Sistema de Telemetria para aplicações Aeroespaciais

Laboratório de Processadores

Arthur Salvador de Almeida - NºUSP: 12675079
Carlos Henrique Silva Thiersch - NºUSP: 11831679

São Paulo, 16 de agosto de 2024

Conteúdo

1 Introdução

1.1 Contexto e Motivação

Sistemas de telemetria são aplicações fundamentais para a Engenharia Aeroespacial, possibilitando a monitoração, o controle e a análise de dados em solo de aviação em tempo real. O funcionamento adequado da telemetria torna-se crucial para o desempenho de veículos aeroespaciais da era moderna, garantindo a performance e a segurança da missão.

A crescente complexidade da Engenharia Aeroespacial moderna e a presença de tripulantes exige a criação de sistemas de computação confiáveis e de alta performance. A utilização de sistemas embarcados para telemetria é corriqueira e utiliza dos conhecimentos adquiridos na disciplina de Laboratório de Processadores.

1.2 Objetivos

O presente trabalho desenvolve um sistema de telemetria simplificado, somente com transmissão para análise em solo, transmitindo alguns dados sensíveis à aviação.

2 Modelo do Problema

Definiu-se 4 variáveis principais para transmissão primeiramente:

- **Pressão** e **Temperatura**: Essenciais para análise de altitude e averiguar a segurança da aplicação
- **Aceleração Angular e Linear**: Provê informações posteriormente utilizadas para projetar a trajetória do corpo. Vital para análise de perfomance e de aerodinâmica.

É necessário transmitir toda esta informação continuamente e de forma confiável para o solo.

3 Modelo da Solução

A solução de telemetria necessita de componentes para a **computação**, **medição** e **transmissão**.

3.1 Computação

Para realizar a orquestração dos processos e cálculo com os dados, utilizou-se de um microcontrolador STM-32, com arquitetura Arm Cortex-M. A linguagem C foi utilizada para programação.

A comunicação com os componentes utiliza dos protocolos I2C e UART. Para evitar a sua implementação baremetal, a dupla utilizou a biblioteca **libopencm3**, uma biblioteca de firmware open-source para microcontroladores Arm Cortex-M.

3.2 Medição

Para a medição, utiliza-se dois componentes.

3.2.1 BMP280

O primeiro destes, o BMP280, é responsável pela medição de temperatura e pressão. O componente está realizando a medição constantemente (normal mode), em uma frequência de 125Hz.

A inicialização do componente ocorre da seguinte forma:

```
bool BMP_init(){
    BMP_i2c_setup(); //Funcao que configura GPIOs e I2C

    bool check_id = BMP_checkID(); //Verifica o ID do chip
    calibration = BMP_read_calibration(); //Le parametros de calibracao

    uint8_t config_data[2] = {BMP_CONFIG_REGISTER, ((0b000 << 5) | (0b000 << 2) | 0b00)}; //Bits 7 a 5 configuram standby time para 0,5ms(para consumo de energia) e bits 4 a 2 definem a ausencia de filtros.
    i2c_transfer7(I2C1, BMP_DEVICE_ADDR, config_data, 2, NULL, 0);

    uint8_t control_data[2] = {BMP_CONTROL_REGISTER, ((0b001 << 5) | (0b001 << 2) | 0b11)}; //Define, em ordem, oversampling de temperatura e pressao e o modo normal de execucao (0b11)
    uint8_t config;
    i2c_transfer7(I2C1, BMP_DEVICE_ADDR, control_data, 2, &config, 1);
    bool check_config = config == ((0b001 << 5) | (0b001 << 2) | 0b11);

    return check_id && check_config;
}
```

A função **BMP_read_calibration()** retorna um struct com valores de calibração posteriormente utilizados para conversão dos valores de medição em unidades de medida convencionais. O código main executa as funções **float BMP_get_temperature()** e **float BMP_get_pressure()** retornando valores em graus celsius e Pa, respectivamente. Estas funções lêem em um conjunto de registradores os valores medidos.

3.2.2 MPU6050

O MPU6050 possui uma frequência de medição de 1kHz. Em sua inicialização, o componente inicia no sleep mode, sendo necessário realizar a saída deste modo.

```
bool MPU_init(){
    MPU_i2c_setup(); // Configura GPIOs e I2C

    //Sai do modo sleep
    uint8_t pwr_mgmt_1[2] = {MPU_PWR_MGMT_1_REGISTER, 0b00001000};
    i2c_transfer7(I2C3, MPU_DEVICE_ADDR, &pwr_mgmt_1, 2, NULL, 0);

    system_delay(100);

    //Seleciona clock de giroscopio
    pwr_mgmt_1[1] = 0b00001001;
    i2c_transfer7(I2C3, MPU_DEVICE_ADDR, &pwr_mgmt_1, 2, NULL, 0);

    uint8_t config[2] = {MPU_CONFIG_REGISTER, 0b00000010};
    i2c_transfer7(I2C3, MPU_DEVICE_ADDR, config, 2, NULL, 0);

    bool check_addr = MPU_check_addr();
    return check_addr;
}
}
```

O código main executa a função **mpu_data MPU_get_data()** para leitura dos dados de aceleração linear e angular armazenados em registradores específicos.

3.3 Transmissão

A transmissão utiliza do componente E32 para transmissão, com um baud rate de 9600. A taxa é mantida baixa para aumentar o alcance da onda.

Pela baixa de transmissão quando comparada a taxa de medição dos outros componentes, utilizou-se do recurso de *fast_interruption* para que assim que uma transmissão se encerre uma nova se inicie de imediato, interrompendo qualquer outro processo acontecendo.

A seguinte rotina de tratamento de interrupção portanto é utilizada:

```
//Interrupcao do pino AUX do transmissor
void exti15_10_isr(){
    if(exti_get_flag_status(EXTI11)){
        exti_reset_request(EXTI11);

        E32_uart_write(u_temp.b_temp, 4);
        E32_uart_write(u_pres.b_pres, 4);
        E32_uart_write(u_a_x.b_a_x, 4);
        E32_uart_write(u_a_y.b_a_y, 4);
        E32_uart_write(u_a_z.b_a_z, 4);
        E32_uart_write(u_w_x.b_w_x, 4);
        E32_uart_write(u_w_y.b_w_y, 4);
        E32_uart_write(u_w_z.b_w_z, 4);
        gpio_toggle(LED_PORT, LED_PIN);
    }
}
```

A função constantemente executada na main, a qual a rotina de interrupção retorna é o seguinte loop:

```
        while (1){
            u_temp.temp = BMP_get_temperature();
            u_pres.pres = BMP_get_pressure();
            imu_data = MPU_get_data();
            u_a_x.a_x = imu_data.a_x;
            u_a_y.a_y = imu_data.a_y;
            u_a_z.a_z = imu_data.a_z;
            u_w_x.w_x = imu_data.w_x;
            u_w_y.w_y = imu_data.w_y;
            u_w_z.w_z = imu_data.w_z;
            system_delay(10);
    }
```

A função main e a rotina de tratamento encontram-se no arquivo **app\src\firmware.c** presente no github.

4 Protótipo

Para a prototipagem foi necessário conectar os componentes para teste. A dupla utilizou-se do trabalho do integrante Arthur Salvador no grupo de extensão Júpiter que confeccionou previamente uma placa integrando os com-

ponentes. Para testagem, uma antena foi utilizada para verificar se os valores enviados eram coerentes. Testes extensivos foram utilizados durante as semanas de maneira incremental que atestaram o funcionamento da solução.

Figura 1: Placa de telemetria desenvolvida.

5 Considerações Finais

O grupo pode aprender imensamente do trabalho e por em prática conceitos aplicados em um projeto aplicável em um cenário real.

A dupla agradece aos professores pelos ensinamentos e pelo oferecimento da matéria