راه اندازی ماژول LoRa :

LoRa (Long Range) یک فناوری بی‌سیم است که امکان ارتباط **برد بلند** با **مصرف توان بسیار کم** رو فراهم می‌کند. تراشه‌هایی مثل SX1278 / SX1276 از این پروتکل پشتیبانی می‌کنند .در فضای باز تا **۱۵ کیلومتر** و حتی بیشتر با آنتن مناسب و دید مستقیم. در مناطق شهری بین **۲ تا ۵ کیلومتر**.

فرکانس کاری: 433MHz، 868MHz، 915MHz

ماژول از SPI برای ارتباط با میکروکنترلر استفاده می‌کند.

کاربرد های رایج:

ارسال دما، رطوبت، فشار از سنسورهای دور	اینترنت اشیاء(IoT)
نظارت بر خاک و آب مزرعه	کشاورزی هوشمند
مانیتورینگ سطل زباله، کیفیت هوا	شهر هوشمند
ارتباط بین واحدهای متحرک بدونWiFi	ربات‌ها
ساخت شبکه خصوصی بدون نیاز به اینترنت	شبکه‌های مستقل

اتصالات سخت افزاری:

SS (NSS/CS)	GPIO5
RESET	GPIO14
DIO0	GPIO2
SCK	GPIO18
MISO	GPIO19
MOSI	GPIO23
GND	GND
VCC	3.3V

مشخصات کلی آنتن BW433FNX75-35B1 :

آنتن استفاده شده BW433FNX75-35B1 یک آنتن مخصوص فرکانس ۴۳۳ مگاهرتز (MHz) هست که برای ماژول‌های LoRa، RF، یا سایر ماژول‌های بی‌سیم با فرکانس ۴۳۳ MHzاستفاده میشود.

ویژگی	توضیح
بازه فرکانسی (Frequency Range)	433 MHz
نوع آنتن	آنتن فنری (Spring Antenna)
تقویت‌کننده سیگنال (Gain)	معمولاً بین 2 تا 5 dBi
امپدانس	50 اهم
طول آنتن	3.5 cm

کتابخانه ها و توابع استفاده شده:

```
#include <SPI.h>
#include <LoRa.h>
```

راه اندازی ماژول:

```
bool LoRa.begin(long frequency); //LoRa.begin(433E6);
```

تنظیم پایه ها:

```
LoRa.setPins(csPin,resetPin,DIO0);
```

ارسال داده:

```
LoRa.beginPacket();
LoRa.print("Hello");
LoRa.endPacket();
```

دریافت داده:

```
int packetSize = LoRa.parsePacket();  // بررسی اینکه دیتایی رسیده یا نه
```

```
int data = LoRa.read();                 // خواندن کاراکتر
String msg = LoRa.readString();         // خواندن کامل پیام
```

توابع پیشرفته تنظیم کیفیت ارتباط:

```
LoRa.setSpreadingFactor(7);         // بین 6 تا 12 (12 = برد بیشتر، سرعت کمتر)
LoRa.setSignalBandwidth(125E3);     // 7.8 تا 500kHz) پهنای باند
LoRa.setCodingRate4(5);             // کدینگ → بین 5 تا 8
LoRa.setTxPower(14);                // توان خروجی (2 تا 20 dBm)

LoRa.packetRssi();          // قدرت سیگنال دریافتی (RSSI)
LoRa.packetSnr();           // نسبت سیگنال به نویز
LoRa.available();           // آیا داده برای خواندن وجود دارد؟

LoRa.idle();                // توقف ارتباط فعلی
LoRa.receive();             // رفتن به حالت گیرنده
LoRa.sleep();               // صرفه‌جویی در انرژی
```

برای متصل کردن دو لورا به هم دیگر:

```
LoRa.setSyncWord(0x12); // یکسان به SyncWord فقط دستگاه‌هایی با هم وصل می‌شن
```

تراشه‌های LoRa مثل SX1276 / SX1278، در زمان دریافت داده، روی پایه DIO0 = HIGH می‌فرستن. این یعنی شما می‌تونی خودت از این پایه، با attachInterrupt() توی ESP32 استفاده کنی تا وقتی داده رسید، وقفه ایجاد بشه .

❖ چرا SPI.h هست ولی ازش مستقیم استفاده نمی‌شه؟

ماژول LoRa از پروتکل SPI برای ارتباط با میکروکنترلر استفاده می‌کند،
اما کنترل SPI به‌طور داخلی توسط کتابخانه LoRa انجام می‌شه . و برای اینکه بتونه با ماژول LoRa ارتباط بگیره، نیاز داره که کتابخانه SPI.h هم در کد شما include شده باشه، حتی اگر مستقیم از SPI. استفاده نکردی .

❖ LoRa.setSyncWord(0x12) چی کار می‌کنه؟

Sync Word یا کلمه هم‌زمان‌سازی، یه عدد ۸ بیتی (۰ تا ۲۵۵) هست که در هنگام ارسال و دریافت، به‌عنوان نوعی "شناسه شبکه" بین دو دستگاه LoRa عمل می‌کنه .

فقط دستگاه‌هایی که Sync Word یکسانی دارن می‌تونن پیام‌های همدیگه رو ببینن و دریافت کنن .

- مثل آیدی شبکه عمل می‌کنه (شبیه SSID در WiFi) باعث میشه دستگاه‌های LoRa در اطراف، تداخل نکنند.

مشکلات و چالش ها:

۱) مشکل: ارتباط یک‌طرفه بین فرستنده و گیرنده

توضیح :ابتدا فقط یک ESP32 پیام می‌فرستاد و دیگری فقط دریافت می‌کرد.

راه‌حل :کدها به‌گونه‌ای بازنویسی شدند که هر دو برد هم ارسال‌کننده و هم دریافت‌کننده باشند، با استفاده از فلگ و بررسی موجود بودن پیام جدید با `LoRa.parsePacket()`

۲) مشکل: ارسال پاسخ به پیام دریافتی

توضیح :سیستم فقط پیام می‌فرستاد و بعد از دریافت پیام، جوابی ارسال نمی‌کرد.

راه‌حل :در کد دریافت، پس از دریافت پیام، ارسال یک پیام "ack" یا پاسخ در نظر گرفته شد.

۳) سوال: چرا در فرستنده `LoRa.receive()` نداریم؟

پاسخ :در حالت عادی دریافت نیازی به این تابع ندارد مگر از **اینترراپت** برای دریافت استفاده شود. در آن صورت باید ماژول را در حالت دریافت نگه داریم `(LoRa.receive())` ، مخصوصاً زمانی که ارسال پیام به‌صورت دستی انجام شود و لازم باشد پس از ارسال دوباره در حالت دریافت قرار بگیرد.

تحلیل سمت فرستنده:

ارسال پیام به‌صورت **دوره‌ای (هر ۱ ثانیه)** بدون استفاده از `delay()`یا اجرای دستی در حلقه‌ی `loop()`، و در عین حال حفظ توانایی **دریافت پیام پاسخ** از گیرنده.

استفاده از esp_timer در ESP32 :

esp_timerیک سیستم **تایمر سطح پایین و دقیق** در ESP32 است که توسط FreeRTOS مدیریت می‌شود و قابلیت اجرای تابع مشخصی را در بازه‌های زمانی مشخص دارد.

```
const esp_timer_create_args_t periodic_timer_args = {
  .callback = &onTimerCallback,
  .arg = nullptr,
  .dispatch_method = ESP_TIMER_TASK,
  .name = "my_periodic_timer"
};

esp_timer_create(&periodic_timer_args, &periodic_timer);
esp_timer_start_periodic(periodic_timer, 1000000); // 1s =
1,000,000 µs
```

esp_timer_create_args_t مشخص می‌کند که تابع چه چیزی را در چه شرایطی و با چه نامی اجرا کند.

esp_timer_start_periodic() به تایمر می‌گوید که هر ۱,۰۰۰,۰۰۰ میکروثانیه (۱ ثانیه) تابع مورد نظر را اجرا کند.

❖ **چرا LoRa.receive() در این کد وجود ندارد؟**

تابع LoRa.receive() فقط در پروژه‌هایی استفاده می‌شود که ماژول LoRa در **حالت گیرنده مداوم (RX mode)** باشد، مثل وقتی از **اینترراپت یا polling گیرنده** استفاده می‌کنیم. در حقیقت بعد از endpacket() ماژول در حالت standby قرار میگیرد. (RX کامل نه)

اما در این کد، چون بعد از هر ارسال، کتابخانه به‌طور پیش‌فرض ماژول را روی حالت آماده دریافت قرار می‌دهد، نیازی به LoRa.receive() نیست.

به همین دلیل همزمان با ارسال دوره‌ای پیام، قابلیت دریافت هم حفظ شده است.

تحلیل سمت گیرنده:

ایجاد سیستمی که در نقش گیرنده اصلی عمل کند، اما هر زمان که پیامی دریافت کرد، بتواند پاسخ مناسبی بفرستد و سپس مجدداً به حالت دریافت بازگردد.

۱) استفاده از وقفه سخت‌افزاری(Interrupt) :

```
attachInterrupt(digitalPinToInterrupt(DIO0), onReceiveInterrupt, RISING);
```

پین DIO0 ماژول LoRa هنگامی که داده‌ای دریافت شد **به لبه‌ی بالا (RISING)** تغییر می‌کند.

با فعال‌سازی این وقفه، تابع onReceiveInterrupt() اجرا می‌شود و تنها کاری که می‌کند این است که یک فلگ (packetReceived)را فعال می‌کند.

مزیت اصلی این روش: نیازی به بررسی دائمی در loop() نیست - CPU تا زمان نیاز به پردازش درگیر نمی‌شود - صرفه‌جویی در منابع.

۲) فعال‌سازی دریافت دائم LoRa :

```
LoRa.receive();
```

این دستور ماژول LoRa را به حالت دریافت دائم می‌برد.(Continues RX)

در این حالت، ماژول منتظر دریافت بسته است و پس از دریافت، وقفه روی DIO0 ایجاد می‌شود.

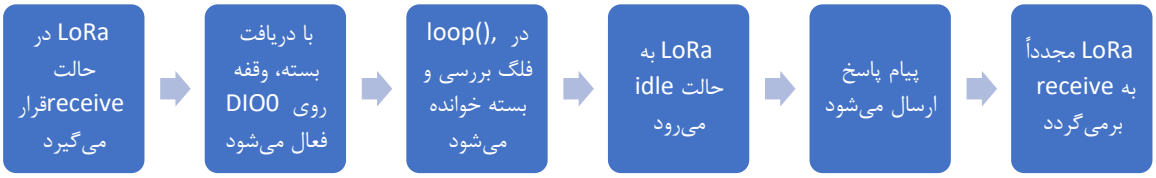
۳) خارج شدن از حالت دریافت برای ارسال پاسخ:

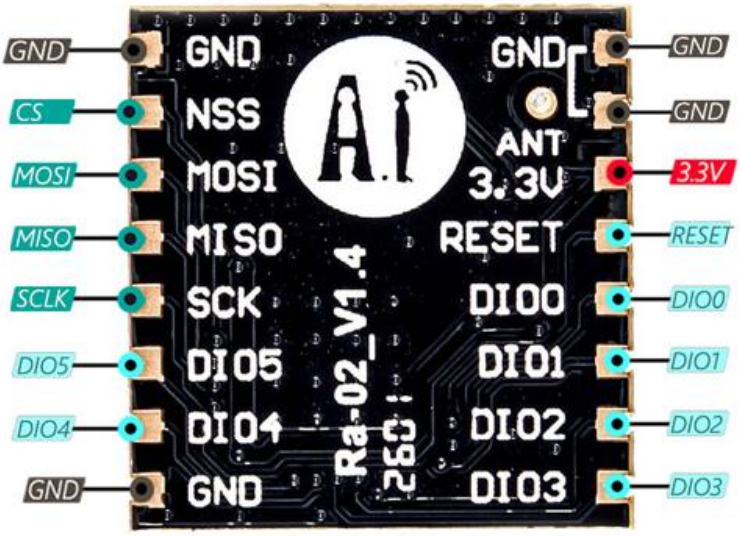
```
LoRa.idle(); توقف حالت  RX برای ارسال
```

حتماً قبل از beginPacket() باید LoRa را از حالت receive خارج کرد.

این کار با LoRa.idle() انجام می‌شود. در غیر این صورت، ارسال انجام نمی‌شود یا به درستی صورت نمی‌گیرد.

پس از ارسال پاسخ، ماژول باید **دوباره به حالت دریافت برگردد** تا پیام‌های بعدی را دریافت کند.





پین‌های **DIO0 تا DIO5** برای **خروجی سیگنال‌های داخلی ماژول لورا** هستند که به میکروکنترلر اطلاع بدن یک اتفاق خاص در ماژول افتاده؛ مثلاً :

- **دیتا دریافت شده**
- **فرستادن دیتا تموم شده**
- **ورود به حالت standby**
- **خطاهای داخلی**

جزئیات کاربرد پین‌ها :

پین	کاربرد اصلی (قابل تغییر در رجیسترها)
DIO0	نشان‌دهنده‌ی **پایان دریافت** یا **پایان ارسال دیتا** (interrupt)
DIO1	نشان‌دهنده‌ی **کشف preamble** یا **timeout**
DIO2	برای **FIFO levels** یا **frequency hopping**
DIO3	نشانگر تنظیمات (channel activity detection) **CAD done**
DIO4و DIO5	به ندرت استفاده می‌شن، قابلیت تنظیم برای شرایط خاص دارن

مثلاً وقتی دیتا از ماژول LoRa دریافت شد، پین DIO0 یک پالس منطقی (interrupt) به آردوینو می‌فرسته، تا MCU بدونه وقتشه بره دیتا رو از رجیستر FIFO بخونه.