

Facultad Regional Paraná

**Técnicas Digitales III**

Trabajo Práctico N°7

Alumnos:

Churichi, Alan

Gonzalez, Exequiel

Profesores:

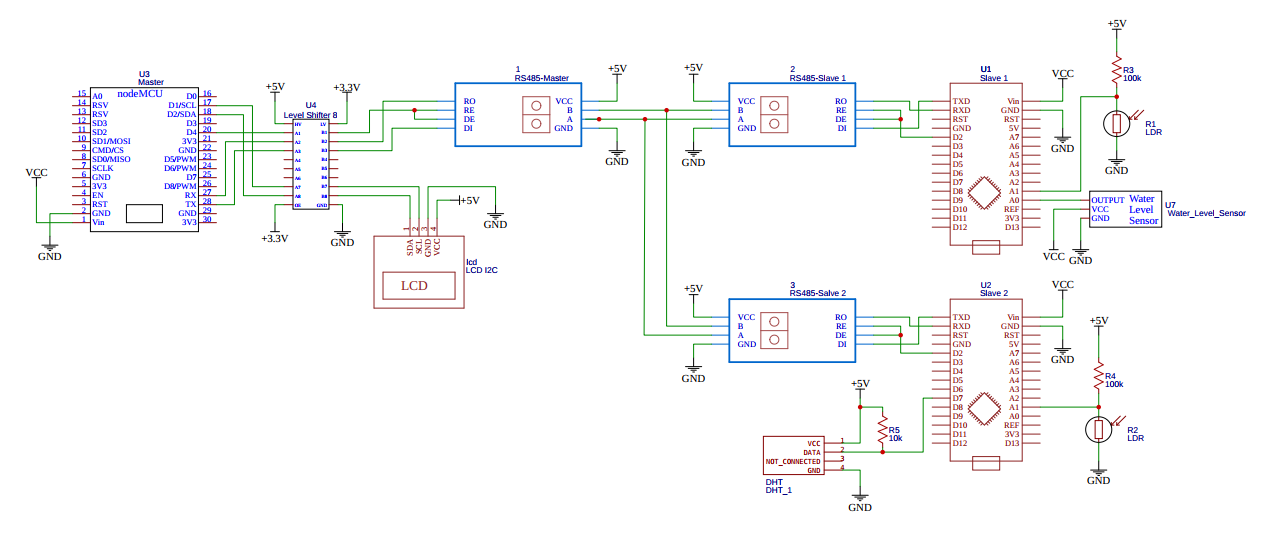
Yarce, Gustavo

Dachary, Alejandro

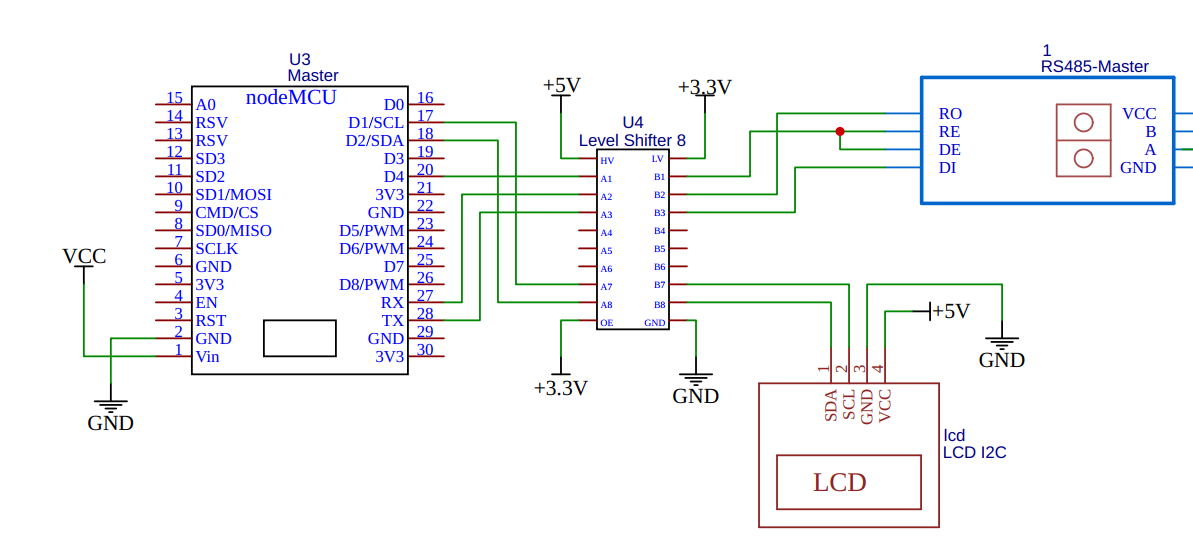
Fecha de Presentación: //2020

**Ejercicio 1**

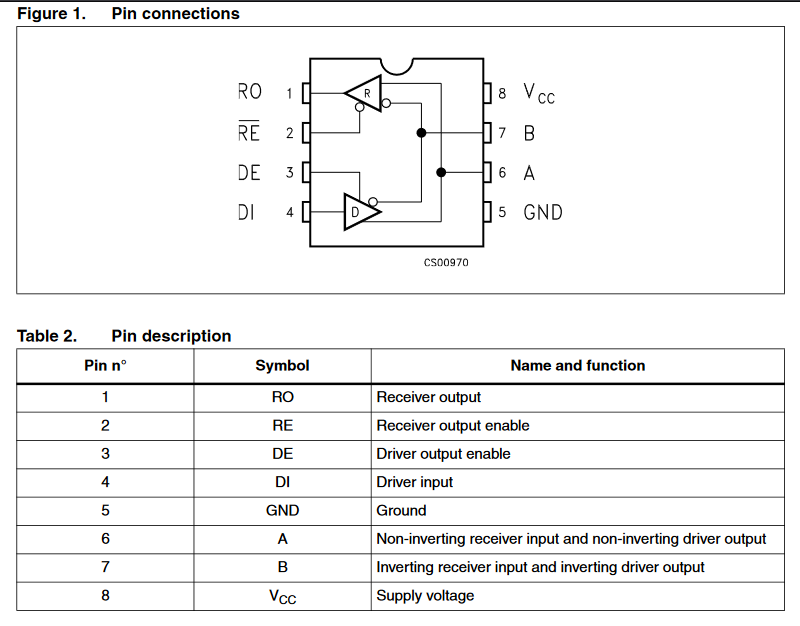
El circuito implementado es el siguiente



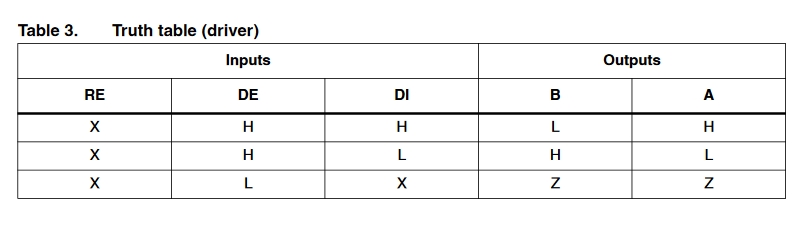
Ahora analicemos por partes. En primer lugar, tenemos el NodeMCU, la cual es una placa que tiene como microcontrolador al ESP8266.

A este se conecta un level shifter el cual se encarga de adaptar los niveles de tensión entre el ESP8266 y el módulo que incluye el Max485, ya que el primero trabaja con 3.3V mientras que el segundo trabaja con 5V. Ademas, al NodeMCU se conecta una pantalla LCD a través del protocolo I2C, también utilizando el level shifter.

Observando el datasheet del rs485 vemos la siguiente tabla que nos ayuda a entender su funcionamiento y el porqué de su conexionado.

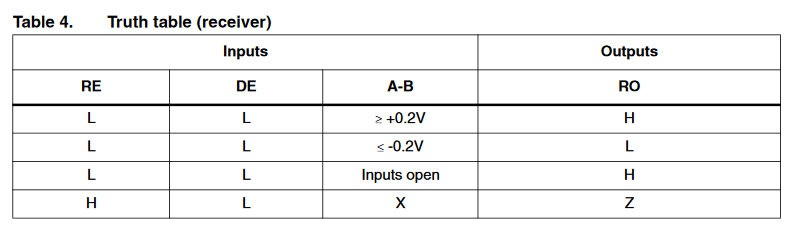


Entendiendo que es cada pin, podemos ver a través de sus tablas de verdad, como debe conectarse para que el RS485 transmita.



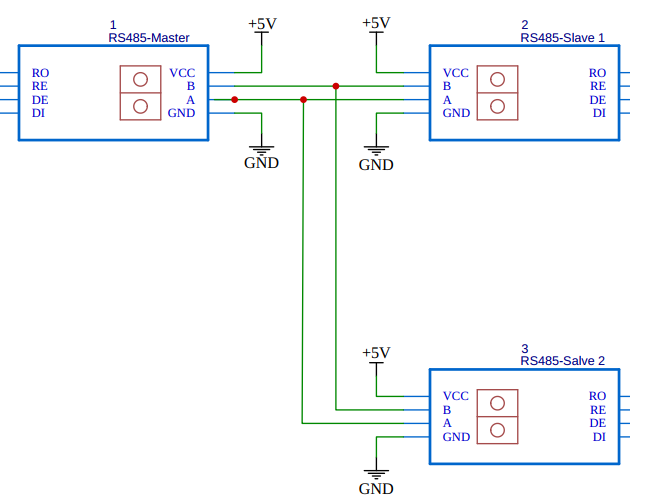
Se observa como para transmitir es imperativo tener el pin de Driver Enable (DE) en un valor alto, asi la señal ingresada en DI aparecerá en las salidas como corresponda. Si DE se encuentra en bajo, las salidas se colocan en alta impedancia.

Realizamos el mismo análisis para la salida.

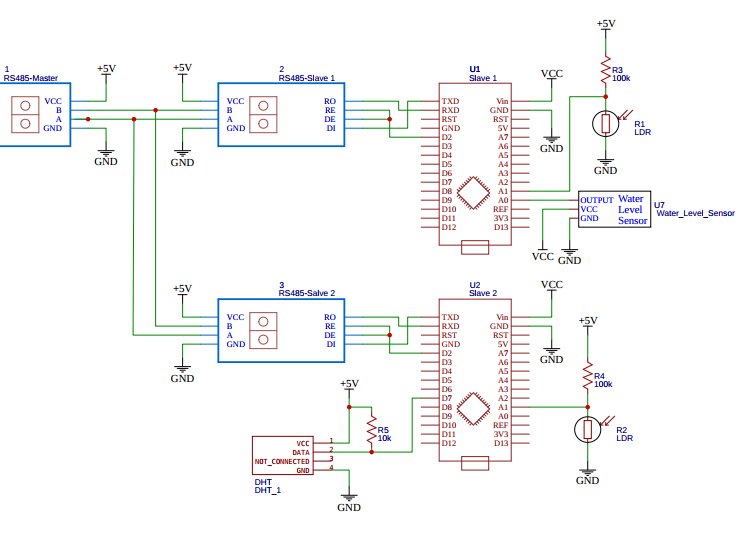
Observamos como en este caso es necesario que tanto RE como DE se encuentren en un valor bajo para que en función de las tensiones en las entradas el pin RO varie como corresponde. En caso de que RE se encuentre en alto, RO se colocará en alta impedancia.

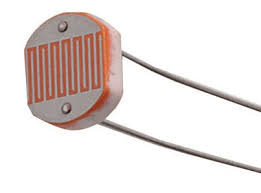
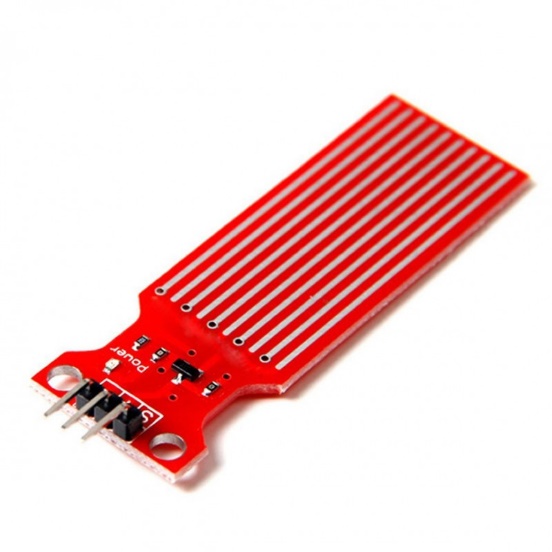
Del análisis anterior, consideramos que la mejor opción era puentear los pines de RE y DE y de esta forma con un pin digital del NodeMCU controlar si se quiere transmitir (pin en alto) o si se quiere recibir información (pin en bajo).

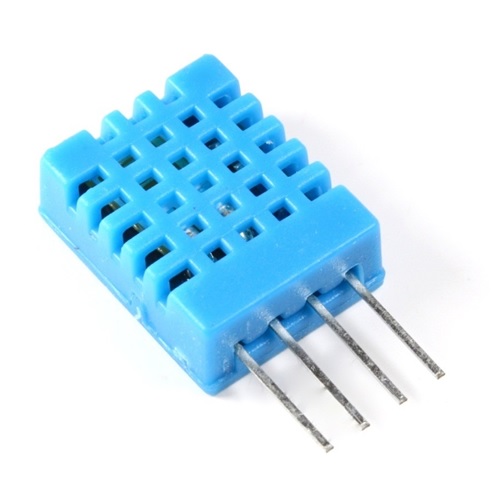
Seguidamente vemos la red RS485 creada. Se debe remarcar que esta red es altamente escalable.



Luego tenemos la parte de los Slave

Para esto se decidió utilizar dos placas Arduino NANO basadas en el microcontrolador ATmega 328p. Al Slave1 se le conectó un sensor LDR y un sensor de nivel de agua.



Al Slave2 se le adicionó un sensor DHT11 para medir la humedad relativa del ambiente. 

Estos Slave leerán y enviarán los valores de sus respectivos sensores cuando el Master se los solicite, para poder mostrar esta información tanto en la página web creada para este fin, como así también en la pantalla LCD conectada al NodeMCU. Todo esto se detallará en la siguiente sección.

A continuación, se explicará, de manera general, el funcionamiento de las distintas partes del código.

|  |
| --- |
| **Master.ino** |
| #include <ESP8266WiFi.h>  #include <WiFiClient.h>  #include <ESP8266WiFiMulti.h>  #include <ESP8266WebServer.h>  #include "html/index\_html.h"  #include <ArduinoJson.h>  #include <Wire.h>  #include <LiquidCrystal\_I2C.h>  ESP8266WiFiMulti wifiMulti; *// Instancia para conectarse al WiFi*  ESP8266WebServer server(80); *// Instancia de la clase WebServer que responde a peticiones HTTP en el puerto 80*  LiquidCrystal\_I2C lcd(0x27, 16, 2); *// Pantalla LCD de 16x2 en la Dirección I2C 0x27*  const int EnTxPin = 0; *// D3 - HIGH:TX y LOW:RX*  const int slave1 = 101; *// Dirección del esclavo 1*  const int slave2 = 102; *// Dirección del esclavo 2*  bool act = false; *// True: Pedir datos y False: No pedir datos*  String slave = "1"; *// El próximo esclavo a leer*  String sensor1; *// Valor en el sensor 1 de esclavo leído*  String sensor2; *// Valor en el sensor 2 de esclavo leído*  unsigned long act\_time; *// Contador para saber cuando leer* |

Primero se cargan todas las librerías a utilizar, la mayoría son relacionadas con la comunicación WiFi, también se cargan librerías para el control del display y se incluyen los archivos html para el diseño de la página web.  
Luego se instancian los objetos a utilizar y se declaran las variables necesarias, como las direcciones de los esclavos, variables para indicar si se va a leer o escribir por el puerto, datos recibidos de los esclavos, estado de la comunicación y tiempo de actualización.

|  |
| --- |
| **Master.ino - setup** |
| void setup(void)  {  *// Comunicación Serial*  Serial.begin(9600);  Serial.setTimeout(100); *//establecemos un tiempo de espera de 100ms*  *// inicialización de los pines*  pinMode(EnTxPin, OUTPUT);  digitalWrite(EnTxPin, HIGH); *//RS485 como Transmisor*  *// LCD*  lcd.init(); *// initialize the lcd*  lcd.backlight();  lcd.setCursor(0, 0);  lcd.print("Esclavo:");  lcd.setCursor(0, 1);  lcd.print("S1:");  lcd.setCursor(8, 1);  lcd.print("S2:");  *// Configuracón del servidor*  wifiMulti.addAP("nombre\_red", "contraseña"); *// Elegir a que red WiFi conectarse*  while (wifiMulti.run() != WL\_CONNECTED) *// Se conecta a la red*  delay(250);  server.on("/", HTTP\_GET, handleRoot); *// Se llama "handleRoot" cuando se solicita la URL "/"*  server.onNotFound(handleNotFound); *// Si se solicita una URL desconocida se llama a "handleNotFound"*  server.begin(); *// Inicializar el servidor*  act\_time = millis(); *// Inicializamos el contador*  } |

En la función setup se inicializa y escribe en la pantalla LCD, además se conecta a la red WiFi y se inicializa el servidor web.

|  |
| --- |
| **Master.ino - loop** |
| void loop(void)  {  server.handleClient(); *// Se escuchan las peticiones HTTP de los clientes*  if (act == true && act\_time + 3000 == millis() && slave == "1")  {  sendSlave(slave1); *// Se piden datos al esclavo 1*  readSlave(slave, "%", "LX"); *// Se leen los datos del esclavo 1 y se muestran por pantalla*  slave = "2"; *// El próximo esclabo a leer es el 2*  act\_time = millis(); *// Se actualiza el contador*  }  else if (act == true && act\_time + 3000 == millis() && slave == "2")  {  sendSlave(slave2); *// Se piden datos al esclavo 2*  readSlave(slave, "%", "LX"); *// Se leen los datos del esclavo 2 y se muestran por pantalla*  slave = "1"; *// El próximo esclabo a leer es el 1*  act\_time = millis(); *// Se actualiza el contador*  }  } |

Función loop donde se contesta a los clientes que realicen peticiones HTTP a nuestro servidor y se leen los datos de los esclavos.

|  |
| --- |
| **Master.ino - handleRoot** |
| void handleRoot()  { *// Las URL que se pueden solicitar junto con los verbos HTTP y la función a la que llaman*  server.send(200, "text/html", INDEX\_HTML);  server.on("/act\_estado", HTTP\_POST, post\_act\_estado);  server.on("/consultar", HTTP\_GET, get\_consultar);  } |

Función con el conjunto de URLs disponibles que pueden ser solicitadas por los clientes.

|  |
| --- |
| **Master.ino - handleRoot** |
| void post\_act\_estado()  { *// Actualiza el estado para leer o no a los esclavos*  StaticJsonDocument<500> jsonDoc;  auto error = deserializeJson(jsonDoc, server.arg("plain"));  if (error)  server.send(400);  else if (server.method() == HTTP\_POST)  {  if (jsonDoc["msg"] == "iniciar")  {  act = true;  act\_time = millis();  }  else if (jsonDoc["msg"] == "detener")  act = false;  server.sendHeader("Location", "/act\_estado/");  server.send(200);  }  } |

Un cliente puede solicitar que se inicie o se detenga la lectura de datos por parte del maestro, esto depende de si el cliente envía el mensaje *“inciar”* en cuyo caso el maestro lee un esclavo cada 3 segundos aproximadamente o si el mensaje es *“detener”* para que el maestro deje de solicitar datos.

|  |
| --- |
| **Master.ino - get\_consultar** |
| void get\_consultar()  { *// Se envía al cliente la información sobre el último sensor leido*  String jsonString;  DynamicJsonDocument jsonDoc(1024);  if (slave == "1")  jsonDoc["slave"] = "2";  if (slave == "2")  jsonDoc["slave"] = "1";  jsonDoc["sensor1"] = sensor1;  jsonDoc["sensor2"] = sensor2;  serializeJson(jsonDoc, jsonString);  server.send(200, "application/javascript", jsonString);  } |

Esta función se ejecuta cuando un cliente solicita datos, lo que se hace es enviar, en formato JSON, la información sobre que esclavo se ha leído, y los valores de los sensores del mismo.

|  |
| --- |
| **Master.ino - handleNotFound** |
| void handleNotFound()  {  server.send(404, "text/plain", "404: Not found"); *// Si se solicita una URL desconocida se muestra un error 404*  } |

En el caso de que se solicite una URL no definida se redireccionará a una página con el mensaje “404: Not Found”

|  |
| --- |
| **Master.ino - sendSlave** |
| void sendSlave(int *slaveDir*)  { *// Se le solicita al esclavo información sobre los sensores 1 y 2*  Serial.print("<"); *// Inicio de trama*  Serial.print(*slaveDir*); *// Dirección del esclavo*  Serial.print("S1"); *// Datos del sensor 1*  Serial.print("S2"); *// Datos del sensor 2*  Serial.print(">"); *// Fin de trama*  Serial.flush(); *// Esperamos hasta que se envíen los datos*  } |

Cuando el maestro quiere pedir datos a alguno de sus esclavos ejecuta esta función donde se envía un mensaje a través de la conexión RS485 solicitando a un esclavo en específico mediante su dirección la información sobre los sensores 1 y 2. Por ejemplo, si se deseara pedir la información sobre los sensores 1 y 2 al esclavo con la dirección 101 el mensaje sería el siguiente:

<101S1S2>

Donde:

< Es el carácter de inicio de mensaje.

101 Es la dirección del esclavo.

S1 Solicita información sobre el sensor 1.

S2 Solicita información sobre el sensor 2.

> Es el carácter de fin del mensaje.

|  |
| --- |
| **Master.ino - readSlave** |
| void readSlave(String *slaveNumber*, String *S1\_unit*, String *S2\_unit*)  { *// Se leen los datos de los sensores del esclavo, se almacenan y se muestran en la pantalla LCD*  digitalWrite(EnTxPin, LOW); *//RS485 como receptor*  delay(500);  if (Serial.available()) *// Si hay datos nuevos*  {  String msj = Serial.readString();  if (msj.startsWith("<100") && msj.endsWith(">")) *// Si la respuesta se recibió bien*  {  msj = msj.substring(4, msj.length() - 1); *// Eliminamos datos no necesarios*  String sensorData;  lcd.setCursor(8, 0);  lcd.print(*slaveNumber*);  while (msj.length() != 0) *//Se leen, almacenan y muestran los datos*  {  if (msj.charAt(0) != 'S')  sensorData += msj.charAt(0);  else  {  lcd.setCursor(3, 1);  lcd.print(" ");  lcd.setCursor(3, 1);  lcd.print(sensorData);  lcd.print(*S1\_unit*);  sensor1 = sensorData + *S1\_unit*;  sensorData = "";  msj = msj.substring(1, msj.length());  }  msj = msj.substring(1, msj.length());  }  lcd.setCursor(11, 1);  lcd.print(" ");  lcd.setCursor(11, 1);  lcd.print(sensorData);  lcd.print(*S2\_unit*);  sensor2 = sensorData + *S2\_unit*;  }  digitalWrite(EnTxPin, HIGH); *//RS485 como Transmisor*  }  } |

Luego de solicitar los datos el maestro ejecuta esta función para leer la respuesta del esclavo. Lo primero que hace cambiar a modo receptor, para que no haya problemas cuando el esclavo trate de enviar su respuesta, luego se esperan 500ms para dar tiempo a que el esclavo pueda leer el mensaje, tomar los datos y enviar la respuesta. Pasado este tiempo el maestro comprueba si hay información disponible y de ser así la lee, comprueba que el mensaje haya llegado correctamente, es decir que los inicios y fin de trama sean correctos y que la dirección del destinatario sea la suya. De ser esto correcto se extrae la información sobre los datos de los sensores, se almacenan (por si algún cliente los solicita) y se muestran en la pantalla LCD.   
Finalmente, el maestro se vuelve a configurar como emisor, para poder enviar datos nuevamente cuando sea necesario.

Los códigos utilizados para los esclavos son muy similares entre sí solo varía la dirección de cada uno y la forma de leer algún sensor, por esto es por lo que decidimos explicar solamente el código para el esclavo 1. Si se desea verse el código completo para el esclavo 2, este se encuentra en el archivo slave\_2.ino en la carpeta Slave\_2

|  |
| --- |
| **slave\_1.ino** |
| #define LIGHT\_CONVERSION(*value*) 2e7 \* pow(value, -1.449) *// Macro para convertir de valor de resistencia a valor de luz en LUX*  #define ADC\_TO\_RESISTANCE(*adc*) -(1e5 \* adc) / (adc - 1023) *// Macro para convertir de valor de adc a resistencia*  const int enTxPin = 2; *// HIGH:TX y LOW:RX*  const int myDir = 101; *// Dirección del esclavo*  const int waterSensorPin = A0; *// Pin del sensor*  const int lightSensorPin = A1; *// Pin del sensor* |

Definimos dos macros, una para convertir la lectura del adc a un valor de resistencia y otra para convertir un valor de resistencia a un valor equivalente en LUX. La ecuación para calcular el valor de resistencia a partir de la lectura del adc no es más que la fórmula del divisor resistivo y la conversión equivalente para un adc de 10 bits.  
Para obtener la ecuación del valor de luz en base a un valor de resistencia, debimos realizar varias mediciones comparando el valor de un luxómetro (aplicación de celular) con el valor de resistencia que obteníamos al incidir más o menos luz en el LDR. Finalmente, con un software como Excel podemos hacer varias aproximaciones y buscar la que mejor se adapte a nuestra curva.   
En esta sección del código también se definen variables globales para definir si el esclavo va a estar en modo lectura o escritura, la dirección del esclavo y los pines de los sensores.

|  |
| --- |
| **slave\_1.ino - setup** |
| void setup()  {  *// Comunicación Serial*  Serial.begin(9600);  Serial.setTimeout(100); *//establecemos un tiempo de espera de 100ms*  *// inicialización de los pines*  pinMode(enTxPin, OUTPUT);  digitalWrite(enTxPin, LOW); *//RS485 como receptor*  pinMode(waterSensorPin, INPUT);  pinMode(lightSensorPin, INPUT);  } |

En la función setup se inicializa el puerto serial y se configuran los pines del esclavo.

|  |
| --- |
| **slave\_1.ino - loop** |
| void loop()  {  if (Serial.available()) *// Si hay datos nuevos*  {  *// Se lee una línea*  String msj = Serial.readString();  if (msj.startsWith(String("<" + String(myDir))) && msj.endsWith(">")) *//Si el inicio y fin del mensaje son correctos y la dirección corresponde a este esclavo*  {  msj = msj.substring(4, msj.length() - 1); *// Se descarta la información que no sea de los sensores a leer*  delay(30);  digitalWrite(enTxPin, HIGH); *//RS485 como emisor*  Serial.print("<100");  while (msj.length() != 0)  {  if (msj.charAt(0) == 'S') *// Se lee todos los sensores solicitados*  {  switch (msj.charAt(1))  {  case '1':  Serial.print("S1");  Serial.print(waterLvlConversion(analogRead(waterSensorPin))); *// Se lee el sensor, se adapta el valor y se envía*  break;  case '2':  Serial.print("S2");  int light = ADC\_TO\_RESISTANCE(analogRead(lightSensorPin));  light = LIGHT\_CONVERSION(light);  Serial.print(light); *// Se lee el sensor, se adapta el valor y se envía*  break;  }  msj = msj.substring(2, msj.length());  }  else  msj = "";  }  Serial.print(">");  Serial.flush(); *//Esperamos hasta que se envíen los datos*  digitalWrite(enTxPin, LOW); *//RS485 como receptor*  }  }  } |

En la función loop el esclavo revisa si hay información entrante por el puerto serie y la lee, corrobora que el mensaje sea correcto y que él sea el destinatario, en caso de cumplirse estas dos condiciones se configura para poder escribir por el pin de TX, seguidamente lee el mensaje recibido carácter por carácter para identificar sobre que sensor se pide información, luego lee el sensor correspondiente, se hacen las conversiones necesarias y se envía el mensaje. Por ejemplo, si el maestro solicita información sobre los sensores 1 y 2 la respuesta del esclavo sería similar a lo siguiente:

<100S125S2192>

Donde:

< Es el carácter de inicio de mensaje.

100 Es la dirección del maestro.

S1 Lo que sigue a continuación es la información del sensor 1.

25 Valor leído del primer sensor (varía según el sensor).

S2 Lo que sigue a continuación es la información del sensor 2.

192 Valor leído del segundo sensor (varía según el sensor).

> Es el carácter de fin del mensaje.

|  |
| --- |
| **slave\_1.ino - waterLvlConversion** |
| int waterLvlConversion(int *adc*)  { *// Función para convertir el valor analógico leido a una valor porcentual*  if (*adc* >= 730)  return 100;  else if (*adc* >= 700)  return 75;  else if (*adc* >= 620)  return 50;  else if (*adc* >= 500)  return 25;  else  return 0;  } |

Esta función es exclusiva del esclavo 1, lo que hace es convertir el valor del adc leído del sensor de nivel de agua a un valor porcentual de 0%, 25%, 50%, 75% o 100%. Estos valores fueron tomados mediante ensayos experimentales y son aproximados ya que en distintas iteraciones del ensayo se obtuvieron valores distintos para niveles de agua similares.