Desafio Técnico - Analista de Firmware Pleno: Módulo de Telemetria

Contexto

Olá! Seja bem-vindo(a) ao nosso processo seletivo.

Nossa startup está desenvolvendo um novo módulo de telemetria (MTE-100) para monitorar plataformas elevatórias em tempo real. Este dispositivo precisa coletar dados vitais do barramento CAN da máquina, processá-los e enviá-los para nossa plataforma na nuvem via MQTT, utilizando uma conexão Ethernet cabeada.

Para este desafio, você desenvolverá o coração do firmware deste dispositivo. **Junto com esta descrição, você receberá um diagrama esquemático (esquematico_mte-100.pdf) do nosso hardware.**

Seu trabalho será dividido em duas partes principais:

1. Escrever o código de inicialização dos periféricos do microcontrolador com base no esquemático.
2. Desenvolver a lógica da aplicação que simula a coleta e o envio de dados.

Parte 1: Análise de Hardware e Configuração de Periféricos

Com base no esquemático fornecido, que utiliza um microcontrolador **STM32F767VIT6**, você deve criar um arquivo (bsp.c - Board Support Package) que contenha a função void BSP_Init(void). Esta função deve realizar a configuração inicial dos seguintes periféricos:

- **System Clock:** Configure o sistema para rodar com um clock estável e de alta performance, **utilizando o oscilador interno de alta velocidade (HSI) e o PLL**, conforme a ausência de um cristal externo (HSE) no esquemático.
- **Interface Ethernet (RMII):** Identifique no esquemático os pinos da interface RMII conectados ao PHY Ethernet **LAN8742A** e configure o periférico ETH (MAC e DMA) do microcontrolador.
- **GPIOs para o CAN:** Identifique no esquemático os pinos CAN_TX (PA12) e CAN_RX (PA11) e configure o periférico CAN1 e os pinos GPIO correspondentes em modo "Alternate Function".
- **Controle do Resistor de Terminação CAN:** O esquemático mostra um jumper (JP1) para habilitar o resistor de terminação. Configure um pino GPIO à sua escolha (ex: PG0) como saída (Output Push-Pull) para simular o controle programático deste recurso

(CAN_TERM_EN).
- **GPIO para LED de Status:** Identifique o pino conectado ao LED de status (PC13) e configure-o como saída (Output Push-Pull).

Importante: Dentro do seu código de inicialização, adicione comentários que justifiquem as configurações, fazendo referência direta ao esquemático. Ex: // Configura o pino PA12 como CAN1_TX (AF11) conforme esquemático MTE-100.

Note que o esquemático não provê uma interface UART para debug. A depuração deverá ser considerada via outros meios (ex: piscando o LED de status).

Parte 2: Lógica da Aplicação

Considerando que a inicialização da Parte 1 foi executada, desenvolva a aplicação principal. Como você não possui o hardware, a lógica da aplicação pode ser desenvolvida de forma mais abstrata, ou utilizando uma placa de desenvolvimento que você possua.

Requisitos Técnicos da Aplicação:

1. Leitura e Simulação do Barramento CAN:

- O firmware não lerá um barramento CAN físico. Em vez disso, ele deve **simular o recebimento de mensagens CAN** periodicamente (a cada 2 segundos).
- As mensagens CAN a serem simuladas são:
  - **ID CAN 0x100 (Status da Plataforma):**
    - Payload (8 bytes): [altura_msb, altura_lsb, peso_msb, peso_lsb, status_flags, 0x00, 0x00, 0x00]
    - altura: uint16_t em centímetros.
    - peso: uint16_t em quilogramas.
    - status_flags: uint8_t, um bitmask:
      - Bit 0: Motor ligado (1) / desligado (0)
      - Bit 1: Cesto nivelado (1) / não nivelado (0)
      - Bit 2: Sobrecarga (1) / normal (0)
  - **ID CAN 0x200 (Bateria e Horímetro):**
    - Payload (8 bytes): [tensao, 0x00, 0x00, 0x00, horimetro_b3, horimetro_b2, horimetro_b1, horimetro_b0]
    - tensao: uint8_t, valor * 0.5 Volts (Ex: valor 48 = 24V).
    - horimetro: uint32_t, total de horas de operação.

2. Processamento dos Dados e Feedback Visual:

- Sua aplicação deve decodificar os bytes para extrair os valores corretos.
- A cada mensagem CAN processada, o LED_STATUS (configurado na Parte 1) deve piscar uma vez.

3. Comunicação MQTT:

- A cada 5 minutos, o firmware deve agregar os últimos dados recebidos e publicar uma única mensagem JSON em um tópico MQTT.
- **Broker MQTT:** Use o broker público broker.hivemq.com na porta 1883.
- **Tópico:** telemetria/plataforma/CLIENT_ID/dados (substitua CLIENT_ID por um identificador único).
- **Payload (JSON):**

```
{
  "altura_m": 15.5,
  "peso_kg": 250,
  "tensao_v": 24.5,
  "horimetro_h": 1234,
  "motor_ligado": true,
  "cesto_nivelado": true,
  "sobrecarga": false
}
```

4. Uso de Sistema Operacional de Tempo Real (Desejável):

- A utilização de um RTOS (como o FreeRTOS) é um grande diferencial.

O que você deve nos entregar:

1. **Código-Fonte Completo:** Incluindo os arquivos bsp.c e bsp.h da Parte 1, e o restante da lógica da aplicação.
2. **Arquivo README.md:** Explique sua arquitetura, decisões de design, instruções de compilação/execução, bibliotecas e premissas.

Prazo:

Você terá **3 dias** para a conclusão e entrega do desafio.

Boa sorte! Estamos ansiosos para ver sua solução.