UNIVERSIDADE FEDERAL DE MINAS GERAIS
Relatório produzido por membro
Fórmula Tesla UFMG

Autor/Autores: Lucas Oliveira Rodrigues

Telemetria NUCLEO

Sistema de transmissão de dados baseado em NUCLEO WL55JC1 LoRa

Belo Horizonte, Minas Gerais
2025

Histórico de Modificações

- 07/07/2025: Primeira versão do relatório [Modificado por Lucas O.]

1 Introdução e Contextualização

A transmissão remota de dados operacionais do veículo é um requisito fundamental para o monitoramento em tempo real e a análise quantitativa de desempenho, além de permitir a detecção imediata de falhas e eventos críticos. O sistema de telemetria anteriormente empregado utilizava módulos XBEE, baseados em protocolo proprietário, que apresentavam limitações em alcance efetivo e elevada suscetibilidade a interferências eletromagnéticas.

Com o intuito de aprimorar a robustez, escalabilidade e confiabilidade da comunicação sem fio, foi desenvolvido o projeto da nova placa Telemetria 2.0, baseada no microcontrolador STM32WL55JC1, que incorpora um transceptor sub-GHz da família Semtech SX126x, compatível com modulação LoRa e outras variantes FSK/GFSK.

Devido a restrições de cronograma e disponibilidade de recursos durante a temporada 2025, optou-se pela utilização das placas de desenvolvimento NUCLEO-WL55JC1 como solução provisória para testes e validação da arquitetura proposta, mantendo-se a compatibilidade com os objetivos de hardware e firmware previstos no projeto da Telemetria 2.0. Este relatório descreve a implementação da Telemetria nessas placas de desenvolvimento, projeto que foi denominado Telemetria NUCLEO.

2 Requisitos

1. Desenvolver um sistema de telemetria com alta confiabilidade e alcance estendido, adequado para comunicação robusta em ambientes ruidosos e de grande área.

2. Utilizar os kits de desenvolvimento STM32WL55JC1 Nucleo, fornecidos por meio de parceria com a STMicroelectronics, como plataforma base para prototipagem e testes do sistema.

3. Estabelecer uma interface de comunicação padronizada entre a unidade de telemetria e os módulos de aquisição de dados, por meio do protocolo I2C, visando compatibilidade e modularidade no sistema embarcado.

4. Projetar uma solução de telemetria de rápida integração e baixo overhead de implementação, compatível com as restrições de tempo e disponibilidade de recursos da temporada competitiva atual.

3 Hardware

O sistema de telemetria foi desenvolvido com base em duas placas de desenvolvimento NUCLEO-WL55JC1, ambas equipadas com o microcontrolador STM32WL55JC1, que integra recursos de comunicação sem fio do tipo LoRa. Uma das placas atua como transmissora e está posicionada na parte frontal do veículo, próxima ao conjunto de sensores e à placa Datalogger, enquanto a outra, receptora, permanece fora do carro, conectada a um computador via cabo USB para recepção e registro dos dados transmitidos.

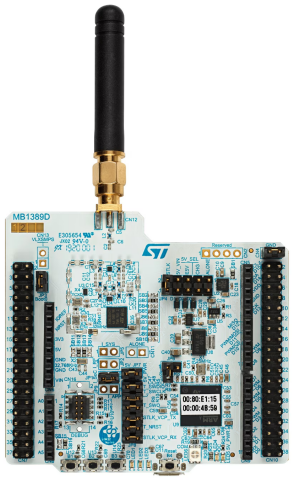

Figura 1: NUCLEO WL55JC1, modelo utilizado na implementação da Telemetria NUCLEO.

A placa transmissora é responsável por receber dados diretamente da Datalogger por meio do barramento I2C, utilizando pinos padrão da interface do STM32. O protocolo I2C foi escolhido por sua simplicidade, por exigir apenas dois fios e por permitir integração modular, o que foi decisivo diante das restrições de tempo e recursos no momento da implementação. O papel da transmissora é essencialmente o de um gateway entre o sistema de aquisição embarcado e o módulo de comunicação LoRa.

Para a transmissão dos dados à unidade receptora, foi empregada a comunicação sem fio utilizando o protocolo LoRa, que oferece boa robustez e alcance para aplicações em campo aberto. A placa receptora, conectada ao computador via USB, atua como interface entre o sistema embarcado e o software de monitoramento, encaminhando os dados recebidos para exibição ou armazenamento. A comunicação entre a placa receptora e o computador é feita por meio da interface USB virtual COM (CDC), o que permite fácil integração com terminais seriais ou scripts personalizados - característica que será explorada na integração com outros sistemas, como a reconstrução de pista.

O posicionamento físico das placas e das antenas foi considerado, já que é um fator importante para o bom desempenho da comunicação. É recomendável que a área ao redor das antenas seja mantida livre de obstáculos e superfícies metálicas, uma vez que isso influencia diretamente na qualidade do sinal LoRa. Além disso, a elevação das placas em relação ao solo também contribui para a melhoria do alcance do sinal, fator especialmente relevante durante os testes de campo e uso em ambientes externos.

4 Software

Podemos dividir o software do projeto em duas partes principais - código e ajuste de parâmetros. Primeiramente mostra-se como configurar o arquivo .ioc do projeto. Em seguida temos a descrição dos principais trechos do código utilizado (que pode ser acessado na íntegra aqui [incluir link github]).

4.1 Configuração de Parâmetros

A seguir são apresentados screenshots de cada aba de configuração modificada durante o projeto.

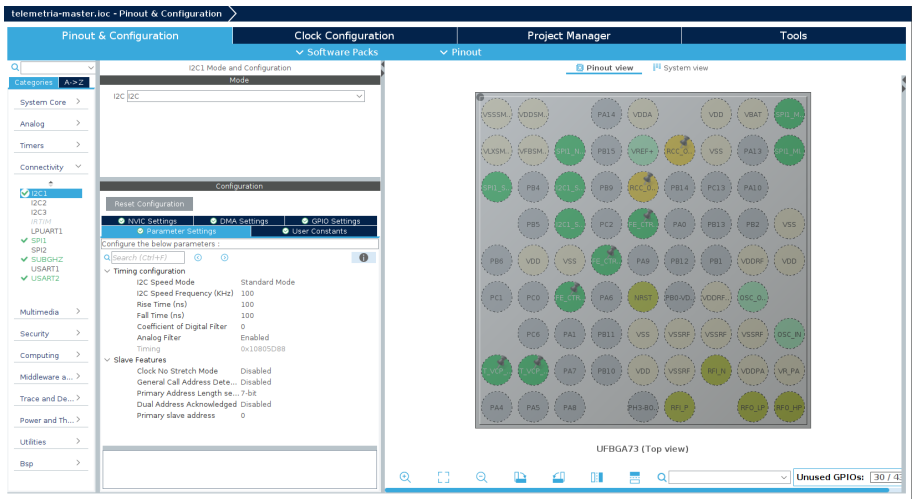

Figura 2: Painel de configuração de parâmetros do I2C.

Primeiramente configuramos o I2C. Basta acessar Connectivity -> I2C1 e, na lista dropdown em frente a "I2C", escolher a opção I2C. Aqui todos os parâmetros foram mantidos no padrão do NUCLEO WL55JC1, porém, caso houver alguma modificação na implementação, é importante verificar a compatibilidade dos parâmetros de transmissão entre os dispositivos envolvidos - especialmente a frequência, que aqui é mantida em 100kHz.

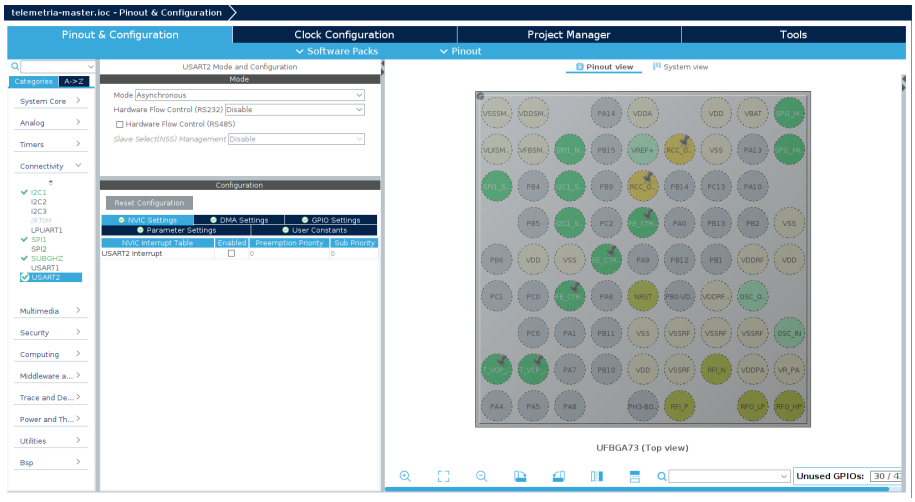

Figura 3: Painel de configuração de parâmetros do UART.

O próximo passo é configurar o UART. Mais uma vez é bem simples. Ainda na aba Connectivity, em USART2 escolha Asychronous. Pronto.

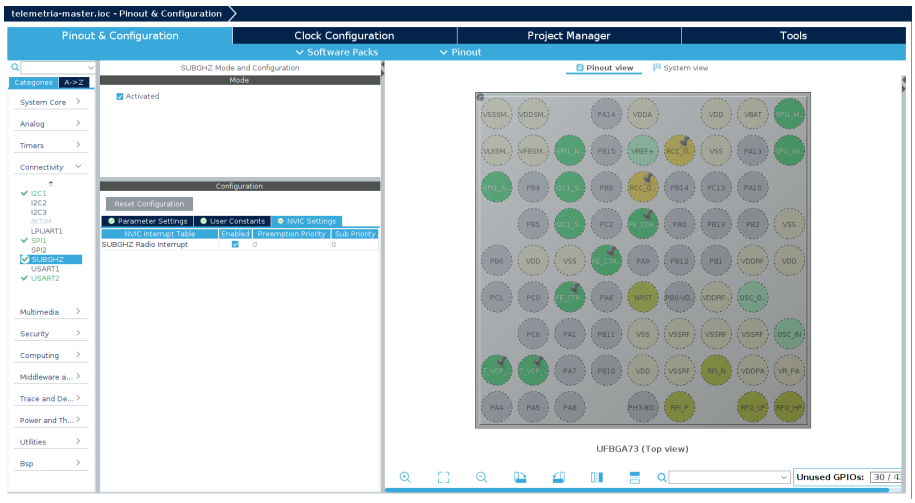

Figura 5: Painel de configuração de parâmetros do SUBGHZ - NVIC Settings

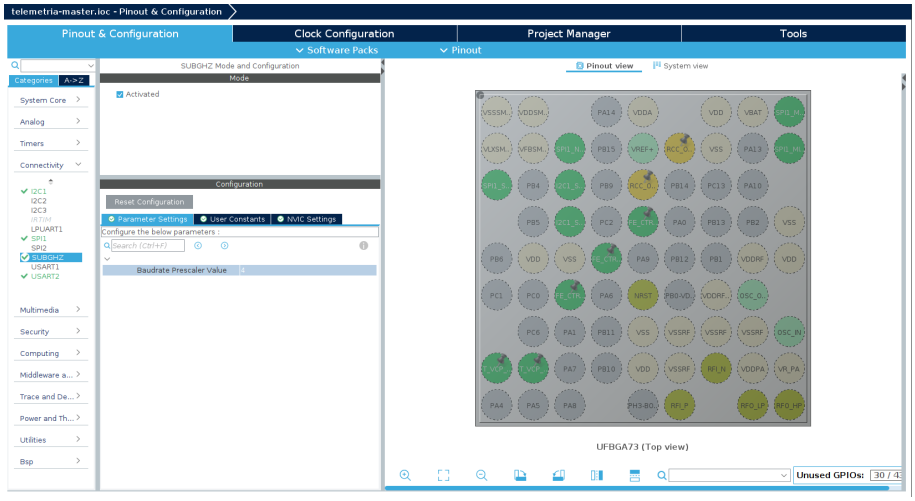

Figura 4: Painel de configuração de parâmetros do SUBGHZ - Parameter Settings.

Mais uma vez em Connectivity, escolha SUBGHZ, marque a opção Activated. Depois confira se a opção Baudrate Prescaler Value está configurada com o valor 4. Na aba NVIC Settings abilite a opção SUBGHZ Radio Interrupt.

4.2 Código

O software está dividido em módulos, cada um responsável por uma parte essencial do sistema.

- **Inicialização:** prepara o HAL, configura o clock e habilita os periféricos.
- **I2C Slave:** só na placa transmissora, recebe dados da Datalogger.
- **LoRa Driver:** controla a configuração, o envio e a recepção pelo rádio Sub-GHz.
- **UART/CLI:** formata strings e envia logs para debug (printf do serial).

ATENÇÃO: Os sketches reproduzidos abaixo são apenas ilustrativos. Trata-se de resumos e simplificações do código real utilizado na telemetria, simplificado para facilitar a compreensão e transmitir as ideias chave com mais clareza.

4.2.1 Macros LoRa

Sketch: Defines LoRa

```
// LoRa     parametros principais
#define RF_FREQUENCY           868000000
#define TX_OUTPUT_POWER        14
#define LORA_BANDWIDTH         0
#define LORA_SPREADING_FACTOR  7
#define LORA_CODINGRATE        1
#define LORA_PREAMBLE_LENGTH   8
#define LORA_SYMBOL_TIMEOUT    5
```

- RF_FREQUENCY: frequência de operação do rádio (868 MHz, ISM).
- TX_OUTPUT_POWER: potência de transmissão em dBm.
- LORA_BANDWIDTH: largura de banda selecionada.
- LORA_SPREADING_FACTOR: fator de espalhamento, que equilibra robustez e taxa.
- LORA_CODINGRATE: taxa de correção de erros (4/5).
- LORA_PREAMBLE_LENGTH: número de símbolos de preâmbulo.
- LORA_SYMBOL_TIMEOUT: tempo de espera antes de timeout na recepção.

4.2.2 Macros I2C (Transmissor)

Sketch: Defines I2C

```
// I2C     slave
#define I2C_SLAVE_ADDR     (0x3C << 1)   /* 0x78 em 8 bit   */
#define RX_TIMEOUT_MS      1000
#define POLL_DELAY_MS      100
```

- **I2C_SLAVE_ADDR**: endereço 7-bit deslocado para 8-bit.
- **RX_TIMEOUT_MS**: tempo máximo de espera por dados I²C.
- **POLL_DELAY_MS**: atraso entre transmissões LoRa.

4.2.3 Inicialização Geral

No início da `main()`, chamamos:

Sketch: Configuração Inicial

```
// Setup HAL e clock
HAL_Init();
SystemClock_Config();
// Habilita e configura perifericos
MX_GPIO_Init();
MX_SUBGHZ_Init();
MX_USART2_UART_Init();
MX_SPI1_Init();
MX_I2C1_Init();
```

As funções `MX_Init()` são geradas pelo STM32CubeMX, garantindo alinhamento com o diagrama de blocos do projeto.

Por fim, inicializamos os LEDs com `BSP_LED_Init()` e enviamos uma mensagem via UART para indicar se aquela placa é *Master* (transmissora) ou *Slave* (receptora).

4.2.4 Recepção I2C

Na placa transmissora, usamos a função abaixo para receber dados da Datalogger voa I2C:

Sketch: Recepção I2C

```
void _i2c1_receivemsg(uint8_t *buf, uint8_t size){
  HAL_StatusTypeDef st = HAL_I2C_Slave_Receive(
    &hi2c1, buf, size, RX_TIMEOUT_MS);
  if (st == HAL_OK) {
    // buf preenchido com dados da Datalogger
  } else {
    // limpa NACK e reinicia I2C em caso de erro
  }
}
```

Ela bloqueia apenas pelo tempo de espera configurado, trata `I2C_FLAG_AF` (NACK) e reinicializa o periférico se for necessário. Assim, o `txBuf` está sempre atualizado para a próxima transmissão LoRa.

Para que o I2C funcione é importante configurar o endereço correto no arquivo i2c.c. No caso do NUCLEO WL, essa configuração não é feita automáticamente, assim, é necessário acessar o arquivo e escrever o endereço correto (com o deslocamento de 1 bit à esquerda) no campo OwnAddress1 da struct hi2c1.Init. O trecho de inicialização dessa struct deve se parecer com o sketch abaixo.

Sketch: I2C Init (no arquivo i2c.c)

```
hi2c1.Instance              = I2C1;
hi2c1.Init.Timing           = 0x10805D88;
hi2c1.Init.OwnAddress1      = 0x3C << 1; // 0x3C (7-bit) -> 0x78
   (8-bit)
hi2c1.Init.AddressingMode   = I2C_ADDRESSINGMODE_7BIT;
hi2c1.Init.DualAddressMode  = I2C_DUALADDRESS_DISABLE;
hi2c1.Init.OwnAddress2      = 0;
hi2c1.Init.OwnAddress2Masks= I2C_OA2_NOMASK;
hi2c1.Init.GeneralCallMode  = I2C_GENERALCALL_DISABLE;
hi2c1.Init.NoStretchMode    = I2C_NOSTRETCH_DISABLE;
```

4.2.5 Driver LoRa

A função `radioInit()` exemplifica como as estruturas da biblioteca Sub-GHz são preenchidas:

Sketch: Inicialização LoRa

```
SUBGRF_Init(RadioOnDioIrq);
SUBGRF_SetRfFrequency(RF_FREQUENCY);
SUBGRF_SetRfTxPower(TX_OUTPUT_POWER);
ModulationParams_t mod = { /* BW, SF, CR */ };
SUBGRF_SetModulationParams(&mod);
PacketParams_t pkt = { /* CRC, pre mbulo */ };
SUBGRF_SetPacketParams(&pkt);
```

Mais adiante, nesta seção, detalharemos os valores de `RF_FREQUENCY`, `LORA_SPREADING_FACTOR`, `LORA_CODINGRATE` e demais macros para garantir operação estável em campo.

4.2.6 Loop Principal

Basicamente o loop principal envolve chamar a função de leitura I2C, depois enviar o pacote de dados por rádio (SUBGHZ) e testar condições de erro/status de transmissão:

Sketch: Loop de Transmissão

```
while(1){
    _i2c1_receivemsg(txBuf,8);
    txDone=txTimeout=false;
    SUBGRF_SetSwitch(RFO_LP,RFSWITCH_TX);
    SUBGRF_SendPayload(txBuf,sizeof txBuf,0);
    while(!txDone&&!txTimeout);
    status=txTimeout?"TIMEOUT":(txDone?"OK":"DESCONHECIDO");
    int len=sprintf(uartBuff,"\r\nStatus: %s\r\nPayload (%d
       bytes): ",
    status,(int)sizeof txBuf);
    HAL_UART_Transmit(&huart2,(uint8_t*)uartBuff,len,
       HAL_MAX_DELAY);
    if(txDone)BSP_LED_Toggle(LED_GREEN);
    HAL_Delay(POLL_DELAY_MS);
}
```

Assim, após chamar `SUBGRF_SendPayload()`, o código espera pelas flags atualizadas em `RadioOnDioIrq()`, formata o relatório usando `sprintf()` e transmite pelo UART, além de alternar o LED para sinalizar o sucesso.

5 Conclusão

Testes com o sistema implementado confirmaram comunicação estável até 700m de distância em área aberta, com taxa de entrega acima de 60%. nessa faixa. Esta é uma distância razoável e adequada para a realização das provas da competição. Nos próximos passos, o sistema será integrado à reconstrução de pista — para rastrear o carro durante a prova e correlacionar dados do veículo com trechos específicos do traçado.