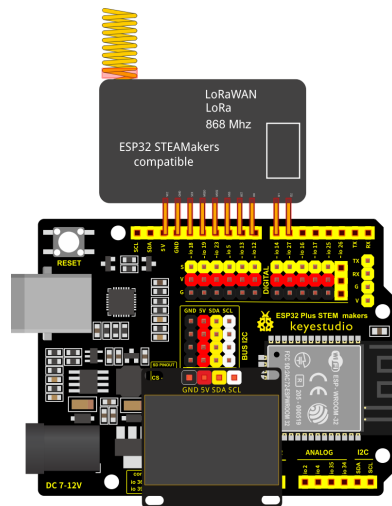


LoRa y LoRaWAN

Arduinoblocks + ESP32 STEAMakers con



Autor: Juanjo López

Índice

LoRa y LoRaWAN	4
Módulo LoRa para ESP32 STEAMakers	8
LoRa con arduinoblocks	10
Bloques LoRa.....	11
JSON formato para intercambio de datos	15
Ejemplos LoRa	19
LoRa 1: Emisor contador + Receptor OLED.....	19
LoRa 2: Emisor T/H + Receptor OLED.....	20
LoRa 3: Timbre remoto.....	22
LoRa 4: Apertura de puerta remota.....	24
LoRa 5: Sensores distribuidos de T/H (maestro-esclavos).....	26
LoRa 6: Datalogger remoto.....	29
LoRa 7: Central de alarma y sensores de presencia.....	31
LoRa 8: Gateway LoRa <-> Telegram.....	33
LoRa 9: Control RGB remoto.....	36
LoRaWAN con arduinoblocks	38
Bloques LoRaWAN:.....	40
Ejemplos LoRaWAN	42
LoRaWAN 1: Estación meteorológica.....	42
LoRaWAN 2: Medidor de luz ambiental.....	44
LoRaWAN 3: Pulsador de emergencia.....	45
LoRaWAN 4: Detector de ocupación para plazas de aparcamiento.....	46
LoRaWAN 4: Geolocalización GPS en tiempo real.....	47
Información y enlaces:	49

LoRa y LoRaWAN

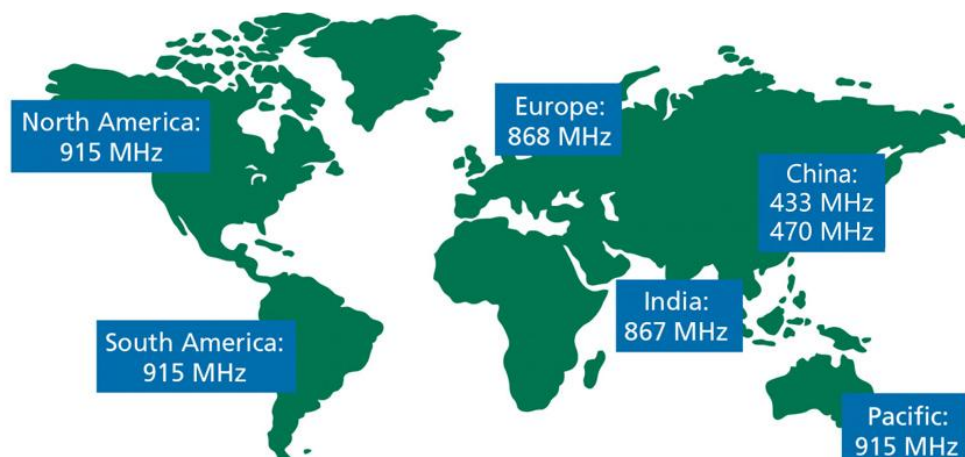
LoRa (Long Range) es una tecnología de comunicación inalámbrica diseñada para transmitir datos a largas distancias con un consumo de energía muy bajo. Permite enviar pequeños paquetes de información, como temperatura o ubicación, de un dispositivo a otro, incluso cuando están bastante separados, como a kilómetros de distancia. Es ideal para aplicaciones como monitoreo de sensores remotos o seguimiento de activos.



La tecnología se desarrolló en Francia en el 2012 por Cycleo y adquirida posteriormente por Semtech que se encargaría de impulsarla. **Usa un tipo de modulación de amplio espectro**, ideal para tolerar el ruido y para que una señal realice caminos múltiples..

Para utilizar LoRa solo necesitamos al menos dos dispositivos con un módulo LoRa para enviar y recibir información entre ellos.

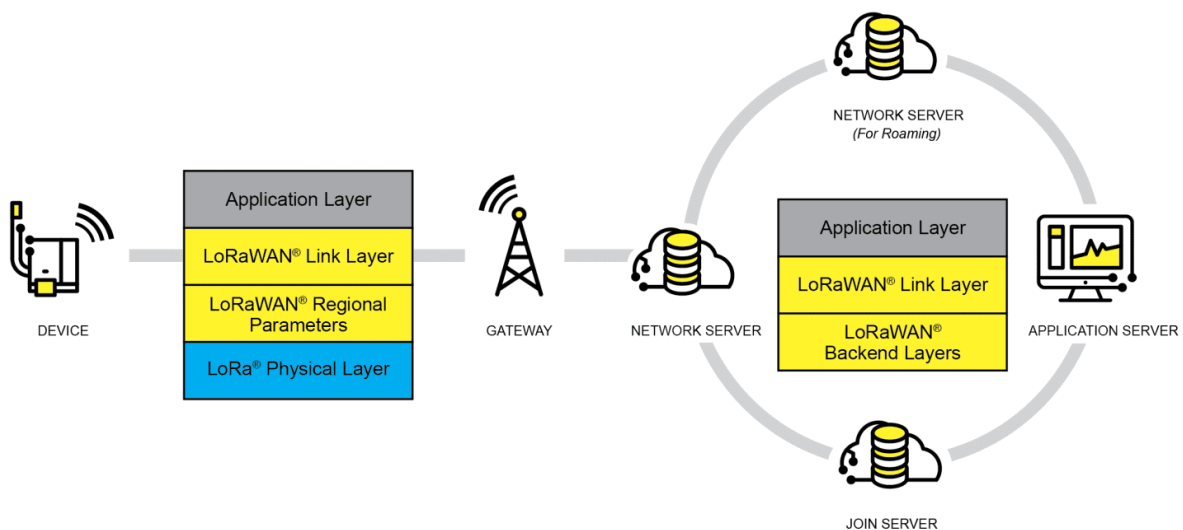
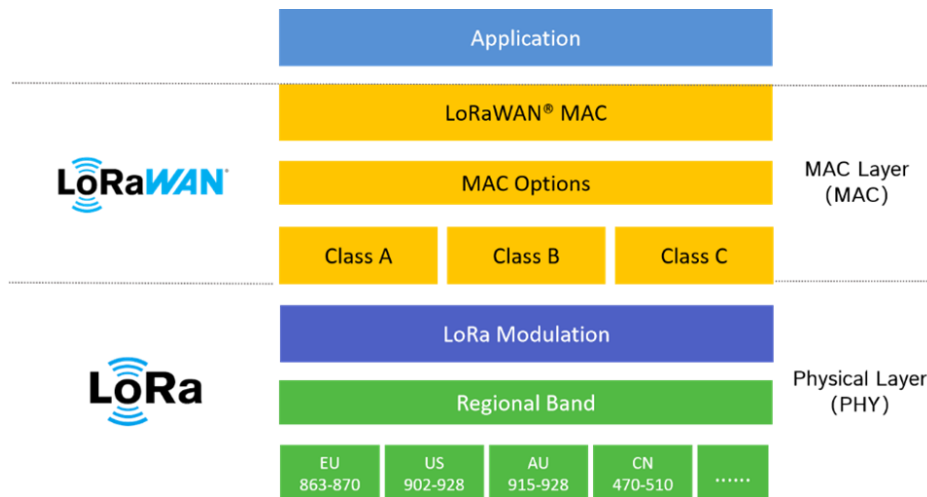
Las frecuencias de funcionamiento varían dependiendo del país y suelen ser las bandas **433 MHz, 868 MHz y 915 MHz**.



LoRaWAN (Long Range Wide Area Network), por otro lado, es un protocolo de red que utiliza la tecnología LoRa para conectar múltiples dispositivos a una red central. Imagina que LoRa es como el camino por el que viajan los datos y LoRaWAN es la autopista que organiza y gestiona ese tráfico de datos. LoRaWAN permite que muchos dispositivos LoRa se comuniquen de manera eficiente con una estación base o una red central, lo que es útil para casos de uso como ciudades inteligentes o agricultura de precisión.

Para utilizar LoRaWAN, necesitas algunos componentes y seguir algunos pasos clave:

1. **Dispositivos LoRaWAN:** Necesitarás dispositivos o sensores que estén equipados con tecnología LoRa para capturar datos y enviarlos a través de la red LoRaWAN. Estos dispositivos pueden variar según tu aplicación, como sensores de temperatura, dispositivos de seguimiento GPS o medidores de humedad, entre otros.
2. **Gateway LoRaWAN:** Debes tener al menos un gateway LoRaWAN. Este es un dispositivo que recibe señales de los dispositivos LoRa y las transmite a la red LoRaWAN. Los gateways suelen estar conectados a Internet y aportan la conectividad a la red global.
3. **Plataforma de Gestión y Aplicaciones:** Utilizarás una plataforma de gestión y aplicaciones para administrar tus dispositivos LoRaWAN y los datos que recopilan. Estas plataformas te permiten configurar y monitorear tus dispositivos, así como recibir y analizar los datos que generan.
4. **Configuración y Programación:** Los dispositivos LoRaWAN necesitarán ser configurados y programados para enviar datos a través de la red. Esto a menudo implica el uso de software o herramientas de desarrollo específicas para tu aplicación.



Conexión de dispositivos

Un solo gateway permite conectar miles de dispositivos, una cifra que aumenta con cada nueva puerta que se añade. Así, se facilita la creación de una red barata, amplia y con un consumo energético bajo. Los aparatos ligados mandarían sus datos en tiempo real y durante largos periodos. Para ello, existen diferentes formas de cobertura LoRawan: redes privadas, redes de terceros y operadoras de dispositivos. Además, estas características ayudan a escalar la arquitectura.

El proceso de escalado se efectúa añadiendo módulos a todo el sistema. Esto hace posible la conexión de millones de sensores que se siguen mediante geolocalización. Cada puerta de enlace se encarga de gestionar miles de dispositivos, toda una ayuda para aumentar la capacidad de la red.

Largo Alcance

Esta es una de las características más destacadas por las implicaciones que conlleva. La tecnología está diseñada para funcionar en ciudades y atravesar todo tipo de obstáculos. En presencia de numerosos edificios y construcciones, dispone de un alcance de tres kilómetros. Sin embargo, **la capacidad aumenta a los 20 km en zonas abiertas.**

El parámetro SF (Spreading Factor) permite variar el factor de “*esparcimiento*” que varía la velocidad y el alcance de la emisión.

Valores con ancho de banda de 125Khz (por defecto)

Spreading Factor	Data Rate	Range	Time on-Air
SF7	5470 bps	2 km	56 ms
SF8	3125 bps	4 km	100 ms
SF9	1760 bps	6 km	200 ms
SF10	980 bps	8 km	370 ms
SF11	440 bps	11 km	40 ms
SF12	290 bps	14 km	1400 ms

<https://www.thethingsnetwork.org/docs/lorawan/spreading-factors/>

<https://docs.arduino.cc/learn/communication/lorawan-101>

Transmisión de datos baja

La velocidad de transmisión de datos es baja, pero no es importante para la mayoría de procesos. La información se envía desde miles de dispositivos de forma continuada. Pese a que la **velocidad es de entre 0,3 y 50 Kbps**, es suficiente para enfrentarse al volumen generado por la red.

Bajo consumo energético y coste

Permite mantener dispositivos con baterías pequeñas sin necesidad de recargarlos durante largos periodos. Esta característica **reduce la intervención humana**, así como las necesidades energéticas. No importa que la red se extienda a través de decenas de kilómetros, ya que estará optimizada desde un punto de vista energético. De este modo, es más sencillo escalarla sin incurrir en grandes costes.

Amplia variedad de dispositivos y aplicaciones

Los dispositivos que se pueden emplear son muy variados y **dependen del objetivo de la red. Ayudan a adelantarse a problemas en ciernes y prevenirlos de manera adecuada**. Algunos ejemplos son los sensores de humedad, temperatura o de conductividad eléctrica.

Estas son sus principales aplicaciones:

- *Smart cities.*
- *Smart industry.*
- *Agricultura.*
- *Logística.*
- *Cadena de suministro.*

Seguridad

La seguridad de la red es primordial en despliegues IoT y LoRaWAN define dos capas de criptografía. Una única clave de sesión de red de 128bit compartida por el end-point y servidor de red. Una única clave (AppSKey) de aplicación de 128bit compartida end-to-end y la capa de aplicación

La regla del 1%

En Europa la banda 868 MHz tiene un par de limitaciones. La primera es la potencia de emisión: 25 mW, que no nos preocupa mucho ya que los módulos que utilizaremos no llegan a tanta. Pero la segunda si que es importante y debemos evitar violar la regulación.

Esta regla simplificada lo que dice es que no podemos transmitir más que el 1% del tiempo. Es decir si por ejemplo enviar un paquete nos lleva 100 ms, tendremos que mantenernos sin emitir por 900 ms. Como norma general, no envíes más de una vez cada 3 minutos y cumplirás la reglamentación. Si quieres enviar más a menudo, existen calculadoras para obtener los tiempos entre envío mínimos.

Calculate the air time of your LoRa frame.
Default values are for EU868 band.

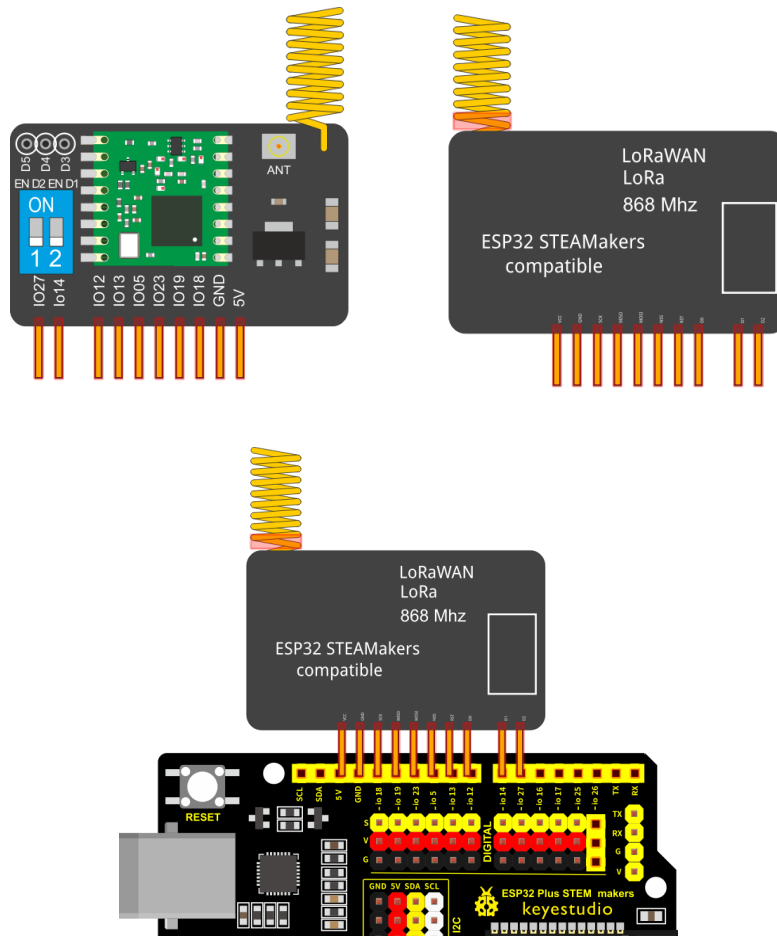
LoRa Modem settings		
Spreading factor	7	7 - 12
Bandwidth	125 kHz	125 kHz default for LoRaWAN. 250 kHz also supported.
Code rate	4	4 / (CR + 4) = 4/8. 4/5 default for LoRaWAN
Frame configuration		
Payload length	4 bytes	
Preamble length	8 symbols	Default for frame = 8, beacon = 10.
Explicit header	<input checked="" type="checkbox"/> Yes	Default on for LoRaWAN
CRC	<input checked="" type="checkbox"/> Yes	Default on for LoRaWAN

<https://loratools.nl/#/airtime>

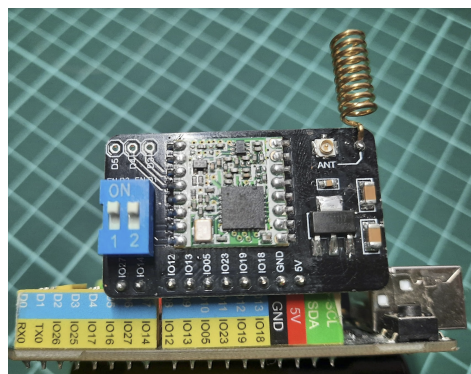
Módulo LoRa para ESP32 STEAMakers

Para utilizar la comunicación LoRa desde ESP32 STEAMakers debemos añadir un módulo capaz de trabajar con la modulación LoRa, para ello utilizaremos un módulo basado en el chip de Semtech SX1276 (módulo RFM95)

El módulo utiliza conexión SPI y algunas señales de control extra dependiendo de si utilizamos LoRa o LoRaWAN en nuestro dispositivo. Para facilidad del conexionado se ha diseñado un módulo específico listo para conectar a la placa ESP32 STEAMakers directamente:



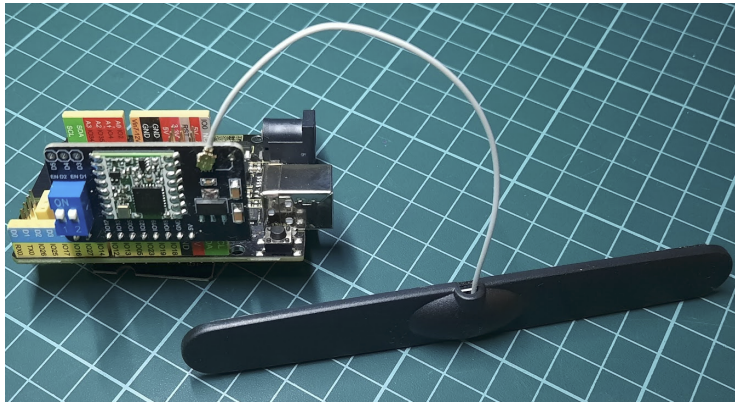
Vista real del módulo LoRa/LoRaWAN conectado a ESP32 STEAMakers:



Antena externa:

El módulo permite conectar una antena externa mediante el conector IPX. En caso de utilizar antena externa se debe quitar la antena integrada. La antena externa debe estar diseñada para trabajar en la frecuencia correcta (868 Mhz, 915 Mhz, etc.)

Módulo LoRa con antena externa para 868Mhz:

**Configuración del módulo:**

El módulo permite habilitar los 2 pines extras D1 (io14) y D2 (io27).

En caso de utilizar LoRa (sin WAN) podemos deshabilitar los dos pines y así utilizar esos pines con otros usos, aunque si los dejamos a ON funcionará el modo LoRa y esos pines debemos no conectarlo a nada.

Por otro lado, si usamos LoRaWAN obligatoriamente el módulo debe tener los dos microinterruptores en posición ON y no podremos conectar ningún sensor a esos pines.

	Modo LoRaWAN	D1 = ON	D2 = ON
	Modo LoRa <i>(dejando libre los pines io27, io14 para otras conexiones)</i>	D1 = OFF	D2 = OFF

Recomendación: Deja siempre los microinterruptores a ON y en caso de necesitar pines extras y utilizar solo conexión tipo LoRa puedes ponerlos a OFF para habilitarlos.

**ESP32 STEAMakers y Módulo LoRa:*

<https://shop.innovadidactic.com/es/standard-perifericos/1663-modulo-lora-lorawan-8436574314656.html>

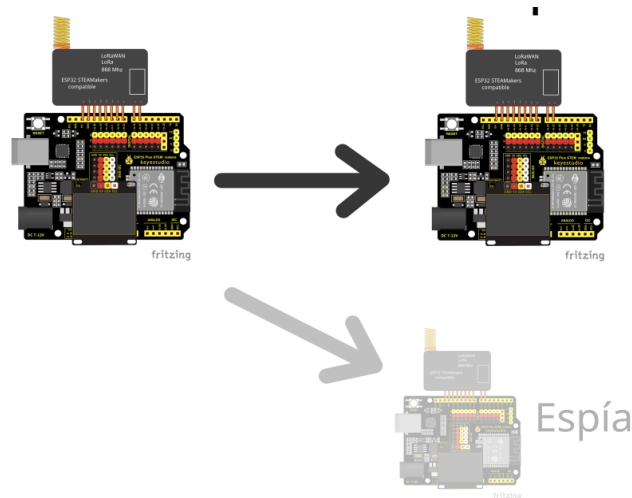
<https://shop.innovadidactic.com/es/standard-placas-shields-y-kits/1567-placa-esp32-steamakers-no-incluye-cable-usb.html>

LoRa con arduinoblocks

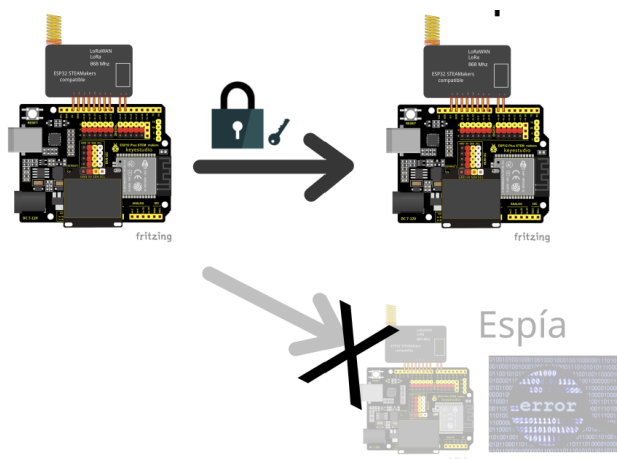
Los bloques LoRa permiten trabajar con comunicación LoRa entre dispositivos, realizando conexiones punto a punto. En realidad la emisión de datos LoRa de esta forma es captada por cualquier dispositivo LoRa dentro del alcance y que trabaje con los mismos parámetros de frecuencia y ajustes internos del módulo LoRa, por eso la importancia de la cifrar los datos para que no sean capturados por cualquier otro módulo y puedan espiar la información.

En este modo tampoco hay un control de acceso al medio ni validación de los datos (ACK) , todo ese proceso lo debemos supervisar nosotros mismos. Lo habitual son sistemas simples de emisor-receptor, o un modelo maestros-esclavos donde un dispositivo inicia siempre la comunicación pidiendo la información a uno de los esclavos que contestará cuando el maestro le requiera.

(En caso de sistemas complejos con multitud de dispositivos se recomienda implementar o utilizar una red LoRaWAN donde los gateways y servidores de aplicación gestionan la comunicación, la seguridad, el encriptado de datos y otros aspectos automáticamente)

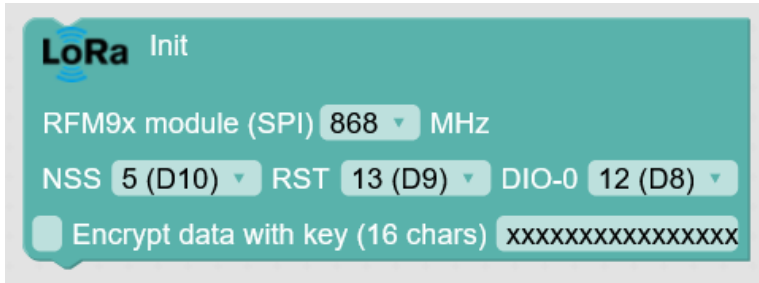


Añadiendo cifrado de datos en emisor-receptor:



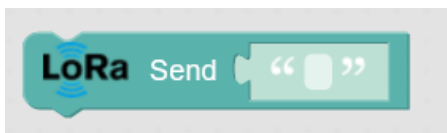
Bloques LoRa

Init: Configura el módulo LoRa (ajustado por defecto para el módulo ESP32 STEAMakers LoRa). Si seleccionamos la opción de encriptación se debe especificar la misma clave de 16 caracteres en el emisor y receptor (clave de encriptación con caracteres alfabéticos).



8

Send: Envía una trama de datos (máxima longitud 51 ~ 242 bytes/ caracteres)



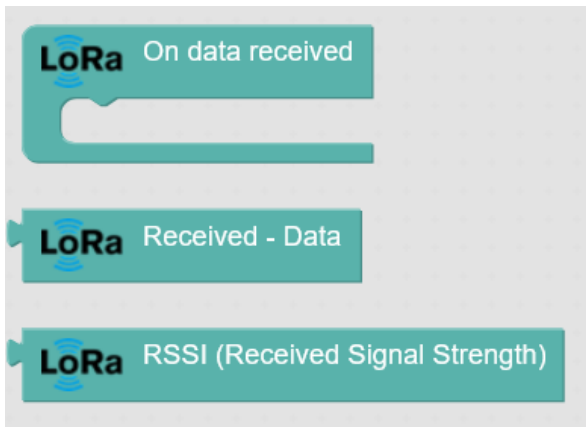
La longitud máxima de la trama de datos dependerá de la frecuencia (zona del mundo) y de parámetros internos de la modulación (SF, Bandwith, etc.)

Ejemplo de longitud máxima de la trama de datos según varias configuraciones:

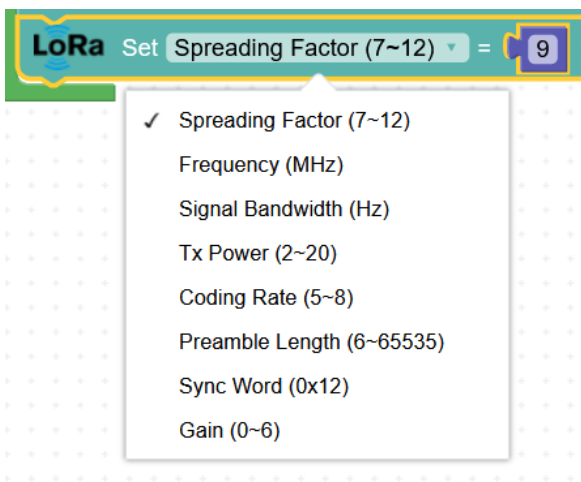
Data Rate	Configuration (SF+BW)	Maximum application payload size (bytes)
0	LoRa: SF12 / 125 kHz	51
1	LoRa: SF11 / 125 kHz	51
2	LoRa: SF10 / 125 kHz	51
3	LoRa: SF9 / 125 kHz	115
4	LoRa: SF8 / 125 kHz	242
5	LoRa: SF7 / 125 kHz	242
6	LoRa: SF7 / 250 kHz	242

<https://www.thethingsnetwork.org/docs/lorawan/regional-parameters/>

On Data Received: Evento que se produce cuando se reciben datos. Dentro de ese bloque de evento el valor del bloque “**Received - Data**” tendrá el texto con la trama de datos recibida y el valor **RSSI** indicando la intensidad de la señal recibida.



El bloque de ajuste de parámetros avanzados permite ajustar algunos valores internos:



Spreading Factor (SF): es el factor de esparcimiento, permite valores entre 7 y 12.

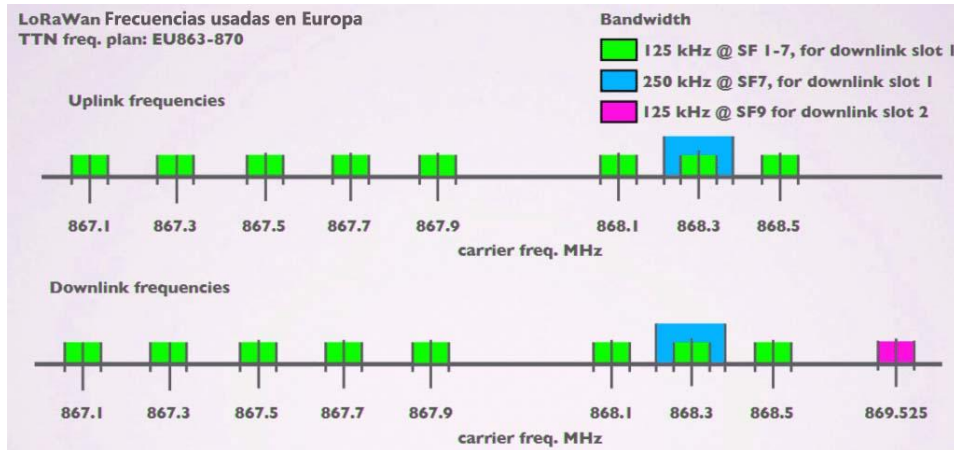
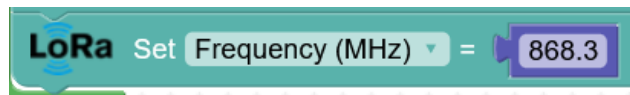
(Valor por defecto = 7)

Spreading Factor	Data Rate	Range	Time on-Air
SF7	5470 bps	2 km	56 ms
SF8	3125 bps	4 km	100 ms
SF9	1760 bps	6 km	200 ms
SF10	980 bps	8 km	370 ms
SF11	440 bps	11 km	40 ms
SF12	290 bps	14 km	1400 ms

Frequency (MHZ): valor en Mhz en la que trabaja el módulo LoRa. Podemos especificar una banda de frecuencia específica dentro de los márgenes de nuestro módulo LoRa.

(Valor por defecto = 868)

Por ejemplo si estamos en zona Europea con 868MHz, podemos variar entre: 867.0 y 869.0 MHz



Signal Bandwith: Ancho de banda de la señal LoRa. Valores posibles:

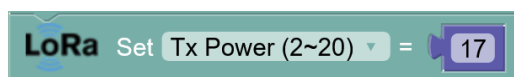
(Valor por defecto = 125000)

7800	10400	15600	20800	312500	41700	62500	125000	250000	500000
------	-------	-------	-------	--------	-------	-------	---------------	--------	--------



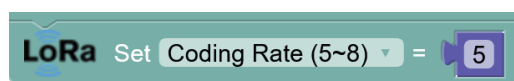
Tx Power: Potencia de transmisión del módulo LoRa. Valor entre 2 y 20 dB.

(Valor por defecto = 17)



Coding Rate: Los valores admitidos están entre 5 y 8, estos corresponden a tasas de codificación de 4/5 y 4/8. El numerador de la tasa de codificación se fija en 4.

(Valor por defecto =5)



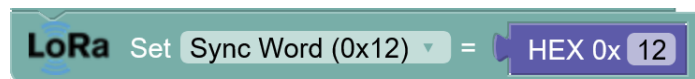
Preamble Length: Los valores admitidos están entre 6 y 65535.

(Valor por defecto = 8)



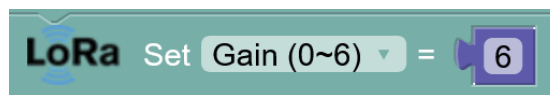
Sync Word: Valor de sincronización, por defecto 0x12

(Valor por defecto = 0x12)



Gain: Configura la ganancia de LNA para obtener una mejor sensibilidad de RX; de forma predeterminada, se usa AGC (control automático de ganancia) y no se usa la ganancia de LNA. Valores de 0 (ganancia automática) hasta el 6.

(Valor por defecto = 0 auto)



JSON formato para intercambio de datos

A la hora de intercambiar información entre dispositivos, y sobre todo en comunicaciones de baja frecuencia como el caso de LoRa, debemos enviar varios valores y de la forma más compacta posible.

El formato de empaquetado de datos "JSON" es una forma sencilla de encapsular varios datos de información en una misma trama de datos con un formato fácilmente entendible y que ocupa poco espacio extra en el envío (existen otros tipos como XML con mucha más carga de datos).

JSON es el acrónimo de "JavaScript Object Notation" y es un formato de texto sencillo que permite encapsular en forma de texto cualquier objeto (un objeto es un concepto dentro de la programación que encapsula las variables, tipos de datos y valores de un estado del programa o de la información necesaria, incluyendo jerarquía, listas, valores, tipos de datos, etc.)

Ejemplo JSON: tenemos en un sistema informático el concepto de objeto "vehículo", una posible representación en formato JSON sería:

<pre>{ "Matricula": "KLC1245", "Año": 2021, "Conductor": { "Principal": "Juan", "Secundario": "María", }, "Seguro": "El piñazo SL", "Gasolina": True, "Gasoil": False, "Eléctrico": False, "Revisiones": [2021, 2023,2024], }</pre>	<pre>{ "Matricula": "KLC1245", "Anyo": 2021, "Conductor": { "Principal": "Juan", "Secundario": "Maria" }, "Seguro": "El piñazo SL", "Gasolina": true, "Gasoil": false, "Eléctrico": false, "Revisiones": [2021, 2023, 2024] }</pre>
---	---

<https://www.freeformatter.com/json-formatter.html>

Para más información consultar la documentación oficial sobre JSON.

<https://es.wikipedia.org/wiki/JSON>

<https://www.youtube.com/watch?v=YYfediyCwAU>

El formato JSON de forma práctica en aplicaciones LoRa/LoRaWAN con ESP32 lo vamos a utilizar para encapsular dentro del bloque de datos ({ }) varios pares "clave" : "valor" y poder recuperarlos fácilmente en el receptor a partir del nombre "clave".

En la práctica con comunicaciones LoRa y ESP32 no utilizaremos estructuras complejas en formato JSON, simplemente es una forma rápida, sencilla y eficiente de codificar y decodificar varios valores (clave/valor) dentro de la misma trama de datos. Ejemplos prácticos:

JSON con datos para controlar un led RGB	{ "R": 100, "G":50, "B": 0 }
JSON con datos para mover dos servos (S1 , S2)	{ "S1": 120, "S2": 45 }
JSON con datos para enviar con un texto para mostrar en varias pantallas remotas (P1, P2)	{ "P1": "Hola", "P2": "Juanjo" }
JSON con datos de eléctrico	{ "V": 232.5, "I": 0.6, "Wh": 326.4, }

Ejemplo sencillo:

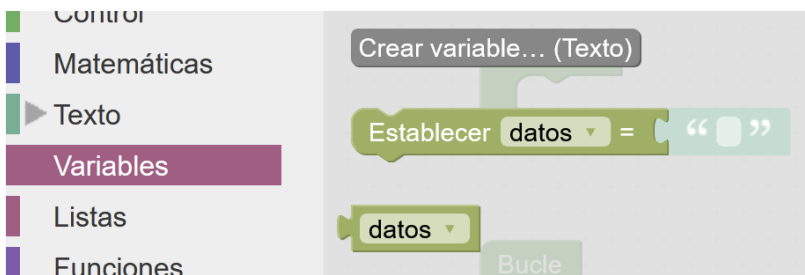
Datos a encapsular de 3 sensores conectados al emisor:

- 1) Valor de temperatura (*Temp*)
- 2) Valor de humedad (*Hum*)
- 3) Nivel de luz (*Luz*)

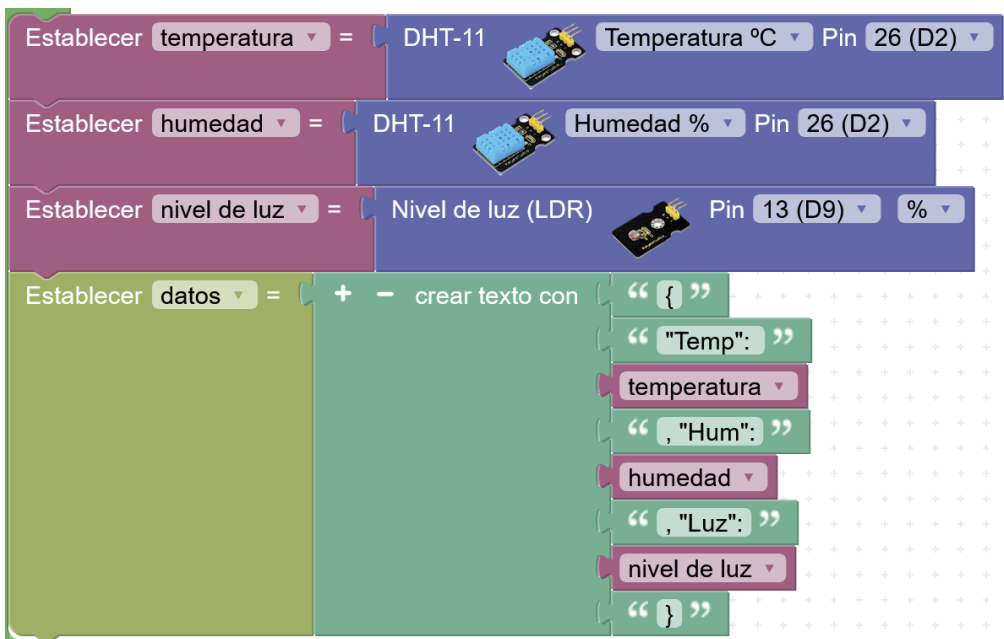
JSON de ejemplo para empaquetar estos 3 valores:

```
{ "Temp": 23.5, "Hum": 68.3, "Luz": 46 }
```

Creación del formato anterior en Arduinoblocks:



1) Opción manual uniendo textos y variables:



2) Opción con bloques para generar JSON de forma automática (*recomendable*)

The screenshot shows a sequence of blocks in an IDE. The first three blocks are 'Establecer' (Set) blocks for variables: 'temperatura' (DHT-11, Temperatura °C, Pin 26 (D2)), 'humedad' (DHT-11, Humedad %, Pin 26 (D2)), and 'nivel de luz' (Nivel de luz (LDR), Pin 13 (D9), %). The fourth block is 'Establecer datos' (Set data) which uses a 'Codificar {K,V}' (Encode {K,V}) block. This block has three entries: 'Temp' (Key) with 'temperatura' (Value), 'Hum' (Key) with 'humedad' (Value), and 'Luz' (Key) with 'nivel de luz' (Value).

LoRa y LoRaWAN permiten tramas de poca longitud, y además la longitud perjudica notablemente el tiempo de envío. Cuanto más ahorremos en espacio mejor, en este ejemplo recomendaría usar nombres "clave" lo más cortos posibles, simplificando lo anterior a "T", "H", "L":

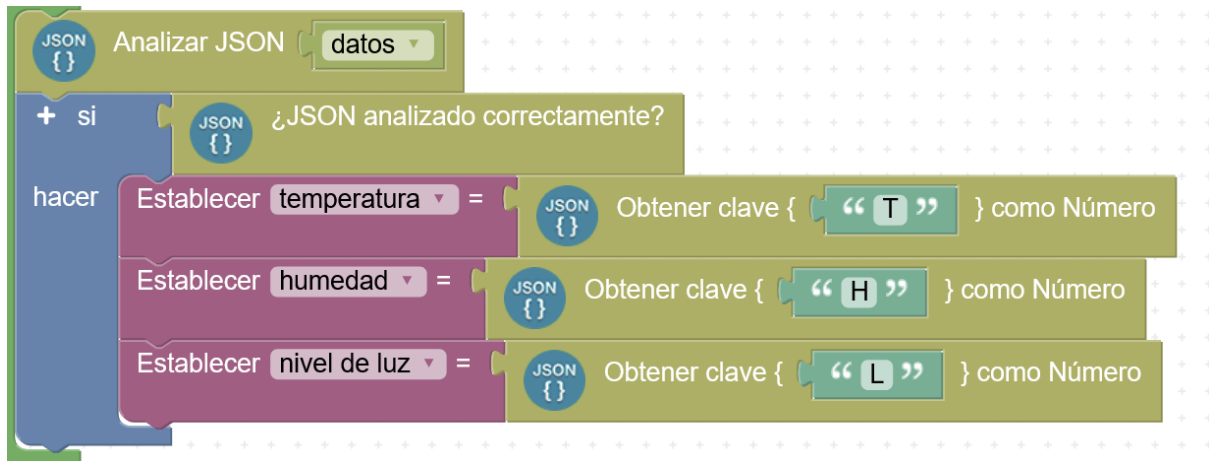
This screenshot is identical to the previous one, but the 'Codificar {K,V}' block now has three entries with single-character keys: 'T' (Key) with 'temperatura' (Value), 'H' (Key) with 'humedad' (Value), and 'L' (Key) with 'nivel de luz' (Value).

El resultado de la trama a enviar sería por ejemplo algo así (los retornos de carro y espacios no afecta, lo importante es mantener la estructura y ahorrar en número de caracteres/bytes a transmitir):

```
{"T": 23.52, "H": 68.31, "L": 89.12}
```

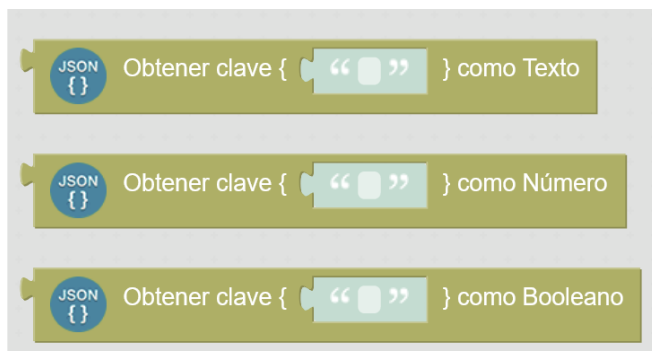
La decodificación de una cadena de texto en formato JSON es sencilla. Se debe analizar y a partir de ahí podremos extraer los datos de ese nivel. Si hay un subnivel (ramificación) se deberá obtener el valor del subnivel y volverlo a analizar para acceder a sus datos de forma directa. No hay límite de niveles y subniveles.

Ejemplo para extraer los datos codificados en el ejemplo anterior:



Durante la comunicación puede haber errores, o el formato del JSON puede ser incorrecto (o no ser datos JSON) por lo que deberíamos comprobar que se ha podido analizar correctamente los datos.

A la hora de extraer los valores de cada clave ("T", "H",...) podemos elegir el bloque que procesará y obtendrá el valor en el formato correspondiente. Esta elección es importante para recuperar el valor de la forma correcta. Por ejemplo si recuperamos un valor de forma numérica y no es un valor numérico válido dará internamente un error.

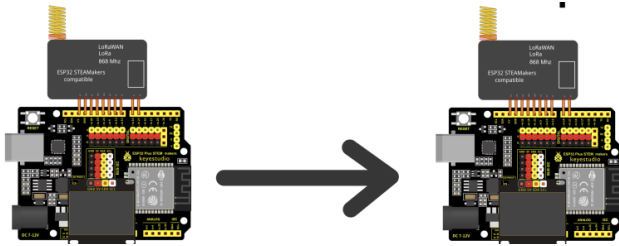


*Más información sobre JSON con Arduinoblocks:

<https://drive.google.com/file/d/1r290vEJVZtt8yp4PELnngxTEFWqUUBZk/view>

Ejemplos LoRa

LoRa 1: Emisor contador + Receptor OLED



Programa Emisor

```

Inicializar
  LoRa Init
  RFM9x module (SPI) 868 MHz
  NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)
  Encrypt data with key (16 chars) xxxxxxxxxxxxxxxxxxxx
  Establecer contador = 0

Bucle
  Ejecutar cada 15000 ms
  LoRa Send Formatear número contador con 0 decimales
  Establecer contador = contador + 1
  
```

Programa receptor:

```

Inicializar
  LoRa Init
  RFM9x module (SPI) 868 MHz
  NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)
  Encrypt data with key (16 chars) xxxxxxxxxxxxxxxxxxxx
  # 1 Iniciar I2C 0x3C Mostrar automáticamente
  # 1 Limpiar

Bucle

LoRa On data received
  # 1 Limpiar
  # 1 Texto X 0 Y 0 LoRa Received - Data Led ON medium
  
```

LoRa 2: Emisor T/H + Receptor OLED

Mediante la codificación de varios datos en formato JSON enviaremos los datos a través de LoRa y en el receptor decodificaremos el formato JSON extrayendo los datos que nos interesen.

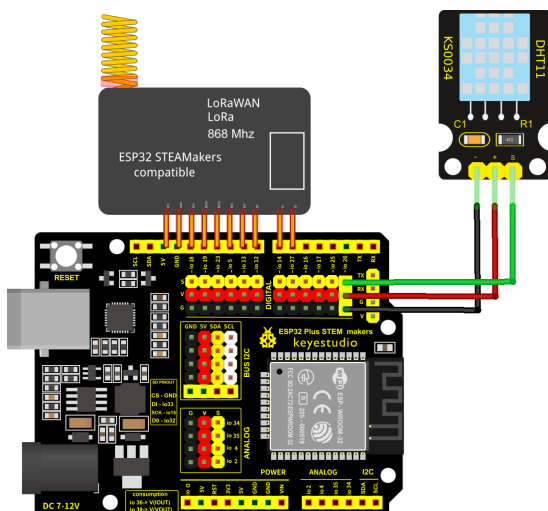
Los datos a enviar serán:

“T” -> Temperatura

“H” -> Humedad

En el receptor se recibe la trama JSON, se analiza y se extraen los datos para visualizar en la pantalla OLED.

Programa en emisor de datos:

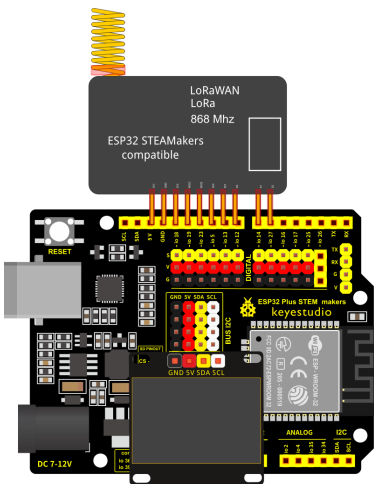


Programa receptor de datos OLED:

```

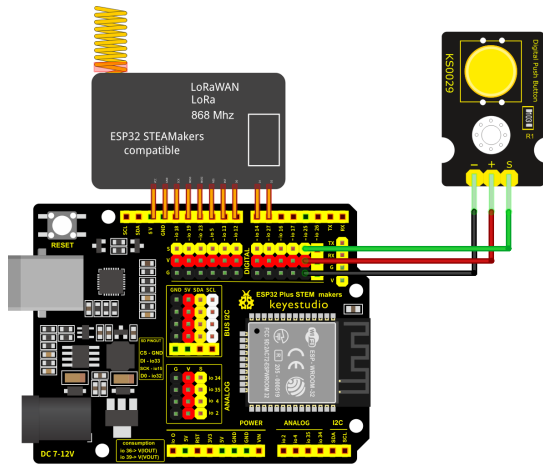
Inicializar
  LoRa Init
  RFM9x module (SPI) 868 MHz
  NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)
  [x] Encrypt data with key (16 chars) juanjlopezxxxxx
  # 1 Iniciar I2C 0x3C [x] Mostrar automáticamente
  # 1 Limpiar

Bucle
  LoRa On data received
  # 1 Limpiar
  Analizar JSON [LoRa Received - Data]
  + si [¿JSON analizado correctamente?]
  hacer
    Establecer temp = [Obtener clave { "T" } como Número]
    Establecer temp = [Obtener clave { "H" } como Número]
    # 1 Texto X 0 Y 0 "TEMP:" Led ON small
    # 1 Texto X 32 Y 0 temp Led ON small
    # 1 Texto X 0 Y 20 "HUM:" Led ON small
    # 1 Texto X 32 Y 20 hum Led ON small
  sino
    # 1 Texto X 0 Y 0 "Datos" Led ON small
    # 1 Texto X 0 Y 10 "incorrectos" Led ON small
  
```

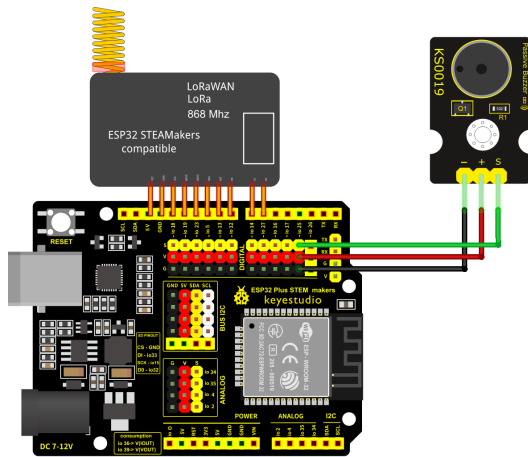


LoRa 3: Timbre remoto

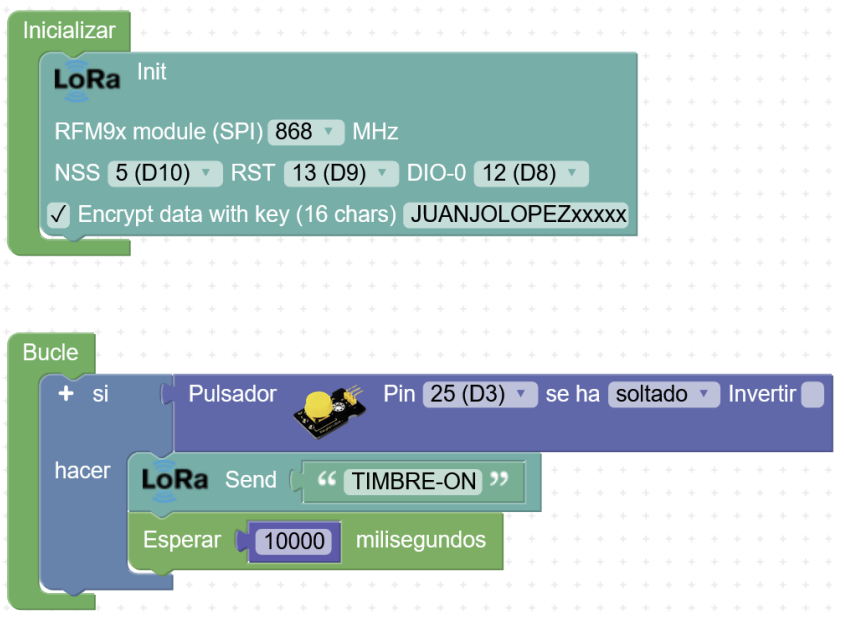
Esquema emisor: pulsador



Esquema receptor: zumbador



Programa emisor:



Programa receptor:

The image shows a Scratch-style code editor with the following blocks:

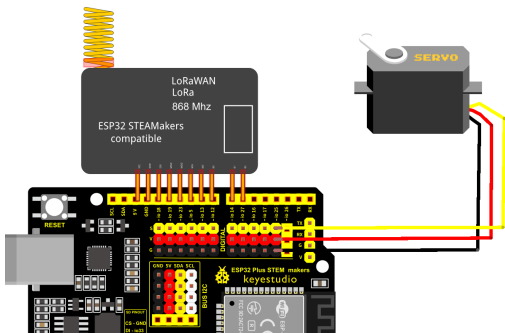
- Inicializar** (Initialize) block:
 - LoRa Init** block with settings:
 - RFM9x module (SPI) **868** MHz
 - NSS **5 (D10)** RST **13 (D9)** DIO-0 **12 (D8)**
 - Encrypt data with key (16 chars) **JUANJOLOPEZxxxxx**
- Bucle** (Loop) block: An empty loop structure.
- LoRa On data received** block:
 - + si** (if) block:
 - LoRa Received - Data** block with condition: **igual a** **“ TIMBRE-ON ”**
 - hacer** (do) block:
 - Zumbador** (Buzzer) block with settings:
 - Pin **25 (D3)**
 - Reproducir RTTTL **Indiana Jones**

LoRa 4: Apertura de puerta remota

Emisor: lector RFID (i2c) + tarjetas o llaveros de identificación



Receptor:



Programa emisor: Lector RFID + tarjeta

```

Inicializar
  MFRC522 Iniciar (I2C) ADDR: 0x28
  LoRa Init
    RFM9x module (SPI) 868 MHz
    NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)
    [x] Encrypt data with key (16 chars) JUANJOLOPEZxxxxx

Bucle
  + si ¿Se ha detectado una nueva tarjeta?
  hacer
    Establecer ID tarjeta = Leer ID y expulsar tarjeta
    Establecer datos json =
      JSON {
        Key "USER" Value "JUANJO"
        Key "RFID" Value ID tarjeta
      }
    LoRa Send datos json
    Esperar 5000 milisegundos
  
```


Programa receptor: apertura de puerta con servo

The code is organized into three main sections:

- Inicializar (Initialize):**
 - cerrar puerta (close door)
 - LoRa Init:
 - RFM9x module (SPI) 868 MHz
 - NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)
 - Encrypt data with key (16 chars) JUANJOLOPEZxxxx
 - Establecer ID valido = "aabbccdeeff"
- LoRa On data received:**
 - Analizar JSON (Analyze JSON) - LoRa Received - Data
 - si ¿JSON analizado correctamente? (if JSON analyzed correctly?)
 - hacer (do):
 - Establecer usuario = Obtener clave { "USER" } como Texto (Set user = Get key { "USER" } as Text)
 - Establecer rfid = Obtener clave { "RFID" } como Texto (Set rfid = Get key { "RFID" } as Text)
 - si rfid igual a ID valido (if rfid equal to valid ID)
 - hacer (do):
 - si puerta abierta (if door open)
 - cerrar puerta (close door)
 - sino (else)
 - abrir puerta (open door)
 - Bucle (Loop):**
 - para cerrar puerta (for close door):
 - Establecer puerta abierta = falso (Set door open = false)
 - Servo Pin 25 (D3) Grados Ángulo 0° Retardo (ms) 0 (Servo Pin 25 (D3) Degrees Angle 0° Delay (ms) 0)
 - para abrir puerta (for open door):
 - Establecer puerta abierta = verdadero (Set door open = true)
 - Servo Pin 25 (D3) Grados Ángulo 90° Retardo (ms) 0 (Servo Pin 25 (D3) Degrees Angle 90° Delay (ms) 0)

LoRa 5: Sensores distribuidos de T/H (maestro-esclavos)

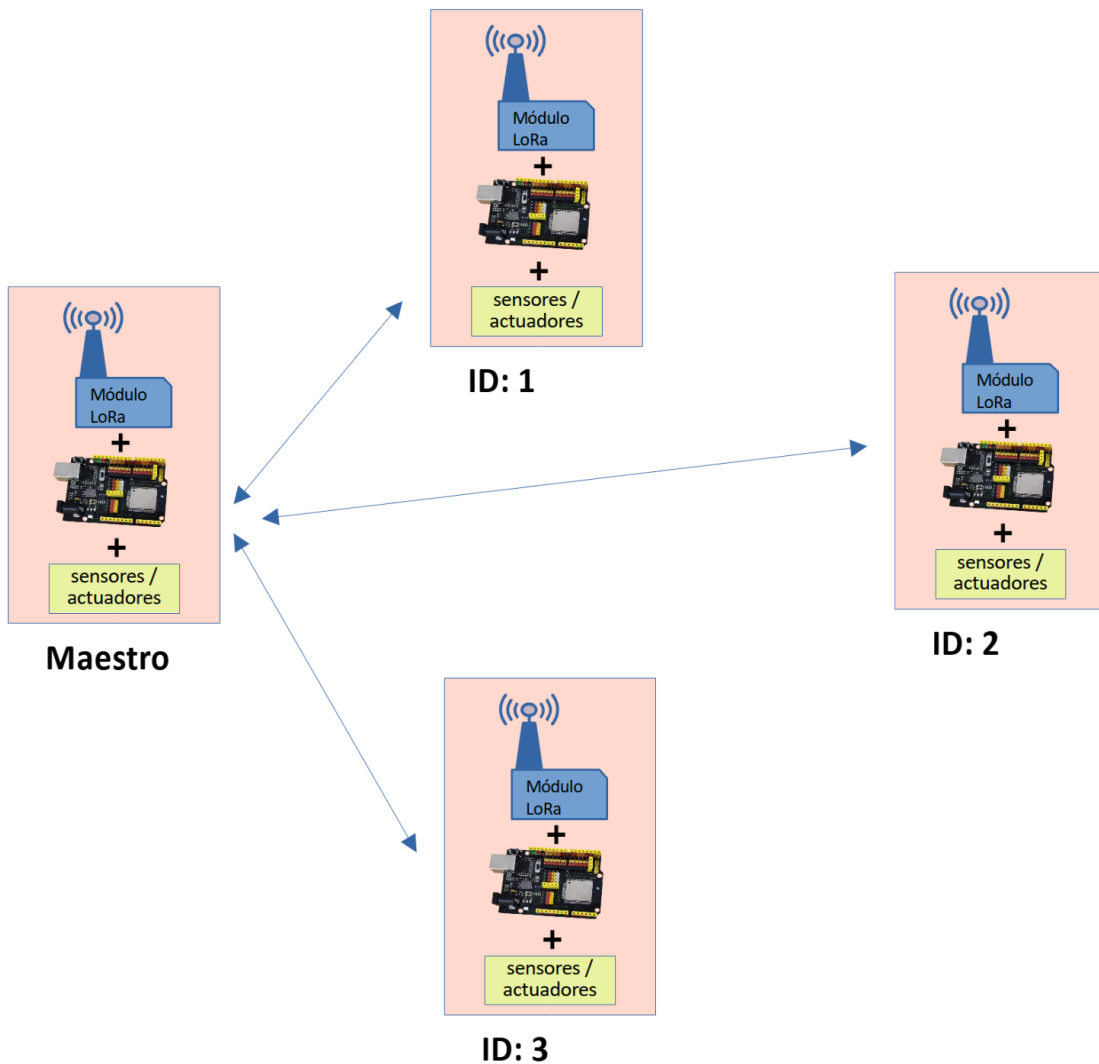
Podemos tener más de dos placas comunicándose entre ellas vía LoRa. Es importante definir un protocolo para evitar que “hablen” varias a la vez y se interfieran. En este ejemplo una placa hace de “*maestro*” y les pide información a los nodos “*esclavos*” que le responden con la información del sensor conectado.

El formato JSON que envía la placa “*maestra*” va a ser simplemente indicando el ID de la placa que debe responder. Cada placa debe tener un ID (número de identificación) asignado en el programa:

```
{ "ID": 1 }
```

Las placas contestarán con un formato JSON con el mismo ID y el valor del sensor de temperatura:

```
{ "ID":1, "T": 24.5 }
```



Programa módulo “maestro”:

Cada 15s pide los datos a un nodo (primero al nodo 1, después al 2, después al 3, y vuelve al 1)
Al recibir la respuesta la visualiza en la consola serie.

The code is written in the Arduino IDE and is organized into three main sections:

- Inicializar (Initialization):**
 - Starts the serial port at 115200 bauds.
 - Initializes the LoRa module with the following settings:
 - RFM9x module (SPI) at 868 MHz.
 - NSS: 5 (D10), RST: 13 (D9), DIO-0: 12 (D8).
 - Encrypt data with key (16 chars): JUANJOLOPEZxxxxx.
 - Establishes the destination ID to 1.
 - Establishes the number of nodes to 3.
- Bucle (Loop):**
 - Executes every 10000 ms.
 - Sends the message "Solicitando datos al nodo:" with a line break.
 - Sends the current destination ID with a line break.
 - Sends a JSON message: `{ "ID": ID destino }`.
 - Increments the destination ID by 1.
 - If the destination ID is greater than the number of nodes, it resets the destination ID to 1.
- LoRa On data received:**
 - Analyzes the received JSON data.
 - If the JSON is analyzed correctly:
 - Extracts the source ID from the JSON and converts it to a number.
 - Extracts the temperature (T) from the JSON and converts it to a number.
 - Sends the message "Datos recibidos del nodo:" with a line break.
 - Sends the source ID with a line break.
 - Sends the message "Temperatura:" with a line break.
 - Sends the temperature value with a line break.
 - Sends a line of dashes with a line break.

Programa de los nodos “esclavos”:

En una variable inicializamos el número de nodo para cambiarlo fácilmente al subir el programa a cada nodo. En este ejemplo todos los nodos van a tener el mismo sensor y van a devolver el mismo tipo de dato (temperatura). Lo único que varía es el número de nodo (ID).

Inicializar

Iniciar Baudios 115200

LoRa Init

RFM9x module (SPI) 868 MHz

NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)

Encrypt data with key (16 chars) JUANJOLOPEZxxxxx

Establecer ID de este nodo = 1

Bucle

LoRa On data received

Analizar JSON Received - Data

si ¿JSON analizado correctamente?

hacer

Establecer ID solicitado = Obtener clave { “ ID ” } como Número

si ID solicitado = ID de este nodo

hacer

Establecer temp = DHT-11 Temperatura °C Pin 26 (D2)

Establecer datos json = Codificar {K,V} - +

Key “ ID ” Value ID de este nodo

Key “ T ” Value temp

LoRa Send datos json

Datos enviados por el módulo maestro en la consola serie:

Baudrate: 115200 Conectar Desconectar Limpiar

Enviar

Solicitando datos al nodo: 1.00
 Datos recibidos del nodo: 1.00
 Temperatura: 23.00

 Solicitando datos al nodo: 2.00

LoRa 6: Datalogger remoto

Programa emisor (datos de sensores):

Inicializar

- LoRa Init**
 - RFM9x module (SPI) 868 MHz
 - NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)
 - Encrypt data with key (16 chars) xxxxxxxxxxxxxxxxxxxx

Bucle

- Ejecutar cada 15000 ms
 - Establecer temperatura = DHT-22 Temperatura °C Pin 25 (D3)
 - Establecer humedad = DHT-22 Humedad % Pin 25 (D3)
 - Establecer nivel de luz = Nivel de luz (LDR) Pin 34 (A3) 0.4095
 - Establecer datos csv = CSV ;
 - + - crear texto con
 - Formatear número temperatura con 1 decimales
 - Formatear número humedad con 1 decimales
 - Formatear número nivel de luz con 0 decimales
 - LoRa Send datos csv

Programa receptor:

Inicializar

- Iniciar Baudios 115200
- LoRa Init**
 - RFM9x module (SPI) 868 MHz
 - NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)
 - Encrypt data with key (16 chars) xxxxxxxxxxxxxxxxxxxx

Bucle

- On data received
 - Enviar LoRa Received - Data Salto de línea

Visualización de datos recibidos (formato CSV):

Consola serie

Baudrate:

```
19.4;53.6;2621;
19.4;53.3;3047;
20.6;87.2;2618;
23.2;94.8;2582;
```

Programa receptor mejorado con grabación de datos en la tarjeta microSD insertada en el zócalo de la ESP32 STEAMakers:

The image shows a screenshot of the Arduino IDE Blocks Editor. The program is organized into two main sections: initialization and a loop.

Inicializar (Initialization):

- SD Iniciar:** A block to initialize the SD card.
- Iniciar Baudios 115200:** A block to set the serial baud rate to 115200.
- LoRa Init:** A block to initialize the LoRa module with the following settings:
 - RFM9x module (SPI) 868 MHz
 - NSS 5 (D10)
 - RST 13 (D9)
 - DIO-0 12 (D8)
 - Encrypt data with key (16 chars) xxxxxxxxxxxxxxxxxxxx

Bucle (Loop):

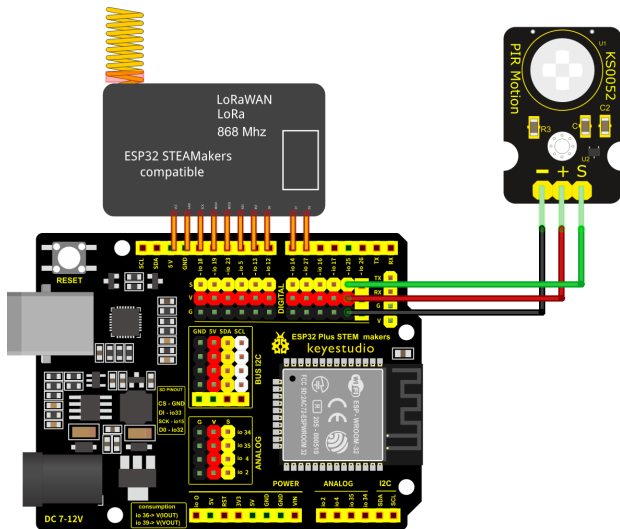
- LoRa On data received:** A block that triggers the following actions when data is received:
 - Enviar:** A block to send the received data via the serial port, with the text "LoRa Received - Data" and the "Salto de línea" (line feed) checkbox checked.
 - Escribir:** A block to write the received data to a file named "datos.csv". The text "LoRa Received - Data" is concatenated with the "Salto de línea" checkbox checked.

LoRa 7: Central de alarma y sensores de presencia

El módulo con el sensor de movimiento emitirá una trama LoRa con el texto “*SENSORMOV*” en caso de detectar movimiento.

El receptor al recibir una trama LoRa con el texto “*SENSORMOV*” activará un led rojo y un zumbador que empezará a pitar. La alarma se podrá desactivar (resetear) pulsando el pulsador.

Emisor (nodo con sensor movimiento):



Inicializar

LoRa Init

RFM9x module (SPI) 868 MHz / SF: 9

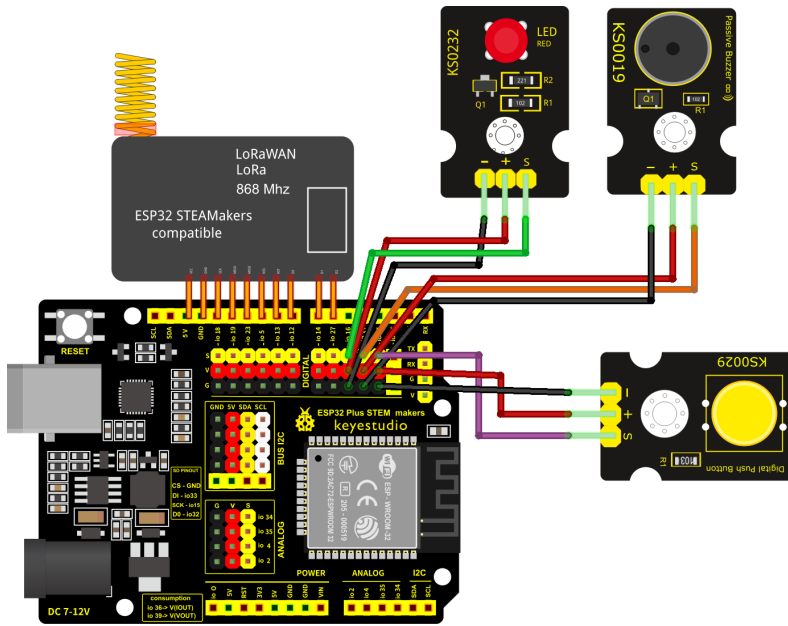
NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)

Encrypt data with key (16 chars) juanjlopezxxxxx

Bucle



Receptor (central de alarma):



- Led rojo -> pin io16
- Zumbador -> pin io17
- Pulsador -> pin io25

```

graph TD
    subgraph Inicializar
        A[Establecer alarma sonando = falso]
        B[Led Pin 16 D5 Estado OFF]
        C[LoRa Init]
        C --- C1[RFM9x module SPI 868 MHz]
        C --- C2[NSS 5 D10 RST 13 D9 DIO-0 12 D8]
        C --- C3[Encrypt data with key 16 chars Juanjlopezxxxx]
    end

    subgraph Bucle
        D[comprobar pulsador reset]
        E[Ejecutar cada 1000 ms]
        F[zumbador pitido]
        D --- E --- F
    end

    subgraph LoRa_On_data_received
        G[si LoRa Received - Data empieza con "SENSORMOV"]
        H[hacer Establecer alarma sonando = verdadero]
        I[Led Pin 16 D5 Estado ON]
        G --- H --- I
    end

    subgraph Comprobar_pulsador_reset
        J[para comprobar pulsador reset]
        K[si Pulsador Pin 25 D3 se ha soltado Invertir]
        L[hacer si alarma sonando]
        M[hacer Establecer alarma sonando = falso]
        N[Led Pin 16 D5 Estado OFF]
        K --- L --- M --- N
    end

    subgraph Zumbador_pitido
        O[para zumbador pitido]
        P[si alarma sonando]
        Q[hacer Zumbador Pin 17 D4 Ms 200 Hz 1000]
        P --- Q
    end
    
```


LoRa 8: Gateway LoRa <-> Telegram

Se creará un bot de Telegram conectado a la ESP32 STEAMakers.

Los datos recibido por LoRa se enviarán directamente al canal de Telegram

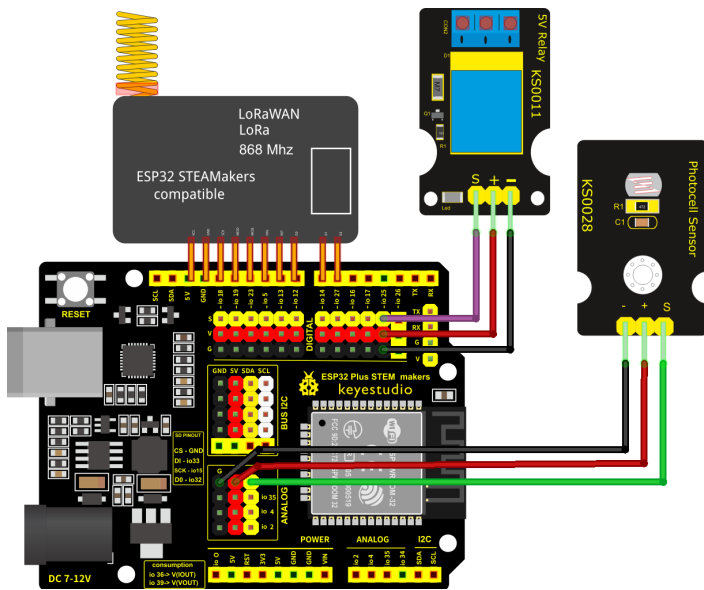
Los datos recibidos del canal de Telegram se enviarán vía LoRa.

El nodo remoto tendrá un sensor de luz cuyo valor enviará cada 60s

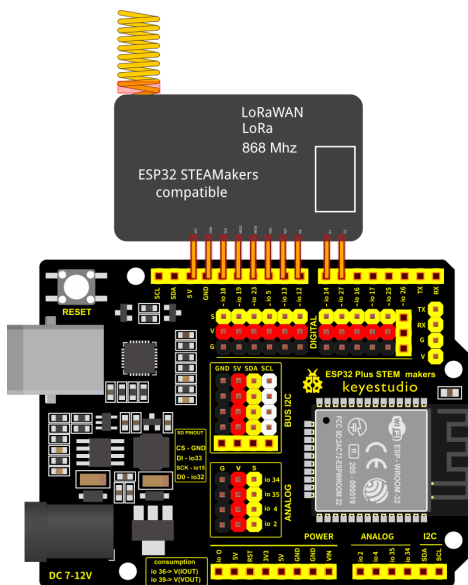
El nodo remoto tendrá un relé conectado, en caso de recibir el comando "RELE-ON" se activará

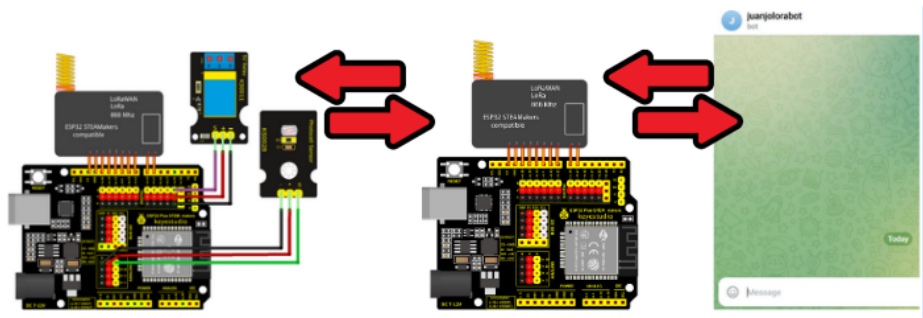
El nodo remoto tendrá un relé conectado, en caso de recibir el comando "RELE-OFF" se apagará

Nodo sensor remoto LoRa:



Nodo gateway LoRa <-> Telegram:





Programa nodo Gateway:

Inicializar

- Conectar a una red WiFi
 - SSID: [redacted]
 - Clave: [redacted]
- Telegram bot: Iniciar API Token [redacted]
- LoRa Init
 - RFM9x module (SPI) 868 MHz
 - NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)
 - Encrypt data with key (16 chars) juanjlopezxxxxx

Bucle

- On data received
 - Telegram bot: Enviar a Chat ID [redacted] Mensaje LoRa Received - Data Formato None
- Nuevo mensaje recibido
 - LoRa Send: Telegram bot Mensaje Texto

Programa nodo LoRa:

Inicializar

- LoRa Init
 - RFM9x module (SPI) 868 MHz
 - NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)
 - Encrypt data with key (16 chars) juanjlopezxxxxx

Bucle

- Ejecutar cada 30000 ms
 - Establecer nivel de luz = Nivel de luz (LDR) Pin 34 (A3) %
 - Establecer datos lora = + - crear texto con "Luz = " nivel de luz
 - LoRa Send: datos lora

LoRa On data received

- si LoRa Received - Data empieza con "RELE-ON"
 - hacer Relé Pin 25 (D3) Estado ON
 - LoRa Send: "Rele ON"
- si LoRa Received - Data empieza con "RELE-OFF"
 - hacer Relé Pin 25 (D3) Estado OFF
 - LoRa Send: "Rele OFF"

Ejemplo en funcionamiento:



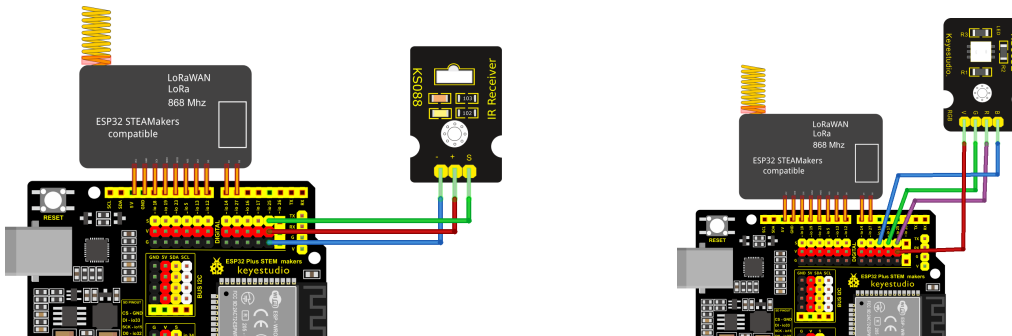
*Más información sobre el uso de Telegram con Arduinoblocks y ESP32 STEAMakers:

<https://drive.google.com/file/d/15cfrsSBGjL8pgq67x7Syr32BBO714F7M/view>

LoRa 9: Control RGB remoto

Mediante un mando IR controlaremos el color de un led RGB remoto via LoRa

Esquema emisor-receptor:



Programa emisor:

Inicializar

LoRa Init

RFM9x module (SPI) 868 MHz

NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)

Encrypt data with key (16 chars) juanjlopezxxxxx

Bucle

Establecer **codigo ir** = Receptor de IR (Texto HEX) Pin 25 (D3)

+ si **codigo ir** igual a "C101E57B"

hacer **LoRa Send** Codificar {Key:Value}

- Key "R" Value 150
- Key "G" Value 0
- Key "B" Value 0

+ si **codigo ir** igual a "97483BFB"

hacer **LoRa Send** Codificar {Key:Value}

- Key "R" Value 0
- Key "G" Value 200
- Key "B" Value 15


+ si **codigo ir** igual a "FOC41643"

hacer **LoRa Send** Codificar {Key:Value}

- Key "R" Value 30
- Key "G" Value 10
- Key "B" Value 180

Programa receptor:

Inicializar

Led RGB  Cátodo común Pin R 25 (D3) Pin G 17 (D4) Pin B 16 (D5) Color

LoRa Init

RFM9x module (SPI) 868 MHz

NSS 5 (D10) RST 13 (D9) DIO-0 12 (D8)

Encrypt data with key (16 chars) juanjlopezxxxxx

Bucle

LoRa On data received

JSON {} Analizar JSON **LoRa** Received - Data


+ si **JSON** {} ¿JSON analizado correctamente?


hacer


Establecer r = **JSON** {} Obtener clave { " R " } como Número

Establecer g = **JSON** {} Obtener clave { " G " } como Número

Establecer b = **JSON** {} Obtener clave { " B " } como Número

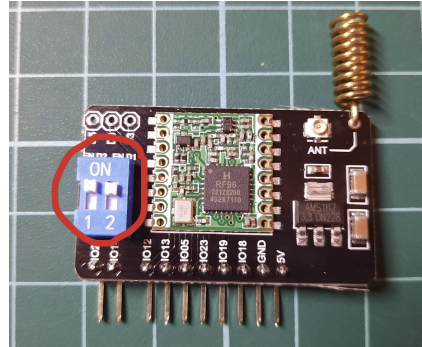
Led intensidad (PWM)  Pin 25 (D3) Valor r

Led intensidad (PWM)  Pin 17 (D4) Valor g

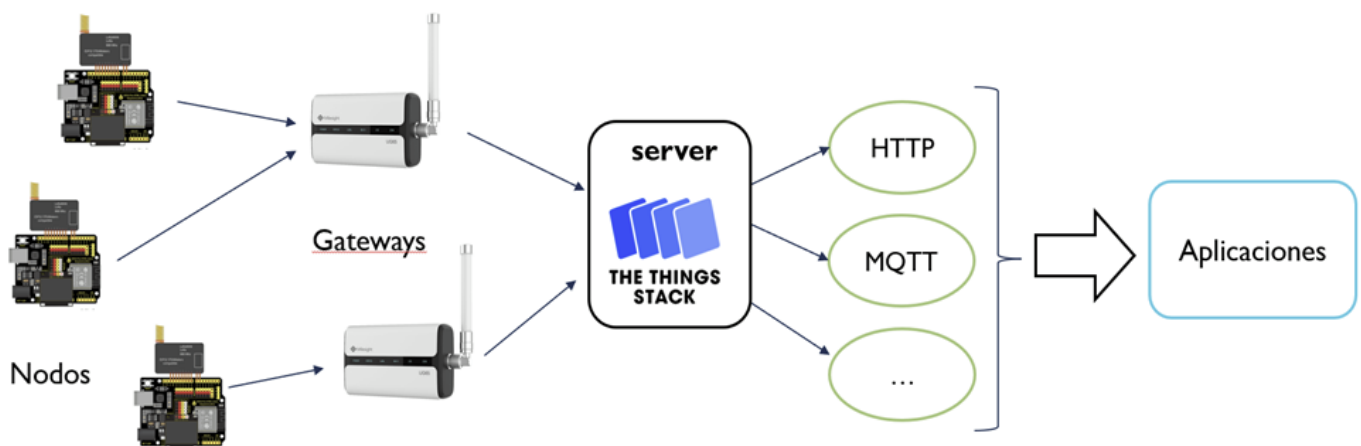
Led intensidad (PWM)  Pin 16 (D5) Valor b

LoRaWAN con arduinoblocks

Para conectar a una red LoRaWAN debemos asegurarnos de tener los dos microinterruptores del módulo LoRa en posición ON conforme se muestra en la siguiente figura:



Esquema de una infraestructura LoRaWAN de ejemplo:



Cada módulo LoRa que se conecte a una infraestructura LoRaWAN necesitará unos datos de conexión que serán configurados dentro del servidor (The Things Stack o similar)

Para la conexión y validación de los nodos a LoRaWAN tenemos dos tipos de conexión:

- **OTAA (Over-the-Air-Activation):** Permite conectar dispositivos de forma dinámica. Cada vez que el dispositivo se conecta recibe una dirección de red automáticamente.
 - Es el método de conexión recomendado
 - Precisa de que el módulo pueda enviar datos al gateway (bidireccional)
 - Es el método asignado a los módulos utilizados en este curso
- **ABP (Activation by Personalization):** Conecta un dispositivo de forma estática indicando durante la conexión la dirección de la red que se va a utilizar.
 - Es un sistema más sencillo de implementar pues se salta el paso de negociación de la dirección pero es desaconsejado.
 - Los parámetros de conexión internos se preconfiguran de forma fija y si uno cambia en el servidor el dispositivo no se conectará (en OTAA se negocian en cada conexión)

Según configuremos el dispositivo en nuestro servidor LoRaWAN debemos iniciar la conexión.

Register end device

From The LoRaWAN Device Repository [Manually](#)

Frequency plan ⓘ *

Europe 863-870 MHz (SF9 for RX2 - recommended) | ▾

LoRaWAN version ⓘ *

LoRaWAN Specification 1.0.3 | ▾

Regional Parameters version ⓘ *

RP001 Regional Parameters 1.0.3 revision A | ▾

Show advanced activation, LoRaWAN class and cluster settings ▾

DevEUI ⓘ *

.....

AppEUI ⓘ *

.....

AppKey ⓘ *

.....

End device ID ⓘ *

my-new-device

Los datos identificativos para cada nodo o dispositivo final serán (OTAA):

- **DevEUI (64 bits):** Identificador único del nodo (especificado en hexadecimal)
- **AppEUI (64 bits):** Identificador de aplicación, puede ser el mismo para todo
- **AppKey (128 bits):** Clave de encriptación para el nodo (recomendable que sea diferente en cada nodo)

*Más información sobre OTAA y ABP:

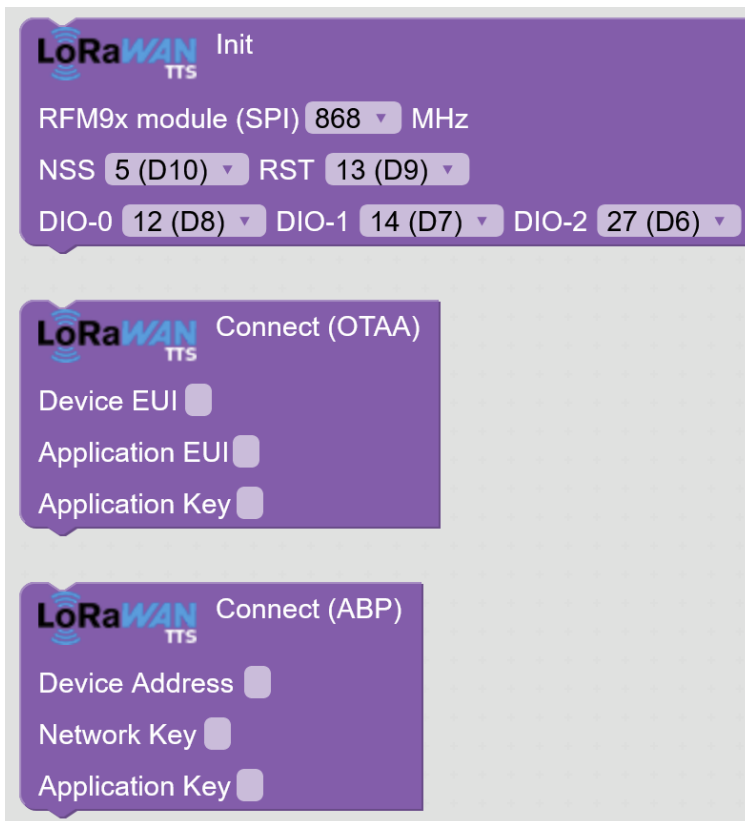
<https://www.thethingsindustries.com/docs/devices/abp-vs-otaa/>

Bloques LoRaWAN:

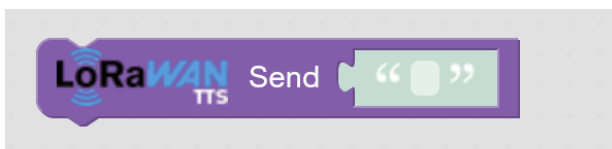
Init: Inicia el módulo para trabajar en modo LoRaWAN (interruptores a ON obligatoriamente). Si usamos el módulo personalizado no debemos cambiar la configuración de pines. La frecuencia LoRaWAN en Europa es 868Mhz, también marcado por defecto.

Connect (OTAA): Conectar (registrarse / JOIN) en la red LoRaWAN mediante autenticación OTAA

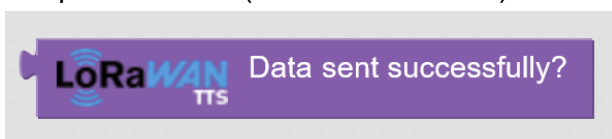
Connect (ABP): Conectar (registrarse / JOIN) en la red LoRaWAN mediante autenticación ABP



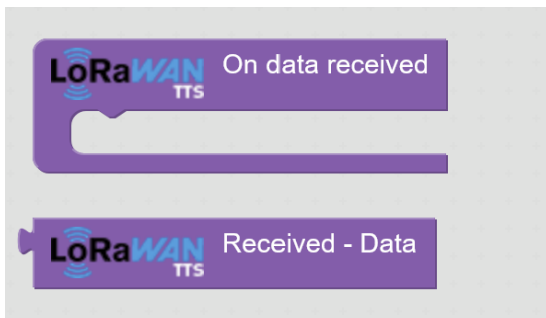
Send: Envía datos a través de la conexión LoRaWAN, los datos llegarán al servidor LoRaWAN quien gestionará el envío o almacenamiento a otras aplicaciones (Web, MQTT, bases de datos, etc.)



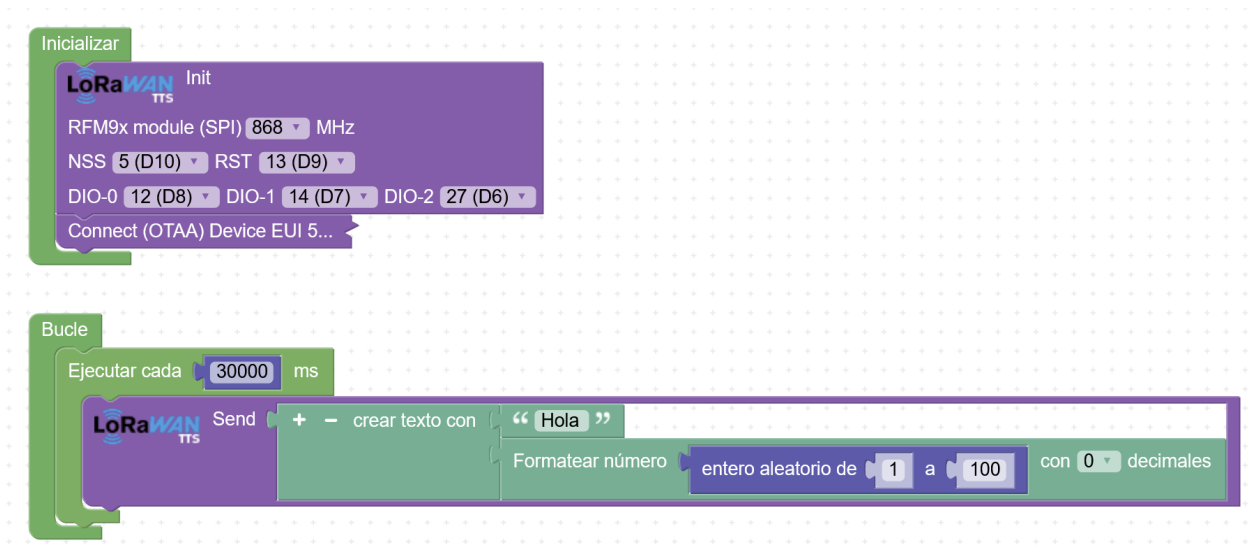
Data sent successfully?: Bloque que permite obtener la confirmación del servidor de que los datos se han podido enviar (en el último "Send")



On Data received: Evento que se produce al recibir datos hacia el dispositivo. En caso de producirse los datos los podemos obtener con el bloque "Received data"



Ejemplo de inicialización del módulo (LoRaWAN + OTAA) y envío de datos de prueba:



Ejemplo de visualización en el servidor TTS de los datos recibidos:

inovación > Live data

Type	Data preview	Verbose stream	Export as JS
01 Forward uplink data message	DevAddr: 00 E0 BE 00 <> Data rate: SF7BW125 SNR: 13.2 RSSI: -42	<input type="checkbox"/>	
01 Forward uplink data message	DevAddr: 00 E0 BE 00 <> Payload: { decoded_data: "Hola 35" } 48	<input type="checkbox"/>	
01 Forward uplink data message	DevAddr: 00 E0 BE 00 <> Data rate: SF7BW125 SNR: 13.5 RSSI: -47	<input type="checkbox"/>	
01 Forward uplink data message	DevAddr: 00 E0 BE 00 <> Payload: { decoded_data: "Hola 89" } 48	<input type="checkbox"/>	
01 Forward uplink data message	DevAddr: 00 E0 BE 00 <> Payload: { decoded_data: "Hola 44" } 48	<input type="checkbox"/>	
01 Forward uplink data message	DevAddr: 00 E0 BE 00 <> Data rate: SF7BW125 SNR: 13.5 RSSI: -49	<input type="checkbox"/>	
01 Forward uplink data message	DevAddr: 00 E0 BE 00 <> Payload: { decoded_data: "Hola 80" } 48	<input type="checkbox"/>	

Ejemplos LoRaWAN

LoRaWAN 1: Estación meteorológica

Nodo LoRaWAN para recoger datos medioambientales de temperatura, humedad y nivel de CO2. En una ciudad con infraestructura LoRaWAN podrían repartirse múltiples nodos en diferentes ubicaciones para recolectar y procesar los datos, así como para emitir avisos.

Sensor DHT22 (temperatura y humedad)

Sensor CO2

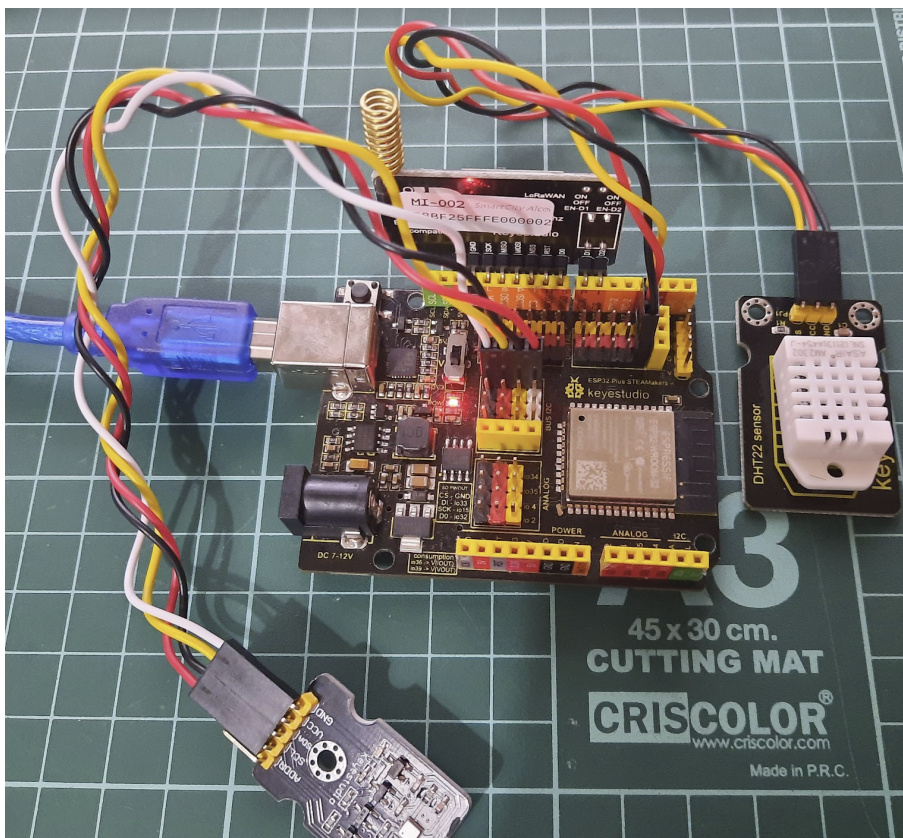
DHT22 -> Pin 25

CO2 -> I2C

Datos en formato JSON:

```
{ "T": x, "H": y, "CO2": z }
```

Imagen del nodo con los sensores y el módulo LoRa:



Programa:

Inicializar

LoRaWAN TTS Init
 RFM9x module (SPI) 868 MHz
 NSS 5 (D10) RST 13 (D9)
 DIO-0 12 (D8) DIO-1 14 (D7) DIO-2 27 (D6)

LoRaWAN TTS Connect (OTAA)
 Device EUI 58BF25FFFE000001
 Application EUI 0000000000000001
 Application Key 00000000000000000000000000000000

Bucle

Ejecutar cada 30000 ms

Establecer temp = DHT-22 Temperatura °C Pin 25 (D3)

Establecer hum = DHT-22 Humedad % Pin 25 (D3)

Establecer co2 = Sensor CO2/TVOC (CCS811) CO2 (ppm)

LoRaWAN TTS Send

JSON Codificar {Key:Value} - +

- JSON Key " T " Value temp
- JSON Key " H " Value hum
- JSON Key " CO2 " Value co2

Datos recibidos en el servidor TTS:

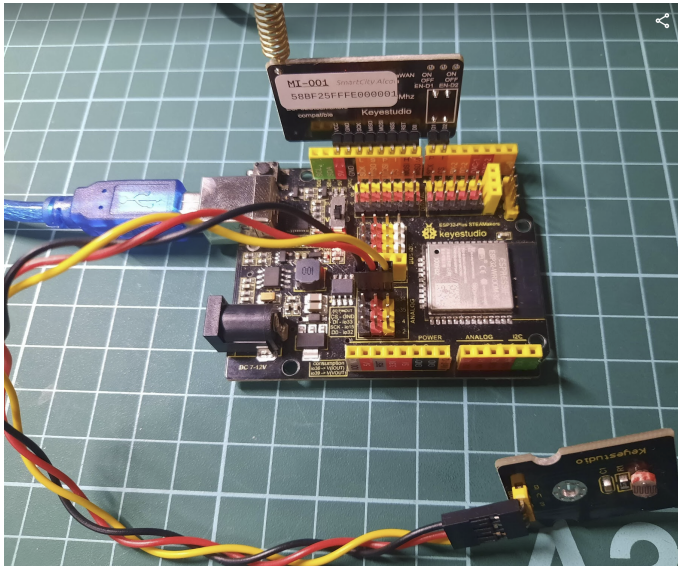
[2023-11-06 19:19:45] ID: 58BF25FFFE000001 , DATOS: {"T":19.10,"H":55.90,"CO2":400.00}

LoRaWAN 2: Medidor de luz ambiental

Nodo LoRaWAN para medición de luz o contaminación lumínica.

Sensor nivel de luz LDR

Sensor LDR -> Pin analógico io34 (A3)



Programa:

Initializar

LoRaWAN TTS Init

RFM9x module (SPI) 868 MHz

NSS 5 (D10) RST 13 (D9)

DIO-0 12 (D8) DIO-1 14 (D7) DIO-2 27 (D6)

Connect (OTAA) Device EUI 5...

Bucle

Ejecutar cada 30000 ms

Establecer luz = Nivel de luz (LDR) Pin 34 (A3) 0..4095

LoRaWAN TTS Send JSON { } Codificar {Key:Value} - + JSON { } Key "L" Value luz

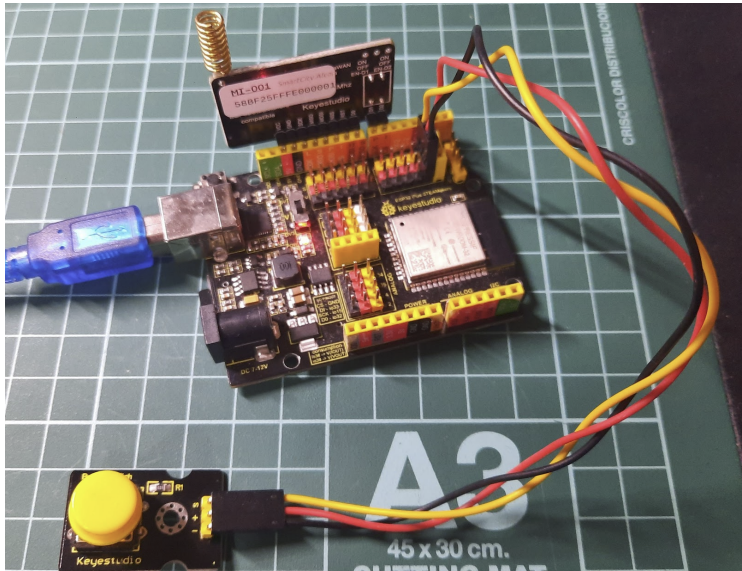
Datos recibidos en el servidor TTS:

[2023-11-06 19:26:17] ID: 58BF25FFFE000001 , DATOS: {"L":1195.00}

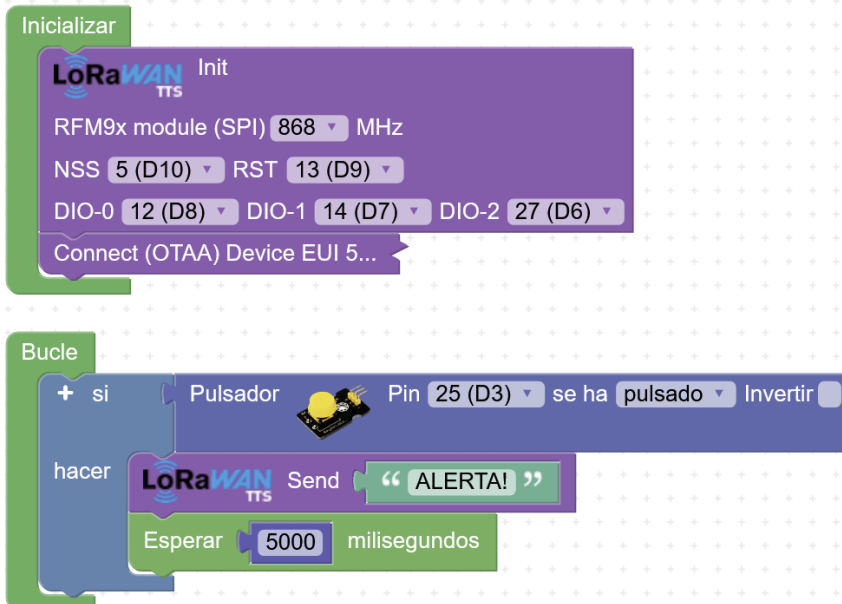
LoRaWAN 3: Pulsador de emergencia

Pulsador para aviso de emergencias, permite lanzar un mensaje de alerta dentro de la cobertura de la red LoRaWAN (por ejemplo para personas mayores dentro de la cobertura de una *smartcity*)

Pulsador -> Pin 25 (pulsador keystudio funciona a la inversa)



Programa:



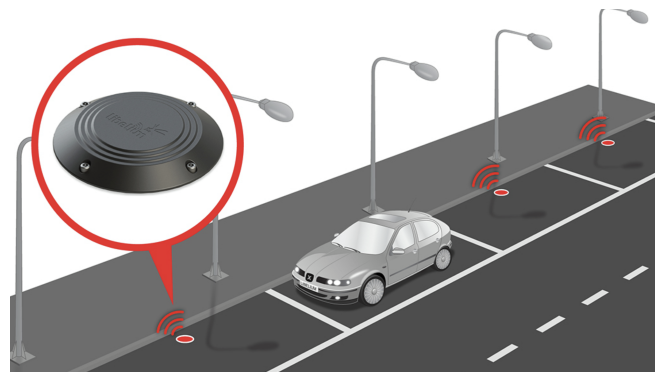
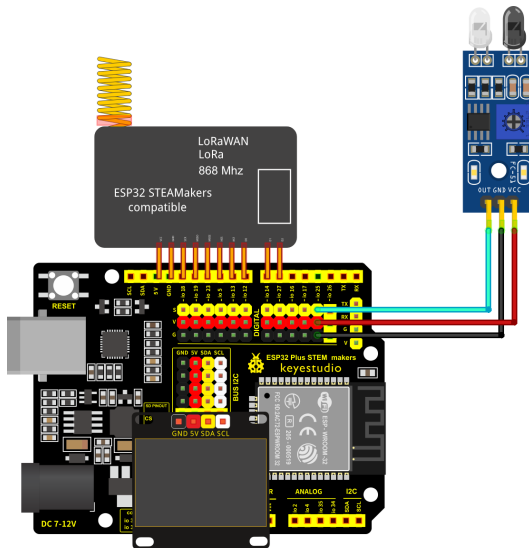
Datos recibidos en el servidor TTS:

[2023-11-06 19:33:13] ID: 58BF25FFFE000001 , DATOS: ALERTA!

LoRaWAN 4: Detector de ocupación para plazas de aparcamiento

Detecta si la plaza de aparcamiento está ocupada (coche encima) y permite crear un mapa de las plazas libres y ocupadas en la ciudad.

Sensor de obstáculo por IR



Programa:

Diagrama de programación en bloques para el detector de ocupación de plazas de aparcamiento.

Inicializar

- LoRaWAN TTS Init
 - RFM9x module (SPI) 868 MHz
 - NSS 5 (D10) RST 13 (D9)
 - DIO-0 12 (D8) DIO-1 14 (D7) DIO-2 27 (D6)
- LoRaWAN TTS Connect (OTAA)
 - Device EUI [oculto]
 - Application EUI [oculto]
 - Application Key [oculto]

Bucle

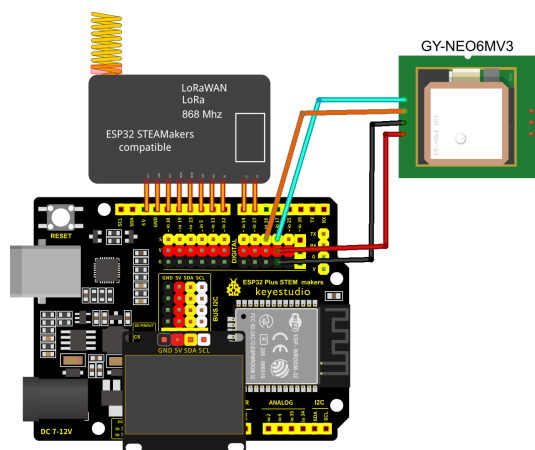
- Ejecutar cada 30000 ms
 - si Detector de obstáculos (IR) Pin 25 (D3)
 - hacer LoRaWAN TTS Send "OCUPADO"
 - sino LoRaWAN TTS Send "LIBRE"

LoRaWAN 4: Geolocalización GPS en tiempo real

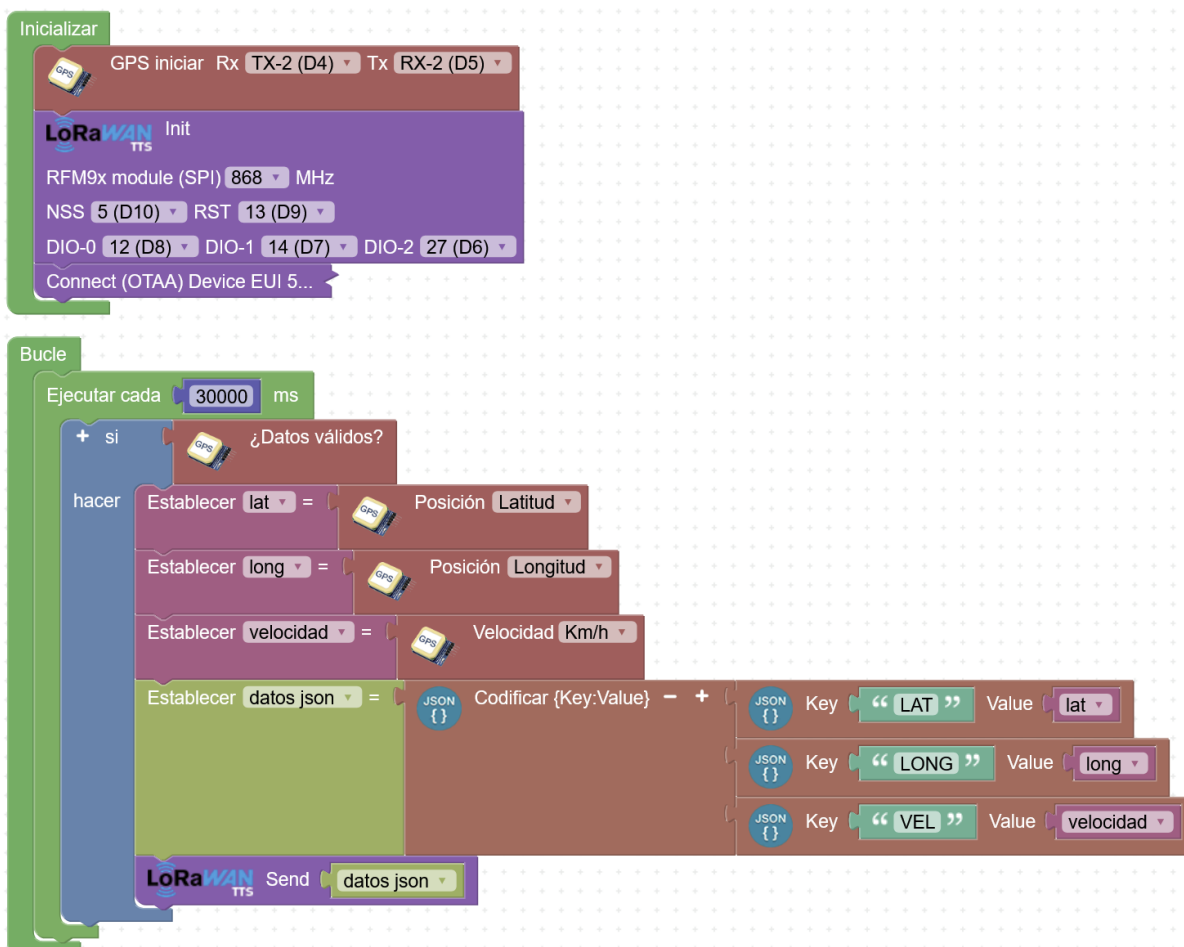
El sistema envía mediante la red LoRaWAN la posición actual obtenida mediante GPS. Permite hacer un seguimiento en tiempo real de un vehículo, personas, etc. y visualizarlo en un mapa.

Sensor -> Módulo GPS Neo6/7

Esquema:



Programa:



Ejemplo de trama de datos recibida:

```
{ "LAT": 38.6265 , "LONG": -0.5727 , "VEL": 17.4 }
```

Posicionamiento en un mapa a partir de los datos recibidos:

Get GPS Coordinates

DD (decimal degrees)*

Latitude

Longitude

Get Address

Lat,Long



<https://www.gps-coordinates.net/>

Información y enlaces:

Arduinoblocks:

<http://www.arduinoblocks.com>

Documentación, manuales, videos:

<http://www.arduinoblocks.com/web/site/doc>

Tienda Innovadidàctic - Arduinoblocks:

<https://shop.innovadidactic.com>

Placa ESP32 STEAMakers:

<https://shop.innovadidactic.com/es/standard-placas-shields-y-kits/1567-placa-esp32-steamakers-no-incluye-cable-usb.html>

Módulo LoRa/LoRaWAN para ESP32 STEAMakers:

<https://shop.innovadidactic.com/es/standard-perifericos/1663-modulo-lora-lorawan-8436574314656.html>

