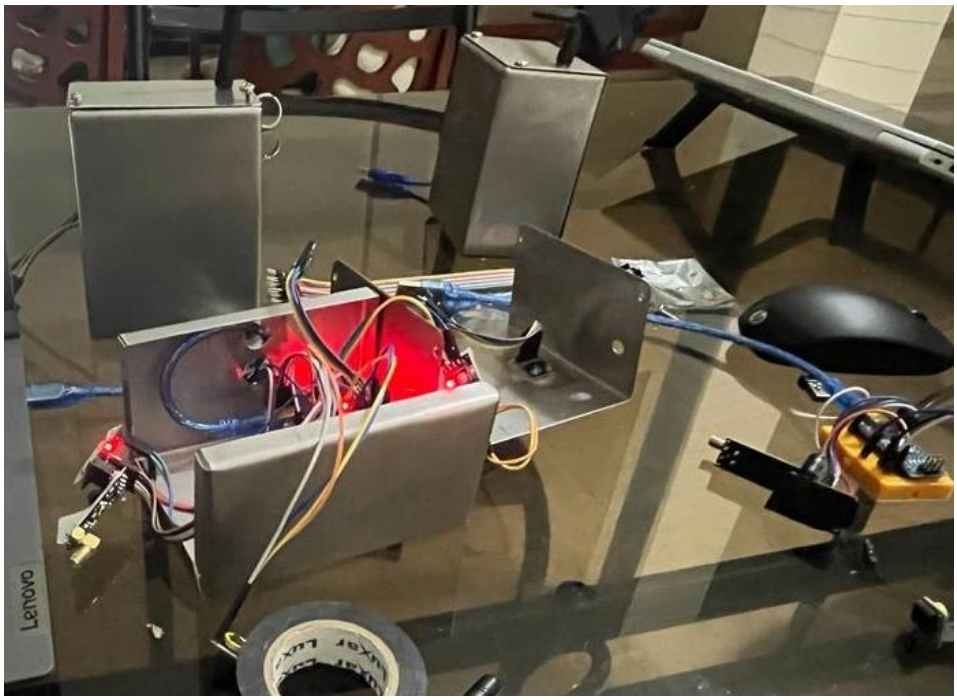


Ilustración 6. Probando correcto funcionamiento de módulo DS3231

Ilustración 7. Nodos listos para instalar

- Demostración de Mejoras en la comunicación de Datos entre nodos: Video

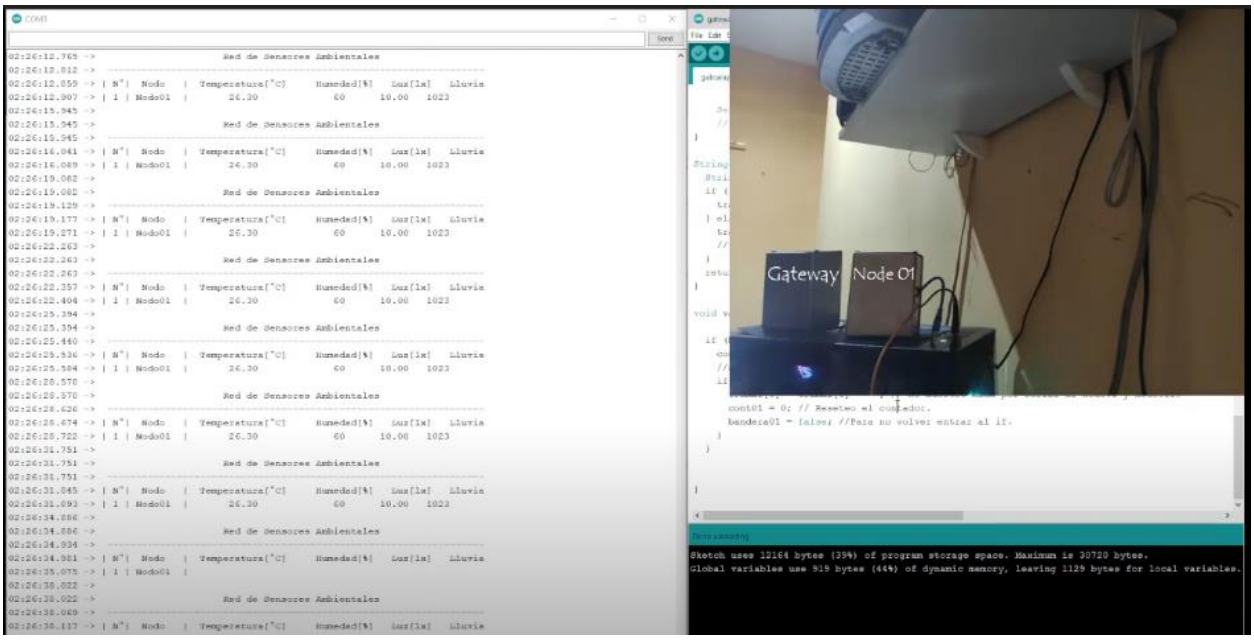


Ilustración 8. Visualización de la trama del nodo padre con nodo hijo fuera de línea.

Anexos

Métodos de Sincronización de Reloj

1. A través del puerto I2C:

El módulo DS3231 – Modelo RTC lleva el control del tiempo dependiente o independiente de la tarjeta Arduino.

En este método el módulo viene ensamblado con batería suministrada listo para usar, para esto es necesario la biblioteca “WIRE”, el cual viene por defecto en la tarjeta Arduino, y la biblioteca DS3231 el cuál puede ser obtenido del repositorio de Github.

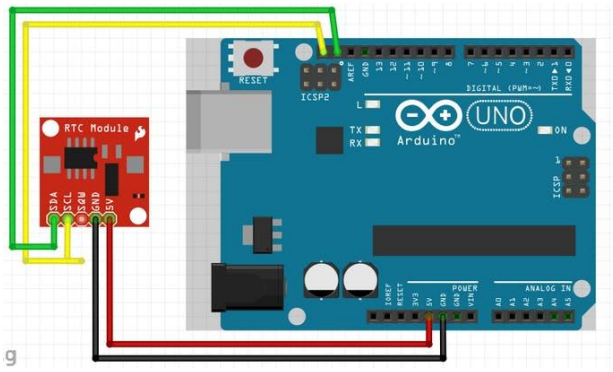


Ilustración 9. Circuito de conexión del módulo DS3231 a través de conexión I2C

2. A través de un servidor de tiempo

Un servidor de tiempo es una computadora en una red que lee la hora de algún reloj de referencia y la distribuye a la red. La fuente de reloj de un servidor de tiempo puede ser otro servidor de tiempo, un reloj atómico o un reloj de radio.

Al requerir una conexión con el servidor es necesario instalar un shield de Ethernet IC, a el equipo Arduino:

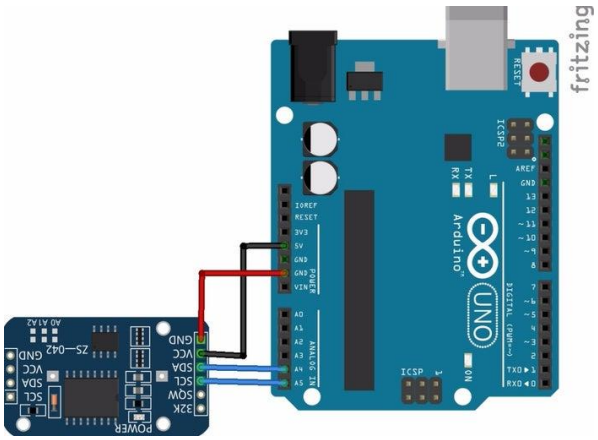
Para establecer la comunicación entre nodo y servidor de tiempo se utiliza el protocolo de tiempo de red (NTP), esta comunicación se basa en un modelo Cliente/Servidor. Un cliente NTP inicia una comunicación con un servidor NTP enviando un paquete de solicitud. Un paquete de solicitud NTP básico tiene una longitud de 48 bytes. Luego, el servidor NTP agrega su propia marca de tiempo al paquete de solicitud y lo envía de vuelta al cliente.

Se utilizan tres bibliotecas

- Biblioteca SPI <SPI.h>: Para comunicarse con Ethernet Shield.
- Biblioteca de tiempo <TimeLib.h>: para actualizar y mostrar la fecha y la hora.
- Biblioteca de Ethernet <Ethernet.h>: para conectar el Arduino a internet o a una red.

3. A través de comunicación serial con PC:

A diferencia del primer método, el reloj es conectado a través de puertos análogos como se muestra:



El dispositivo Arduino recibe una trama desde el PC por la conexión USB, esta trama debe ser parseada de bytes codificados a bytes decimales.

La trama es enviada gracias a un ejecutable .bat programado en MS-Dos, en el que se especifica el puerto COM al que se enviará la trama.

```
Status for device COM3:
-----------------------
    Baud:             9600
    Parity:           None
    Data Bits:        8
    Stop Bits:        1
    Timeout:          OFF
    XON/XOFF:         OFF
    CTS handshaking:  OFF
    DSR handshaking:  OFF
    DSR sensitivity:  OFF
    DTR circuit:      ON
    RTS circuit:      ON

S46,
D28,
H15,
T05,
M10,
J2015,
```

Métodos de Sincronización de Reloj

1. **Librería Low Power:**

Arduino está diseñado más para ser fácil de usar que para consumir poco, ya que la eficiencia no estuvo nunca dentro de los parámetros de diseño. Las pruebas de estrés han demostrado que un Arduino UNO tiende a consumir en vacío un aproximado 45MA y un Arduino Nano 15mA.

Esto hace poco rentable usar los equipos como nodos alimentados por baterías o fuentes limitadas de batería.

Para ello existen métodos para deshabilitar considerablemente el número de procesos durante un tiempo definido o hasta que sean activadas por señales externas ejecutadas por un reloj.

Una de estas librerías es LowPower.h, la cual es ligera y sencilla de usar, podemos activar el modo de reposo de un Arduino con tan solo la siguiente línea:

LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);

Como se ve, simplemente pedimos a la librería que duerma a Arduino 8 segundos y mantenga apagados los convertidores analógicos a digital, y el circuito BOD (O Brown Out Detection).

Si se necesita desactivar el Arduino por más tiempo esta línea puede ser combinad por un loop finito que mantega el dispositivo dormido:

O en su defecto tiene otras tres maneras de despertar un Arduino del modo Sleep indefinidos:

• Interrupciones hardware

• Timers y WDT que es un timer un tanto peculiar

• UART o puerta serie.

Referencias:

1. https://create.arduino.cc/projecthub/MisterBotBreak/how-to-use-a-real-time-clock-module-ds3231-bc90fe
2. https://www.circuitbasics.com/using-an-arduino-ethernet-shield-for-timekeeping/
3. https://www.instructables.com/Synchronise-DS3221-RTC-with-PC-via-Arduino/
4. https://www.prometec.net/el-modo-sleep-en-arduino/

Anexos

Enlace de Tesis: https://espolec.sharepoint.com/:b:/s/ComunitariasCIDIS2022/ESvj90NLC-hEniuwD7-8YI4BDjZ9jqo7PWLHqtd4EXdyFw?e=ajgq0M

Librerías utilizadas:

RF24

Comunicación

Controlador de radio, biblioteca OSI capa 2 para módulos nrf24L01(+).
Biblioteca central para comunicación nRF24L01(+). Fácil de usar para principiantes, pero ofrece opciones de configuración avanzadas. Se incluyen muchos ejemplos para demostrar varios modos de comunicación.

SPI

[Comunicación]

Descripción

Esta biblioteca le permite comunicarse con dispositivos SPI, con Arduino como dispositivo controlador. Esta biblioteca se incluye con todas las plataformas Arduino (avr, megaavr, mbed, samd, sam, arc32), por **lo que no necesita instalar** la biblioteca por separado.

WIRE

[Comunicación]

Descripción

Esta biblioteca le permite comunicarse con dispositivos I2C/TWI. En las placas Arduino con el diseño R3 (pinout 1.0), SDA (línea de datos) y SCL (línea de reloj) están en los encabezados de los pines cerca del pin AREF. El Arduino Due tiene dos interfaces I2C/TWI SDA1 y SCL1 que están cerca del pin AREF y la adicional está en los pines 20 y 21.

BH1750

Sensores

Biblioteca Arduino para las placas de conexión del sensor de luz digital que contiene el BH1750FVI IC
Biblioteca BH1750
bastante simple y robusta. Compatible con Arduino, ESP8266 y ESP32.

SIMPLE TIMER

Sincronización

Temporizador arduino simple.
Una biblioteca simple de Arduino para trabajar con el tiempo.

Código de Nodo Gateway implementado, con comentarios de uso en funciones.

```
// Nodo 00, Nodo Central o Nodo Gateway
#include <RF24.h>
#include <RF24Network.h>
#include <SPI.h>
#include <SimpleTimer.h>
#include <Wire.h>
#include "LowPower.h"

#define DS3231_I2C_ADDRESS 0x68

RF24 radio(8, 10); // Crea una instancia y envía los pines (CE,CSN) del nrf24l01+
RF24Network network(radio); // Construye la red
const uint16_t this_node = 00; // Dirección del nodo actual en formato Octal

float data[8]; // N° total de sensores del nodo del nivel 1 y nivel 2
String tramaFinal; // Trama de toda la red.
String trama1[5]; // Trama nivel 1; trama1[0] = nodo01, ….. , trama1[4] = nodo05.
String trama2[4]; // Trama nivel 2; trama2[0] = nodo011, ..... , trama2[3] = nodo014.

int cont01 = 0, cont02 = 0, cont03 = 0, cont04 = 0, cont05 = 0;
bool bandera_Red = false;
static bool bandera01 = false, bandera02 = false, bandera03 = false, bandera04 = false, bandera05 =
false;

SimpleTimer firstTimer;

// Timer DS3231 Variables
```

```
String current_time = "";
int init_hour = 18; // 6PM
int last_hour = 5; // 5AM

byte year, month, day, hour, minute;
byte second;

void setup() {
  Serial.begin(9600);
  SPI.begin();
  radio.begin();
  network.begin(120, this_node); // Canal y dirección del nodo
  radio.setDataRate(RF24_250KBPS); // Velocidad de transmisión de datos
  //radio.setPALevel(RF24_PA_MAX,1); // Nivel PA, estado LNA

  //Si el modulo es nrf24l01+PA+LNA          PA,    LNA,   mA
  //radio.setPALevel(RF24_PA_MIN,1); //    -18 dBm, -6dBm, 7 mA
  //radio.setPALevel(RF24_PA_LOW,1); //    -12 dBm, 0dBm,  7.5 mA
  //radio.setPALevel(RF24_PA_HIGH,1); //   -6 dBm,  3dBm,  9 mA
  radio.setPALevel(RF24_PA_MAX, 1); //      0 dBm,  7dBm,  11.3 mA
}

void loop() {
  if(!isOperativeTime()){
    enableSleepMode();
  }else{
    network.update(); // Mantiene la capa en funcionamiento
    while (network.available() && !bandera_Red) {
      //Serial.println("Nodos hijos detectados...");
      //bandera_Inicio = true;
      firstTimer.setInterval(3000);
      bandera_Red = true; // Para no volver entrar al while?
    }

    if (firstTimer.isReady() /*&& bandera_Inicio*/ /*&& !flag*/) {
      firstTimer.reset();
      mostrar_Datos();
      //flag = true; //Para no volver entrar al if.
    }

    leer_Red_Sensores();
  }

}
```

```
void leer_Red_Sensores() {

  network.update(); //Esta función se debe llamar con regularidad para mantener la capa en funcionamiento. Aquí es donde las cargas útiles se redirigen, reciben y ocurre toda la acción.

  while (network.available()) {  // ¿Hay datos entrantes en el NC?
    RF24NetworkHeader header2;

    //Lee el siguiente header disponible sin avanzar al siguiente mensaje entrante.
    //Útil para cambiar el tipo de mensaje
    //network.peek(header2);

    //Lee la siguiente payload disponible sin avanzar al siguiente mensaje entrante.
    //Útil para hacer una capa de manipulación de paquetes transparente encima de RF24Network .
    //network.peek(header2,&data, sizeof(data));


    network.read(header2, &data, sizeof(data));
    if (header2.from_node == 1) {
      //Serial.print("-1");
      trama1[0] = validar_Datos_n1(data[0], data[1], data[2], data[3]); //Trama nodo01
      //trama2[0] = validar_Datos_n2(data[4], data[5]); //Trama nodo011
      cont01 = 0; // Reseteo el contador.
      bandera01 = true; //Para entrar al if
      //network.peek(header2,&data, sizeof(data));
      //network.peek(header2);
    }


  }
  validar_Trama();

}


void mostrar_Datos() {

  ///*
  Serial.println("\n\t\t     Red de Sensores Ambientales");
  Serial.println(" ----------------------------------------------------------------");
  Serial.println("| N°|  Nodo   |  Temperatura[°C]     Humedad[%]   Luz[lx]   Lluvia");
  Serial.println("| 1 | Nodo01  |\t      " + trama1[0]);
  displayTimeDS3231();
```

```
  /*Serial.println("| 2 | Nodo02  |\t     " + trama1[1]);
   Serial.println("| 3 | Nodo03  |\t     " + trama1[2]);
   Serial.println("| 4 | Nodo04  |\t     " + trama1[3]);
   Serial.println("| 5 | Nodo05  |\t     " + trama1[4]);
   Serial.println("| 6 | Nodo011 |\t     " + trama2[0]);
   Serial.println("| 7 | Nodo012 |\t     " + trama2[1]);
   Serial.println("| 8 | Nodo013 |\t     " + trama2[2]);
   Serial.println("| 9 | Nodo014 |\t     " + trama2[3]);
   Serial.println(" ----------------------------------------------------------------");
   //*/


  /*
   tramaFinal = "\n1-" + trama1[0] + //  1-31.00-0-19-1023
        "\n2-" + trama1[1] +      //  2-30.74-0-83-1023
        "\n3-" + trama1[2] +      //  3-30.83-0-55-1023
        "\n4-" + trama1[3] +      //  4-30.27-0-89-1023
        "\n5-" + trama1[4] +      //  5-30.87-0-35-1023
        "\n6-" + trama2[0] +      //  6-30.76-0-41-1023
        "\n7-" + trama2[1] +      //  7-30.91-0-21-1023
        "\n8-" + trama2[2] +      //  8-30.76-0-22-1023
        "\n9-" + trama2[3];       //  9-30.60-0-49-1023
   Serial.println(tramaFinal + "\n");
   //*/
}

String validar_Datos_n1(float sensor1 , float sensor2, float sensor3, float sensor4) {
  String trama1; //sensor1 = temperatura ; sensor2 = %HR ; sensor3 = Radiacion solar ; sensor4 =
precipitacion
  if ( sensor1 == 0 && sensor2 == 0 && sensor3 == 0 && sensor4 == 0) {
    trama1 = "";
  } else {
    trama1 = String(sensor1) + "\t\t" + String(int(sensor2)) + "\t" + String(sensor3) + "\t" +
String(int(sensor4));
    //trama1 = String(sensor1)+"-"+ String(int(sensor2))+"-"+String(sensor3)+"-"+String(int(sensor4));
//Formato para enviarlo a la raspberry
  }
  return trama1;
}

void validar_Trama() {

  if (bandera01) {
    cont01++;
```

```
   //Serial.println(cont01);
   if (
      /**
       * @author William Joseph Ayala Quinde
       * @author Gabriel Stefano Villanueva Rosero
       * Se modifica el valor de comparación del contador para que
       * el cambio en la comunicación se note inmediatamente si no hay mensajes
       * por parte del nodo hijo
       */
     cont01 == 2) {
     trama1[0] = trama2[0] = ""; // No muestro nada por serial al nodo01 y nodo011.
     cont01 = 0; // Reseteo el contador.
     bandera01 = false; //Para no volver entrar al if.
   }
  }
}

/**
 * Este método realiza la conversión de un valor BCD a DEC
 * @author Erick García, Debbie Donoso
 * @params val es un valor en byte del decimal en codigo binario
 * @return Un byte que representa un valor decimal (hora|minutos|segudo|dia|mes|año)
*/
byte bcdToDec(byte val){
  return ( (val / 16 * 10) + (val % 16) );
}

/**
 * Este método obtiene la hora actual del timer DS3231 y determina si se encuentra en una hora
operativa
 * @author Erick García, Debbie Donoso
 * @return Verdadero si es hora operativa, falso si no
*/
boolean isOperativeTime(){
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  // retrieve data from DS3231
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);

  boolean sleep_flag = int(hour) >= init_hour && (int(hour) <= last_hour && int(minute) <= 0 );
  return sleep_flag ;
}

/**
 * Este método mantiene el dispositivo arduino en estado de reposo durante 8S por iteración
```

```
 * @author Erick García, Debbie Donoso
 * @return No retorna.
*/
void enableSleepMode(){
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);

  boolean sleep_flag = int(hour) >= init_hour && (int(hour) <= last_hour && int(minute) <= 0 );
  do {
     sleep_flag = int(hour) >= init_hour && (int(hour) <= last_hour && int(minute) <= 0 );
     LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
  } while (sleep_flag);
}

/**
 * Este método mantiene el dispositivo arduino en estado de reposo durante 8S por iteración
 * @author Erick García, Debbie Donoso
 * @params:
 *    *second puntero hacia un espacio de 1 byte de memoria reservado para almacenar segundos
 *    *minute puntero hacia un espacio de 1 byte de memoria reservado para almacenar minutos
 *    *hour puntero hacia un espacio de 1 byte de memoria reservado para almacenar horas
 *    *dayOfWeek puntero hacia un espacio de 1 byte de memoria reservado para almacenar días de la
semana
 *    *dayOfMonth puntero hacia un espacio de 1 byte de memoria reservado para almacenar días del
mes
 *    *month puntero hacia un espacio de 1 byte de memoria reservado para almacenar meses
 *    *year puntero hacia un espacio de 1 byte de memoria reservado para almacenar años
 * @return No retorna.
*/
void readDS3231time(byte *second,
                    byte *minute,
                    byte *hour,
                    byte *dayOfWeek,
                    byte *dayOfMonth,
                    byte *month,
                    byte *year)
{
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // set DS3231 register pointer to 00h
  Wire.endTransmission();
  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);
  // request seven bytes of data from DS3231 starting from register 00h
  *second = bcdToDec(Wire.read() & 0x7f);
  *minute = bcdToDec(Wire.read());
```

```
  *hour = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month = bcdToDec(Wire.read());
  *year = bcdToDec(Wire.read());
}

/**
 * Este método imprime por consola serial la hora y fecha actual del timer DS3231
 * @author Erick García, Debbie Donoso
 * @return Imprime por consola.
*/
void displayTimeDS3231() {
  current_time = "";
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  // retrieve data from DS3231
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);
  // send it to the serial monitor
  Serial.print(hour, DEC);
  current_time += String(hour);

  // convert the byte variable to a decimal number when displayed
  Serial.print(":");
  current_time += ":";
  if (minute < 10)
  {
    Serial.print("0");
    current_time += "0";
  }
  Serial.print(minute, DEC);
  current_time += String(minute);
  Serial.print(":");
  current_time += ":";
  if (second < 10)
  {
    Serial.print("0");
    current_time += "0";
  }
  Serial.print(second, DEC);
  current_time += String(second);
  Serial.print(" ");
  Serial.print(dayOfMonth, DEC);
  Serial.print("/");
  Serial.print(month, DEC);
```

```
  Serial.print("/");
  Serial.print(year, DEC);
  Serial.println("");
  delay(1000);
}
```

Código de Nodo 01 implementado, con comentarios de uso en funciones.

```
// Nodo01 (Transceptor) Primer nodo hijo del Nodo Central

#include <RF24.h>
#include <RF24Network.h>
#include <SPI.h>
#include <Wire.h>
#include <BH1750.h>
#include "DHT.h"
#include "LowPower.h"

#define pin_Lluvia A1
#define DS3231_I2C_ADDRESS 0x68

#define DHTPIN 2 // Digital pin connected to the DHT sensor
#define DHTTYPE DHT11   // DHT 11
DHT dht(DHTPIN, DHTTYPE);

const int valor_Capacitivo_Aire = 800; // Valor calculado con el sensor al aire
const int valor_Capacitivo_Agua = 427 ; // Valor calculado con el sensor sumergido en agua

BH1750 medidorLuz;
RF24 radio(8, 10); // nRF24L01 (CE,CSN)
RF24Network network(radio); // Incluir la radio en la red
const uint16_t this_node = 01; // Dirección de este nodo en formato Octal
const uint16_t node00 = 00;

float data[4]; // Matriz para recibir los datos del nodo011 del nivel 2.
float dataa[8]; // Matriz para enviar los datos del nodo actual y nodo011.
float temp; // Variable para almacenar el valor de Temperatura
float hum; // Variable para almacenar el valor de Humedad
float lux; // Variable para almacenar el valor de Luminosidad
int lluvia; //0 - 1023

/**
 * @author William Joseph Ayala Quinde
 * @author Gabriel Stefano Villanueva Rosero
```

```
 * Se modifica el tipo de variable del contador para ahorrar espacio en la memoria flash
 */
byte cont = 0;

// Timer DS3231 Variables
String current_time = "";
int init_hour = 18; // 6PM
int last_hour = 5; // 5AM

byte year, month, day, hour, minute;
byte second;

void setup() {

  Serial.begin(9600);
  SPI.begin();
  Wire.begin();
  radio.begin();
  dht.begin();
  network.begin(120, this_node); //(channel, node address)
  //radio.setDataRate(RF24_1MBPS);
  //radio.setDataRate(RF24_2MBPS);
  radio.setDataRate(RF24_250KBPS);

  //Si el modulo es nrf24l01+PA+LNA          PA,    LNA,   mA
  //radio.setPALevel(RF24_PA_MIN,1); //    -18 dBm, -6dBm, 7 mA
  //radio.setPALevel(RF24_PA_LOW,1); //    -12 dBm, 0dBm,  7.5 mA
  radio.setPALevel(RF24_PA_HIGH, 1); //     -6 dBm,  3dBm,  9 mA
  //radio.setPALevel(RF24_PA_MAX,1); //      0 dBm,  7dBm,  11.3 mA

  medidorLuz.begin(BH1750::CONTINUOUS_HIGH_RES_MODE_2);
  /*if (medidorLuz.begin(BH1750::CONTINUOUS_HIGH_RES_MODE)) {
    Serial.println(F("BH1750 Advanced begin"));
    }
    else {
    Serial.println(F("Error initialising BH1750"));
    }
  */
}

void loop() {
   if (!isOperativeTime()){
    enableSleepMode();
  }else{
```

```
   network.update(); //Devuelve el tipo de la última carga útil recibida. //las cargas útiles se redirigen,
reciben y ocurre toda la acción.
   RF24NetworkHeader header(011); // Encabezado para recibir los datos del Nodo011.
   RF24NetworkHeader header2(node00); // Encabezado para enviar los datos al nodo Gateway.
   leerSensores();
   //recibir_Datos(header);
   enviar_Datos(header2);
   displayTimeDS3231();
  }
}

void leerSensores() {

  temp = dht.readTemperature();
  hum = dht.readHumidity();
  lux = medidorLuz.readLightLevel();
  lluvia = analogRead(pin_Lluvia);
  if (medidorLuz.measurementReady()) {
    lux = medidorLuz.readLightLevel();
  }

  //Datos sensados del nodo actual
  dataa[0] = temp;
  dataa[1] = hum;
  dataa[2] = lux;
  dataa[3] = lluvia;
}

void recibir_Datos(RF24NetworkHeader header) {

  static bool bandera = false;
  network.update();
  while (network.available()) { //True, si hay un mensaje disponible para este nodo.

   network.read(header, &data, sizeof(data)); // Leer los datos del nodo011 devuelve //El número total
de bytes copiados en message
   delay(random(5, 20));
   // 011 octal equals to 9 in decimal
   if (header.from_node == 9) { //Si los datos provienen del nodo011, entra al if.
     dataa[4] = data[0]; // Guardar los datos recibidos en el nodo actual.
     dataa[5] = data[1];
     cont = 0; // Reseteo el contador.
     bandera = true; //Para entrar al if
```

```
   }
  }

  if ( (network.available() == false) && bandera ) {
   cont++;
   //Serial.println(cont);
   if (
    /**
     * @author William Joseph Ayala Quinde
     * @author Gabriel Stefano Villanueva Rosero
     * Se modifica el valor de comparación del contador para que
     * el cambio en la comunicación se note inmediatamente si no hay mensajes
     * por parte del nodo hijo
     */
    cont == 1) {
    dataa[4] = dataa[5] = 0; // Reset de valores
    cont = 0; // Reset de contador.
    bandera = false; //Para no volver entrar al if.
   }
  }
}

void enviar_Datos(RF24NetworkHeader header) {


  // El tamaño de payload (carga útil) máximo predeterminado es 120 bytes.
  bool ok = network.write(header, &dataa, sizeof(dataa)); // Enviando datos del nodo01 o nodo011 al
nodo central.
  delay(random(5, 20));

  if (ok) {

   Serial.println("------------------Nodo 01--------------------");
   Serial.println("Temp: " + String(dataa[0]) + " C \t Hum: " + String(dataa[1]) + " %");
   Serial.println("Luz: " + String(dataa[2]) + " lx" + "\tLluvia: " + String(dataa[3]));
   Serial.println("------------------Nodo 011------------------");
   Serial.println("Temp: " + String(dataa[4]) + " C \t Hum: " + String(dataa[5]) + " % RH");
   Serial.println("!Se envio los datos al Nodo Gateway con exito!\n");

  } else {

   delay(random(5, 20));
   network.write(header, &dataa, sizeof(dataa));
  }
```

```
}

/**
 * Este método realiza la conversión de un valor BCD a DEC
 * @author Erick García, Debbie Donoso
 * @params val es un valor en byte del decimal en codigo binario
 * @return Un byte que representa un valor decimal (hora|minutos|segudo|dia|mes|año)
*/
byte bcdToDec(byte val){
  return ( (val / 16 * 10) + (val % 16) );
}

/**
 * Este método obtiene la hora actual del timer DS3231 y determina si se encuentra en una hora
operativa
 * @author Erick García, Debbie Donoso
 * @return Verdadero si es hora operativa, falso si no
*/
boolean isOperativeTime(){
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  // retrieve data from DS3231
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);

  boolean sleep_flag = int(hour) >= init_hour && (int(hour) <= last_hour && int(minute) <= 0 );
  return sleep_flag ;
}

/**
 * Este método mantiene el dispositivo arduino en estado de reposo durante 8S por iteración
 * @author Erick García, Debbie Donoso
 * @return No retorna.
*/
void enableSleepMode(){
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
  readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);

  boolean sleep_flag = int(hour) >= init_hour && (int(hour) <= last_hour && int(minute) <= 0 );
  do {
     sleep_flag = int(hour) >= init_hour && (int(hour) <= last_hour && int(minute) <= 0 );
     LowPower.powerDown(SLEEP_8S, ADC_OFF, BOD_OFF);
  } while (sleep_flag);
}

/**
```

```
 * Este método mantiene el dispositivo arduino en estado de reposo durante 8S por iteración
 * @author Erick García, Debbie Donoso
 * @params:
 *    *second puntero hacia un espacio de 1 byte de memoria reservado para almacenar segundos
 *    *minute puntero hacia un espacio de 1 byte de memoria reservado para almacenar minutos
 *    *hour puntero hacia un espacio de 1 byte de memoria reservado para almacenar horas
 *    *dayOfWeek puntero hacia un espacio de 1 byte de memoria reservado para almacenar días de la
semana
 *    *dayOfMonth puntero hacia un espacio de 1 byte de memoria reservado para almacenar días del
mes
 *    *month puntero hacia un espacio de 1 byte de memoria reservado para almacenar meses
 *    *year puntero hacia un espacio de 1 byte de memoria reservado para almacenar años
 * @return No retorna.
*/
void readDS3231time(byte *second,
                    byte *minute,
                    byte *hour,
                    byte *dayOfWeek,
                    byte *dayOfMonth,
                    byte *month,
                    byte *year)
{
  Wire.beginTransmission(DS3231_I2C_ADDRESS);
  Wire.write(0); // set DS3231 register pointer to 00h
  Wire.endTransmission();
  Wire.requestFrom(DS3231_I2C_ADDRESS, 7);
  // request seven bytes of data from DS3231 starting from register 00h
  *second = bcdToDec(Wire.read() & 0x7f);
  *minute = bcdToDec(Wire.read());
  *hour = bcdToDec(Wire.read() & 0x3f);
  *dayOfWeek = bcdToDec(Wire.read());
  *dayOfMonth = bcdToDec(Wire.read());
  *month = bcdToDec(Wire.read());
  *year = bcdToDec(Wire.read());
}

/**
 * Este método imprime por consola serial la hora y fecha actual del timer DS3231
 * @author Erick García, Debbie Donoso
 * @return Imprime por consola.
*/
void displayTimeDS3231() {
  current_time = "";
  byte second, minute, hour, dayOfWeek, dayOfMonth, month, year;
```

```
 // retrieve data from DS3231
 readDS3231time(&second, &minute, &hour, &dayOfWeek, &dayOfMonth, &month, &year);
 // send it to the serial monitor
 Serial.print(hour, DEC);
 current_time += String(hour);

 // convert the byte variable to a decimal number when displayed
 Serial.print(":");
 current_time += ":";
 if (minute < 10)
 {
  Serial.print("0");
  current_time += "0";
 }
 Serial.print(minute, DEC);
 current_time += String(minute);
 Serial.print(":");
 current_time += ":";
 if (second < 10)
 {
  Serial.print("0");
  current_time += "0";
 }
 Serial.print(second, DEC);
 current_time += String(second);
 Serial.print(" ");
 Serial.print(dayOfMonth, DEC);
 Serial.print("/");
 Serial.print(month, DEC);
 Serial.print("/");
 Serial.print(year, DEC);
 Serial.println("");
 delay(1000);
}
```