Relatório Técnico de Desenvolvimento e Validação

Sistema de Retransmissão de Telemetria LoRa para Radiosonda

Autor: Alisson

Data: 06 de Agosto de 2025

Índice

1.0 Resumo Executivo

Este relatório detalha o processo de desenvolvimento e validação do firmware para um sistema

de retransmissão de telemetria LoRa. O objetivo foi criar um sistema robusto utilizando duas placas **STM32 NUCLEO-WL55JC1**, capaz de receber um fluxo de dados de uma radiosonda, validar sua integridade, e retransmiti-lo via LoRa para uma estação base que exibe os dados em um computador. A arquitetura de software foi construída sobre as camadas de abstração de hardware (HAL) e de suporte à placa (BSP) da STMicroelectronics, utilizando DMA para recepção de dados eficiente. O sistema foi implementado com sucesso, validando a comunicação de ponta a ponta e demonstrando a viabilidade da arquitetura proposta.

2.0 Arquitetura do Sistema

O sistema foi projetado com uma arquitetura de dois nós para garantir modularidade e eficiência energética.

- **Placa 1 (Nó de Campo):** Atua como uma ponte inteligente. Utiliza uma porta **USART1** com DMA em modo circular para receber dados de uma radiosonda (19200 baud). Um parser com máquina de estados valida a integridade dos pacotes via `SYNC_WORD` e `checksum`. Pacotes válidos são então transmitidos via LoRa.
- **Placa 2 (Estação Base):** Atua como um gateway. O rádio LoRa fica em modo de recepção contínua. Ao receber um pacote, a Placa 2 decodifica a estrutura de dados e a envia de forma legível, via **USART2** (115200 baud), para um computador, utilizando a Porta COM Virtual (VCP) do ST-LINK integrado.

3.0 Ferramentas e Ambiente de Desenvolvimento

O desenvolvimento foi realizado utilizando um conjunto de ferramentas padrão da indústria para sistemas embarcados.

- **Hardware:** 2x Placas de desenvolvimento STMicroelectronics NUCLEO-WL55JC1. Esta placa foi escolhida por integrar o microcontrolador dual-core de ultra-baixo consumo STM32WL55JC e o transceptor LoRa, ideal para a aplicação.
- **IDE (Ambiente de Desenvolvimento Integrado):** **STM32CubeIDE**, que integra o configurador gráfico STM32CubeMX e um ambiente de desenvolvimento C/C++ baseado em Eclipse.
- **Configurador de Hardware (`.ioc`):** O **STM32CubeMX** foi utilizado para a configuração

gráfica de todos os periféricos (RCC, GPIOs, UARTs, DMA, SUBGHZ), garantindo uma inicialização de hardware rápida e livre de erros.

- **Terminal Serial:** O software **PuTTY** foi utilizado como terminal serial no computador para monitorar as saídas de debug da Placa 1 e para visualizar os dados de telemetria finais da Placa 2.

4.0 Arquitetura de Software e Uso de Drivers

O firmware foi construído sobre uma arquitetura de software em camadas para maximizar a portabilidade e a abstração do hardware, utilizando os pacotes de drivers fornecidos pela STMicroelectronics.

1. **Aplicação (`main.c`):** Contém a lógica de alto nível do sistema: a máquina de estados do parser, o controle de fluxo e as chamadas para as camadas inferiores.
2. **BSP (Board Support Package):** A camada BSP específica para a `NUCLEO-WL55JC1` foi utilizada para interações de alto nível com os componentes da placa. Exemplo: `BSP_LED_Toggle(LED_GREEN)` em vez de manipulação direta de registradores GPIO. Isso torna o código mais legível e portável.
3. **Driver de Rádio (`radio_driver.c`):** Uma camada de abstração de nível médio que fornece uma API funcional para o rádio LoRa (ex: `SUBGRF_SendPayload`, `SUBGRF_SetRx`). Este driver, por sua vez, utiliza o BSP para controlar o hardware específico da placa, como o switch da antena.
4. **HAL (Hardware Abstraction Layer):** A camada HAL da STMicroelectronics foi a base para a configuração de todos os periféricos (ex: `HAL_UART_Receive_DMA`). Ela fornece uma API consistente para interagir com o hardware do microcontrolador.
5. **Hardware (MCU STM32WL55JC):** A camada física do microcontrolador.

5.0 Implementação do Firmware

5.1 Firmware da Placa 1 (Nó de Campo / Transmissor)

O objetivo deste firmware é receber, validar e retransmitir os dados da radiosonda de forma autônoma.

Lógica Principal

A recepção de dados da radiosonda na `USART1` é gerenciada por DMA em modo circular, não bloqueando o processador. O loop `while(1)` verifica continuamente se novos dados chegaram ao buffer do DMA e alimenta cada novo byte a uma função de processamento (`ProcessByte`) que implementa a máquina de estados do parser.

Funções-Chave

A função `ProcessByte` implementa a lógica central de parsing, mudando de estado conforme recebe os bytes de sincronia, payload e checksum. Se o checksum for válido, a função `SendLoRaPacket` é chamada.

```
void ProcessByte(uint8_t receivedByte)
{
  switch (currentState)
  {
    case AWAITING_SYNC:
      if (receivedByte == SYNC_WORD) {
        currentState = RECEIVING_PAYLOAD;
        byteCounter = 0;
      }
      break;
    case RECEIVING_PAYLOAD:
      if (byteCounter < PAYLOAD_SIZE) {
        payloadBuffer[byteCounter++] = receivedByte;
      }
      if (byteCounter >= PAYLOAD_SIZE) {
        currentState = AWAITING_CHECKSUM;
      }
      break;
    case AWAITING_CHECKSUM:
      {
        uint8_t receivedChecksum = receivedByte;
        uint8_t calculatedChecksum = calculate_checksum(payloadBuffer,
PAYLOAD_SIZE);
        if (receivedChecksum == calculatedChecksum) {
          printf("Checksum OK. Pacote da radiosonda validado.\r\n");
          BSP_LED_Toggle(LED_GREEN);
          SendLoRaPacket((LoRaPayload_t*)payloadBuffer);
        }
        currentState = AWAITING_SYNC;
      }
      break;
  }
}
```

A função `SendLoRaPacket` é responsável por usar o driver de rádio para transmitir o payload validado. Ela utiliza uma flag `tx_done` para garantir que uma nova transmissão só ocorra após a anterior ter sido concluída.

```
void SendLoRaPacket(LoRaPayload_t* payload)
{
    if (tx_done == true) // Só transmite se a transmissão anterior já
terminou
    {
        tx_done = false;
        printf("Transmitindo pacote LoRa ID: %lu\r\n", payload-
>packet_id);
        SUBGRF_SendPayload((uint8_t*)payload, PAYLOAD_SIZE, 0);
    }
}
```

5.2 Firmware da Placa 2 (Estação Base / Receptor)

O objetivo deste firmware é receber os pacotes LoRa e exibi-los no PC de forma clara.

Lógica Principal

Este firmware é altamente eficiente, sendo totalmente orientado a eventos. Após a inicialização, o rádio é colocado em modo de recepção contínua (`SUBGRF_SetRx(0)`) e o processador entra em modo de baixo consumo (`HAL_PWR_EnterSLEEPMode`). Toda a lógica é acionada por interrupções do rádio, tornando o sistema reativo e com baixo consumo de energia.

Funções-Chave

A função `RadioOnDioIrq` é o callback principal para eventos do rádio. O caso `IRQ_RX_DONE` é acionado quando um pacote válido é recebido. Dentro dele, os dados são extraídos do rádio, o feedback visual é dado (LED), o status do sinal (RSSI/SNR) é impresso, e os dados são passados para a função de impressão. Crucialmente, o rádio é instruído a voltar a ouvir em seguida.

```
void RadioOnDioIrq(RadioIrqMasks_t radioIrq)
{
    switch (radioIrq)
    {
        case IRQ_RX_DONE:
            {
                uint8_t received_size = 0;
                PacketStatus_t packetStatus;
                BSP_LED_Toggle(LED_GREEN);

                SUBGRF_GetPayload(LoRa_rx_buffer, &received_size, 255);
                SUBGRF_GetPacketStatus(&packetStatus);
                printf("Pacote LoRa Recebido! RSSI: %d dBm, SNR:
%d\r\n", packetStatus.Params.LoRa.RssiPkt,
packetStatus.Params.LoRa.SnrPkt);

                ProcessAndPrintPayload(LoRa_rx_buffer, received_size);

                SUBGRF_SetRx(0);
            }
            break;
        // ... outros casos como CRC_ERROR e TIMEOUT ...
    }
}
```

A função `ProcessAndPrintPayload` decodifica a estrutura binária, converte os valores brutos para unidades legíveis (como graus decimais e metros) e usa `printf` para exibir a telemetria formatada no terminal do PC.

```
void ProcessAndPrintPayload(uint8_t* buffer, uint8_t size) {
  if (size != PAYLOAD_SIZE) { /* ... validação de tamanho ... */
return; }

  LoRaPayload_t* telemetry = (LoRaPayload_t*)buffer;

  float latitude = telemetry->latitude_raw / 10000000.0f;
  float longitude = telemetry->longitude_raw / 10000000.0f;
  // ... resto da decodificação ...

  printf("\r\n---[ PACOTE DE TELEMETRIA DECODIFICADO ]---\r\n");
  printf("  ID do Pacote:   %lu\r\n", telemetry->packet_id);
  printf("  Latitude:       %f\r\n", latitude);
  // ... resto dos printf's ...
}
```

6.0 Resultados e Validação Final

O teste de integração final, com as duas placas programadas e a Placa 1 conectada a uma fonte de dados seriais, foi executado com sucesso. As saídas de terminal observadas em cada placa validaram o funcionamento de ponta a ponta do sistema.

Saída Observada no Terminal da Placa 1 (Transmissor)

```
Checksum OK. Pacote da radiosonda validado.
Transmitindo pacote LoRa ID: 1
LoRa TX Done.
```

Saída Observada no Terminal da Placa 2 (Receptor)

```
Pacote LoRa Recebido! RSSI: -50 dBm, SNR: 9
---[ PACOTE DE TELEMETRIA DECODIFICADO ]---
  ID do Pacote:   1
  Latitude:       -3.740123
  Longitude:      -38.570456
  Altitude:       50.12 m
  Voltagem:       3300 mV
  Temp. Radio:    25 C
  Status GPS:     FIX OK (Satelites: 8)
-----------------------------------------
```

O feedback visual dos LEDs em ambas as placas também correspondeu ao comportamento esperado, piscando a cada transmissão e recepção bem-sucedida.

7.0 Conclusão e Próximos Passos

O desenvolvimento atingiu com sucesso todos os objetivos propostos. Foi criado um sistema funcional e robusto capaz de atuar como um repetidor de telemetria, utilizando técnicas eficientes como DMA e uma arquitetura de software em camadas. A metodologia de configuração via CubeMX e o uso dos drivers HAL e BSP provaram ser eficazes para um desenvolvimento rápido e confiável. O resultado é uma base de código sólida para futuras expansões.

Próximos passos recomendados:

- Implementação final dos modos de baixo consumo de energia (`SUBGRF_SetSleep`) na Placa 1 para otimizar a vida útil da bateria em campo.
- Desenvolvimento de uma interface gráfica ou script de logging na estação base (PC) para visualização e armazenamento dos dados recebidos.
- Realização de testes de alcance em campo aberto para validar o desempenho do link LoRa.