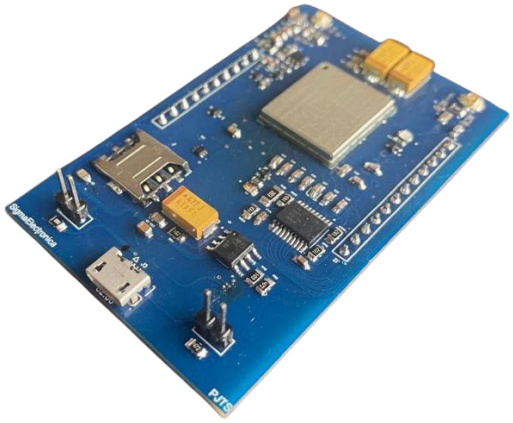
TARJETA EG800AK-LA

Módulo de desarrollo 4G LTE GNSS

Código	Tarjeta EG800AK-LA	**Versión**	[1.2]
Fecha	[08/08/2026]	**Elaboró**	Sigma Electronica LTDA

1. DESCRIPCIÓN GENERAL

La tarjeta de desarrollo celular LTE diseñada a partir del módulo Quectel EG800AK-LA ofrece una plataforma robusta y versátil para la implementación de comunicaciones móviles de banda ancha con posicionamiento satelital integrado. Esta tarjeta integra el módulo EG800AK-LA y está equipada con un regulador de voltaje conmutado, lo que permite su alimentación en un rango amplio y estable (4V, mínimo 2A), asegurando un funcionamiento confiable durante los picos de transmisión LTE. Incluye conectores U.FL duales para antena principal y GNSS, ranura para SIM estándar, puerto micro USB para depuración y actualización de firmware, y un LED indicador de estado de red. Para facilitar su uso en proyectos y pruebas, la tarjeta cuenta con dos headers de expansión de 10 y 12 pines que exponen las señales principales del módulo: alimentación 5V/GND, control ON/OFF y RESET, habilitación de antena GNSS, dos entradas ADC, y las interfaces UART principal (DBG) y auxiliar (AUX) con control de flujo completo (RTS/CTS/DTR/DCD/RI). Basada en tecnología LTE Cat.1, la tarjeta soporta bandas FDD-LTE y TDD-LTE multirregión, con velocidades de hasta 150 Mbps en descarga y 50 Mbps en subida, además de conectividad de respaldo según la variante. El consumo se optimiza mediante modos de bajo consumo (PSM, eDRX y sleep vía DTR/AT+QSCLK), haciendo viable su uso en aplicaciones alimentadas por batería. Es una herramienta ideal para el desarrollo de aplicaciones IoT como telemetría remota, rastreo GNSS, monitoreo industrial, medidores inteligentes y envío de datos a servidores vía HTTP/TCP.



2. ESPECIFICACIONES

Parámetros de operación del módulo.

Especificación	Detalle
Voltaje de alimentación	5 V (mínimo 2 A recomendado por picos de transmisión)
Rango de voltaje del módulo	3.0 V – 5.5 V (convertidor buck-boost integrado)
Consumo de corriente	600mA Variable según transmisión LTE (picos de corriente en TX)
Tecnología	LTE Cat 1 bis (single mode, sin 2G/3G)
Velocidad de datos (FDD)	10 Mbps (DL) / 5 Mbps (UL)
Velocidad de datos (TDD)	8.96 Mbps (DL) / 3.1 Mbps (UL)
Bandas LTE-FDD	B1/B2/B3/B4/B5/B7/B8/B12/B13/B14/B17/B18/B19/B20/B25/B26/B28/B66/B71
Bandas LTE-TDD	B34/B38/B39/B40/B41
Protocolos	TCP/UDP/HTTP/HTTPS/SSL/MQTT
Posicionamiento	GNSS integrado
Interfaz principal	UART (TX, RX, DTR), lógica 1.8 V a 5.0 V
Comandos AT	3GPP TS 27.007, 27.005 y comandos AT enriquecidos Quectel
Conectividad	Ranura SIM (Nano/Micro), USB micro, antenas U.FL (LTE + GNSS)
Temperatura de operación	-40 °C a +85 °C (extendida)

3.PINOUT

Distribución de pines y descripción funcional de cada conector.

PIN	Detalle
5V	Entrada de alimentación de **5 VDC / 2 A**. Conectar a una fuente externa de 5 V.
GND	Conectar al **GND de la fuente de alimentación**. Utilizar un segundo pin GND para establecer la referencia común con el microcontrolador.
TX MCU	Conectar al pin **RX del microcontrolador** para la comunicación serial UART.
RX MCU	Conectar al pin **TX del microcontrolador** para la comunicación serial UART.
ON OFF MCU	Entrada de control utilizada para **encender o apagar el módulo mediante un pulso** generado por el microcontrolador.

VDD MCU	Salida de alimentación para el microcontrolador externo.
RESET MCU	Señal de control utilizada para reiniciar el microcontrolador.
RST MCU	Entrada de control utilizada para realizar un reinicio del módulo.
CTS MCU	Señal CTS (Clear To Send) para el control de flujo por hardware de la comunicación UART.
DCD MCU	Señal DCD (Data Carrier Detect) que indica la detección de una conexión o portadora.
RI MCU	Señal RI (Ring Indicator) utilizada para indicar una llamada o evento de notificación.
DTR MCU	Señal DTR (Data Terminal Ready) utilizada para control del estado o modo de operación del módulo.
AUX TX	Salida de transmisión UART para comunicación con dispositivos o periféricos externos.
AUX RX	Entrada de recepción UART para comunicación con dispositivos o periféricos externos.
ADC 0	Entrada analógica para la medición de señales de voltaje.
ADC 1	Entrada analógica para la medición de señales de voltaje.
GNSS ANT EN	Señal de control para habilitar o deshabilitar la alimentación de la antena GNSS activa.
DBG TX	Salida UART utilizada para funciones de depuración y diagnóstico del módulo.
DBG RX	Entrada UART utilizada para funciones de depuración y diagnóstico del módulo.

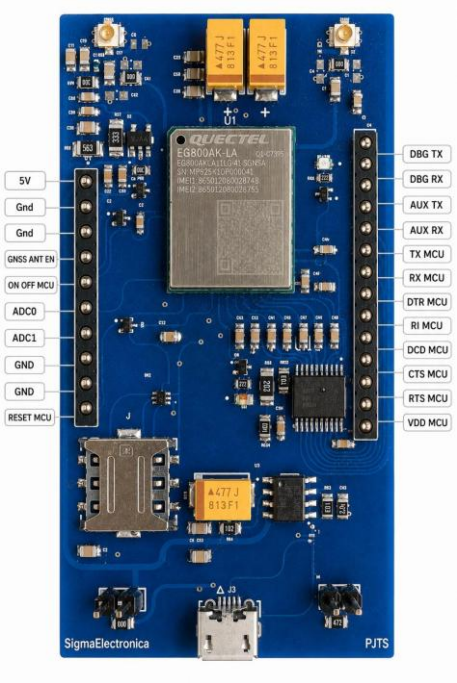


Figura 1: Pineout de la TARJETA EG800AK, autoría propia.

4. ESQUEMÁTICO

Diagrama esquemático de la placa.

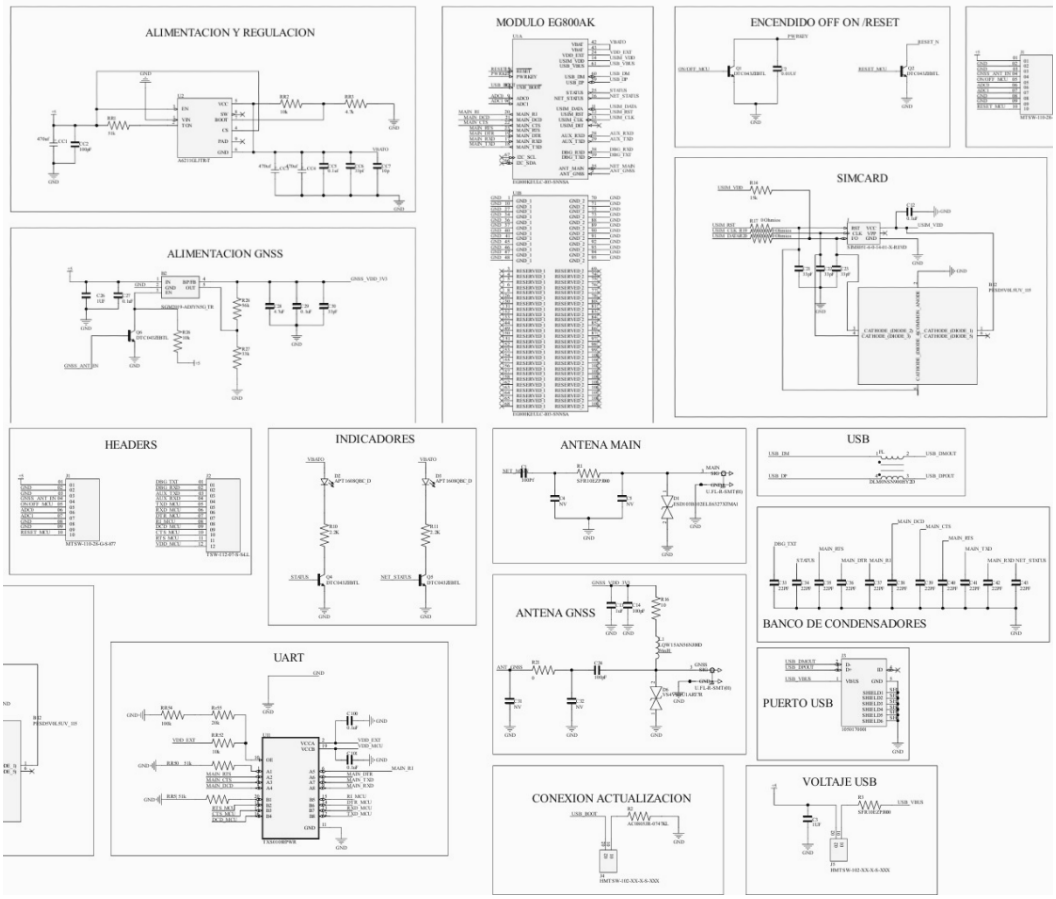


Figura 2: Esquemático TARJETA EG800AK, autoría propia.

5. DISEÑO 3D

Vista 3D y dimensiones físicas de la placa.

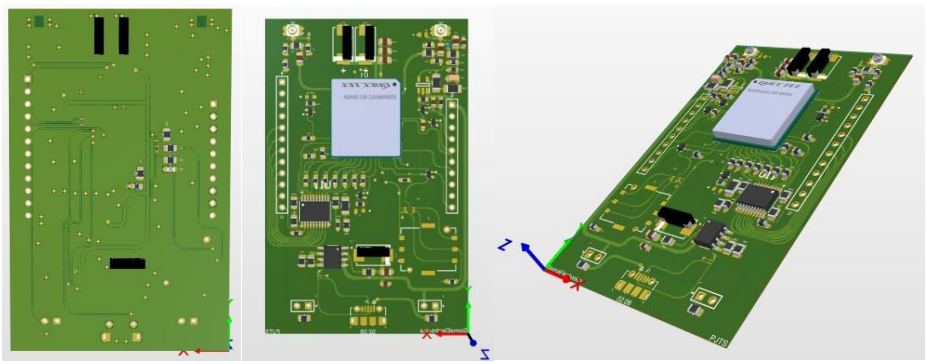


Figura 3: Diseña 3D TARJETA EG800AK, autoría propia.

6. GUÍA DE USO

El módulo está diseñado para ofrecer una integración Plug & Play con diferentes plataformas de desarrollo y microcontroladores, como Arduino, ESP32, Raspberry Pi, Raspberry Pi Pico, entre otros.

Dependiendo de los requerimientos del proyecto o prototipo, es posible utilizar conexiones adicionales para aprovechar las diferentes funciones y periféricos disponibles en el módulo.

En esta sección se presenta la conexión básica mediante la interfaz Main UART, la cual proporciona una configuración sencilla y adecuada como punto de partida para la mayoría de aplicaciones.

A continuación, se muestran ejemplos de conexión del módulo con un ESP32 y un Arduino Uno.

DIAGRAMA ESP32

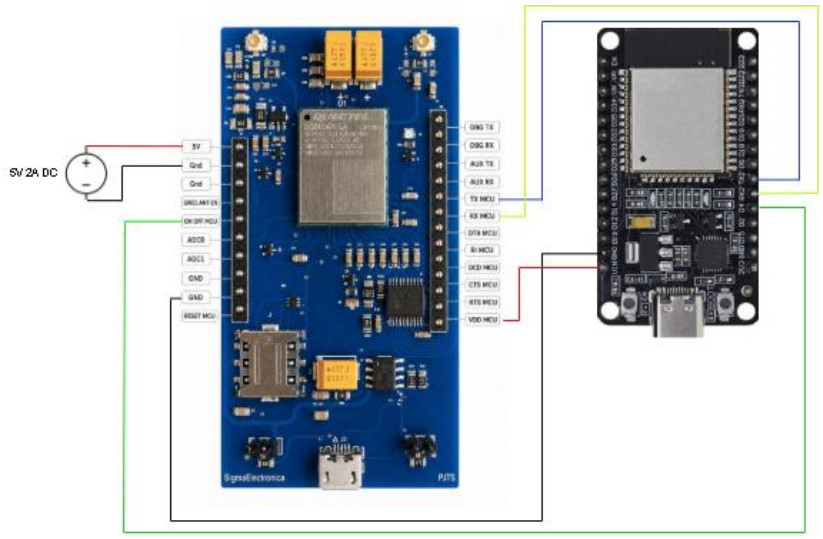


Figura 4: Diagrama TARJETA EG800AK Con ESP32, autoría propia.

DIAGRAMA ARDUINO UNO

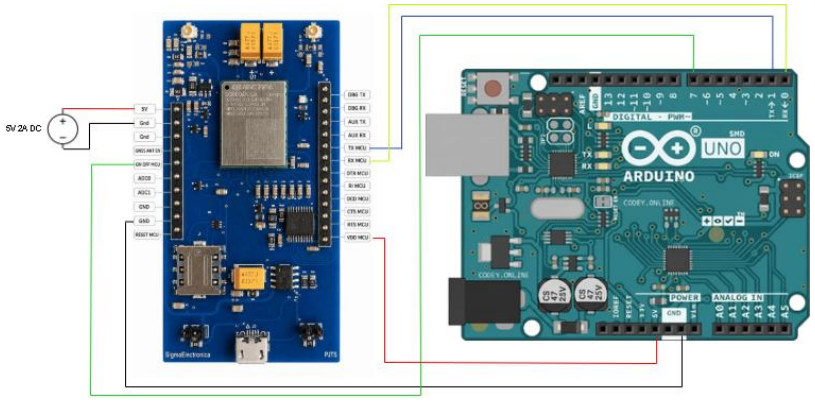


Figura 5: Diagrama TARJETA EG800AK Con ARDUINO UNO SMD, autoría propia.

7. PROGRAMACIÓN Y COMANDOS

A continuación, se presentan los sketches de referencia para ESP32 utilizados en las pruebas de fábrica y control de energía del módulo EG800AK-LA.

7.1 Prueba de fábrica de hardware (EG800AK_FACTORY_TEST.ino)

Verifica la comunicación UART con el módem, lectura de entradas ADC, control de GNSS_EN, señales DTR/RTS/CTS y el puerto auxiliar (AUX).

```
/*
 EG800AK_FACTORY_TEST.ino
 Prueba básica de hardware ESP32 + EG800AK
*/

#include <HardwareSerial.h>

#define MODEM_TX 17
#define MODEM_RX 16
#define MODEM_PWRKEY 4

#define GNSS_EN 14
#define ADC0_PIN 34
#define ADC1_PIN 35

#define AUX_TX 26
#define AUX_RX 27

#define DTR_PIN 25
#define RTS_PIN 33
#define CTS_PIN 32

HardwareSerial LTE(1);
HardwareSerial AUX(2);

void powerOnModem(){
  pinMode(MODEM_PWRKEY,OUTPUT);
  digitalWrite(MODEM_PWRKEY,LOW);
  delay(1000);
  digitalWrite(MODEM_PWRKEY,HIGH);
  delay(3000);
  digitalWrite(MODEM_PWRKEY,LOW);
  delay(8000);
}

String sendAT(String cmd,uint32_t timeout=3000){
  while(LTE.available()) LTE.read();
  LTE.println(cmd);
  String r="";
  uint32_t t=millis();
  while(millis()-t<timeout){
    while(LTE.available()) r+=(char)LTE.read();
  }
  return r;
}

void setup(){
```

```
  Serial.begin(115200);
  LTE.begin(115200,SERIAL_8N1,MODEM_RX,MODEM_TX);
  AUX.begin(115200,SERIAL_8N1,AUX_RX,AUX_TX);

  pinMode(GNSS_EN,OUTPUT);
  pinMode(DTR_PIN,OUTPUT);
  pinMode(RTS_PIN,OUTPUT);
  pinMode(CTS_PIN,INPUT_PULLUP);

  powerOnModem();

  Serial.println("=== EG800AK FACTORY TEST ===");
  Serial.println(sendAT("AT"));
  Serial.println(sendAT("ATI"));
}

void loop(){

  Serial.println("---- ADC ----");
  Serial.printf("ADC0=%d\n",analogRead(ADC0_PIN));
  Serial.printf("ADC1=%d\n",analogRead(ADC1_PIN));

  digitalWrite(GNSS_EN,!digitalRead(GNSS_EN));
  Serial.printf("GNSS_EN=%d\n",digitalRead(GNSS_EN));

  digitalWrite(DTR_PIN,!digitalRead(DTR_PIN));
  digitalWrite(RTS_PIN,!digitalRead(RTS_PIN));

  Serial.printf("CTS=%d DTR=%d RTS=%d\n",
      digitalRead(CTS_PIN),
      digitalRead(DTR_PIN),
      digitalRead(RTS_PIN));

  AUX.println("SIGMA TEST");
  delay(100);
  while(AUX.available()){
    Serial.write(AUX.read());
  }

  Serial.println(sendAT("AT+CSQ"));
  Serial.println("------------------------");
  delay(3000);
}
```

7.2 Control de encendido, apagado y modos de bajo consumo — v1

Menú interactivo por puerto serie para encender/apagar el módem, alternar CFUN, activar sleep (QSCLK) y probar el Deep Sleep del ESP32.

```
#include <HardwareSerial.h>
#include "esp_sleep.h"

#define MODEM_TX 17
#define MODEM_RX 16
#define PWRKEY 4
#define DTR 25

HardwareSerial LTE(1);

String sendAT(String c,int t=3000){
 while(LTE.available()) LTE.read();
 LTE.println(c);
 String r="";
 unsigned long s=millis();
 while(millis()-s<t){
  while(LTE.available()) r+=(char)LTE.read();
 }
 return r;
}
void modemOn(){
 pinMode(PWRKEY,OUTPUT);
 digitalWrite(PWRKEY,LOW);delay(1000);
 digitalWrite(PWRKEY,HIGH);delay(3000);
 digitalWrite(PWRKEY,LOW);delay(8000);
}
void modemOff(){
 digitalWrite(PWRKEY,HIGH);
 delay(2500);
 digitalWrite(PWRKEY,LOW);
 delay(5000);
}
void setup(){
 Serial.begin(115200);
 LTE.begin(115200,SERIAL_8N1,MODEM_RX,MODEM_TX);
 pinMode(DTR,OUTPUT);
 pinMode(PWRKEY,OUTPUT);
 Serial.println("\nEG800AK POWER TEST");
 Serial.println("1 ON");
 Serial.println("2 OFF");
 Serial.println("3 REBOOT");
 Serial.println("4 AT");
 Serial.println("5 CFUN=0");
 Serial.println("6 CFUN=1");
 Serial.println("7 QSCLK");
 Serial.println("8 WAKE");
 Serial.println("9 ESP SLEEP 10s");
 Serial.println("0 ESP RESTART");
}
void loop(){
 if(!Serial.available()) return;
 char op=Serial.read();
 switch(op){
  case '1': modemOn(); Serial.println("MODEM ON"); break;
  case '2': modemOff(); Serial.println("MODEM OFF"); break;
  case '3': modemOff(); delay(2000); modemOn(); break;
  case '4': Serial.println(sendAT("AT")); break;
  case '5': Serial.println(sendAT("AT+CFUN=0",10000)); break;
  case '6': Serial.println(sendAT("AT+CFUN=1",10000)); break;
  case '7':
    Serial.println(sendAT("AT+QSCLK=1"));
    digitalWrite(DTR,HIGH);
```

```
    Serial.println("Sleep solicitado");
    break;
  case '8':
    digitalWrite(DTR,LOW);
    delay(100);
    Serial.println(sendAT("AT"));
    break;
  case '9':
    Serial.println("ESP Deep Sleep");
    delay(500);
    esp_sleep_enable_timer_wakeup(10000000ULL);
    esp_deep_sleep_start();
    break;
  case '0':
    ESP.restart();
    break;
 }
}
```

7.3 Ejemplo funcional — Envío de datos por HTTP POST (Webhook) con datos simulados

Nota: Ejemplo funcional que enciende el módulo EG800AK, registra la sesión LTE (SIM, red, APN, PDP context), abre un socket TCP y envía una petición HTTP POST con datos simulados en formato JSON hacia un endpoint webhook, usando comandos AT nativos (AT+QIOPEN, AT+QISEND, AT+QICLOSE) sin librerías intermedias.
PORFAVOR CAMBIAR ESTAS VARIABLE EN EL CODIGO.

Parámetro en el código	Línea / variable	Qué cambiar
URL del webhook	WEBHOOK_ID = "68110b84-90d6-46d8-87a5-c92f5f2c725e"	Es un endpoint de prueba en webhook.site, expira. Reemplazar por el endpoint real de destino (servidor propio, backend, plataforma IoT).
APN	AT+CGDCONT=1,"IP","internet.movistar.com.co"	Hardcodeado para SIM Movistar Colombia. Cambiar según el operador de la SIM que se use en campo (Claro, Tigo, etc. tienen APN distinto).
TX MCU	"webhook.site" en AT+QIOPEN y en el header Host:	Debe coincidir con el dominio/IP real del servidor final.
RX MCU	80 en AT+QIOPEN	Es HTTP plano. Si el servidor final requiere HTTPS, hay que migrar a AT+QSSLOPEN/contexto SSL (el EG800AK sí soporta TLS vía comandos QSSL, este ejemplo no lo usa).
ON OFF MCU	MODEM_TX=17, MODEM_RX=16, MODEM_PWRKEY=4	Verificar que coincidan con el diseño físico de la placa donde se use (pueden cambiar si es otra revisión de PCB).
VDD MCU	delay(15000) en loop()	Ajustar según el plan de datos disponible y el consumo energético permitido.
RESET MCU	temperature, humidity, voltage, signal (generados con random())	Son simulados. Reemplazar por lecturas reales de sensores antes de integrarlo al producto final.

```
/*
=========================================================
   ESP32 + EG800AK LTE WEBHOOK FUNCIONAL FINAL
=========================================================

FUNCION:
- Conecta LTE Movistar
- Abre TCP real
- Envia HTTP POST real
- Envia JSON a webhook.site

=========================================================
WEBHOOK
=========================================================

https://webhook.site/68110b84-90d6-46d8-87a5-c92f5f2c725e

=========================================================
CONEXIONES
=========================================================

EG800AK        ESP32
-------------------------
TX         -> GPIO16
RX         -> GPIO17
GND        -> GND
```

```
PWRKEY     -> GPIO4

=========================================================
IMPORTANTE
=========================================================

FUENTE:
4V / 2A MINIMO

=========================================================
*/

#include <HardwareSerial.h>
#include "soc/soc.h"
#include "soc/rtc_cntl_reg.h"

#define MODEM_TX       17
#define MODEM_RX       16
#define MODEM_PWRKEY    4

HardwareSerial LTE(1);

String WEBHOOK_ID = "68110b84-90d6-46d8-87a5-c92f5f2c725e";

int counter = 0;

// =========================================================
// POWER ON
// =========================================================

void powerOnModem() {

  Serial.println("\n[POWER] ENCENDIENDO MODEM");

  pinMode(MODEM_PWRKEY, OUTPUT);

  digitalWrite(MODEM_PWRKEY, LOW);
  delay(1000);

  digitalWrite(MODEM_PWRKEY, HIGH);
  delay(3000);

  digitalWrite(MODEM_PWRKEY, LOW);

  delay(8000);

  Serial.println("[POWER] MODEM LISTO");
}

// =========================================================
// SEND AT
// =========================================================

String sendAT(String cmd, int timeout = 3000) {

  String response = "";

  Serial.println("\n================================================");
  Serial.print(">> ");
  Serial.println(cmd);

  while (LTE.available()) LTE.read();

  LTE.println(cmd);

  long startTime = millis();

  while ((millis() - startTime) < timeout) {
    while (LTE.available()) {
```

```
      char c = LTE.read();
      response += c;
    }
  }

  Serial.println(response);

  return response;
}

// ====================================================
// WAIT FOR
// ====================================================

String waitFor(String target, int timeout = 10000) {

  String response = "";
  long t = millis();

  while ((millis() - t) < timeout) {
    while (LTE.available()) {
      char c = LTE.read();
      response += c;
      Serial.write(c);
      if (response.indexOf(target) != -1) {
        return response;
      }
    }
  }

  return response;
}

// ====================================================
// INIT LTE
// ====================================================

bool initLTE() {

  Serial.println("\n================================================");
  Serial.println("INICIANDO LTE");
  Serial.println("================================================");

  sendAT("AT");
  sendAT("ATE0");
  sendAT("AT+CMEE=2");

  // ====================================================
  // SIM — reintentar hasta 10 veces
  // ====================================================

  String sim = "";
  for (int i = 0; i < 10; i++) {
    sim = sendAT("AT+CPIN?", 5000);
    if (sim.indexOf("READY") != -1) break;
    Serial.println("[SIM] Esperando... intento " + String(i+1));
    delay(2000);
  }

  if (sim.indexOf("READY") == -1) {
    Serial.println("[ERROR] SIM");
    return false;
  }

  Serial.println("[OK] SIM READY");

  sendAT("AT+CSQ");

  // ====================================================
```

```
  // CEREG — esperar stat=1 o stat=5
  // ====================================================

  Serial.println("\n[RED] Esperando registro LTE...");

  bool registered = false;
  for (int i = 0; i < 15; i++) {
    String reg = sendAT("AT+CEREG?");
    if (reg.indexOf(",1") != -1 || reg.indexOf(",5") != -1) {
      registered = true;
      break;
    }
    Serial.println("[RED] No registrado, esperando... (" + String(i+1) + "/15)");
    delay(2000);
  }

  if (!registered) {
    Serial.println("[ERROR] No se registro en LTE");
    return false;
  }

  Serial.println("[OK] LTE REGISTRADO");

  sendAT("AT+QNWINFO");

  // ====================================================
  // APN
  // ====================================================

  sendAT("AT+CGDCONT=1,\"IP\",\"internet.movistar.com.co\"");

  // ====================================================
  // ATTACH — timeout 30s (manual dice max 140s)
  // ====================================================

  sendAT("AT+CGATT=1", 30000);

  String attach = sendAT("AT+CGATT?");

  if (attach.indexOf("+CGATT: 1") == -1) {
    Serial.println("[ERROR] NO LTE");
    return false;
  }

  Serial.println("[OK] LTE ATTACHED");

  // ====================================================
  // ACTIVATE PDP
  // ====================================================

  sendAT("AT+CGACT=1,1", 15000);

  // ====================================================
  // GET IP
  // ====================================================

  String ip = sendAT("AT+CGPADDR=1");

  if (ip.indexOf(".") == -1) {
    Serial.println("[ERROR] NO IP");
    return false;
  }

  Serial.println("[OK] LTE INTERNET");

  return true;
}

// ====================================================
```

```
// SEND WEBHOOK
// ================================================

bool sendWebhook() {

  Serial.println("\n================================================");
  Serial.println("ENVIANDO DATOS LTE");
  Serial.println("================================================");

  sendAT("AT+CSQ");

  // ================================================
  // JSON
  // ================================================

  float temperature = random(200, 350) / 10.0;
  float humidity    = random(400, 900) / 10.0;
  float voltage     = random(360, 420) / 100.0;
  int   signal      = random(10, 31);

  counter++;

  String json =
    "{"
    "\"temperature\":" + String(temperature, 1) + ","
    "\"humidity\":"    + String(humidity, 1)    + ","
    "\"voltage\":"     + String(voltage, 2)     + ","
    "\"signal\":"      + String(signal)         + ","
    "\"counter\":"     + String(counter)        +
    "}";

  Serial.println("\nJSON:");
  Serial.println(json);

  // ================================================
  // CLOSE SOCKET PREVIO
  // ================================================

  sendAT("AT+QICLOSE=0", 3000);
  delay(2000);

  // ================================================
  // OPEN TCP
  // 1. Enviar AT+QIOPEN → esperar OK
  // 2. Esperar URC +QIOPEN: 0,0 por separado
  // ================================================

  while (LTE.available()) LTE.read();

  LTE.println("AT+QIOPEN=1,0,\"TCP\",\"webhook.site\",80,0,1");

  Serial.println("\n================================================");
  Serial.println(">> AT+QIOPEN=1,0,\"TCP\",\"webhook.site\",80,0,1");

  String okResp = waitFor("OK", 5000);

  if (okResp.indexOf("ERROR") != -1) {
    Serial.println("[ERROR] QIOPEN rechazo el comando");
    return false;
  }

  Serial.println("[TCP] Esperando confirmacion de conexion...");

  String urc = "";
  long tUrc = millis();

  while ((millis() - tUrc) < 20000) {
    while (LTE.available()) {
      char c = LTE.read();
```

```
    urc += c;
    Serial.write(c);
  }
  if (urc.indexOf("+QIOPEN:") != -1 &&
      urc.length() > urc.indexOf("+QIOPEN:") + 12) {
    break;
  }
}

Serial.println("\n[TCP] URC: " + urc);

if (urc.indexOf("+QIOPEN: 0,0") == -1) {
  Serial.println("[ERROR] TCP OPEN FAIL");
  Serial.println("[INFO] err=3 DNS fallo | err=5 sin internet | err=2 rechazado");
  sendAT("AT+QICLOSE=0", 3000);
  return false;
}

Serial.println("[OK] TCP CONNECTED");
delay(500);

// ================================================
// HTTP REQUEST
// ================================================

String request =
  "POST /" + WEBHOOK_ID + " HTTP/1.1\r\n"
  "Host: webhook.site\r\n"
  "User-Agent: ESP32-EG800AK\r\n"
  "Content-Type: application/json\r\n"
  "Connection: close\r\n"
  "Content-Length: " + String(json.length()) + "\r\n\r\n" +
  json;

Serial.println("\n================================================");
Serial.println("HTTP REQUEST");
Serial.println("================================================");
Serial.println(request);

// ================================================
// AT+QISEND
// ================================================

int len = request.length();

String sendCmd = "AT+QISEND=0," + String(len);

Serial.println("\n================================================");
Serial.print(">> ");
Serial.println(sendCmd);

while (LTE.available()) LTE.read();

LTE.println(sendCmd);

// ================================================
// ESPERAR PROMPT >
// ================================================

bool gotPrompt = false;
long t = millis();

while ((millis() - t) < 10000) {
  while (LTE.available()) {
    char c = LTE.read();
    Serial.write(c);
    if (c == '>') {
      gotPrompt = true;
      break;
```

```
      }
    }
    if (gotPrompt) break;
  }

  if (!gotPrompt) {
    Serial.println("\n[ERROR] NO PROMPT");
    sendAT("AT+QICLOSE=0", 3000);
    return false;
  }

  Serial.println("\n[OK] SEND PROMPT");

  // ================================================
  // ENVIAR HTTP — una sola vez
  // ================================================

  LTE.print(request);

  Serial.println("\n[HTTP ENVIADO]");

  // ================================================
  // SERVER RESPONSE
  // ================================================

  Serial.println("\n================================================");
  Serial.println("SERVER RESPONSE");
  Serial.println("================================================");

  String response = "";
  long timeout = millis();

  while ((millis() - timeout) < 20000) {
    while (LTE.available()) {
      char c = LTE.read();
      response += c;
      Serial.write(c);
      timeout = millis();
    }
  }

  // ================================================
  // CLOSE SOCKET
  // ================================================

  sendAT("AT+QICLOSE=0", 3000);

  // ================================================
  // VERIFY
  // ================================================

  if (response.indexOf("HTTP/1.1") != -1) {
    Serial.println("\n[OK] WEBHOOK ENVIADO");
    return true;
  }

  Serial.println("\n[ERROR] WEBHOOK FAILED");
  return false;
}

// ================================================
// SETUP
// ================================================

void setup() {

  WRITE_PERI_REG(RTC_CNTL_BROWN_OUT_REG, 0);

  Serial.begin(115200);
```

```
  delay(3000);

  Serial.println("\n================================================");
  Serial.println(" EG800AK LTE WEBHOOK FINAL");
  Serial.println("================================================");

  randomSeed(micros());

  powerOnModem();

  LTE.begin(115200, SERIAL_8N1, MODEM_RX, MODEM_TX);
  delay(3000);

  bool ok = initLTE();

  if (!ok) {
    Serial.println("\n================================================");
    Serial.println("LTE FAILED");
    Serial.println("================================================");
    while (1);
  }

  Serial.println("\n================================================");
  Serial.println("SYSTEM READY");
  Serial.println("================================================");
}

// ======================================================
// LOOP
// ======================================================

void loop() {

  bool ok = sendWebhook();

  if (ok) {
    Serial.println("\n[SUCCESS]");
  } else {
    Serial.println("\n[FAILED]");
  }

  Serial.println("\nWAIT 15 SECONDS...\n");
  delay(15000);
}
```

8. APLICACIONES

El módulo EG800AK-LA está orientado a soluciones de conectividad IoT que requieren comunicación celular LTE Cat 1 de bajo consumo, con soporte adicional de GNSS. Su diseño compacto y su configuración de doble SIM lo hacen apto para escenarios donde se requiere redundancia de operador o continuidad de servicio ante fallas de cobertura.

- Telemetría y monitoreo remoto de variables (temperatura, humedad, voltaje, sensores industriales).
- Sistemas de rastreo y localización GNSS para activos móviles.

- Gateways IoT con envío de datos a plataformas en la nube (HTTP/HTTPS, MQTT, WebSocket).
- Soluciones de respaldo de conectividad con doble SIM (operador principal + operador de respaldo).
- Dispositivos portátiles o embebidos donde el tamaño reducido de la placa es un requisito de diseño.
- Equipos que requieren configuración o diagnóstico local vía puerto Micro USB, sin depender únicamente de comandos AT por UART.

9. ROADMAP

El desarrollo de este módulo continuará enfocado en mejorar la conectividad, mantener actualizado el firmware base de Quectel y optimizar el tamaño físico de la placa sin sacrificar funcionalidad.

- **Conectividad:** evaluación y prueba de mejoras en estabilidad de registro LTE y manejo de roaming entre operadores.
- **Actualizaciones de firmware:** seguimiento activo a los releases de firmware de Quectel para el chipset EG800AK, con plan de actualización periódica y validación de compatibilidad con los comandos AT usados en los sketches de fábrica.
- **Doble SIM:** implementación y validación funcional del soporte de doble SIM en la placa, permitiendo selección/conmutación entre operadores para mayor disponibilidad de servicio
- **Optimización de tamaño:** rediseño de la placa (PJTS) orientado a reducir sus dimensiones físicas, manteniendo la doble SIM y el resto de periféricos (GNSS, ADC, puerto AUX).
- **Modo Micro USB:** activación del modo de configuración/diagnóstico vía puerto Micro USB del módulo, como alternativa a la comunicación por UART, para facilitar pruebas de fábrica y actualización de firmware en campo.

10. CONTACTO

[Pedro Jose Torres Sequera] · [ingenieria@sigmaelectronica.net] · [3153291044]

Sigma Electrónica — www.sigmaelectronica.com