Space Systems Lab Firmware and Mission Requirements Document

Hersch Nathan

November 2024

0.1 General Requirements

This software is to support the KRUPS missions. The current mission is KRUPS Aboard Norwegian GhostSat (KANGS). The hardware paradim is based around an ESP32-S3.

This work is porting Matt Ruffner's firmware for Adafruit Feather M0/M4 to the ESP32-S3. This including switching from the Arduino IDE to ESP-IDF.

The goal of this project is to make the firmware more modular and configurable for each mission and hardware.

The first chapter of this document outlines the over all architechure design and background information for this project. The second chapter rt details each libraries, drives, wrappers, and other software components that we wrote for this project. It provides design decisions and the reasoning behind them aswell as how to interface with them. The third and on details the software requirements for each hardware component for each mission.

Contents

Chapter 1

Architechure Design

1.1 Background Information

1.1.1 freeRTOS

freeRTOS is a real time operating system. A real time operating system works by having tasks that run at different priorities. The tasks are scheduled by the operating system and those tasks occur through a determanistic prosess. The tasks can communicate with each other using message queues. The tasks can also synchronize with each other using semaphores.

1.1.2 ESP-IDF

ESP-IDF is the Espressif IoT Development Framework. It is the official development framework for the ESP32 and ESP32-S3. It is a set of libraries and tools that are used to develop software for the ESP32 and ESP32-S3.

1.2 Overall Architecture Design

The firmware's framework is designed to be modular and configerable thmission and each hardware. The backbone of the firmware is using freeRTOS tasks to handle the different modules. Each taks is its own module that encapsulates the relevant hardware. The tasks communicate with each other using a publisher/subscriber model.

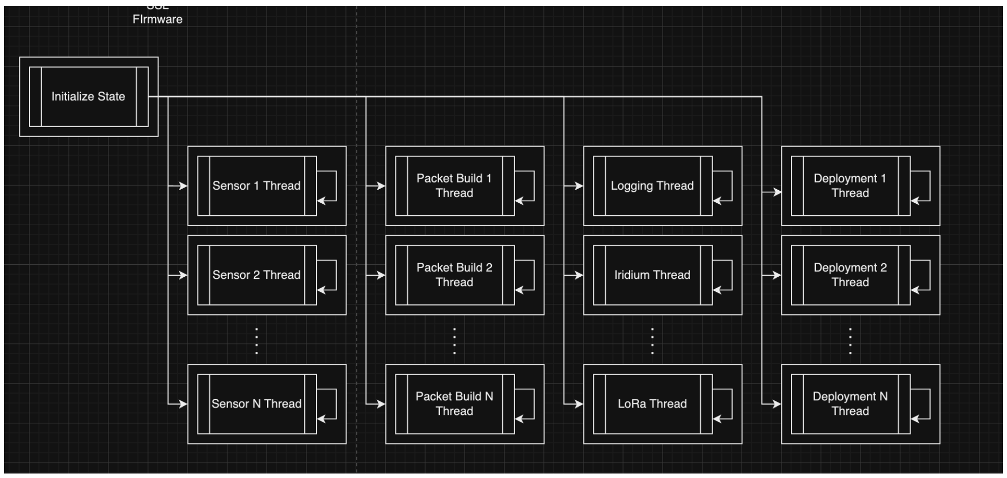

Figure 1.1: Firmware Architecture

In the figure 1.1, the main tasks there are four types of tasks.

1.2.1 Task

Sensor Tasks

Each sensor task manages one sensor or group of sensors, see 1.2. At startup it configures the sensors with the necessary parameters, then it calls a looping task. That task, takes the semaphore for the used serial communication interface (I2C, SPI, UART, etc). It sends the command to the sensor to get the data, reads the response, and then releases the semaphore. The task then sends the data to the packet build tasks queues.

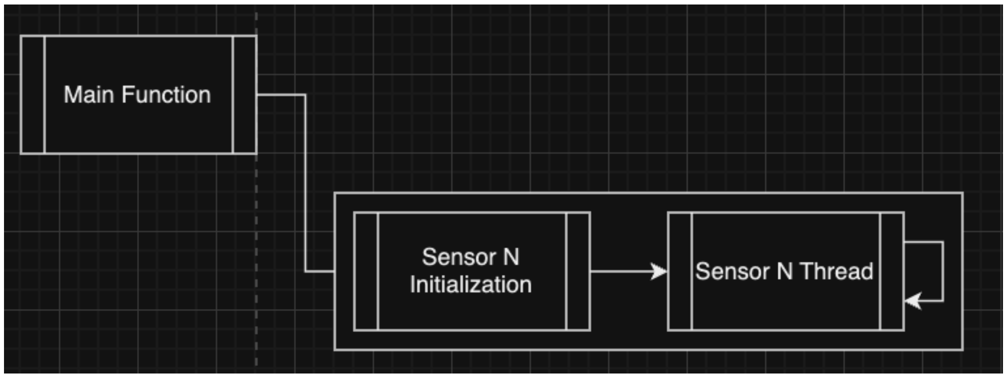

Figure 1.2: Sensor Task Architecture

Packet Build Tasks

Wireless Communication Tasks

Deployment Tasks

1.2.2 Middleware

Chapter 2

Software Components

2.1 Wrappers

2.1.1 I2C Wrapper

the I2C wrapper is a thread safe wrapper of the I2C driver from ESP-IDF. It mostly uses semaphores to allow and disallow access to th i2c bus. This is not the optimal implementation. If needing higher effiency a single thread should manage the each i2c bus.

Note the timeout ticks is hard defined to 1000 in the I2C wrapper header file.

this wrapper is based on the i2cdev library from UncleRus.

- **i2c_dev_init**: Initializes the I2C device, setting up the necessary configurations.
- **i2c_dev_done**: Deinitializes the I2C device, releasing resources.
- **i2c_dev_create_mutex**: Creates a mutex for the I2C device to ensure thread-safe operations.
- **i2c_dev_delete_mutex**: Deletes the mutex associated with the I2C device.
- **i2c_dev_take_mutex**: Takes the mutex for the I2C device, blocking other threads from accessing it.
- **i2c_dev_give_mutex**: Releases the mutex for the I2C device, allowing other threads to access it.
- **i2c_dev_probe**: Checks if the I2C device is available and ready for communication.
- **i2c_dev_read**: Reads data from the I2C device.
- **i2c_dev_write**: Writes data to the I2C device.
- **i2c_dev_read_reg**: Reads data from a specific register of the I2C device.
- **i2c_dev_write_reg**: Writes data to a specific register of the I2C device.

Chapter 3

KRUPS Aboard Norwegian GhostSat (KANG)

3.1 Kentucky Flight Computer (KFC) Requirements

3.2 FemptoSats Requirements

The FemptoSats submission is to test the viability of using Wifi/LoRa for intercapsuole communication.

3.2.1 Sensors Requirements

The FemptoSats will have the following sensors:

- BME280 Temperature/pressure/humidity sensor
- BNO086 9 - Axis IMU

BME280 Temperature/pressure/humidity sensor

The BME280 will be run with the following configuration settings:

BNO086 9 - Axis IMU

The BNO086 will be run with the following configuration settings:

3.2.2 Wireless Communcation Requirements

the FemptoSat will use the following Wireless Communcation modules:

- Integrated Wifi
- LoRa

The Wifi will fail over to the LoRa when the Wifi gets out of range

3.3 Rocketstation Transmitter (RST) Requirements

uart to raspi to shut off need

3.4 Rocketstation Requirements

uart to raspi to shut off need

3.5 Groundstation Requirements