Centro de Engenharia, Modelagem e Ciências Sociais Aplicadas
Programa de Graduação em Engenharia de Informação

MONITOR IOT PARA ARMAS INTELIGENTES

ADAILSON ALVES DE SOUZA

Santo André, Dezembro de 2025

ADAILSON ALVES DE SOUZA

MONITOR IOT PARA ARMAS INTELIGENTES

Monografia apresentada como requisito parcial à conclusão do Curso de Engenharia de Informação da UFABC.

Orientador: Mario Alexandre Gazziro.

Santo André - SP

2025

Sistema de Bibliotecas da Universidade Federal do ABC
Elaborada pelo Sistema de Geração de Ficha Catalográfica da UFABC
com os dados fornecidos pelo(a) autor(a)

Souza, Adailson Alves
Monitor IOT para armas inteligentes / Adailson Alves de Souza. – 2025.
89 fls. : il.
Orientador: Mario Alexandre Gazziro

Trabalho de Conclusão de Curso – Universidade Federal do ABC, Bacharelado em Engenharia de Informação, Santo André, 2025.

1. Arma Inteligente. 2. Datalogger. 3. IOT. 4. MQTT. I. Gazziro, Mario Alexandre. II Bacharelado em Engenharia de Informação, 2025. III. Título.

ATA DE DEFESA DE TRABALHO DE GRADUAÇÃO EM ENGENHARIA DE INFORMAÇÃO

Ata de Defesa do Trabalho de Graduação em Engenharia de Informação da Universidade Federal do ABC

No dia **28 de Novembro de 2025** reuniu-se a banca examinadora do trabalho apresentado como Trabalho de Graduação em Engenharia de Informação de **Adailson Alves de Souza**, intitulado: **"MONITOR IOT PARA ARMAS INTELIGENTES"**. Após a exposição oral, o aluno foi arguido pelos componentes da banca que se reuniram reservadamente, e decidiram atribuir o conceito final **A**.

Orientador

Prof. Dr. Mario Alexandre Gazziro

Documento assinado digitalmente
gov.br
HUGO PUERTAS DE ARAUJO
Data: 28/11/2025 15:51:03-0300
Verifique em https://validar.iti.gov.br

Avaliador 1

Prof. Dr. Hugo Puertas de Araújo

Avaliador 2

Prof. Dr. João Paulo Pereira do Carmo

Santo André - SP

2025

AGRADECIMENTOS

Agradeço ao meu orientador Mario Gazziro pela direção da definição do tema do projeto, pela oportunidade e pelo apoio durante todo o processo de construção do mesmo.

“A ciência não é apenas um conjunto de conhecimentos,
mas uma forma de pensar” (Carl Sagan).

RESUMO

Com o relaxamento das restrições para aquisição de armas de fogo no Brasil, tem-se observado um aumento significativo no número de Colecionadores, Atiradores e Caçadores (CACs) nos últimos anos. Esse cenário levanta preocupações quanto à segurança pública, especialmente em relação ao uso indevido de armas de origem legal por parte do crime organizado. O presente trabalho propõe o desenvolvimento de um registrador de dados de arma de fogo, um dispositivo eletrônico projetado para registrar eventos como disparos e manipulações, contribuindo para um controle mais eficaz, resultando em uma redução nos casos de violência, e assim mitigando os problemas relacionados com armas de fogo e promovendo segurança no contexto brasileiro. Pretendemos utilizar tecnologia de IOT do estado da arte para atingir os objetivos propostos. Esperamos que esse projeto contribua com a sugestão de políticas públicas de tecnologia para que haja uma desmotivação para o uso indevido dessas armas em atividades criminosas.

Palavras-chave: Arma Inteligente; datalogger; IOT, MQTT.

ABSTRACT

With the relaxation of restrictions on firearm acquisition in Brazil, there has been a significant increase in the number of Collectors, Shooters, and Hunters (CACs) in recent years. This scenario raises concerns regarding public safety, especially regarding the misuse of legally obtained weapons by organized crime. This paper proposes the development of a firearm data logger, an electronic device designed to record events such as shots fired and manipulations, contributing to more effective control, resulting in a reduction in violence cases, and thus mitigating issues related to firearms and promoting security in the Brazilian context. We intend to utilize state-of-the-art IoT technology to achieve the proposed objectives. We hope that this project contributes to suggesting public technology policies to discourage the misuse of these weapons in criminal activities.

Keywords: Smartgun; data logger; IOT, MQTT.

LISTA DE SIGLAS

AI	–	Thinker – Fabricante do módulo A9G
API	–	Application Programming Interface
CAC	–	Colecionadores, Atiradores e Caçadores
CSV	–	Comma-Separated Values
DC-DC	–	Conversor de corrente contínua para corrente contínua
EMQX	–	Broker MQTT em nuvem
GMS	–	Global System for Mobile Communications
GNSS	–	Global Navigation Satellite System
GPRS	–	General Packet Radio Service
GPS	–	Global Positioning System
I2C	–	Inter-Integrated Circuit
IoT	–	Internet of Things
LBS	–	Location-Based Service
LTE	–	Long Term Evolution (citada indiretamente em contexto)
MQTT	–	Message Queuing Telemetry Transport
RTC	–	Real-Time Clock
SDK	–	Software Development Kit
SIM	–	Subscriber Identity Module
SMS	–	Short Message Service
SoC	–	System on Chip
SPI	–	Serial Peripheral Interface
SHA	–	Secure Hash Algorithm
TCU	–	Tribunal de Contas da União
UART	–	Universal Asynchronous Receiver/Transmitter

LISTA DE TABELAS

LISTA DE FIGURAS

SUMÁRIO

1 Introdução

O 17° Anuário Brasileiro de Segurança Pública (FÓRUM BRASILEIRO DE SEGURANÇA PÚBLICA, 2023) traz dados sobre armas de fogo, e é baseado em informações fornecidas pelas secretarias de segurança públicas estaduais, pelas polícias civis, militares e federais, entre outras fontes oficiais da segurança pública. Com a análise dos dados é possível notar o aumento expressivo do número de armas nas mãos de civis no Brasil, sendo que até 2022 a soma chegava a mais de 1,5 milhões, considerando somente as armas ativas, as com registro expirado ultrapassou 1,5 milhões e as apreendidas chegou a mais de 100 mil (.

Com os CACs estão 42,5% das armas particulares compradas no Brasil, 25% pertencem a membros das Forças Armadas, o restante das armas fica nas mãos de caçadores de subsistência, cidadãos com registro de defesa pessoal e servidores civis, com o exército sendo o órgão responsável por fiscalizar, registrar e controlar esse arsenal.

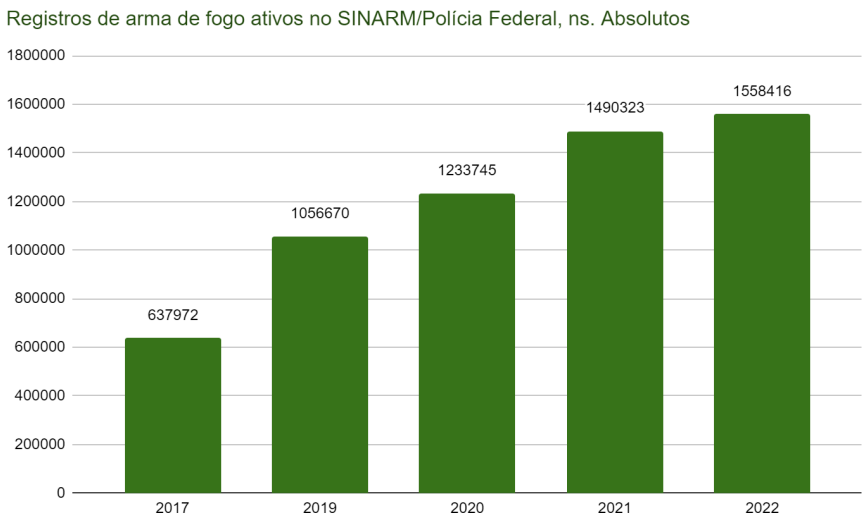

Figura 1 - Registros de arma de fogo ativos. Fonte: SINARM/PF.

No período de 2018 a 2023 houve um aumento de 666% de certificados de registros ativos para atividades de caçador, atirador esportivo e colecionador (CAC). Em 2018 existiam 117 mil CACs, mas ao final de 2022 o número ultrapassou 780 mil, 7 vezes maior. Neste mesmo período, devido a maiores flexibilidades da gestão

do governo (MILITÃO, 2022), Além de contarem com um maior acervo, este ficou mais potente e com menos controle, porém, em 2023, com uma nova gestão, foram feitos edições nos decretos para revogar uma série de normas que facilitava e ampliava o acesso da população a armas de fogo e munição, essa nova legislação de 2023 fez o número de registros de arma de fogo caírem em 82% em relação a 2022, de 114 mil para 20 mil, e é o menor desde 2004 (RADIO SENADO, 2024).

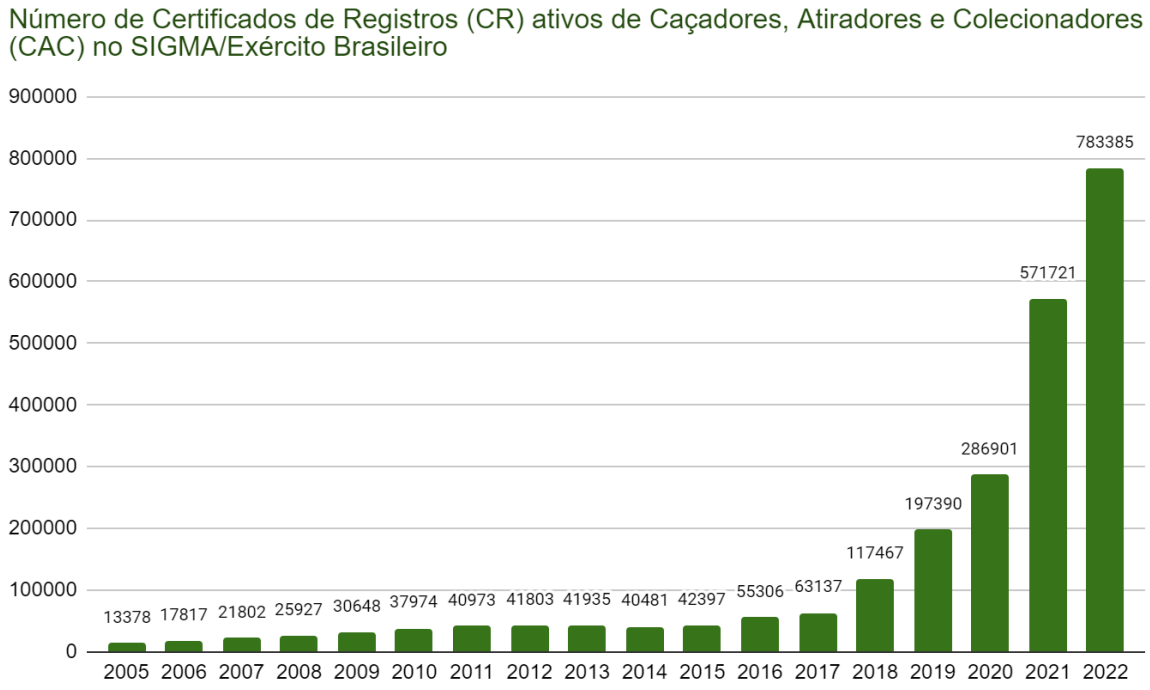

Figura 2 - Número de certificados de registro ativos de CAC. Fonte: SINARM/PF.

É necessário salientar que as armas de fogo seguem como o principal instrumento utilizado para matar no Brasil, segundo o anuário, em 2022 76,5% dos homicídios fizeram uso de armas de fogo.

O controle e a rastreabilidade de armas de fogo constituem um desafio contínuo para as autoridades de segurança e a sociedade em geral, sendo cruciais para a prevenção e investigação de crimes (SOUZA; LIMA, 2020). A necessidade de mecanismos mais eficazes para monitorar o uso de armamentos tem impulsionado a pesquisa e o desenvolvimento de dispositivos eletrônicos de rastreamento (SILVA et al., 2018). Tais dispositivos visam registrar, de forma autônoma e segura, eventos cruciais como o disparo e a manipulação da arma.

A proposição de um dispositivo eletrônico acoplado ao corpo da arma representa um avanço potencial nesse campo, buscando oferecer uma solução robusta para o registro de eventos e a detecção de violação mecânica do sistema (PEREIRA, 2022). A principal dificuldade reside em assegurar a integridade dos registros e evitar fraudes, dado o contexto de uso e as implicações legais dos dados coletados. Este trabalho se insere nessa lacuna, propondo o desenvolvimento de um sistema capaz de fornecer dados confiáveis e à prova de adulteração sobre o histórico operacional de uma arma de fogo.

2 Justificativa

Posto o fato inegável do aumento de armas nas mãos dos brasileiros, torna-se necessário olhar para as reais consequências disso. Segundo matéria da A Pública de 2021 (FONSECA, 2021), os CACs perdem 3 armas por dia, no mesmo ano, 840 armas de fogo foram roubadas ou extraviadas. Uma matéria do jornal Folha de São Paulo (LOPES, 2024) mostra que ao menos 8% das armas apreendidas após crimes em São Paulo pertenciam a CACs, após o TCU ter acesso integral ao sistema SIGMA (Sistema de Gerenciamento Militar de Armas), base de dados do Exército, e cruzar as informações com o banco de dados de armas apreendidas em São Paulo de 2015 a 2020, chegou-se a uma análise que mostra que das 47748 armas recolhidas em São Paulo no período de 2015 a 2020, 3873 foram de CACs. Ao menos 1312 armas foram adquiridas e registradas no Exército a partir de 2019. Matéria do jornal UOL de 2022 (MILITÃO, 2022) mostra que policiais descobriram que CACs fornecem a assaltantes, e segundo o delegado entrevistado, a aquisição de armas de fogo, de forma lícita, por meio de certificados de registro de colecionador e de atirador, obtidos, perante o Exército Brasileiro, tem sido uma estratégia usada por grupos criminosos, para obter instrumentos para ações criminosas. Matéria do jornal JM Online, no dia 3 de junho de 2023 (PAIVA, 2023), mostra o caso em que policiais suspeitam que falsos despachantes emprestavam ou alugavam suas armas, adquiridas como CAC, para criminosos.
Na matéria do jornal Gazeta do Povo (GAZETA DO POVO, 2023), é falado sobre o inquérito do STM que investiga armas desviadas do exército e das polícias para abastecer facções criminosas.

3 Objetivos

Projetar um dispositivo eletrônico para registrar eventos de disparo e de manipulação, bem como detectar violação mecânica do sistema, uma vez que o dispositivo estará acoplado ao corpo da arma, assim como garantir uma maneira de evitar fraudes nos registros.

4 Metodologia e Implementação

4.1 Visão geral

O monitor IOT para armas inteligentes é um projeto desenvolvido com o intuito de realizar o monitoramento eletrônico de armas de fogo. Para isso, o projeto foi dividido em etapas que contemplam desde o estudo do problema e definição dos requisitos até a implementação da solução embarcada e o teste funcional do protótipo.

Inicialmente, foi realizada uma análise detalhada dos cenários no contexto Brasil, principalmente mostrando casos reais de desvios e os riscos associados ao uso indevido de armas legalizadas. Esta etapa foi a que especificou quais as características físicas e de software para chegar a uma solução de projeto.

Foi realizada uma pesquisa de diferentes microcontroladores, módulos GPS, GPRS, acelerômetros e também foi levado em consideração o baixo consumo de energia para cada um desses. O módulo A9G Pudding foi o selecionado pois oferece um equilíbrio adequado entre conectividade, armazenamento e baixo consumo.

Depois da concepção da eletrônica, iniciou-se o desenvolvimento do firmware embarcado. Nessa etapa foram implementados os algoritmos de detecção de disparos, conexão GSM/GPRS, manipulação de dados em cartões SD.

Por fim, foram feitos testes funcionais do protótipo, que consistiram em avaliar o comportamento do sensor, integridade dos registros, precisão do GPS. Estes testes permitiram validar o funcionamento como um todo e verificar se a solução atende aos objetivos propostos pelo trabalho.

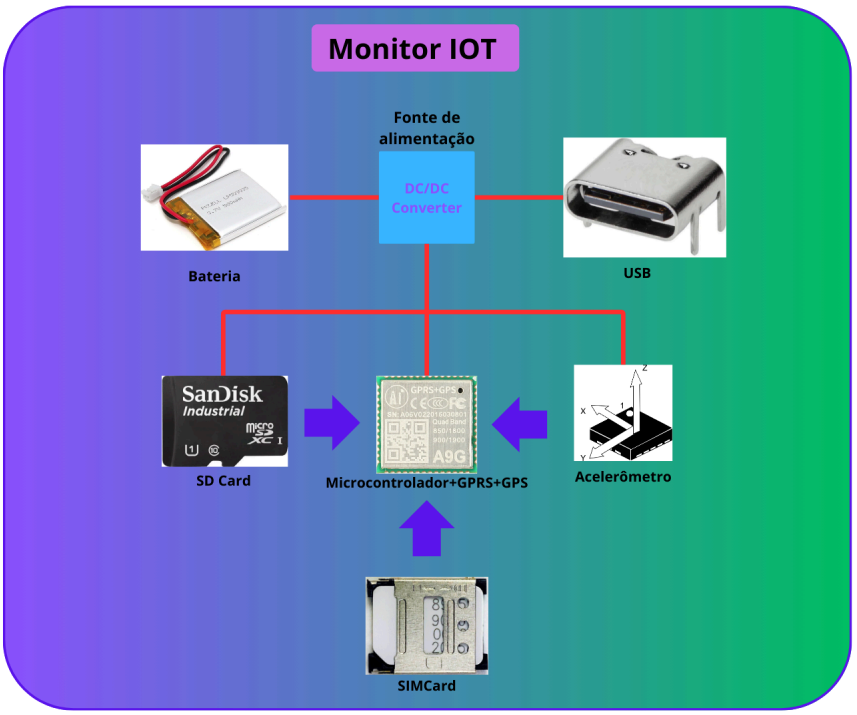

Figura 3 - Diagrama de blocos do registrador de dados de arma de fogo. Fonte: autoria própria.

4.2 Arquitetura eletrônica

O projeto eletrônico foi concebido utilizando os recursos já encontrados no módulo A9G Pudding. Neste módulo existe o submódulo A9G, componentes para gerar alimentação adequada, acelerômetro, slot para cartão SIM, slot para cartão SD, antena GPS, antena GPRS e conector para acoplar a bateria (AI-THINKER, s.d.).

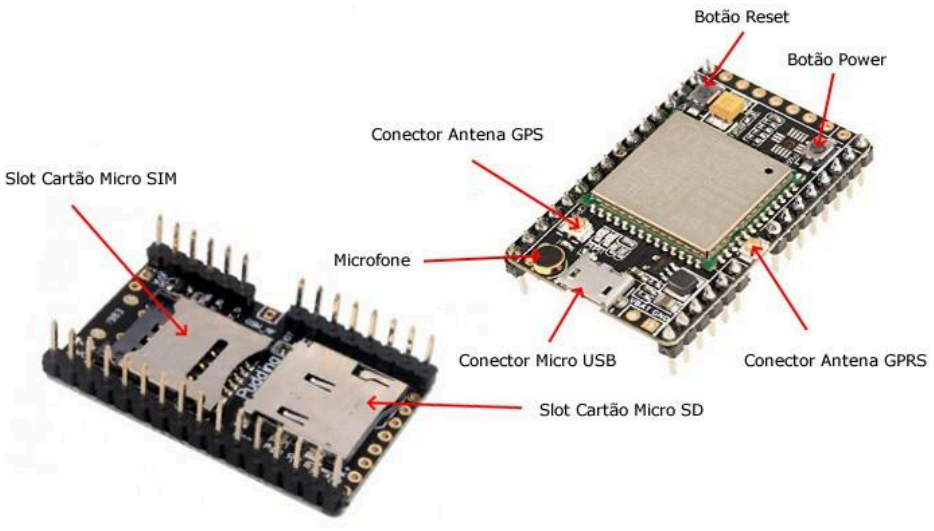

Figura 4 - Módulo A9G Pudding e seus recursos físicos. Fonte: Usinainfo.

4.2.1 Submódulo A9G

O módulo A9G foi projetado pela AI-Thinker e contempla num mesmo encapsulamento um módulo SoC RDA8955 32-bit RISC core com 4MB de flash que incorpora GSM/GPRS (2G) quad band (850MHz,1800MHz,900MHz,1900MHz), e também um receptor GNSS GK9501.

A interface com este módulo pode ser feita de duas formas, através de comandos AT utilizando um microcontrolador externo e comandos AT via UART, ou no modo “SDK On Chip Development” que não necessita de microcontrolador externo e o firmware é gravado diretamente no módulo. Para este projeto foi utilizado o modo SDK pois assim não seria necessário utilizar um outro microcontrolador.

Figura 5 - Submódulo A9G da empresa AI-Thinker. Fonte: AI-Thinker.

O firmware é gravado no módulo através de uma interface serial UART dedicada, com o auxílio de um conversor USB-Serial TTL e um software para Windows chamado CoolWatcher.

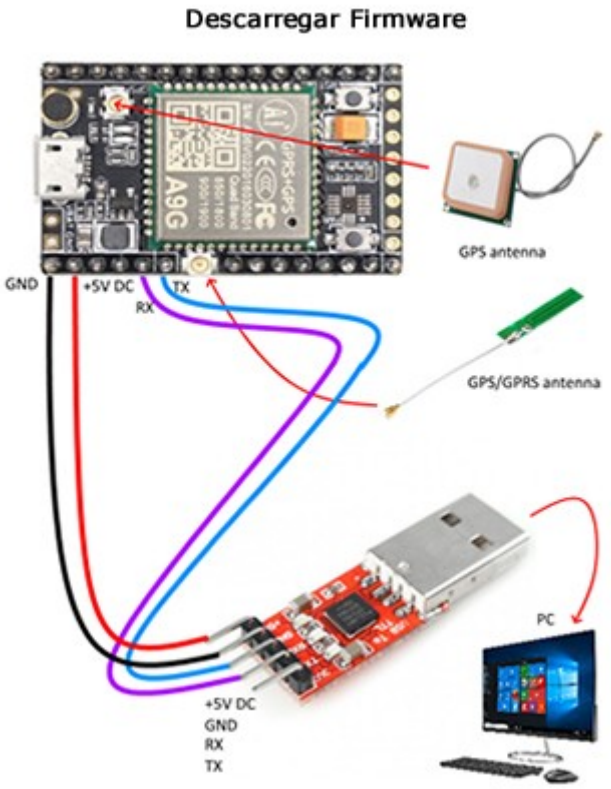

Figura 6 - Hardware para gravação de firmware. Fonte: Usinainfo.

4.2.2 Acelerômetro

O LIS3DH é um acelerômetro de 3 eixos fabricado pela STMicroelectronics. Possui alta performance e baixo consumo, em modo de baixo consumo pode consumir menos que 2 uA. A interface de comunicação é I2C ou SPI, e pode fornecer dados de aceleração com taxas de 1 Hz a 53 kHz, também possui pinos configuráveis de interrupção.

Figura 7 - Circuito integrado do acelerômetro LIS3DH. Fonte: STMicroelectronics.

4.2.3 Cartão SIM

Para garantir uma identificação única na rede GSM/GPRS, o módulo faz uso de um slot de cartão SIM do padrão micro SIM e o cartão microSIM de operadora brasileira.

4.2.4 Cartão SD

O cartão SD (Secure Digital) é uma solução de armazenamento compacta e portátil usada para armazenar e transferir dados em diversos dispositivos, como câmeras digitais e smartphones. Eles são diferenciados pela capacidade e velocidade de escrita e leitura.

4.2.5 Fonte de alimentação

Para suprir as demandas energéticas do módulo A9G, que no momento do registro na rede GSM/GPRS pode dar picos de 2A, e dos demais periféricos, a placa possui o SY8089 que é um regulador Step-down DC-DC síncrono de alta eficiência.

E o acumulador de energia, que manterá o módulo funcionando por longos períodos é um bateria de Lithium 3.7V com 300mAh.

Figura 8 - Bateria de lithium 3.7V. Fonte: autoria própria.

4.2.6 Detecção de violação

Para a detecção de violação, foi considerado o uso de um mecanismo simples e confiável baseado em chave mecânica de contato seco (como um micro switch ou push button de pressão). Esse dispositivo é instalado entre o módulo eletrônico e a parede interna da caixa metálica que fica colada à arma. Caso a caixa seja removida, descolada ou aberta, o botão muda de estado instantaneamente, disparando um evento de violação.

Essa solução possui baixo consumo energético, operação confiável e não exige sensores complexos, sendo adequada ao ambiente hostil e às restrições de espaço do projeto.

4.2.7 Esquema eletrônico

Por padrão de fabricação, o acelerômetro LIS3DH não vem soldado no módulo, portanto foi necessário adquiri-lo e soldá-lo, além disso foi necessário fazer um conexão via fio no pino de interrupção para o pino IO2 do A9G já que não foi previsto o uso do pino de interrupção do acelerômetro.

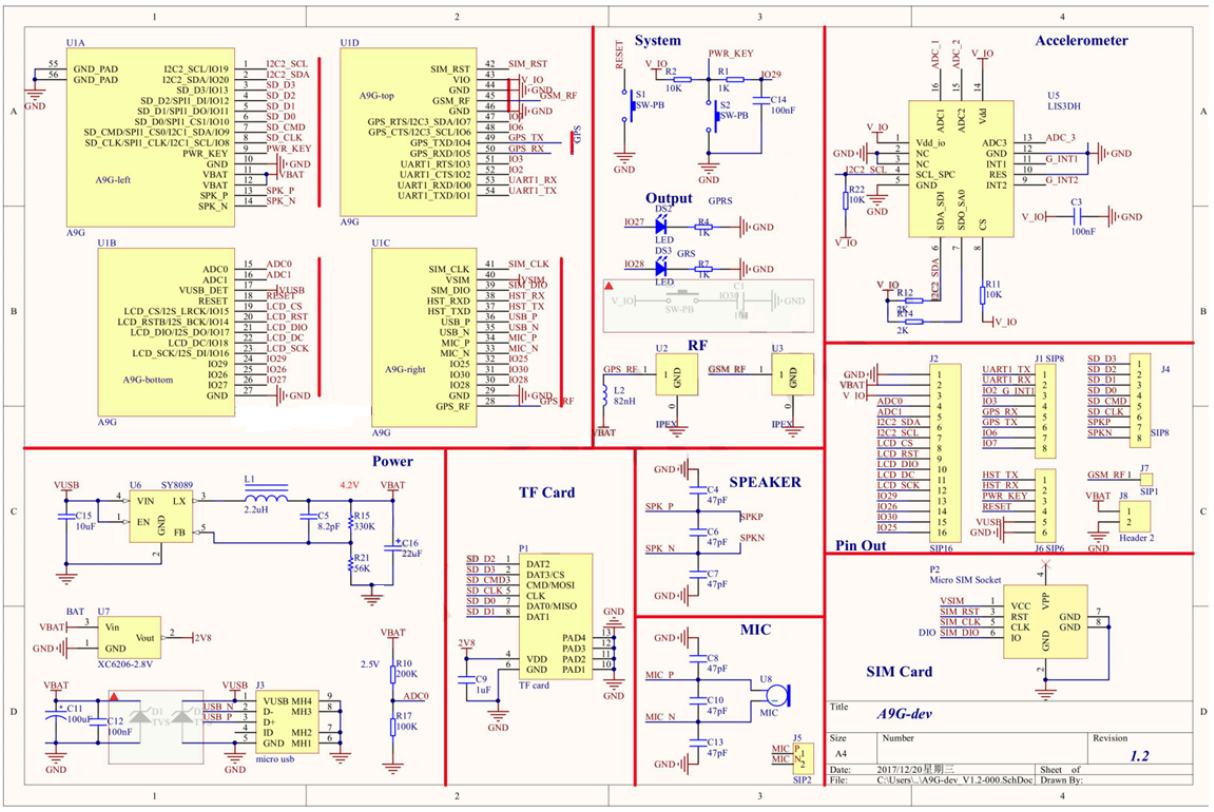

Figura 9 - Esquema eletrônico do módulo A9G Pudding. Fonte: AI-Thinker, adaptado.

4.3 Fluxo de registro de eventos

O fluxo de registro de eventos descreve as etapas que o sistema executa desde a detecção de um evento relevante no armamento até o armazenamento final das informações no datalogger e sua posterior transferência para validação e monitoramento. Esse fluxo garante a rastreabilidade e a integridade dos dados de uso da arma, minimizando a possibilidade de fraudes.

Figura 10 - Fluxo de eventos. Fonte autoria própria.

4.4 Arquitetura de software

O firmware do dispositivo foi desenvolvido com a utilização do SDK da AI-Thinker que fornece um sistema operacional em tempo real proprietário, utilizando uma arquitetura modular baseada em tarefas. A aplicação foi estruturada em componentes independentes, cada um responsável por uma tarefa, como aquisição de dados, comunicação, armazenamento e gestão de energia.

Há três tarefas principais, “MainTask”, “Task_CaptureGPS” e “EventTask”.

4.4.1 MainTask

A tarefa “MainTask” é a que contém uma máquina de estados com 4 estados: STATE_INIT -> STATE_CAPTURE -> STATE_SEND -> STATE_SLEEP, e essa tarefa é a que controla o ciclo operacional da aplicação.

O estado STATE_INIT é responsável pela inicialização dos periféricos, como GPS, GSM, I2C e SD Card.

O estado STATE_CAPTURE é responsável pela captura de dados de geolocalização, tanto por GPS quanto por LBS.

O estado STATE_SEND é o responsável pela conexão GPRS, envio dos dados via rede e SMS.

O estado STATE_SLEEP é o responsável por entrar em modo de baixo consumo. Nele é feito uma rotina de desligar periféricos, desconectar da rede e executar o comando para o A9G entrar em sleep mode.

4.4.2 Task_CaptureGPS

A tarefa “Task_CaptureGPS” é responsável por ligar e desligar o GPS, e capturar os dados de geolocalização de forma eficiente e contínua.

4.4.3 EventTask

Nesta tarefa a maior parte dos eventos são de origem da rede GSM/GPRS, como falha ou sucesso ao conseguir registro, ativar rede GPRS, anexar à rede GPRS, dados LBS, recebimento e leitura de SMS, mas também há eventos do sistema, como UARTs, informação de gerenciamento de energia e botões .

4.4.4 Comunicação com periféricos e drivers

O firmware utiliza diversos subsistemas do A9G:

Módulo	Interface	Finalidade
GPS GNSS	UART1	Coleta de latitude/longitude
Rede GSM/GPRS	API de rede do SDK	Envio de dados e SMS
Acelerômetro LIS3DH	I2C	Detecção de disparos e movimento
SD Card	SPI	Registro local permanente dos eventos

Tabela 1 - Subsistemas do A9G e suas interfaces. Fonte: Autoria própria.

4.4.5 Armazenamento local e formato de registro

Os dados coletados são armazenados em cartão microSD no formato CSV, contendo identificação do evento, timestamp, geolocalização e tipo de ocorrência. Para aumentar a confiabilidade do sistema, mecanismos de integridade podem ser incorporados ao processo de gravação.

Além do registro convencional, cada linha gravada no arquivo pode receber um hash criptográfico (por exemplo, SHA-256), calculado no momento da escrita. Esse hash funciona como uma assinatura de integridade: qualquer alteração posterior nos dados pode ser detectada ao se recalcular o hash e compará-lo com o gravado.

O SDK utilizado no módulo A9G já dispõe de componentes voltados à comunicação segura, incluindo suporte a SSL/TLS para conexões criptografadas. Embora o armazenamento local no cartão SD ainda não utilize diretamente a criptografia fornecida pelo SDK, essa infraestrutura já disponível abre caminho para que, em versões futuras, o dispositivo possa empregar criptografia simétrica (como AES) ou até mesmo encapsular os arquivos em estruturas cifradas antes de serem gravados no cartão.

A combinação de hashing para integridade e a possibilidade de utilizar os mecanismos criptográficos já embutidos no SDK torna o sistema mais robusto frente a tentativas de adulteração ou manipulação dos dados registrados.

Registro
Timestamp
Motivo gerador do evento
Coordenadas GPS
Fonte de localização (GPS/LBS)
Força do sinal celular
Identificadores das células GSM
IMEI e ICCID da rede móvel
Nível da bateria

Tabela 2 - Formato de registro de um evento. Fonte: autoria própria.

4.4.6 Comunicação MQTT

Após a captura, os dados são enviados ao broker na nuvem utilizando GPRS.

O software executa:

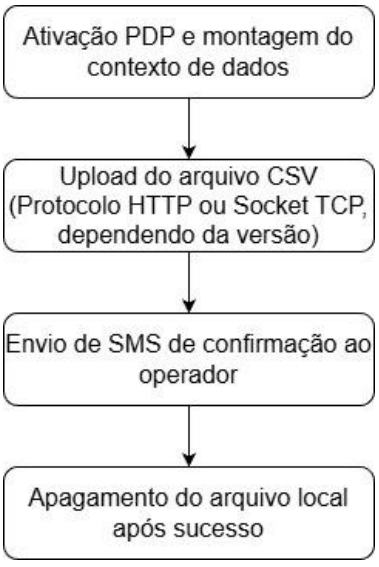

Figura 11 - Sequência para envio dos dados. Fonte: autoria própria.

Esse ciclo garante que nenhum registro é perdido.

4.4.7 Gestão de energia

O módulo alterna entre dois modos de operação, o de alta performance durante a captura e envio de dados, e o modo de baixo consumo durante o período ocioso. No período ocioso o acelerômetro permanece ativo em low-power para gerar interrupção ao detectar disparo, despertando o sistema automaticamente através de uma interrupção no pino do A9G.

Todos os estados anteriores ao STATE_SLEEP possuem timeout, pois assim garantimos uma janela de tempo para o sistema conseguir capturar os dados e enviá-los, caso não consiga, ao fim da janela de tempo o sistema força a entrada no estado STATE_SLEEP para economizar bateria. Esse formato reduz o consumo de energia por deixar o sistema a maior parte do tempo dormindo, despertando apenas para capturar e transmitir os eventos de disparo ou violação.

4.4.8 Sistema Keep Alive

O sistema pode incluir um mecanismo de keep alive mensal, enviado via SMS, cujo objetivo é confirmar que cada dispositivo permanece operacional, mesmo que nenhum evento de disparo ou violação tenha ocorrido no período. Cada unidade possui um identificador único, permitindo que o sistema central registre quais dispositivos estão ativos.

O envio de SMS consome menos energia do que o modo de operação normal, via GPRS, pois exige somente o registro na rede GSM por um período curto. Considerando que esse envio ocorre apenas uma vez por mês, o impacto energético é relativamente pequeno quando comparado a alternativas como conexão periódica via GPRS/MQTT, que exigem mais tempo de rádio ativo e consumo significativamente maior.

Assim, o SMS mensal apresenta o melhor compromisso entre custo energético, simplicidade operacional e confiabilidade — sendo, portanto, a solução mais eficiente para dispositivos alimentados por baterias de longa duração.

5 Resultados

O projeto do Monitor IOT para Armas Inteligentes foi desenvolvido e testado em condições controladas, demonstrando capacidade de registrar eventos de disparo, armazená-los e enviá-los corretamente.

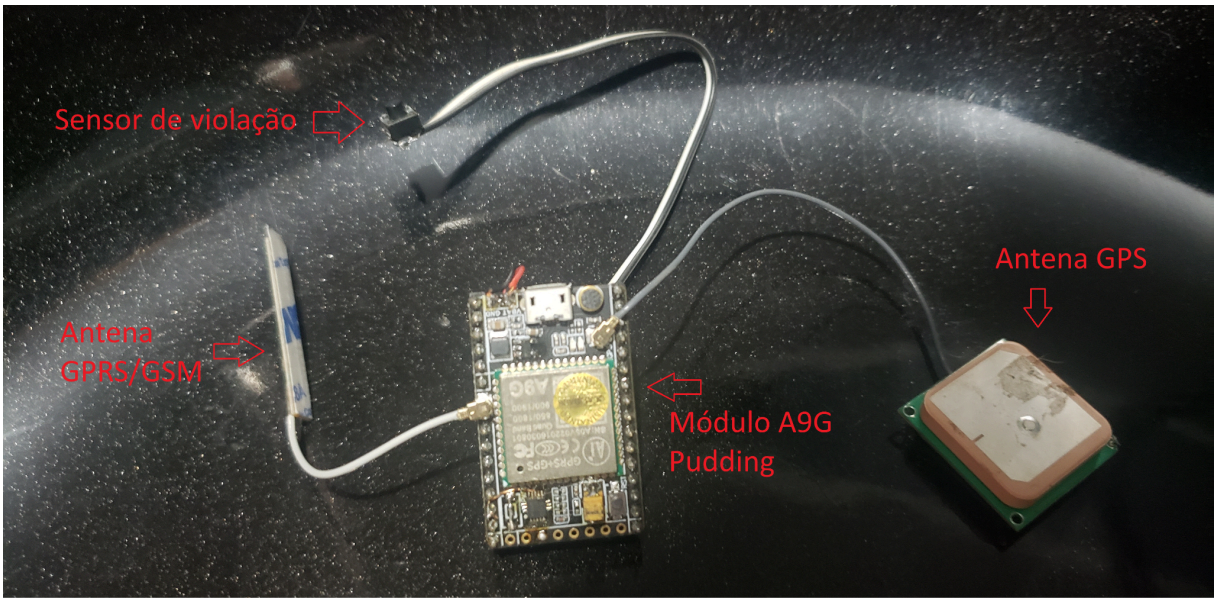

Figura 12 - Montagem final do monitor IOT. Fonte: autoria própria.

Para constatar o funcionamento do projeto, foi utilizado o broker em nuvem gratuíto da EMQX (broker.emqx.io:1883), e para visualizar os dados foi utilizado o software cliente MQTTX, nele é possível assinar tópicos, publicar tópicos e visualizar os dados.

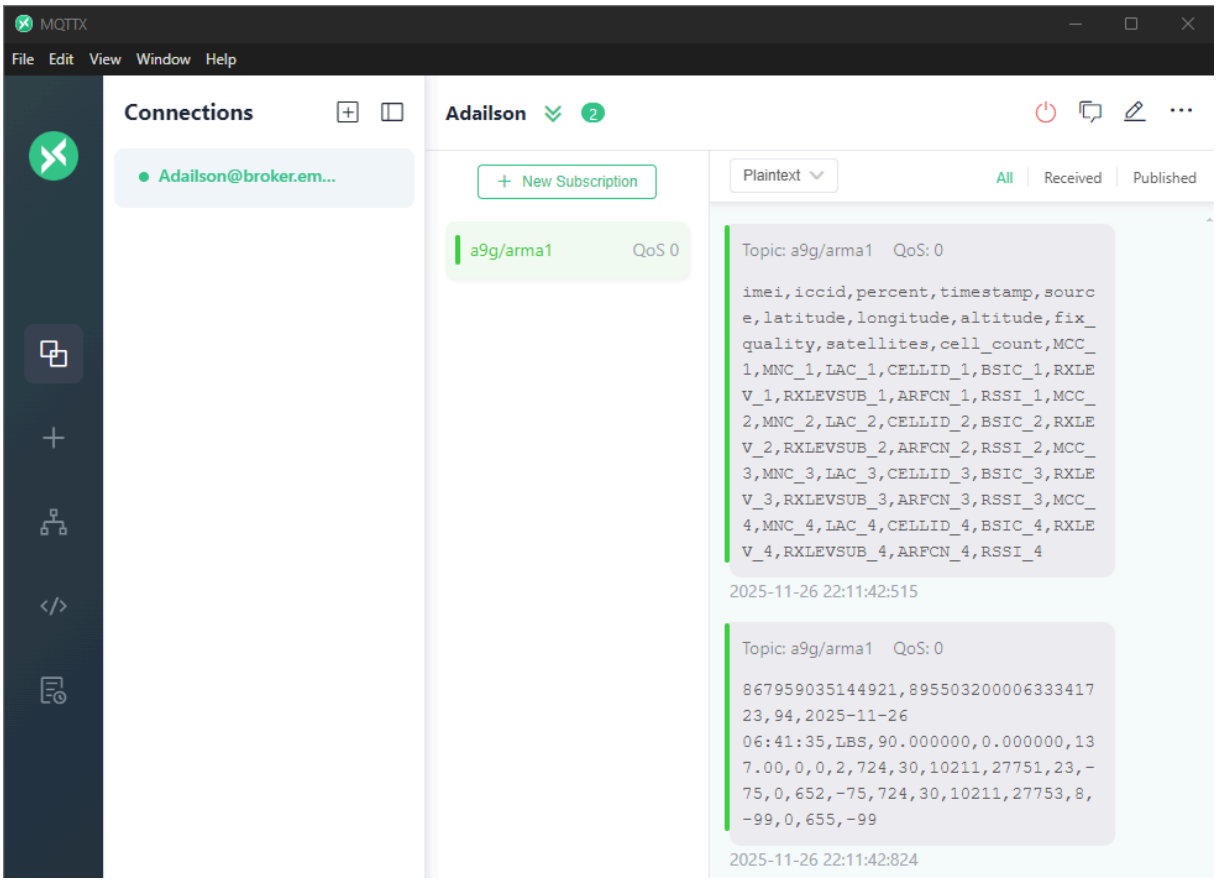

Figura 13 - Software MQTTX com dados recebidos do Monitor IOT. Fonte: autoria própria.

Para ajudar a debugar e analisar a performance do projeto, uma UART foi utilizada para fazer log do funcionamento do código, e com um terminal serial foi possível ver o que aconteceu desde o momento da captura dos dados até o envio e entrada no modo sleep (apêndice A).

Com estes dados em mãos, foi possível constatar que um evento de disparo foi detectado, que a localização foi registrada e transmitida via MQTT com sucesso.

6 Conclusões

Neste trabalho foi desenvolvido um monitor IOT para armas de fogo, com capacidade de registrar e transmitir dados de rastreabilidade de equipamentos legalizados. O projeto integrou sensores de movimento, sensor de violação, módulo GPS, comunicação GPRS, demonstrando a viabilidade técnica.

Os teste realizados comprovam o correto funcionamento, desde a detecção de eventos de disparo ou violação, captura de dados de geolocalização, armazenamento dos dados com marcação temporal, envio de dados via rede GPRS com protocolo MQTT, o que reforça sua aplicabilidade em cenários reais de fiscalização periódica.

Com base nos resultados obtidos, conclui-se que o projeto proposto atende aos requisitos essenciais definidos neste trabalho, oferecendo uma solução concreta para o monitoramento seguro e inviolável de armas de fogo.

7 Trabalhos futuros

Ao longo do desenvolvimento do projeto foi descoberto erros de hardware e software no SDK, erros estes que não serão resolvidos pela AI-Thinker, pois eles abandonaram os repositórios no seu git oficial. Um dos problemas de software é que com as últimas versões do SDK não é mais possível deixar o módulo inteiro em baixo consumo, antes era possível desligar o core do processador e religar com interrupção externa, agora o baixo consumo do A9G ainda consome uma corrente significante para o propósito deste trabalho. Para atender a autonomia de 1 ano e sem modificar muito do hardware atual, seria necessário utilizar um microcontrolador externo de baixíssimo consumo e transferir o gerenciamento do acelerômetro LIS3DH para ele, e toda a alimentação do restante seria cortada ou conectada pelo GND do circuito através de um MOSFET do tipo N, e este microcontrolador também gerenciaria a janela de tempo para o módulo A9G pudding fazer suas tarefas. Ou no pior dos casos, substituir o A9G por um equivalente, por exemplo o SIM808, mas isso implicaria em fazer um novo hardware.

Uma melhoria seria a miniaturização de todos os componentes eletrônicos, pois assim a estrutura mecânica ficaria menor e seria possível embarcar o monitor IOT em mais tipos de armas de fogo, mesmo as menores.

Um outro ponto de melhoria significativa seria a substituição da antena ativa do GPS por uma passiva, já que esta última consome menos energia e espaço físico.

Referências bibliográficas

FÓRUM BRASILEIRO DE SEGURANÇA PÚBLICA. 17º Anuário Brasileiro de Segurança Pública 2023. 2023. Disponível em: https://publicacoes.forumseguranca.org.br/bitstreams/e84399ab-7cf2-4411-b893-35ce521cbc95/download. Acesso em: 13/07/2024.

MILITÃO, E. Registro de armas cresce com Bolsonaro e homicídios sobem em estados. UOL Notícias, 4 jun. 2022. Disponível em: https://noticias.uol.com.br/cotidiano/ultimas-noticias/2022/06/04/registro-de-armas-jair-bolsonaro-cacs-violencia-homicidios-lobby.htm. Acesso em: 12/07/2024.

LOPES, R. Ao menos 8% das armas apreendidas após crimes em São Paulo pertenciam a CACs. Folha de S. Paulo, 12 mar. 2024. Disponível em: https://www1.folha.uol.com.br/cotidiano/2024/03/ao-menos-8-das-armas-apreendidas-apos-crimes-em-sao-paulo-pertenciam-a-cacs.shtml. Acesso em: 13/07/2024.

RADIO SENADO. Sob nova legislação, registro de armas para defesa pessoal cai. Senado Federal, 18 jan. 2024. Disponível em: https://www12.senado.leg.br/radio/1/noticia/2024/01/18/sob-nova-legislacao-registro-de-armas-para-defesa-pessoal-cai. Acesso em: 13/07/2024.

FONSECA, B; MARTINS L. Caçadores, atiradores e colecionadores perdem três armas por dia no Brasil. 3 nov. 2021. Disponível em: https://apublica.org/2021/11/cacadores-atiradores-e-colecionadores-perdem-tres-armas-por-dia-no-brasil/?utm_source=webstories&utm_medium=botao&utm_campaign=armasperdidas. Acesso em: 12/07/2024.

PEREIRA, João Victor. Sistemas eletrônicos de rastreamento de armamentos: desafios de hardware e segurança de dados. Revista Brasileira de Tecnologia Aplicada, São Paulo, v. 15, n. 2, p. 45-60, jul./dez. 2022.

SILVA, Ana Carolina et al. Integridade de dados em dispositivos embarcados para controle de uso de armas. In: SEMINÁRIO INTERNACIONAL DE CIÊNCIA E TECNOLOGIA, 5., 2018, Curitiba. Anais [...]. Curitiba: Editora Tecnológica, 2018. p. 112-125.

SOUZA, Marcos Roberto; LIMA, Cláudia Ferreira. O uso de tecnologia no combate ao crime: rastreabilidade de armas de fogo. Segurança Pública em Foco, Brasília, v. 10, n. 1, p. 78-95, jan./jun. 2020.

PAIVA, C. Armas apreendidas em operação seriam emprestadas ou alugadas a criminosos. 3 jun. 2023. Disponível em: https://jmonline.com.br/policia/armas-apreendidas-em-operac-o-seriam-emprestadas-ou-alugadas-a-criminosos-1.279446. Acesso em: 21/07/2024.

GAZETA DO POVO. Inquérito do STM investiga armas desviadas do Exército e das polícias para facções criminosas. Gazeta do Povo, 2023. Disponível em: https://www.gazetadopovo.com.br/republica/armas-de-quarteis-abastecem-faccoes/. Acesso em: 21/07/2024.

POLÍCIA FEDERAL. Registro de Armas. Disponível em: https://www.gov.br/pf/pt-br/assuntos/armas/registro. Acesso em: 21/07/2024.

AI-THINKER. A9/A9G development board: pudding. Disponível em: https://ai-thinker-open.github.io/GPRS_C_SDK_DOC/en/hardware/pudding-dev-board.html. Acesso em: 23/11/2025.

Conhecendo o MQTT. IBM. Disponível em: https://www.ibm.com/developerworks/br/library/iot-mqtt-why-good-for-iot/index.html. Acesso em: 25/11/2025.

VAGNER, J. (UFG) Tecnologias de rede sem fio. Disponível em:https://ww2.inf.ufg.br/~vagner/courses/mobilecomputing/docs/Tecnologia-GSM.pdf. Acesso em: 25/11/2025.

Apêndice A - Log da UART de um registro de posição

```
Initializing System
Inicializando perifericos...
SD not initialized
Task_CaptureGPS started
Sistema esta pronto
start mqtt task
IMEI:867959035144921


******************** network register success ****************

********************************
**  Received EVENT_GPS_START  **
********************************
Iniciando captura de dados...
network attach success
network activate success

Tempo restante (8 s).
Esperando dados de localizacao...
GPS: --
GSM: OK
LBS: Cell 0 info:0,0,0,0,0,0,0,0
LBS: Cell 1 info:0,0,0,0,0,0,0,0
LBS: Cell 2 info:0,0,0,0,0,0,0,0
LBS: Cell 3 info:0,0,0,0,0,0,0,0
GPS: latitude: 0.000000
GPS: longitude: 0.000000
GPS: altitude: 0.000000
GPS: fixQuality: 0
GPS: satellites: 0
set gps ret:1
gps firmware version:,GOKE9501_1.4_18022400,GOKE microsemi
GPS Init OK!

Tempo restante (5 s).
Esperando dados de localizacao...
GPS: --
GSM: OK
```

LBS: Cell 0 info:724,30,10211,27751,23,-73,0,652
LBS: Cell 1 info:0,0,0,0,0,0,0,0
LBS: Cell 2 info:0,0,0,0,0,0,0,0
LBS: Cell 3 info:0,0,0,0,0,0,0,0
GPS: latitude: 90.000000
GPS: longitude: 0.000000
GPS: altitude: 137.000000
GPS: fixQuality: 0
GPS: satellites: 0

Tempo restante (3 s).
Esperando dados de localizacao...
GPS: --
GSM: OK
LBS: Cell 0 info:724,30,10211,27751,23,-73,0,652
LBS: Cell 1 info:0,0,0,0,0,0,0,0
LBS: Cell 2 info:0,0,0,0,0,0,0,0
LBS: Cell 3 info:0,0,0,0,0,0,0,0
GPS: latitude: 90.000000
GPS: longitude: 0.000000
GPS: altitude: 137.000000
GPS: fixQuality: 0
GPS: satellites: 0

Tempo restante (1 s).
Esperando dados de localizacao...
GPS: --
GSM: OK
LBS: Cell 0 info:724,30,10211,27751,23,-75,0,652
LBS: Cell 1 info:0,0,0,0,0,0,0,0
LBS: Cell 2 info:0,0,0,0,0,0,0,0
LBS: Cell 3 info:0,0,0,0,0,0,0,0
GPS: latitude: 90.000000
GPS: longitude: 0.000000
GPS: altitude: 137.000000
GPS: fixQuality: 0
GPS: satellites: 0
Timeout: closing window.
Dados capturados.

```
power:4155 -- 94%
IMEI: 867959035144921
ICCID: 89550320000633341723
GPS: Timestamp: 2025-11-26 06:41:35
GPS: Source: LBS
GPS: latitude: 90.000000
GPS: longitude: 0.000000
GPS: altitude: 137.000000
GPS: fixQuality: 0
GPS: satellites: 0
GPS: cellCount: 2
LBS: cell 0 info:724,30,10211,27751,23,-75,0,652
LBS: cell 1 info:724,30,10211,27753,8,-99,0,655
LBS: cell 2 info:0,0,0,0,0,0,0,0
LBS: cell 3 info:0,0,0,0,0,0,0,0


Dados armazenados.
********************************
**   Received EVENT_GPS_STOP  **
********************************
attach status:1
GPRS conectado!
Esperando conexao ao broker MQTT...

Tempo restante (8 s).
MQTT CONNECTED!
Esperando conexao ao broker MQTT...

Tempo restante (5 s).
Comecar a enviar!
Sending [1]:
imei,iccid,percent,timestamp,source,latitude,longitude,altitude,fix_quality,satellites,cell_count,MCC_1,
MNC_1,LAC_1,CELLID_1,BSIC_1,RXLEV_1,RXLEVSUB_1,ARFCN_1,RSSI_1,MCC_2,MNC_2,LAC
_2,CELLID_2,BSIC_2,RXLEV_2,RXLEVSUB_2,ARFCN_2,RSSI_2,MCC_3,MNSending [2]:
867959035144921,89550320000633341723,94,2025-11-26
06:41:35,LBS,90.000000,0.000000,137.00,0,0,2,724,30,10211,27751,23,-75,0,652,-75,724,30,10211,
27753,8,-99,0,655,-99
MQTT FAILED! Code=256
MQTT CONNECTED!
No more data to send! Total: 2
SendAllRecordsNow() finished OK!
```

Envio completo!

Esperando desativar...

Esperando desativar...

Esperando desatachar...

Timeout: closing window.

Entrando em modo de baixo consumo...

Apêndice B - Códigos fonte

Código fonte principal - App.c

```
#include "api_hal_gpio.h"
#include "stdint.h"
#include "stdbool.h"
#include "api_debug.h"
#include "api_os.h"
#include "api_hal_pm.h"
#include "api_event.h"
#include "api_network.h"

#include <api_hal_uart.h>
#include "gps.h"
#include "gps_parse.h"
#include <api_gps.h>
#include "api_lbs.h"
#include "api_sms.h"
#include "api_call.h"
#include "api_sim.h"

#include "api_fs.h"
#include "api_socket.h"
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "api_info.h"

#include "app.h"
#include "gpio.h"
#include "lis3dh.h"
#include "power.h"
#include "sd.h"

#include "api_mqtt.h"

#define TRACE_GLOBAL_LEVEL TRACE_LEVEL_INFO

#define CAPTURE_CHECK_INTERVAL_MS 2000
```

```
#define MAIN_TASK_STACK_SIZE (1024 * 2)
#define MAIN_TASK_PRIORITY 0
#define MAIN_TASK_NAME "MAIN Test Task"

#define TEST_TASK_STACK_SIZE (1024 * 2)
#define TEST_TASK_PRIORITY 1
#define TEST_TASK_NAME "GPIO Test Task"

#define EVENT_TASK_STACK_SIZE (1024 * 2)
#define EVENT_TASK_PRIORITY 1
#define EVENT_TASK_NAME "Event Task"

#define GPS_TASK_STACK_SIZE (1024 * 2)
#define GPS_TASK_PRIORITY 2
#define GPS_TASK_NAME "GPS Task"

#define GPRS_TASK_STACK_SIZE (1024 * 2)
#define GPRS_TASK_PRIORITY 2
#define GPRS_TASK_NAME "GPRS Task"

static HANDLE mainTaskHandle = NULL;
static HANDLE secondTaskHandle = NULL;

// ************************************************************

static HANDLE gpsTaskHandle = NULL;
static HANDLE gprsTaskHandle = NULL;

static HANDLE semCaptureGPS;
static HANDLE semConnectGPRS;

static volatile bool gsmReady = false;
static volatile bool gpsReady = false;
static volatile bool capture_4_cell = false;
// static volatile bool registerReady = false;
static volatile bool gprsReady = false;

static HANDLE eventTaskHandle = NULL;

// ************************************************************
// Acelerometer
```

```
#define LIS3DH_ADDR 0x19         // ou 0x19 dependendo do AD0/SA0
#define LIS3DH_I2C I2C2          // barramento I2C que você usa
#define LIS3DH_INT_PIN GPIO_PIN2 // IO1 ligado no INT1 do LIS3DH
#define I2C_DEFAULT_TIME_OUT 10  // 10ms

// ****************************************************************
// GPS

typedef struct
{
   uint32_t id;
   uint32_t param1;
   uint32_t param2;
} MyEvent_t;

#define EVENT_GPS_START 0x7001
#define EVENT_GPS_STOP 0x7002
#define EVENT_GPS_DATA 0x7003

bool isGpsOn = true;

// static HANDLE gpsEventHandle = NULL;

// ****************************************************************
// AGPS
HANDLE semGetCellInfo = NULL;
float latitudeLbs = 0.0;
float longitudeLbs = 0.0;
// ****************************************************************
// SMS
const uint8_t unicodeMsg[] = {0x00, 0x61, 0x00, 0x61, 0x00, 0x61, 0x6d, 0x4b, 0x8b, 0xd5, 0x77,
0xed, 0x4f, 0xe1}; // unicode:
const uint8_t gbkMsg[] = {0x62, 0x62, 0x62, 0xB0, 0xA1, 0xB0, 0xA1, 0xB0, 0xA1, 0xB0, 0xA1, 0x63,
0x63, 0x63};     // GBK    :

void extractPhoneNumber(const char *header, char *phone, size_t maxLen);
void SMSInit();
void ServerCenterTest();
void cmd_interpret(const char *content);

// ****************************************************************
// HTTP
```

```
// #define SD_LOG_PATH "/sd/log.csv"
#define HTTP_SERVER "0x0.st"
#define HTTP_PORT 80
uint8_t buffer[1024];

#define CHUNK_SIZE 512
#define HTTP_TIMEOUT_SEC 30

bool UploadCsvTo0x0(void);
// void uploadCsvTo0x0(void);
int Http_Post(const char *domain, int port, const char *path, uint8_t *body, uint16_t bodyLen, char
*retBuffer, int bufferLen);

// *****************************************************************
// MQTT
#define MQTT_TASK_STACK_SIZE (1024 * 2)
#define MQTT_TASK_PRIORITY 1
#define MQTT_TASK_NAME "Mqtt Task"

static HANDLE mqttTaskHandle = NULL;

static HANDLE semMqttStart = NULL;
MQTT_Connect_Info_t ci;

typedef enum
{
   MQTT_EVENT_CONNECTED = 0,
   MQTT_EVENT_DISCONNECTED,
   MQTT_EVENT_MAX
} MQTT_Event_ID_t;

typedef struct
{
   MQTT_Event_ID_t id;
   MQTT_Client_t *client;
} MQTT_Event_t;

typedef enum
{
   MQTT_STATUS_DISCONNECTED = 0,
   MQTT_STATUS_CONNECTED,
```

```
    MQTT_STATUS_MAX
} MQTT_Status_t;

MQTT_Status_t mqttStatus = MQTT_STATUS_DISCONNECTED;

char willMsg[50] = "GPRS 123456789012345 disconnected!";
static volatile bool mqttReady = false;

static MQTT_Client_t *mqttClient = NULL;

void MqttTask(void *pData);
void OnMqttConnection(MQTT_Client_t *client, void *arg, MQTT_Connection_Status_t status);
void MqttTaskEventDispatch(MQTT_Event_t *pEvent);
void Mqtt_connect(void);
void OnMqttReceived(void *arg, const char *topic, uint32_t payloadLen);
void OnMqttReceiedData(void *arg, const uint8_t *data, uint16_t len, MQTT_Flags_t flags);
void OnMqttSubscribed(void *arg, MQTT_Error_t err);

void OnTimerPublish(void *param);
void OnPublish(void *arg, MQTT_Error_t err);

void Mqtt_Start(void);
void OnMqttConnection(MQTT_Client_t *client, void *arg, MQTT_Connection_Status_t status);
bool Mqtt_Publish(const char *topic, const char *payload);
bool Mqtt_Subscribe(const char *topic);

// ****************************************************************
// GERAL
SD_Record_t record;
void OnCaptureTimeout(void *param);
static bool waitingCapture = false;
static bool timeoutExpired = false;
static bool sysReady = false;

uint32_t CAPTURE_TIMEOUT_MS = 60000;

char file_address[50];
static bool f_file_sended = false;
static bool f_sms_sent = false;

HANDLE captureTimer = NULL; //
```

```
uint8_t imei[16];
uint8_t iccid[21];

#define SD_OFFSET_PATH "sd:/log_offset.txt"
static int lastSentLine = 0;
bool SendAllRecordsNow();

// ****************************************************************
// Definitions

#define USE_TRACE 1
#define USE_CALL 1

SystemState state = STATE_INIT;

// protótipos
// void EventDispatch(API_Event_t* event);
void MainTask(void *pData);
void GPIO_TestTask();
void Task_CaptureGPS(void *pData);
void Task_ConnectGPRS(void *pData);
static void InitLIS3DH_Driver(void);
void OnPinFalling(GPIO_INT_callback_param_t *param);

int send_uart(const char *fmt, ...);

void SendUtf8(char *sender, char *sms_message);

//
int interv_led = 100;


void MainTask(void *pData)
{
   OS_Sleep(2000);

   // open UART1 to print NMEA infomation
   UART_Config_t config = {
      .baudRate = UART_BAUD_RATE_115200,
      .dataBits = UART_DATA_BITS_8,
      .stopBits = UART_STOP_BITS_1,
      .parity = UART_PARITY_NONE,
```

```
        .rxCallback = NULL,
        .useEvent = true};
    UART_Init(UART1, config);

    send_uart("\r\nInitializing System\r\n");

    // Tasks ---------------------------------------------------------------------------
    /*secondTaskHandle = OS_CreateTask(GPIO_TestTask,
        NULL, NULL, TEST_TASK_STACK_SIZE, TEST_TASK_PRIORITY, 0, 0, TEST_TASK_NAME);*/

    gpsTaskHandle = OS_CreateTask(Task_CaptureGPS,
                            &gpsTaskHandle, NULL, GPS_TASK_STACK_SIZE, GPS_TASK_PRIORITY, 0,
0, GPS_TASK_NAME);

    mqttTaskHandle = OS_CreateTask(MqttTask,
                             NULL, NULL, MQTT_TASK_STACK_SIZE, MQTT_TASK_PRIORITY, 0, 0,
MQTT_TASK_NAME);

    // ---------------------------------------------------------------------------------------
    Trace(10, "MainTask started v1");

    state = STATE_INIT;

    while (1)
    {
        switch (state)
        {
        case STATE_INIT:
        {
            PM_SleepMode(false);
            PM_SetSysMinFreq(PM_SYS_FREQ_312M);
            // OS_Sleep(2000);

            Trace(10, "Inicializando perifericos...");
            send_uart("Inicializando perifericos...\r\n");

            // Tira do modo avião
            Network_StartAttach();
            Network_SetFlightMode(false);

            PM_PowerEnable(POWER_TYPE_VPAD, true);
            PM_PowerEnable(POWER_TYPE_MMC, false);
```

```
        PM_PowerEnable(POWER_TYPE_LCD, false);
        PM_PowerEnable(POWER_TYPE_CAM, false);

        gsmReady = false;
        gpsReady = false;
        gprsReady = false;
        capture_4_cell = 0;

        for (int i = 0; i < MAX_CELLS; ++i)
        {
            record.cells[i].Mcc = 0;
            record.cells[i].Mnc = 0;
            record.cells[i].Lac = 0;
            record.cells[i].CellId = 0;
            record.cells[i].Bsic = 0;
            record.cells[i].RxLev = 0;
            record.cells[i].RxLevSub = 0;
            record.cells[i].Arfcn = 0;
        }

        record.latitude = 0.0;
        record.longitude = 0.0;
        record.altitude = 0.0;
        record.fixQuality = 0;
        record.satellites = 0;

        // Inicializar barramento I2C
        I2C_Config_t i2cCfg = {
            .freq = I2C_FREQ_100K, // ou 400kHz
        };

        I2C_Init(LIS3DH_I2C, i2cCfg);

        // Testar leitura do WHO_AM_I (0x0F deve retornar 0x33)
        uint8_t whoami = 0;
        if (I2C_ReadMem(LIS3DH_I2C, LIS3DH_ADDR, 0x0F, 1, &whoami, 1,
I2C_DEFAULT_TIME_OUT) == I2C_ERROR_NONE)
        {
            Trace(10, "LIS3DH WHO_AM_I=0x%02X", whoami);

            // Encapsulado: inicializa driver + GPIO callback + registra callback da app
            InitLIS3DH_Driver();
```

```
        }
        else
        {
            Trace(10, "Erro ao comunicar com LIS3DH");
        }

        // Atualiza RTC com dados da ERB
        TIME_SetIsAutoUpdateRtcTime(true);

        // SD Card
        if (!SD_Init())
        {
            send_uart("SD initialized\r\n");
            Trace(10, "SD initialized");
        }
        else
        {
            send_uart("SD not initialized\r\n");
        }

        while (!sysReady)
            OS_Sleep(2000);

        // Start GPS
        MyEvent_t *evStart = (MyEvent_t *)OS_Malloc(sizeof(MyEvent_t));
        if (evStart)
        {
            memset(evStart, 0, sizeof(MyEvent_t));
            evStart->id = EVENT_GPS_START;
            evStart->param1 = 123; // exemplo numérico

            bool sent = OS_SendEvent(gpsTaskHandle, evStart, OS_TIME_OUT_WAIT_FOREVER,
OS_EVENT_PRI_NORMAL);
            // NOTA: NÃO liberar evStart aqui — SecondTask fará OS_Free(evStart)
        }

        state = STATE_CAPTURE;
        break;
    }

    case STATE_CAPTURE:
    {
```

```
        Trace(10, "Iniciando captura de dados...
********************************************************************************");
        send_uart("Iniciando captura de dados...\r\n");

        timeoutExpired = false;
        CAPTURE_TIMEOUT_MS = 10000;
        OS_StartCallbackTimer(mainTaskHandle, CAPTURE_TIMEOUT_MS, OnCaptureTimeout,
NULL);
        uint32_t start = (uint32_t)(clock() / CLOCKS_PER_MSEC);

        strcpy(record.source, "LBS");

        while (!(gpsReady && gsmReady) && !timeoutExpired)
        {

            OS_Sleep(CAPTURE_CHECK_INTERVAL_MS);

            if (!timeoutExpired)
            {
                send_uart("\r\nTempo restante (%d s).\r\n", (CAPTURE_TIMEOUT_MS -
((uint32_t)(clock() / CLOCKS_PER_MSEC) - start)) / 1000);
                send_uart("Esperando dados de localizacao...\r\n");

                if (gpsReady)
                {
                    send_uart("GPS: OK\r\n");
                    strcpy(record.source, "GPS");
                }
                else
                    send_uart("GPS: --\r\n");

                if (gsmReady)
                    send_uart("GSM: OK\r\n");
                else
                    send_uart("GSM: --\r\n");

                if (gsmReady /*&& !capture_4_cell*/)
                {
                    //  LBS
                    if (!Network_GetCellInfoRequst())
                    {
                        send_uart("(LSB) network get cell info fail.\r\n");
```

```
                }
                else
                {
                    // send_uart("(LSB) network get cell info succsess.\r\n");
                }

                for (int i = 0; i < MAX_CELLS; ++i)
                {
                    send_uart("LBS: Cell %d info:%d,%d,%d,%d,%d,%d,%d,%d\r\n",
                            i,
                            record.cells[i].Mcc,
                            record.cells[i].Mnc,
                            record.cells[i].Lac,
                            record.cells[i].CellId,
                            record.cells[i].Bsic,
                            record.cells[i].RxLev,
                            record.cells[i].RxLevSub,
                            record.cells[i].Arfcn);
                }
            }

            send_uart("GPS: latitude: %f\r\n", record.latitude);
            send_uart("GPS: longitude: %f\r\n", record.longitude);
            send_uart("GPS: altitude: %f\r\n", record.altitude);
            send_uart("GPS: fixQuality: %d\r\n", record.fixQuality);
            send_uart("GPS: satellites: %d\r\n", record.satellites);
        }
    }

    OS_StopCallbackTimer(mainTaskHandle, OnCaptureTimeout, NULL);

    Trace(10, "Dados capturados.");
    send_uart("Dados capturados.\r\n\r\n");

    // send_uart("Data & Hora. ********************************\r\n");
    TIME_System_t localTime;
    TIME_GetLocalTime(&localTime);
    // send_uart("localtime:%d-%d-%d %02d:%02d:%02d", localTime.year, localTime.month,
localTime.day, localTime.hour, localTime.minute, localTime.second);
    // send_uart("\r\nData & Hora. ********************************\r\n");

    snprintf(record.timestamp, sizeof(record.timestamp),
```

```
        "%04d-%02d-%02d %02d:%02d:%02d",
        localTime.year,
        localTime.month,
        localTime.day,
        localTime.hour,
        localTime.minute,
        localTime.second);

uint8_t percent;
uint16_t v = PM_Voltage(&percent);
record.percent = percent;
send_uart("\r\n\r\npower:%d -- %d%%\r\n", v, percent);

memset(imei, 0, sizeof(imei));
INFO_GetIMEI(imei);
memcpy(record.imei, imei, sizeof(record.imei));
send_uart("IMEI: %s\r\n", imei);

memset(iccid, 0, sizeof(iccid));
SIM_GetICCID(iccid);
memcpy(record.iccid, iccid, sizeof(record.iccid));
send_uart("ICCID: %s\r\n", iccid);

send_uart("GPS: Timestamp: %s\r\n", record.timestamp);
send_uart("GPS: Source: %s\r\n", record.source);
send_uart("GPS: latitude: %f\r\n", record.latitude);
send_uart("GPS: longitude: %f\r\n", record.longitude);
send_uart("GPS: altitude: %f\r\n", record.altitude);
send_uart("GPS: fixQuality: %d\r\n", record.fixQuality);
send_uart("GPS: satellites: %d\r\n", record.satellites);
send_uart("GPS: cellCount: %d\r\n", record.cellCount);

for (int i = 0; i < MAX_CELLS; ++i)
{
   send_uart("LBS: cell %d info:%d,%d,%d,%d,%d,%d,%d,%d\r\n",
            i,
            record.cells[i].Mcc,
            record.cells[i].Mnc,
            record.cells[i].Lac,
            record.cells[i].CellId,
            record.cells[i].Bsic,
            record.cells[i].RxLev,
```

```
                record.cells[i].RxLevSub,
                record.cells[i].Arfcn);
        }

        send_uart("\r\n\r\n");

        SD_AppendRecord(&record);
        send_uart("Dados armazenados.\r\n");

        // Stop GPS
        MyEvent_t *evStop = (MyEvent_t *)OS_Malloc(sizeof(MyEvent_t));
        if (evStop)
        {
            memset(evStop, 0, sizeof(MyEvent_t));
            evStop->id = EVENT_GPS_STOP;
            evStop->param1 = 123; // exemplo numérico

            bool sent = OS_SendEvent(gpsTaskHandle, evStop, OS_TIME_OUT_WAIT_FOREVER,
OS_EVENT_PRI_NORMAL);
            // NOTA: NÃO liberar evStop aqui — SecondTask fará OS_Free(evStop)
        }

        state = STATE_SEND;
        break;
    }

    case STATE_SEND:
    {
        Trace(10, "Enviando dados...");

        // gprsReady = false;
        uint8_t status;
        bool ret = Network_GetAttachStatus(&status);
        if (!ret)
        {
            Trace(1, "get attach staus fail");
            send_uart("get attach staus fail\r\n");
        }
        Trace(1, "attach status:%d", status);
        send_uart("attach status:%d\r\n", status);

        if ((status == 0))
```

```
        {
            ret = Network_StartAttach();
            if (!ret)
            {
                Trace(1, "network attach fail");
                send_uart("network attach fail");
            }
        }
        else
        {
            Network_PDP_Context_t context = {
                .apn = "timbrasil.br",
                .userName = "tim",
                .userPasswd = "tim"};
            Network_StartActive(context);
        }

        timeoutExpired = false;
        CAPTURE_TIMEOUT_MS = 30000;
        OS_StartCallbackTimer(mainTaskHandle, CAPTURE_TIMEOUT_MS, OnCaptureTimeout,
NULL);
        uint32_t start = (uint32_t)(clock() / CLOCKS_PER_MSEC);

        while (!gprsReady && !timeoutExpired)
        {
            OS_Sleep(CAPTURE_CHECK_INTERVAL_MS);

            if (!timeoutExpired)
            {
                send_uart("Esperando conexao GPRS...\r\n");

                send_uart("\r\nTempo restante (%d s).\r\n", (CAPTURE_TIMEOUT_MS -
((uint32_t)(clock() / CLOCKS_PER_MSEC) - start)) / 1000);
            }
        }

        OS_StopCallbackTimer(mainTaskHandle, OnCaptureTimeout, NULL);

        if (gprsReady)
        {
            send_uart("GPRS conectado!\r\n");
```

```
            timeoutExpired = false;
            CAPTURE_TIMEOUT_MS = 10000;
            OS_StartCallbackTimer(mainTaskHandle, CAPTURE_TIMEOUT_MS, OnCaptureTimeout,
NULL);
            uint32_t start = (uint32_t)(clock() / CLOCKS_PER_MSEC);

            while ((mqttStatus != MQTT_STATUS_CONNECTED) && !timeoutExpired)
            {
               OS_Sleep(CAPTURE_CHECK_INTERVAL_MS);

               if (!timeoutExpired)
               {
                  send_uart("Esperando conexao ao broker MQTT...\r\n");

                  send_uart("\r\nTempo restante (%d s).\r\n", (CAPTURE_TIMEOUT_MS -
((uint32_t)(clock() / CLOCKS_PER_MSEC) - start)) / 1000);
               }

               // Mqtt_connect();
               Mqtt_Start();
            }

            if (mqttStatus == MQTT_STATUS_CONNECTED)
            {
               // Mqtt_Publish("a9g/app", "Hello MQTT!");
               send_uart("Comecar a enviar!\r\n");

               while (1)
               {
                  if (SendAllRecordsNow())
                  {
                     Trace(1, "Envio completo!");
                     send_uart("Envio completo!\r\n");
                     break;
                  }
                  else
                  {
                     Trace(1, "Erro durante o envio!");
                     send_uart("Erro durante o envio!\r\n");
                  }
               }
```

```
            // MqttSyncTask();
        }

        MQTT_Error_t err = MQTT_Disconnect(mqttClient);

        SD_DeleteAll();

        /*f_file_sended = false;
        UploadCsvTo0x0();

        if (f_file_sended)
        {
            f_sms_sent = false;
            SendUtf8("11973707375", file_address);
            SD_DeleteAll();
            OS_StopCallbackTimer(mainTaskHandle, OnCaptureTimeout, NULL);
            captureTimer = NULL;

            timeoutExpired = false;
            CAPTURE_TIMEOUT_MS = 5000;
            OS_StartCallbackTimer(mainTaskHandle, CAPTURE_TIMEOUT_MS,
OnCaptureTimeout, NULL);
            uint32_t start = (uint32_t)(clock() / CLOCKS_PER_MSEC);
            while (!f_sms_sent && !timeoutExpired)
            {
                OS_Sleep(CAPTURE_CHECK_INTERVAL_MS);

                if (!timeoutExpired)
                {
                    send_uart("Esperando SMS...\r\n");

                    send_uart("\r\nTempo restante (%d s).\r\n", (CAPTURE_TIMEOUT_MS -
((uint32_t)(clock() / CLOCKS_PER_MSEC) - start)) / 1000);
                }
            }
            OS_StopCallbackTimer(mainTaskHandle, OnCaptureTimeout, NULL);
        }*/
    }

    // while(true);

    // Network_StartDeactive
```

```
    uint8_t active = 0;
    Network_GetActiveStatus(&active);

    if (active == 1)
    {
        Network_StartDeactive(1); // geralmente o contexto é 1

        while (active == 1)
        {
            send_uart("Esperando desativar...\r\n");
            OS_Sleep(1000);
            Network_GetActiveStatus(&active);
        }
    }

    uint8_t attached = 0;
    Network_GetAttachStatus(&attached);

    if (attached == 1)
    {
        Network_StartDetach();

        while (attached == 1)
        {
            send_uart("Esperando desatachar...\r\n");
            OS_Sleep(1000);
            Network_GetAttachStatus(&attached);
        }
    }

    Network_SetFlightMode(true);

    OS_Sleep(1000);

    state = STATE_SLEEP;
    send_uart("Entrando em modo de baixo consumo...\r\n");
    break;
}

case STATE_SLEEP:
{
    // Trace(10, "Entrando em modo de baixo consumo...");
```

```
            // ------------------------------------------------------
            // LIS3DH_EnterLowPower();

            // GPIO_ClearInt(LIS3DH_INT_PIN);

            OS_StopCallbackTimer(mainTaskHandle, OnCaptureTimeout, NULL);

            PM_SleepMode(true);
            // PM_ShutDown();

            OS_Sleep(10000); //
            break;
        }

        default:
        {
            // Segurança: retorne ao INIT se algo estranho acontecer
            state = STATE_INIT;
            break;
        }
        }

        OS_Sleep(100); // Aguarda ciclo
    }
}

void EventTask(void *pData)
{
    API_Event_t *event = NULL;
    static uint8_t lbsCount = 0;

    while (1)
    {
        if (OS_WaitEvent(mainTaskHandle, (void **)&event, OS_TIME_OUT_WAIT_FOREVER))
        {
            switch (event->id)
            {
            // SMS Events ------------------------
            case API_EVENT_ID_SMS_SENT:
                send_uart("\r\nSMS SENT!!!!\r\n");
                f_sms_sent = true;
```

```
            break;

        case API_EVENT_ID_SMS_RECEIVED:
        {
            Trace(1, "SMS RECEIVED!!!");
            send_uart("\r\nSMS RECEIVED!!!!\r\n");

            SMS_Encode_Type_t encodeType = event->param1;
            uint32_t contentLength = event->param2;
            uint8_t *header = event->pParam1;
            uint8_t *content = event->pParam2;

            send_uart("message header: %s\r\n", header);
            // send_uart("message content length: %d\r\n", contentLength);
            send_uart("message content: %s\r\n", content);

            char phone[32];
            extractPhoneNumber(header, phone, sizeof(phone));
            // send_uart("message sender: %s\r\n", phone);

            cmd_interpret(content);

            // send_uart("\r\nListando SMSs... **********************************************************\r\n\r\n");
            SMS_ListMessageRequst(SMS_STATUS_ALL, SMS_STORAGE_SIM_CARD);

            break;
        }

        case API_EVENT_ID_SMS_LIST_MESSAGE:
        {
            SMS_Message_Info_t *messageInfo = (SMS_Message_Info_t *)event->pParam1;

            send_uart("\r\nmessage header index:%d,status:%d,number
type:%d,number:%s,time:\"%u/%02u/%02u,%02u:%02u:%02u+%02d\"\r\n",
                    messageInfo->index,
                    messageInfo->status,
                    messageInfo->phoneNumberType,
                    messageInfo->phoneNumber,
                    messageInfo->time.year, messageInfo->time.month, messageInfo->time.day,
                    messageInfo->time.hour, messageInfo->time.minute, messageInfo->time.second,
                    messageInfo->time.timeZone);
```

```
            send_uart("message content len:%d, data:%s\r\n", messageInfo->dataLen,
messageInfo->data);

            ServerCenterTest();

            if (!SMS_DeleteMessage(messageInfo->index, SMS_STATUS_ALL,
SMS_STORAGE_SIM_CARD))
            {
               send_uart("delete sms fail\r\n");
            }
            else
            {
               send_uart("delete sms index %d success\r\n", messageInfo->index);
            }

            /*SMS_Storage_Info_t storageInfo;

            SMS_GetStorageInfo(&storageInfo,SMS_STORAGE_SIM_CARD);
            send_uart("sms storage sim card info,
used:%d,total:%d\r\n",storageInfo.used,storageInfo.total);


if(!SMS_DeleteMessage(messageInfo->index,SMS_STATUS_ALL,SMS_STORAGE_SIM_CARD))
            send_uart("delete sms fail\r\n");
            else send_uart("delete sms success\r\n");*/

            // SMS_ListMessageRequst(SMS_STATUS_ALL,SMS_STORAGE_SIM_CARD);

            // need to free data here
            OS_Free(messageInfo->data);
            break;
         }

         case API_EVENT_ID_SMS_ERROR:
            send_uart("SMS error occured! cause:%d\r\n", event->param1);
            break;
// ------------------------------------

// CALL Events ------------------------
#if USE_CALL
         case API_EVENT_ID_CALL_HANGUP: // param1: is remote release call, param2:error
code(CALL_Error_t)
```

```
            Trace(1, "Hang up,is remote hang up:%d, error code:%d", event->param1, event->param2);
            send_uart("Hang up,is remote hang up:%d, error code:%d\r\n", event->param1,
event->param2);
            break;
        case API_EVENT_ID_CALL_INCOMING: // param1: number type, pParam1:number
            Trace(1, "Receive a call, number:%s, number type:%d", event->pParam1, event->param1);
            send_uart("Receive a call, number:%s, number type:%d\r\n", event->pParam1,
event->param1);
            OS_Sleep(2000);
            if (!CALL_Answer())
            {
                Trace(1, "answer fail");
                send_uart("answer fail\r\n");
            }
            else
            {
                Trace(1, "Call answered");
                send_uart("Call answered\r\n");
            }
            break;
        case API_EVENT_ID_CALL_ANSWER:
            Trace(1, "answer success");
            send_uart("answer success\r\n");
            break;
        case API_EVENT_ID_CALL_DTMF: // param1: key
            Trace(1, "received DTMF tone:%c", event->param1);
            break;
#endif
        // -------------------------------------
        case API_EVENT_ID_KEY_DOWN:
            Trace(1, "power key press down now");
            break;

        case API_EVENT_ID_KEY_UP:
            Trace(1, "power key press up now");
            break;

        case API_EVENT_ID_NO_SIMCARD:
            Trace(10, "!!NO SIM CARD%d!!!!", event->param1);
            // Network_SetFlightMode(true);
            Trace(1, "MODO AVIAOOOOOOOOO");
```

```
            // networkFlag = false;
            break;

        case API_EVENT_ID_POWER_ON:
            Trace(1, "Power On - motivo: %d", event->param1);
            break;

        case API_EVENT_ID_POWER_INFO:
            // uint16_t charge_level = event->param1;
            // uint16_t battery_voltage = event->param2;
            // Trace(1, "Battery status: \n charge_level: %d%% \n battery_voltage: %d (mV)",
charge_level, battery_voltage );
            break;

        case API_EVENT_ID_SYSTEM_READY:
            Trace(1, "Sistema esta pronto
**********************************************************************");
            send_uart("Sistema esta pronto\r\n");
            sysReady = true;
            if (semMqttStart)
                OS_ReleaseSemaphore(semMqttStart);

            SMSInit();

            SMS_ListMessageRequst(SMS_STATUS_ALL, SMS_STORAGE_SIM_CARD);

            // Network_StartAttach();
            // Network_SetFlightMode(false);
            // Trace(1, "Network_StartAttach, boraaa! ******************************");
            break;

        case API_EVENT_ID_NETWORK_REGISTER_SEARCHING:
            Trace(1, "Network register searching
**********************************************************************");
            send_uart("Network register searching...\r\n");
            gsmReady = false;
            break;
        case API_EVENT_ID_NETWORK_REGISTER_DENIED:
            Trace(1, "network register denied");
            gsmReady = false;
            break;
```

```
        case API_EVENT_ID_NETWORK_REGISTER_NO:
            Trace(1, "network register no");
            gsmReady = false;
            break;

        case API_EVENT_ID_NETWORK_REGISTERED_HOME:
        case API_EVENT_ID_NETWORK_REGISTERED_ROAMING:
        {
            gsmReady = true;
            send_uart("\r\n\r\n********************* network register success ****************\r\n\r\n");
            break;
        }

        case API_EVENT_ID_NETWORK_ATTACHED:
            Trace(1, "network attach success");
            send_uart("network attach success\r\n");

            Network_PDP_Context_t context = {
                .apn = "timbrasil.br",
                .userName = "tim",
                .userPasswd = "tim"};
            Network_StartActive(context);
            break;

        case API_EVENT_ID_NETWORK_ACTIVATED:
            Trace(1, "network activate success");
            send_uart("network activate success\r\n");
            gprsReady = true;
            break;

        case API_EVENT_ID_NETWORK_CELL_INFO:
        {
            uint8_t number = event->param1;
            Network_Location_t *location = (Network_Location_t *)event->pParam1;
            Trace(1, "network cell infomation,serving cell number:1, neighbor cell number:%d", number -
1);
            // send_uart("network cell infomation,serving cell number:1, neighbor cell number:%d\r\n",
number - 1);

            // SD_Record_t record;
            // memset(&record, 0, sizeof(SD_Record_t));
            // strcpy(record.source, "LBS");
```

```
            record.cellCount = number;

            for (int i = 0; i < number; ++i)
            {
                /*record.cells[i].Mcc = location[i].sMcc[0] * 100 + location[i].sMcc[1] * 10 +
location[i].sMcc[2];
                record.cells[i].Mnc = location[i].sMnc[0] * 100 + location[i].sMnc[1] * 10 +
location[i].sMnc[2];
                record.cells[i].Lac = location[i].sLac;
                record.cells[i].CellId = location[i].sCellID;
                record.cells[i].rssi = location[i].iRxLev;

                location[i].sMcc[0],
                location[i].sMcc[1],
                location[i].sMcc[2],
                location[i].sMnc[0],
                location[i].sMnc[1],
                location[i].sMnc[2],*/

                record.cells[i].Mcc = location[i].sMcc[0] * 100 + location[i].sMcc[1] * 10 +
location[i].sMcc[2];
                record.cells[i].Mnc = location[i].sMnc[0] * 100 + location[i].sMnc[1] * 10 +
location[i].sMnc[2];

                record.cells[i].Lac = location[i].sLac;
                record.cells[i].CellId = location[i].sCellID;
                record.cells[i].Bsic = location[i].iBsic;
                record.cells[i].RxLev = location[i].iRxLev;
                record.cells[i].RxLevSub = location[i].iRxLevSub;
                record.cells[i].Arfcn = location[i].nArfcn;

                record.cells[i].rssi = location[i].iRxLev;

                /*send_uart("cell %d info:%d%d%d,%d%d%d,%d,%d,%d,%d,%d,%d\r\n", i,
                        location[i].sMcc[0], location[i].sMcc[1], location[i].sMcc[2],
                        location[i].sMnc[0], location[i].sMnc[1], location[i].sMnc[2],
                        location[i].sLac, location[i].sCellID, location[i].iBsic,
                        location[i].iRxLev, location[i].iRxLevSub, location[i].nArfcn);*/
            }

            if (number == 4)
            {
```

```
            capture_4_cell = 1;
        }

        OS_ReleaseSemaphore(semGetCellInfo);
        lbsCount = 0;
        break;
    }

    // ----------------------------------------------------------
    case API_EVENT_ID_GPS_UART_RECEIVED:
    {
        // Trace(1, "GPS UART RECEIVEDDDD");
        //  dentro do handler UART do GPS (ou onde o módulo envia NMEA)
        // Trace(1, "RAW GPS data: %s", (char*)event->pParam1);
        // send_uart("\r\nRAW GPS data: %s\r\n", (char*)event->pParam1);

        // Refresh data GPS to buffer
        MyEvent_t *evData = (MyEvent_t *)OS_Malloc(sizeof(MyEvent_t));
        if (evData)
        {
            memset(evData, 0, sizeof(MyEvent_t));
            evData->id = EVENT_GPS_DATA;
            evData->param1 = 123; // dummy

            bool sent = OS_SendEvent(gpsTaskHandle, evData, OS_TIME_OUT_WAIT_FOREVER,
OS_EVENT_PRI_NORMAL);
        }

        GPS_Update(event->pParam1, event->param1);
        break;
    }

    case API_EVENT_ID_UART_RECEIVED:
        Trace(1, "UART %d recebeu %d bytes", event->param1, event->param2);
        // GPS
        if (event->param1 == UART1)
        {
            uint8_t data[event->param2 + 1];
            data[event->param2] = 0;
            memcpy(data, event->pParam1, event->param2);
            Trace(1, "uart received data,length:%d,data:%s", event->param2, data);
            if (strcmp(data, "close") == 0)
```

```
                {
                    Trace(1, "close gps");
                    GPS_Close();
                    isGpsOn = false;
                }
                else if (strcmp(data, "open") == 0)
                {
                    Trace(1, "open gps");
                    GPS_Open(NULL);
                    isGpsOn = true;
                }
            }
            break;

        case API_EVENT_ID_SIGNAL_QUALITY:
            // Trace(1,"SQ: %d, RXQUAL: %d, RSSI: %d*2-113", event->param1, event->param2,
event->param1 );
            break;

        default:
            Trace(1, "Evento desconhecido: 0x%x", event->id);
            break;
        }

        // EventDispatch(event);

        // Liberação de memória alocada automaticamente pelo sistema para os ponteiros
        if (event->pParam1)
            OS_Free(event->pParam1);
        if (event->pParam2)
            OS_Free(event->pParam2);
        OS_Free(event);
    }
  }
}

void app_Main()
{
  mainTaskHandle = OS_CreateTask(MainTask,
                        NULL, NULL, MAIN_TASK_STACK_SIZE, MAIN_TASK_PRIORITY, 0, 0,
MAIN_TASK_NAME);
  OS_SetUserMainHandle(&mainTaskHandle);
```

```
    eventTaskHandle = OS_CreateTask(EventTask, NULL, NULL, EVENT_TASK_STACK_SIZE,
EVENT_TASK_PRIORITY, 0, 0, EVENT_TASK_NAME);
}

// --- MQTT task ---
void MqttTask(void *pData)
{
    MQTT_Event_t *event = NULL;

    semMqttStart = OS_CreateSemaphore(0);
    OS_WaitForSemaphore(semMqttStart, OS_WAIT_FOREVER);
    OS_DeleteSemaphore(semMqttStart);
    semMqttStart = NULL;

    Trace(1, "start mqtt task");
    send_uart("start mqtt task\r\n");

    INFO_GetIMEI(imei);
    Trace(1, "IMEI:%s", imei);
    send_uart("IMEI:%s\r\n", imei);

    while (1)
    {
        if (OS_WaitEvent(mqttTaskHandle, (void **)&event, OS_TIME_OUT_WAIT_FOREVER))
        {
            MqttTaskEventDispatch(event);
            OS_Free(event);
        }
    }
}


void MqttTaskEventDispatch(MQTT_Event_t *pEvent)
{
    switch (pEvent->id)
    {
    case MQTT_EVENT_CONNECTED:

        mqttStatus = MQTT_STATUS_CONNECTED;
        Trace(1, "MQTT connected, now subscribe topic:%s", SUBSCRIBE_TOPIC);
        send_uart("MQTT connected, now subscribe topic:%s\r\n", SUBSCRIBE_TOPIC);
```

```
        // reconnectInterval = 3000;
        MQTT_Error_t err;
        MQTT_SetInPubCallback(pEvent->client, OnMqttReceived, OnMqttReceiedData, NULL);
        err = MQTT_Subscribe(pEvent->client, SUBSCRIBE_TOPIC, 2, OnMqttSubscribed, (void
*)SUBSCRIBE_TOPIC);
        if (err != MQTT_ERROR_NONE)
        {
            Trace(1, "MQTT subscribe error, error code:%d", err);
            send_uart("MQTT subscribe error, error code:%d\r\n", err);
        }
        // StartTimerPublish(PUBLISH_INTERVAL,pEvent->client);
        break;
    case MQTT_EVENT_DISCONNECTED:
        mqttStatus = MQTT_STATUS_DISCONNECTED;
        // StartTimerConnect(reconnectInterval,pEvent->client);
        break;
    default:
        break;
    }
}

void Mqtt_connect(void)
{
    MQTT_Client_t *client = MQTT_ClientNew();

    MQTT_Error_t err;
    memset(&ci, 0, sizeof(MQTT_Connect_Info_t));
    ci.client_id = imei;
    ci.client_user = CLIENT_USER;
    ci.client_pass = CLIENT_PASS;
    ci.keep_alive = 20;
    ci.clean_session = 1;
    ci.use_ssl = false;
    ci.will_qos = 2;
    ci.will_topic = "will";
    ci.will_retain = 1;
    memcpy(strstr(willMsg, "GPRS") + 5, imei, 15);
    ci.will_msg = willMsg;

    err = MQTT_Connect(client, BROKER_IP, BROKER_PORT, OnMqttConnection, NULL, &ci);
    if (err != MQTT_ERROR_NONE)
    {
```

```
        Trace(1, "MQTT connect fail,error code:%d", err);
        send_uart("MQTT connect fail,error code:%d\r\n", err);
        mqttReady = false;
    }
    else
    {
        mqttReady = true;
        send_uart("MQTT connect success!\r\n");
    }
}

void OnMqttReceived(void *arg, const char *topic, uint32_t payloadLen)
{
    Trace(1, "MQTT received publish data request, topic:%s, payload length:%d", topic, payloadLen);
    send_uart("MQTT received publish data request, topic:%s, payload length:%d\r\n", topic,
payloadLen);
}

void OnMqttReceiedData(void *arg, const uint8_t *data, uint16_t len, MQTT_Flags_t flags)
{
    Trace(1, "MQTT recieved publish data,  length:%d,data:%s", len, data);
    send_uart("MQTT recieved publish data,  length:%d,data:%s\r\n", len, data);
    if (flags == MQTT_FLAG_DATA_LAST)
        Trace(1, "MQTT data is last frame");
    send_uart("MQTT data is last frame\r\n");
}

void OnMqttSubscribed(void *arg, MQTT_Error_t err)
{
    if (err != MQTT_ERROR_NONE)
    {
        Trace(1, "MQTT subscribe fail,error code:%d", err);
        send_uart("MQTT subscribe fail,error code:%d\r\n", err);
    }
    else
        Trace(1, "MQTT subscribe success,topic:%s", (const char *)arg);
    send_uart("MQTT subscribe success,topic:%s\r\n", (const char *)arg);
}

void OnTimerPublish(void *param)
{
    MQTT_Error_t err;
```

```
    MQTT_Client_t *client = (MQTT_Client_t *)param;

    send_uart("MQTT OnTimerPublish\r\n");
    err = MQTT_Publish(client, PUBLISH_TOPIC, PUBLISH_PAYLOEAD,
strlen(PUBLISH_PAYLOEAD), 1, 2, 0, OnPublish, NULL);
    if (err != MQTT_ERROR_NONE)
    {
        Trace(1, "MQTT publish error, error code:%d", err);
        send_uart("MQTT publish error, error code:%d\r\n", err);
    }
    else
    {
        send_uart("MQTT publish payload: %s \r\n", PUBLISH_PAYLOEAD);
    }

    /* MQTT_Error_t err;
     MQTT_Client_t* client = (MQTT_Client_t*)param;
     uint8_t status = MQTT_IsConnected(client);
     Trace(1,"mqtt status:%d",status);
     send_uart("mqtt status:%d\r\n",status);
     if(mqttStatus != MQTT_STATUS_CONNECTED)
     {
         Trace(1,"MQTT not connected to broker! can not publish");
         send_uart("MQTT not connected to broker! can not publish\r\n");
         return;
     }
     Trace(1,"MQTT OnTimerPublish");
     send_uart("MQTT OnTimerPublish\r\n");
     err =
MQTT_Publish(client,PUBLISH_TOPIC,PUBLISH_PAYLOEAD,strlen(PUBLISH_PAYLOEAD),1,2,0,O
nPublish,NULL);
     if(err != MQTT_ERROR_NONE)
         {
         Trace(1,"MQTT publish error, error code:%d",err);
         send_uart("MQTT publish error, error code:%d\r\n",err);
         }
     else
     {
         send_uart("MQTT publish payload: %s \r\n",PUBLISH_PAYLOEAD);
     }
     //StartTimerPublish(PUBLISH_INTERVAL,client); */
}
```

```
void OnPublish(void *arg, MQTT_Error_t err)
{
   if (err == MQTT_ERROR_NONE)
   {
      Trace(1, "MQTT publish success");
      send_uart("MQTT publish success\r\n");
   }
   else
   {
      Trace(1, "MQTT publish error, error code:%d", err);
      send_uart("MQTT publish error, error code:%d\r\n", err);
   }
}




void Mqtt_Start(void)
{
   mqttClient = MQTT_ClientNew();
   memset(&ci, 0, sizeof(ci));

   INFO_GetIMEI(imei);

   ci.client_id = imei;
   ci.clean_session = 1;
   ci.keep_alive = 30;
   ci.use_ssl = false;

   MQTT_SetInPubCallback(mqttClient, OnMqttReceived, OnMqttReceiedData, NULL);

   MQTT_Connect(
      mqttClient,
      BROKER_IP,
      BROKER_PORT,
      OnMqttConnection,
      NULL,
      &ci);
}

void OnMqttConnection(MQTT_Client_t *client, void *arg, MQTT_Connection_Status_t status)
```

```
{
    if (status == MQTT_CONNECTION_ACCEPTED)
    {
        mqttStatus = MQTT_STATUS_CONNECTED;
        Trace(1, "MQTT CONNECTED!");
        send_uart("MQTT CONNECTED!\r\n");
    }
    else
    {
        mqttStatus = MQTT_STATUS_DISCONNECTED;
        Trace(1, "MQTT FAILED! Code=%d", status);
        send_uart("MQTT FAILED! Code=%d\r\n", status);
    }
}

bool Mqtt_Publish(const char *topic, const char *payload)
{
    if (mqttStatus != MQTT_STATUS_CONNECTED)
    {
        Trace(1, "Cannot publish, not connected");
        send_uart("Cannot publish, not connected\r\n");
        return false;
    }

    MQTT_Error_t err = MQTT_Publish(
        mqttClient,
        topic,
        payload,
        strlen(payload),
        1, // QoS
        0, // Retain
        0, // Duplicate
        NULL,
        NULL);

    if (err != MQTT_ERROR_NONE)
    {
        Trace(1, "Publish error: %d", err);
        send_uart("Publish error: %d\r\n", err);
        return false;
    }
```

```
    return true;
}

bool Mqtt_Subscribe(const char *topic)
{
    if (mqttStatus != MQTT_STATUS_CONNECTED)
        return false;

    MQTT_Error_t err = MQTT_Subscribe(
        mqttClient,
        topic,
        1,
        OnMqttSubscribed,
        (void *)topic);

    return (err == MQTT_ERROR_NONE);
}

void MqttSyncTask(void *pData)
{
    LoadLastSentLine();
    while (1)
    {
        if (!SendNextRecordFromSD())
            OS_Sleep(60000); // Tenta de novo em 1 minuto
        else
            OS_Sleep(2000); // Pequeno delay entre envios
    }
}




// Lê uma linha do arquivo manualmente (até \n ou EOF)
static int ReadLine(int fd, char *buffer, int maxLen)
{
    int i = 0;
    char c;
    int readBytes;

    while (i < (maxLen - 1))
    {
```

```
        readBytes = API_FS_Read(fd, &c, 1);
        if(readBytes <= 0)
        {
            break; // EOF ou erro
        }

        buffer[i++] = c;

        if(c == '\n')
            break;
    }

    buffer[i] = '\0';

    if(i == 0)
        return -1; // Nenhum byte lido

    return i;
}


bool SendAllRecordsNow()
{
    if(mqttStatus != MQTT_STATUS_CONNECTED)
    {
        Trace(1, "MQTT not connected");
        send_uart("MQTT not connected\r\n");
        return false;
    }

    int fd = API_FS_Open(SD_LOG_PATH, FS_O_RDONLY, 0);
    if(fd < 0)
    {
        Trace(1, "Failed to open SD log");
        send_uart("Failed to open SD log\r\n");
        return false;
    }

    char line[512];
    int count = 0;
    int len;
```

```
    while(true)
    {
        len = ReadLine(fd, line, sizeof(line));
        if(len <= 0)
        {
            Trace(1, "No more data to send! Total: %d", count);
            send_uart("No more data to send! Total: %d\r\n", count);
            break;
        }

        // Remover \r e \n ao final
        while(len > 0 && (line[len-1] == '\n' || line[len-1] == '\r'))
        {
            line[--len] = '\0';
        }

        Trace(1, "Sending [%d]: %s", count + 1, line);
        send_uart("Sending [%d]: %s\r\n", count + 1, line);

        if(!Mqtt_Publish(PUBLISH_TOPIC, line))
        {
            Trace(1, "Failed to publish line %d", count + 1);
            send_uart("Failed to publish line %d\r\n", count + 1);
            API_FS_Close(fd);
            return false; // Interrompe envio se falhar
        }

        count++;
        OS_Sleep(1000); // anti-flood no modem
    }

    API_FS_Close(fd);

    Trace(1, "SendAllRecordsNow() finished OK!");
    send_uart("SendAllRecordsNow() finished OK!\r\n");
    return true;
}
```

```
void Task_CaptureGPS(void *pData)
{
    MyEvent_t *ev = NULL;

    GPS_Info_t *gpsInfo = Gps_GetInfo();
    uint8_t buffer[300];

    Trace(1, "Task_CaptureGPS started");
    send_uart("Task_CaptureGPS started\r\n");

    while (!gsmReady)
    {
        Trace(1, "wait for gsm register complete");
        OS_Sleep(1000);

        int rssi = 0;
        Network_GetSignalQuality(&rssi);
        Trace(1, "RSSI inicial: %d", rssi);

        bool rfMode = true;
        Network_GetFlightMode(&rfMode);
        Trace(1, "RF ligado? %d", rfMode);
    }

    while (1)
    {
        //-----------------------------------------
        // Espera evento enviado PARA ESTA task
        //-----------------------------------------
        // OS_WaitEvent(hSelf, &ev, OS_TIME_OUT_WAIT_FOREVER);
        // OS_WaitEvent(gpsTaskHandle, (void **)&ev, OS_TIME_OUT_WAIT_FOREVER);

        if (OS_WaitEvent(gpsTaskHandle, (void **)&ev, OS_TIME_OUT_WAIT_FOREVER))
```

```
{
    // Aqui você processa o evento conforme seu tipo
    switch (ev->id)
    {
    // SMS Events ------------------------
    case EVENT_GPS_START:
    {
        send_uart("*********************************\r\n");
        send_uart("**  Received EVENT_GPS_START  **\r\n");
        send_uart("*********************************\r\n");

        // --- ligar GPS, configurar etc ---
        GPS_Init();
        GPS_Open(NULL);
        isGpsOn = true;

        // Espera inicialização
        while (gpsInfo->rmc.latitude.value == 0)
            OS_Sleep(1000);

        // set gps nmea output interval
        for (uint8_t i = 0; i < 5; ++i)
        {
            bool ret = GPS_SetOutputInterval(10000);
            send_uart("set gps ret:%d\r\n", ret);
            if (ret)
                break;
            OS_Sleep(1000);
        }

        if (!GPS_GetVersion(buffer, 150))
            send_uart("get gps firmware version fail\r\n");
        else
            send_uart("gps firmware version:%s\r\n", buffer);

        // if(!GPS_SetFixMode(GPS_FIX_MODE_LOW_SPEED))
        // Trace(1,"set fix mode fail");

        if (!GPS_SetOutputInterval(3000))
            send_uart("set nmea output interval fail\r\n");

        send_uart("GPS Init OK!\r\n");
```

```
            break;
        }

        case EVENT_GPS_STOP:
        {
            send_uart("*********************************\r\n");
            send_uart("**   Received EVENT_GPS_STOP  **\r\n");
            send_uart("*********************************\r\n");

            // Caso pegue stop fora da captura
            GPS_Close();
            // GPS_PowerOff();
            isGpsOn = false;
            gpsReady = false;
            break;
        }

        case EVENT_GPS_DATA:
        {
            //send_uart("\r\nGPS DATA!!!!\r\n");

            uint8_t isFixed = gpsInfo->gsa[0].fix_type > gpsInfo->gsa[1].fix_type ?
gpsInfo->gsa[0].fix_type : gpsInfo->gsa[1].fix_type;

            char *isFixedStr;
            if (isFixed == 2)
                isFixedStr = "2D fix";
            else if (isFixed == 3)
            {
                if (gpsInfo->gga.fix_quality == 1)
                    isFixedStr = "3D fix";
                else if (gpsInfo->gga.fix_quality == 2)
                    isFixedStr = "3D/DGPS fix";
            }
            else
                isFixedStr = "no fix";

            if (isFixed > 1)
            {
                gpsReady = true;
            }
```

```
            if (isGpsOn)
            {
                // convert unit ddmm.mmmm to degree(°)
                int temp = (int)(gpsInfo->rmc.latitude.value / gpsInfo->rmc.latitude.scale / 100);
                double latitude = temp + (double)(gpsInfo->rmc.latitude.value - temp *
gpsInfo->rmc.latitude.scale * 100) / gpsInfo->rmc.latitude.scale / 60.0;
                temp = (int)(gpsInfo->rmc.longitude.value / gpsInfo->rmc.longitude.scale / 100);
                double longitude = temp + (double)(gpsInfo->rmc.longitude.value - temp *
gpsInfo->rmc.longitude.scale * 100) / gpsInfo->rmc.longitude.scale / 60.0;

                snprintf(buffer, sizeof(buffer), "GPS fix mode:%d, BDS fix mode:%d, fix quality:%d,
satellites tracked:%d, gps sates total:%d, is fixed:%s, coordinate:WGS84, Latitude:%f, Longitude:%f,
unit:degree,altitude:%f", gpsInfo->gsa[0].fix_type, gpsInfo->gsa[1].fix_type,
                        gpsInfo->gga.fix_quality, gpsInfo->gga.satellites_tracked, gpsInfo->gsv[0].total_sats,
isFixedStr, latitude, longitude, gpsInfo->gga.altitude);

                /*send_uart("GPS fix mode:%d, BDS fix mode:%d, fix quality:%d, satellites tracked:%d,
gps sates total:%d, is fixed:%s, coordinate:WGS84, Latitude:%f, Longitude:%f,
unit:degree,altitude:%f\r\n\r\n", gpsInfo->gsa[0].fix_type, gpsInfo->gsa[1].fix_type,
                        gpsInfo->gga.fix_quality, gpsInfo->gga.satellites_tracked,
gpsInfo->gsv[0].total_sats, isFixedStr, latitude, longitude, gpsInfo->gga.altitude);*/

                record.latitude = latitude;
                record.longitude = longitude;
                record.fixQuality = gpsInfo->gga.fix_quality;
                record.satellites = gpsInfo->gga.satellites_tracked;
                if (gpsInfo->gga.altitude.scale > 0)
                {
                    record.altitude = (double)gpsInfo->gga.altitude.value /
                                (double)gpsInfo->gga.altitude.scale;
                }
                else
                {
                    record.altitude = 0.0; // ou um valor especial
                }
            }

            break;
        }
        }

        if (ev->param1)
```

```
            {
                OS_Free(ev->param1);
                ev->param1 = NULL;
            }
            if (ev->param2)
            {
                OS_Free(ev->param2);
                ev->param2 = NULL;
            }

            // liberar o próprio evento (alocado pelo produtor)
            OS_Free(ev);
        }
    }
}


// LIS3DH
// Callback de aplicação: muda estado quando disparo detectado
static void OnLIS3DH_Interrupt(void)
{
    Trace(10, ">>> Disparo detectado! Mudando estado...");
    // g_state = STATE_INIT;
}

void OnPinFalling(GPIO_INT_callback_param_t *param)
{
    // Trace(10,"OnPinFalling");
    switch (param->pin)
    {
    case GPIO_PIN2:
        send_uart("\r\nAccelerometer sensor interrupt!\r\n");
        // GPIO_LEVEL statusNow;
        // GPIO_Get(GPIO_PIN2,&statusNow);
        // Trace(10,"gpio2 status now:%d",statusNow);

        state = STATE_INIT;

        // Melhor reiniciar todo o sistema
        // PM_Restart();

        break;
```

```
    default:
        break;
    }
}

static void InitLIS3DH_Driver(void)
{
    // Inicializa driver do sensor
    LIS3DH_Init(LIS3DH_I2C, LIS3DH_ADDR);

    // Configura pino do INT1 (GPIO_PIN2)
    GPIO_config_t gpioINT = {
        .mode = GPIO_MODE_INPUT_INT,
        .pin = GPIO_PIN2,
        .defaultLevel = GPIO_LEVEL_LOW,
        .intConfig.debounce = 1,
        .intConfig.type = GPIO_INT_TYPE_RISING_EDGE,
        .intConfig.callback = OnPinFalling};

    GPIO_Init(gpioINT);

    // Registra o callback de aplicação (alto nível) dentro do driver
    LIS3DH_EnableInterrupt(OnLIS3DH_Interrupt);

    Trace(10, "LIS3DH driver inicializado e interrupcao configurada.");
}

int send_uart(const char *fmt, ...)
{
    char buffer[256];
    va_list args;
    va_start(args, fmt);
    int len = vsnprintf(buffer, sizeof(buffer), fmt, args);
    va_end(args);

    if (len > 0)
        UART_Write(UART1, (uint8_t *)buffer, strlen(buffer));

    return len;
}

void SendUtf8(char *sender, char *sms_message)
```

```
{
    uint8_t *unicode = NULL;
    uint32_t unicodeLen;

    Trace(1, "sms start send UTF-8 message");

    if (!SMS_LocalLanguage2Unicode(sms_message, strlen(sms_message), CHARSET_UTF_8,
&unicode, &unicodeLen))
    {
        Trace(1, "local to unicode fail!");
        return;
    }

    if (!SMS_SendMessage(sender, unicode, unicodeLen, SIM0))
    {
        Trace(1, "sms send message fail");
        send_uart("sms send message fail\r\n");
    }
    else
        send_uart("sms send message success\r\n");
    OS_Free(unicode);
}

void extractPhoneNumber(const char *header, char *phone, size_t maxLen)
{
    const char *start = strchr(header, '"'); // Encontra a primeira aspa
    if (!start)
        return; // Nenhuma aspa encontrada

    start++; // Avança um caractere (depois da aspa)

    const char *end = strchr(start, '"'); // Encontra a próxima aspa
    if (!end)
        return; // Falha se não houver fechamento

    size_t len = end - start; // Calcula o tamanho do número
    if (len >= maxLen)
        len = maxLen - 1; // Evita overflow

    strncpy(phone, start, len);
    phone[len] = '\0'; // Finaliza a string
```

```
    // Remove espaços extras no início, se houver
    while (*phone == ' ')
        memmove(phone, phone + 1, strlen(phone));
}


void SMSInit()
{
    if (!SMS_SetFormat(SMS_FORMAT_TEXT, SIM0))
    {
        send_uart("sms set format error\r\n");
        return;
    }
    SMS_Parameter_t smsParam = {
        .fo = 17,
        .vp = 167,
        .pid = 0,
        .dcs = 8, // 0:English 7bit, 4:English 8 bit, 8:Unicode 2 Bytes
    };
    if (!SMS_SetParameter(&smsParam, SIM0))
    {
        send_uart("sms set parameter error\r\n");
        return;
    }
    if (!SMS_SetNewMessageStorage(SMS_STORAGE_SIM_CARD))
    {
        send_uart("sms set message storage fail\r\n");
        return;
    }
}

void ServerCenterTest()
{
    uint8_t addr[32];
    uint8_t temp;
    SMS_Server_Center_Info_t sca;
    sca.addr = addr;
    SMS_GetServerCenterInfo(&sca);
    // send_uart("server center address:%s,type:%d\r\n",sca.addr,sca.addrType);
    temp = sca.addr[strlen(sca.addr) - 1];
    sca.addr[strlen(sca.addr) - 1] = '0';
    if (!SMS_SetServerCenterInfo(&sca))
```

```
    {
        // send_uart("SMS_SetServerCenterInfo fail\r\n");
    }
    else
    {
        // send_uart("SMS_SetServerCenterInfo success\r\n");
    }
    SMS_GetServerCenterInfo(&sca);
    // send_uart("server center address:%s,type:%d\r\n",sca.addr,sca.addrType);
    sca.addr[strlen(sca.addr) - 1] = temp;
    if (!SMS_SetServerCenterInfo(&sca))
    {
        // send_uart("SMS_SetServerCenterInfo fail\r\n");
    }
    else
    {
        // send_uart("SMS_SetServerCenterInfo success\r\n");
    }
}

void OnCaptureTimeout(void *param)
{
    timeoutExpired = true;
    send_uart("Timeout: closing window.\r\n");
}
```

Código fonte do gerenciador do cartão SD - sd.c

```
#include "sd.h"
#include "api_debug.h"
#include "stdio.h"
#include "string.h"

// ------------------------------------------------
// Inicialização do SD card
// ------------------------------------------------
bool SD_Init(void)
{
    int fd = API_FS_Open(SD_LOG_PATH, FS_O_RDONLY, 0);
    if (fd < 0)
    {
```

```
        Trace(1, "Arquivo de log não existe, criando novo...");
        fd = API_FS_Open(SD_LOG_PATH, FS_O_CREAT | FS_O_RDWR | FS_O_TRUNC, 0);
        if (fd < 0)
        {
            Trace(1, "Falha ao criar arquivo SD!");
            return false;
        }

        const char *header =
            "timestamp,source,latitude,longitude,altitude,fix_quality,"
            "satellites,MCC,MNC,LAC,CELLID,RSSI\n";
        API_FS_Write(fd, header, strlen(header));
    }
    API_FS_Close(fd);

    Trace(1, "SD inicializado com sucesso");
    return true;
}

// ------------------------------------------------
// Grava um novo registro no arquivo CSV
// ------------------------------------------------
bool SD_AppendRecord(const SD_Record_t *record)
{
    if (record == NULL)
        return false;

    int fd = API_FS_Open(SD_LOG_PATH, FS_O_APPEND | FS_O_WRONLY, 0);
    if (fd < 0)
    {
        Trace(1, "Falha ao abrir arquivo SD para append!");
        return false;
    }

    char buffer[512];
    int offset = 0;

    // ---- Cabeçalho principal (dados GPS/LBS) ----
    offset += snprintf(buffer + offset, sizeof(buffer) - offset,
                       "%s,%s,%d,%s,%s,%.6f,%.6f,%.2f,%d,%d,%d",
                       record->imei,
                       record->iccid,
```

```
                record->percent,
                record->timestamp,
                record->source,
                record->latitude,
                record->longitude,
                record->altitude,
                record->fixQuality,
                record->satellites,
                record->cellCount);

    // ---- Células capturadas ----
    for (uint8_t i = 0; i < record->cellCount && i < MAX_CELLS; i++)
    {
        const CellInfo_t *c = &record->cells[i];
        offset += snprintf(buffer + offset, sizeof(buffer) - offset,
                           ",%d,%d,%d,%d,%d,%d,%d,%d,%d",
                           c->Mcc,
                           c->Mnc,
                           c->Lac,
                           c->CellId,
                           c->Bsic,
                           c->RxLev,
                           c->RxLevSub,
                           c->Arfcn,
                           c->rssi);
    }

    // ---- Nova linha ----
    offset += snprintf(buffer + offset, sizeof(buffer) - offset, "\n");

    int written = API_FS_Write(fd, buffer, strlen(buffer));
    API_FS_Close(fd);

    if (written < 0)
    {
        Trace(1, "Erro ao escrever no SD!");
        return false;
    }

    Trace(1, "Registro salvo no SD com sucesso");
    return true;
}
```

```
// ------------------------------------------------
// Apaga uma linha específica do arquivo CSV
// (recria o arquivo sem essa linha)
// ------------------------------------------------
bool SD_DeleteRecord(uint32_t lineNumber)
{
   int fd = API_FS_Open(SD_LOG_PATH, FS_O_RDONLY, 0);
   if (fd < 0)
   {
      Trace(1, "Falha ao abrir arquivo para leitura");
      return false;
   }

   char buffer[512];
   char newContent[4096];
   newContent[0] = '\0';

   uint32_t currentLine = 0;
   int bytesRead;
   char line[256];
   int linePos = 0;

   while ((bytesRead = API_FS_Read(fd, buffer, sizeof(buffer))) > 0)
   {
      for (int i = 0; i < bytesRead; i++)
      {
         char c = buffer[i];
         line[linePos++] = c;

         if (c == '\n' || linePos >= sizeof(line) - 1)
         {
            line[linePos] = '\0';
            if (currentLine != lineNumber)
               strcat(newContent, line);
            linePos = 0;
            currentLine++;
         }
      }
   }

   API_FS_Close(fd);
```

```
    // Recria o arquivo
    fd = API_FS_Open(SD_LOG_PATH, FS_O_CREAT | FS_O_WRONLY | FS_O_TRUNC, 0);
    if (fd < 0)
    {
        Trace(1, "Falha ao reabrir arquivo para regravação");
        return false;
    }

    API_FS_Write(fd, newContent, strlen(newContent));
    API_FS_Close(fd);

    Trace(1, "Linha %d removida com sucesso", lineNumber);
    return true;
}

bool SD_DeleteAll(void)
{
    // Remove o arquivo existente
    if (API_FS_Delete(SD_LOG_PATH) < 0)
    {
        Trace(1, "Falha ao apagar arquivo de log!");
        return false;
    }

    Trace(1, "Arquivo de log apagado com sucesso");

    // Cria novamente o arquivo vazio com o cabeçalho completo
    int fd = API_FS_Open(SD_LOG_PATH, FS_O_CREAT | FS_O_RDWR | FS_O_TRUNC, 0);
    if (fd < 0)
    {
        Trace(1, "Falha ao recriar arquivo de log!");
        return false;
    }

    // ---- Cabeçalho principal (dados GPS/LBS) ----
    const char *headerBase =
        "imei,iccid,percent,timestamp,source,latitude,longitude,altitude,fix_quality,"
        "satellites,cell_count";

    // ---- Cabeçalho para células capturadas ----
    // Cada célula possui 9 campos, e até MAX_CELLS podem ser registradas
```

```
    char header[1024];
    strcpy(header, headerBase);

    for (uint8_t i = 0; i < MAX_CELLS; i++)
    {
        char cellHeader[128];
        snprintf(cellHeader, sizeof(cellHeader),

",MCC_%d,MNC_%d,LAC_%d,CELLID_%d,BSIC_%d,RXLEV_%d,RXLEVSUB_%d,ARFCN_%d,RSS
I_%d",
                i + 1, i + 1, i + 1, i + 1, i + 1, i + 1, i + 1, i + 1, i + 1);
        strcat(header, cellHeader);
    }

    strcat(header, "\n");

    // Escreve o cabeçalho no arquivo
    API_FS_Write(fd, header, strlen(header));
    API_FS_Close(fd);

    Trace(1, "Novo arquivo de log criado com cabeçalho completo");
    return true;
}
```

Código fonte do acelerômetro - lis3dh.c

```
#include "lis3dh.h"
#include "api_hal_i2c.h"
#include "api_os.h"
#include "api_debug.h"




static I2C_ID_t g_i2c;
static uint8_t g_addr;
static LIS3DH_InterruptCallback_t g_callback = NULL;

static void LIS3DH_WriteReg(uint8_t reg, uint8_t val)
{
```

```
    uint8_t data = val;
    I2C_Error_t err = I2C_WriteMem(g_i2c, g_addr, reg, 1, &data, 1, I2C_DEFAULT_TIME_OUT);
    if(err != I2C_ERROR_NONE)
        Trace(10, "I2C write failed: reg=0x%02X, err=%d", reg, err);
}

static uint8_t LIS3DH_ReadReg(uint8_t reg)
{
    uint8_t val = 0;
    I2C_Error_t err = I2C_ReadMem(g_i2c, g_addr, reg, 1, &val, 1, I2C_DEFAULT_TIME_OUT);
    if(err != I2C_ERROR_NONE)
        Trace(10, "I2C read failed: reg=0x%02X, err=%d", reg, err);
    return val;
}




LIS3DH_Status_t LIS3DH_Init(I2C_ID_t i2c, uint8_t addr)
{
    g_i2c = i2c;
    g_addr = addr;
    Trace(10, "LIS3DH Init: addr=0x%02X", addr);

    // CTRL1: ODR=400Hz, all axis enabled, normal mode
    LIS3DH_WriteReg(LIS3DH_REG_CTRL1, 0x77);

    // CTRL4: ±16g, High resolution disabled
    LIS3DH_WriteReg(LIS3DH_REG_CTRL4, 0x30);

    // CTRL3: INT1 signal on INT1 pin
    LIS3DH_WriteReg(LIS3DH_REG_CTRL3, 0x40);

    LIS3DH_Clear_Interrupt();

    return LIS3DH_OK;
}

LIS3DH_Status_t LIS3DH_EnableInterrupt(LIS3DH_InterruptCallback_t callback)
{
    g_callback = callback;
```

```
    // Threshold ~3g (48 * 62.5mg = 3g aprox)
    LIS3DH_WriteReg(LIS3DH_REG_INT1_THS, 24);

    // Duration = 1 sample (~2.5ms at 400Hz)
    LIS3DH_WriteReg(LIS3DH_REG_INT1_DURATION, 2);

    // INT1_CFG: Enable interrupt on high event X, Y, Z
    LIS3DH_WriteReg(LIS3DH_REG_INT1_CFG, 0x2A ); // 0x3F = XH+YH+ZH+XL+YL+ZL

    LIS3DH_WriteReg(LIS3DH_REG_CTRL5, 0x08); // Latch interrupt

    // Clear the previous Interrupt
    LIS3DH_ReadReg(LIS3DH_REG_INT1_SRC);

    Trace(10, "LIS3DH interrupt enabled for gunshot detection 2");

    return LIS3DH_OK;
}

void LIS3DH_Clear_Interrupt ( void )
{
    // Clear the previous Interrupt
    uint8_t dummy = LIS3DH_ReadReg(LIS3DH_REG_INT1_SRC);

    Trace(1, "LIS3DH INT1_SRC = 0x%02X (latch cleared)", dummy);

    dummy = LIS3DH_ReadReg(LIS3DH_REG_INT1_SRC);

    Trace(1, "LIS3DH INT1_SRC = 0x%02X (latch cleared)", dummy);
}

LIS3DH_Status_t LIS3DH_EnterLowPower(void)
{
    // CTRL1: ODR=1Hz, low power enable, XYZ on
    LIS3DH_WriteReg(LIS3DH_REG_CTRL1, 0x17);

    Trace(10, "LIS3DH low power mode");
    return LIS3DH_OK;
}

LIS3DH_Status_t LIS3DH_WakeFromInterrupt(void)
{
```

```
    // Read source register to clear interrupt
    uint8_t src = LIS3DH_ReadReg(LIS3DH_REG_INT1_SRC);
    Trace(10, "LIS3DH interrupt source: 0x%02X", src);

    if(g_callback)
        g_callback();

    return LIS3DH_OK;
}

void LIS3DH_DumpRegisters(void)
{
    uint8_t regs[] = {0x20, 0x22, 0x23, 0x24, 0x30, 0x32, 0x33};
    for (int i = 0; i < sizeof(regs); i++)
    {
        uint8_t val = LIS3DH_ReadReg(regs[i]);
        Trace(10, "Reg 0x%02X = 0x%02X", regs[i], val);
    }
}


static int16_t readAxis(uint8_t laddr)
{
    uint8_t low = LIS3DH_ReadReg(laddr);
    uint8_t high = LIS3DH_ReadReg(laddr + 1);
    return (int16_t)((high << 8) | low); // two's complement
}

void Lis3dhDebugTask(void)
{
    while(1)
    {
        // espera semáforo liberado pela ISR
        //if (OS_WaitForSemaphore(semLis3dh, 5000) == OS_OK) {
            // Ler INT1_SRC (IA bit = 6)
            uint8_t src = LIS3DH_ReadReg(LIS3DH_REG_INT1_SRC);
            Trace(10, "INT1_SRC = 0x%02X", src);

            // Ler e mostrar X/Y/Z bruto
            int16_t x = readAxis(OUT_X_L);
            int16_t y = readAxis(OUT_Y_L);
            int16_t z = readAxis(OUT_Z_L);
```

```
            // Converte para g aproximado (com ±16g, 16-bit?) - escala depende do modo; aqui só log
bruto
            Trace(10, "Accel RAW X=%d Y=%d Z=%d", x, y, z);

            // Se usou latch (CTRL5=0x08), a leitura de INT1_SRC limpa a interrupção
        /*} else {
            // timeout: sem interrupção nos últimos 5s -> opcional polling read
            uint8_t src = LIS3DH_ReadReg(LIS3DH_REG_INT1_SRC);
            if (src & 0x40) {
                Trace(10, "POLLED INT1_SRC = 0x%02X", src);
            }
        }*/

        LIS3DH_Clear_Interrupt();

        OS_Sleep(100);
    }
}
```