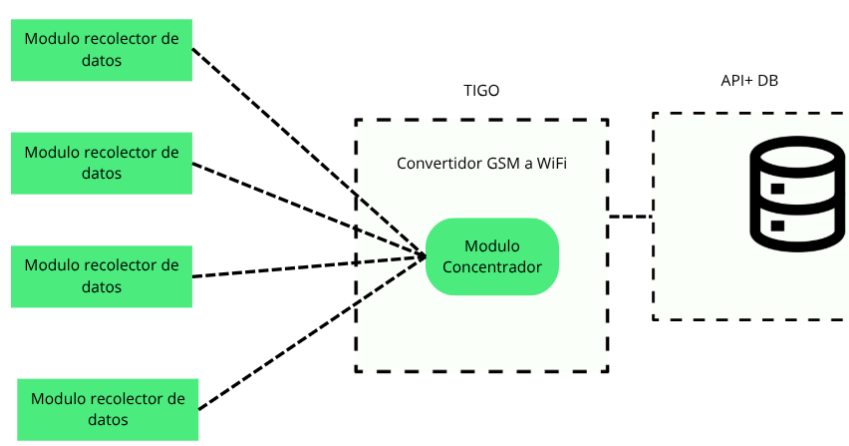


Tiveg: Integración de Concentrador SMS-API

1. Descripción General



El **Concentrador TIVEG** es un sistema modular para ESP32 con módem SIM800L (TTGO T-Call o DualMCU ONE). Está diseñado para:

- **Recepción de SMS en formato PDU/UDH** con reconstrucción de mensajes largos.
- **Sanitización de datos** y conversión a JSON.
- **Envío confiable de datos vía HTTP/HTTPS** hacia APIs.
- **Monitoreo en tiempo real** mediante modo consola.
- **Prevención de saturación de memoria** mediante limpieza automática.

Se integra fácilmente en aplicaciones IoT críticas donde la confiabilidad es fundamental.

2. Estructura de Archivos

Archivo	Descripción
secrets.h	Contiene credenciales privadas (WiFi, URLs API). No versionar.
config.h	Selección de tarjeta, pines, baudrate, configuración HTTPS.
wifi_config.h	Funciones de conexión, reconexión y escaneo WiFi.
http_utils.h	Manejo de POST con cabeceras comunes, retries y soporte HTTPS.

SIM800SmsManager.* ver: Clase SIM800SmsManager	Envío confiable de SMS y ejecución de comandos AT.
SmsPduParser.*	Conversión de PDU a texto plano y detección de multipartes.
SmsConcatManager.*	Ensamblado de SMS largos, manejo de timeouts y borrado de partes.
JsonQueue.h	Cola FIFO en RAM para almacenar mensajes JSON.
main.ino	Lógica principal: recepción SMS, envío API, barridos y modo consola.

3. Configuración de `secrets.h`

```

1  #pragma once
2
3  // ===== WiFi =====
4  #define WIFI_SSID      "MiRedWiFi"
5  #define WIFI_PASSWORD "MiPassword"
6
7  // ===== API =====
8  #define API_URL        "http://192.168.1.10:5000/api/data"
9  #define MIRROR_URL     ""    // opcional
10

```

Notas:

- Mantener este archivo fuera del control de versiones.

4. Configuración de `config.h`

Este archivo concentra las **constantes de configuración global** del sistema.

Permite elegir **qué placa de hardware se está usando** y **qué modo de comunicación HTTP/HTTPS emplear** para el envío de datos a la API.

```

1  // Selección de tarjeta
2  #define USE_BOARD 0    // 0 = TTGO T-Call, 1 = DualMCU ONE
3
4  // HTTPS
5  #define USE_HTTPS 1
6  #define HTTPS_INSECURE 1    // 0 = validar certificado, 1 = ignorarlo en
7
8  // SIM800L
9  #define SIM_BAUD 9600
10
11 #if USE_BOARD == 0
12     #define MODEM_RST      5
13     #define MODEM_PWKEY    4
14     #define MODEM_POWER_ON 23
15     #define MODEM_TX       27
16     #define MODEM_RX       26
17 #elif USE_BOARD == 1
18     #define MODEM_RST      21
19     #define MODEM_PWKEY    22
20     #define MODEM_POWER_ON 19
21     #define MODEM_TX       17

```

```

22 #define MODEM_RX      16
23 #endif
24
25 #define STATUS_LED 25
26

```

1. Selección de Tarjeta

```

1 // 0 = TTGO T-Call (ESP32 + SIM800L)
2 // 1 = DualMCU ONE (ESP32 + RP2040)
3 #define USE_BOARD 0
4

```

- **USE_BOARD 0** → **TTGO T-Call**

Usa los pines estándar del TTGO T-Call, que integra un **ESP32** con un **SIM800L** en la misma tarjeta.

- **USE_BOARD 1** → **DualMCU ONE**

Está pensada para la tarjeta interna desarrollada con **ESP32 + RP2040** y módem SIM800L.

En este caso, se redefinen los pines para coincidir con el diseño de la placa.

1.1 Configuración de Pines según la Tarjeta

Si seleccionas **TTGO T-Call** (**USE_BOARD 0**):

```

1 #define MODEM_RST      5
2 #define MODEM_PWKEY    4
3 #define MODEM_POWER_ON 23
4 #define MODEM_TX       27
5 #define MODEM_RX       26
6


```

Si seleccionas **DualMCU ONE** (**USE_BOARD 1**):

```

1 #define MODEM_RST      21
2 #define MODEM_PWKEY    22
3 #define MODEM_POWER_ON 19
4 #define MODEM_TX       17
5 #define MODEM_RX       16
6

```

 **Nota:** Estos defines permiten al firmware saber qué pines usar para controlar y comunicarse con el SIM800L. Si el valor de **USE_BOARD** no coincide con tu hardware, el módem no arrancará correctamente.

2. Configuración de HTTP/HTTPS

```

1 // 0 = usar HTTP plano
2 // 1 = usar HTTPS (WiFiClientSecure)
3 #define USE_HTTPS 1
4

```

- **USE_HTTPS 0** → el envío de datos se hace en **HTTP sin cifrado**.

✓ Útil en pruebas rápidas dentro de una red local.

✗ No recomendable en producción (los datos viajan en claro).

- **USE_HTTPS 1** → el envío se hace en **HTTPS (cifrado TLS/SSL)** usando **WiFiClientSecure**.

✓ Obligatorio en producción si se conecta a un servidor en Internet.

2.1 Validación de Certificados

```
1 // Si USE_HTTPS = 1:
2 //    0 = validar certificado (producción)
3 //    1 = ignorar certificado (solo pruebas)
4 #define HTTPS_INSECURE 1
5
```

- **HTTPS_INSECURE 0** → valida los certificados SSL del servidor.
 - ✓ Configuración correcta en producción.
 - ✗ Requiere cargar el certificado válido en el ESP32.
 - **HTTPS_INSECURE 1** → ignora la validación del certificado.
 - ✓ Útil en pruebas, laboratorios o servidores con certificados autofirmados.
 - ✗ No seguro para producción: acepta cualquier certificado.
-

3. Configuración SIM800L y Baudrate

```
1 #define SIM_BAUD 9600
2
```

- Define la **velocidad de comunicación UART** entre el ESP32 y el SIM800L.
 - Por defecto se usa **9600 bps**, que es el valor estándar del SIM800L.
 - Si el módem se configuró a otra velocidad, debe actualizarse aquí.
-

4. LED de Estado

```
1 #define STATUS_LED 25
2
```

- Pin asignado al **LED de estado** en la placa.
 - Se utiliza para indicar que el sistema está encendido o para debug visual.
-

✓ En resumen:

- Para **cambiar de tarjeta**, modificar **#define USE_BOARD** a **0** (TTGO T-Call) o **1** (DuaIMCU ONE).
 - Para **cambiar el modo de conexión a la API**, usar **USE_HTTPS = 0** (HTTP) o **USE_HTTPS = 1** (HTTPS).
 - Si se usa HTTPS, decidir si se **validan certificados** (**HTTPS_INSECURE=0**) o se ignoran en pruebas (**HTTPS_INSECURE=1**).
-

5. Configuración de `wifi_config.h`

Funciones principales:

- **conectarWiFi()** :
 - Inicia en modo estación.
 - Imprime redes visibles.
 - Conecta a **WIFI_SSID / WIFI_PASSWORD** .
 - Timeout: 30 s.

- `checkWiFiLoop()` :
 - Monitorea estado.
 - Reconecta automáticamente si la conexión se pierde.

🔧 Recomendación: usar **red de 2.4 GHz** y hostname opcional `WIFI_HOSTNAME` .

6. Flujo de Operación

1. Encendido del módem (`powerOnBoard()`).
2. Conexión WiFi (`conectarWiFi()`).
3. Inicialización del SIM800L con comandos AT (`AT` , `AT+CMGF=0` , `AT+CNMI=2,1,0,0,0`).
4. Recepción de SMS mediante URC `+CMTI` .
5. Parseo PDU (`SmsPduParser`).
6. Reconstrucción multipartes (`SmsConcatManager`).
7. Sanitización JSON y push en `JsonQueue` .
8. Envío a API vía `http_utils.h` .
9. Sweep de seguridad cada 60 s (`sweepMessages()`).
10. Modo consola (`###`) para enviar AT manuales.

7. Notas Importantes

- **JsonQueue**: mensajes simultáneos en RAM.
- Los SMS reconstruidos se borran automáticamente de la memoria **ME**.
- En producción: `USE_HTTPS=1` , `HTTPS_INSECURE=0` .
- El SIM800L requiere fuente **4 V – 2 A picos**.
- El **modo consola** se activa con `###` por Serial.

Ver anexo: [Guía de uso – Modo Consola y Concentrador](#)

8. Problemas Comunes y Soluciones

Problema	Posible causa	Solución
No conecta WiFi	SSID incorrecto, red 5 GHz	Revisar <code>secrets.h</code> y usar red 2.4 GHz
SMS no recibidos	CNMI mal configurado	Confirmar <code>AT+CNMI=2,1,0,0,0</code>

API rechaza datos	JSON inválido o falta de headers	Revisar sanitización y cabeceras en <code>http_utils.h</code>
Respuestas AT vacías	Baudrate incorrecto	Ajustar <code>SIM_BAUD</code>
Caídas aleatorias	Fuente insuficiente	Asegurar >2 A con capacitores de desacoplo

9. Ejemplo de Uso Mínimo

¡Perfecto! 🚀

Aquí te dejo un **Ejemplo de Uso Mínimo** más realista y enfocado en el flujo **recepción de SMS** → **parseo** → **envío a API**, quitando la parte de `sendSmsReliable`.

📌 Ejemplo de Uso Mínimo – Recepción de SMS y envío a API

```

1  #include "secrets.h"
2  #include "config.h"
3  #include "wifi_config.h"
4  #include "http_utils.h"
5
6  #include "SIM800SmsManager.h"
7  #include "SmsPduParser.h"
8  #include "SmsConcatManager.h"
9  #include "JsonQueue.h"
10
11  HardwareSerial MODEM(2);
12  SIM800SmsManager smsMgr(MODEM);
13  SmsConcatManager concatMgr;
14  JsonQueue queueJson;
15
16  // Buffer temporal para URCs
17  static String modemLine;
18
19  void setup() {
20      Serial.begin(115200);
21      Serial.println("\n=== TIVE6 concentrador (ejemplo mínimo) ===");
22
23      // Encender módem y configurar UART
24      pinMode(MODEM_POWER_ON, OUTPUT);
25      pinMode(MODEM_PWKEY, OUTPUT);
26      pinMode(MODEM_RST, OUTPUT);
27      digitalWrite(MODEM_POWER_ON, HIGH);
28      delay(100);
29      digitalWrite(MODEM_RST, HIGH);
30      delay(100);
31      digitalWrite(MODEM_PWKEY, LOW);
32      delay(1200);
33      digitalWrite(MODEM_PWKEY, HIGH);
34      delay(3500);
35
36      MODEM.begin(SIM_BAUD, SERIAL_8N1, MODEM_RX, MODEM_TX);
37
38      // Conexión WiFi
39      conectarWiFi();
40
41      // Inicialización SIM800L
42      smsMgr.at("AT");

```

```

43 smsMgr.at("ATE0");
44 smsMgr.at("AT+CMEE=2");
45 smsMgr.at("AT+CMGF=0"); // modo PDU
46 smsMgr.at("AT+CPMS=\"ME\", \"ME\", \"ME\"");
47 smsMgr.at("AT+CNMI=2,1,0,0,0"); // URC +CMTI cuando llega SMS
48 smsMgr.at("AT+CSClk=0");
49 }
50
51 void loop() {
52     // --- Procesar URCS entrantes del módem ---
53     while (MODEM.available()) {
54         char c = MODEM.read();
55         if (c == '\n') {
56             modemLine.trim();
57             if (modemLine.startsWith("+CMTI")) {
58                 int cpos = modemLine.lastIndexOf(',');
59                 if (cpos > 0) {
60                     int idx = modemLine.substring(cpos + 1).toInt();
61                     String pdu;
62                     if (SmsPduParser::readPduAtIndex(MODEM, idx, pdu)) {
63                         SmsPduInfo info;
64                         if (SmsPduParser::parseSmsDeliverPdu(pdu, info)) {
65                             String assembled = concatMgr.addPart(info, idx);
66                             if (!assembled.isEmpty()) {
67                                 assembled.trim();
68
69                                 // Formatear como JSON simple
70                                 String json = String("{\"from\":\"\" + info.sender +
71                                     "\", \"text\":\"\" + assembled + \"\"}");
72
73                                 // Intentar enviar a API
74                                 if (enviarJsonAPI(json)) {
75                                     Serial.println("[API] Enviado correctamente");
76                                 } else {
77                                     Serial.println("[API] Error al enviar");
78                                     queueJson.push(json); // guardar en cola RAM
79                                 }
80
81                                 // Borrar mensaje de memoria ME
82                                 smsMgr.at("AT+CMGD=" + String(idx));
83                             }
84                         }
85                     }
86                 }
87             }
88             modemLine = "";
89         } else if (c != '\r') {
90             modemLine += c;
91         }
92     }
93
94     // --- Intentar enviar mensajes pendientes en cola ---
95     if (!queueJson.empty()) {
96         String &j = queueJson.front();
97         if (enviarJsonAPI(j)) {
98             Serial.println("[API] Reenvio OK");
99             queueJson.pop();
100         }
101     }
102
103     // --- Mantener WiFi activo ---
104     checkWifiLoop();
105 }
106

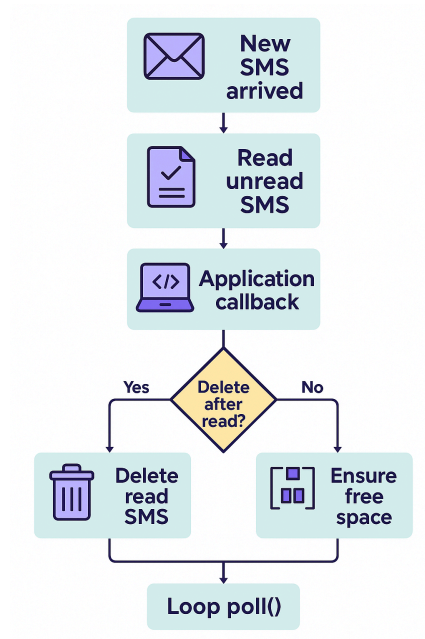
```

🔑 Explicación rápida

- **Recepción SMS** → detectada por URC **+CMTI**.

- **Lectura PDU** → `SmsPduParser::readPduAtIndex` .
- **Parseo PDU** → convierte remitente y contenido a texto plano.
- **JSON armado** → `{"from":"remitente","text":"contenido"}` .
- **Envío a API** → con `enviarJsonAPI(json)` .
- **Cola RAM (JsonQueue)** → guarda temporalmente mensajes si falla el envío.
- **Sweep automático** → cada loop intenta reenviar los pendientes.

10. Diagrama de Flujo del Proceso



Explicación rápida de cada etapa:

1. **Llegada de SMS (+CMTI)** → El módem notifica que entró un nuevo mensaje.
2. **Lectura de SMS no leídos (REC UNREAD)** → Se listan y procesan todos los mensajes pendientes.
3. **Callback de aplicación** → El SMS es entregado a tu función (ejemplo: parsear JSON y subirlo a la API).
4. **¿Borrado automático activado?**
 - **Sí** → Se borra el mensaje leído con `AT+CMGD=<idx>` .
 - **No** → Se mantiene en memoria.
5. **Gestión de memoria** → Limpia la memoria ME si está al límite, eliminando leídos o antiguos.
6. **Loop principal** → El sistema continúa ejecutando `poll()` sin bloquear otras tareas.