**Technical Document: LoRa Spacecraft Telemetry Transmitter**

Version: 1.0

Date: 2025-04-11

Target Hardware: LilyGO T-Beam S3 Supreme (with SX1262 LoRa Module)

**1. Introduction & Overview**

This document describes a firmware system designed to function as a telemetry transmitter, simulating aspects of spacecraft data transmission. The system runs on specific microcontroller hardware (LilyGO T-Beam S3 Supreme), collects data from a suite of onboard sensors, formats this data according to a structure inspired by the Consultative Committee for Space Data Systems (CCSDS) standards, enhances data reliability using Cyclic Redundancy Check (CRC) and Reed-Solomon Forward Error Correction (RS-FEC), and transmits the resulting data packets wirelessly using LoRa (Long Range) radio technology.

The system is composed of three main code components:

1. **Application Logic (**TelemetryTx.ino**):** The main Arduino sketch handling sensor reading, data packaging, error correction encoding, initiating transmissions, and managing the display.
2. **Hardware Abstraction Layer - Definitions (**LoRaBoards.h**):** A header file defining the specific pin mappings, hardware features, and peripheral configurations for the target board (T-Beam S3 Supreme).
3. **Hardware Abstraction Layer - Implementation (**boards.cpp**):** A C++ file providing the functions to initialize and manage the hardware based on the definitions in LoRaBoards.h. This includes complex power management, peripheral detection, and setup routines.

The goal is to provide a functional example of embedded system design incorporating sensor integration, standard data formatting concepts, error control coding, and wireless communication, drawing parallels to real-world systems like the Voyager probes.

**2. System Architecture**

The firmware follows a layered architecture:

+---------------------------------+

| Application Layer |

| (`TelemetryTx.ino`) |

| - Sensor Reading |

| - Data Processing (CCSDS-like)|

| - Error Control Encoding |

| - LoRa Transmission Control |

| - Display Management |

+---------------------------------+

| (Uses Objects/Functions)

V

+---------------------------------+

| Board Support Package (BSP) |

| / HAL Implementation |

| (`boards.cpp`) |

| - `setupBoards()` |

| - `beginPower()` (PMU Init) |

| - `beginDisplay()` |

| - `beginSDCard()` |

| - `beginGPS()` |

| - I2C/SPI/Serial Init |

| - Utility Functions |

+---------------------------------+

| (Uses Definitions)

V

+---------------------------------+

| Hardware Definitions |

| (`LoRaBoards.h`) |

| - Pin Mappings (GPIO #s) |

| - Feature Flags (`HAS\_...`) |

| - Peripheral Configs |

| - Board Identification |

+---------------------------------+

| (Maps To)

V

+---------------------------------+

| Physical Hardware |

| (LilyGO T-Beam S3 Supreme) |

| - ESP32-S3 Microcontroller |

| - SX1262 LoRa Radio |

| - AXP2101 PMU |

| - GPS Module (L76K/UBlox) |

| - IMU (QMI8658) |

| - Mag (QMC6310 - Implied) |

| - Env (BME280 - Implied) |

| - Display (SH1106) |

| - SD Card Slot |

| - Button, LED |

+---------------------------------+

**3. Hardware Configuration (**LoRaBoards.h **- Analysis of Code 2)**

This header file acts as the crucial link between the abstract code and the physical T-Beam S3 Supreme board. It uses C preprocessor #define directives to establish constants and flags used throughout the codebase.

* **Board Identification:**
  + T\_BEAM\_S3\_SUPREME\_SX1262, T\_BEAM\_S3\_SUPREME: Flags indicating the specific board model.
  + USING\_SX1262: Flag confirming the SX1262 LoRa chip is present, enabling specific RadioLib configurations in Code 1.
  + BOARD\_VARIANT\_NAME "T-Beam S3": Human-readable name.
* **Pin Definitions (GPIO Mappings):** Defines which ESP32-S3 GPIO pins connect to which components.
  + I2C\_SDA (17), I2C\_SCL (18): Primary I2C bus (likely for Display, BME280, QMC6310).
  + I2C1\_SDA (42), I2C1\_SCL (41): Secondary I2C bus (Wire1).
  + PMU\_IRQ (40): Power Management Unit interrupt pin.
  + GPS\_RX\_PIN (9), GPS\_TX\_PIN (8): UART pins for GPS communication.
  + GPS\_WAKEUP\_PIN (7), GPS\_PPS\_PIN (6): GPS control/timing pins.
  + BUTTON\_PIN (0): User button pin.
  + RADIO\_SCLK\_PIN (12), RADIO\_MISO\_PIN (13), RADIO\_MOSI\_PIN (11), RADIO\_CS\_PIN (10): LoRa Radio SPI pins.
  + RADIO\_RST\_PIN (5), RADIO\_DIO1\_PIN (1), RADIO\_BUSY\_PIN (4): LoRa Radio control pins.
  + SPI\_MOSI (35), SPI\_SCK (36), SPI\_MISO (37): *Second* SPI bus pins (HSPI).
  + IMU\_CS (34), IMU\_INT (33): IMU (QMI8658) Chip Select and Interrupt pins (uses second SPI bus).
  + SDCARD\_CS (47): SD Card Chip Select pin (uses second SPI bus). Aliases SDCARD\_MOSI/MISO/SCLK to the second SPI bus pins.
  + RTC\_INT (14): Pin for an external Real-Time Clock interrupt (if present).
* **Peripheral Configuration:**
  + GPS\_BAUD\_RATE (9600): Default baud rate for the GPS module.
  + PMU\_WIRE\_PORT Wire1: Specifies the PMU uses the *secondary* I2C bus (Wire1).
  + DISPLAY\_MODEL U8G2\_SH1106\_128X64\_NONAME\_F\_HW\_I2C: Defines the exact U8g2 constructor for the onboard SH1106 OLED display (128x64, Hardware I2C on the primary bus).
* **Hardware Feature Flags:**
  + HAS\_SDCARD, HAS\_GPS, HAS\_DISPLAY, HAS\_PMU, HAS\_SENSOR: Flags indicating the presence of these components, enabling conditional code compilation.
  + \_\_HAS\_SPI1\_\_: Indicates the presence of the secondary SPI bus.

**4. Board Support Package (BSP) Implementation (**boards.cpp **- Analysis of Code 3)**

This file provides the functions that bring the hardware definitions to life. It initializes peripherals correctly for the T-Beam S3 Supreme.

* setupBoards(bool disable\_u8g2 = false)**:**
  + The main initialization routine called once from the application's setup().
  + Initializes Serial for debugging (115200 baud).
  + Prints chip information (getChipInfo).
  + Initializes the primary SPI bus (SPI) for the LoRa radio.
  + Initializes the secondary SPI bus (SDCardSPI) if HAS\_SDCARD.
  + Initializes both I2C buses (Wire, Wire1) and scans them for devices using scanDevices().
  + Initializes the GPS serial port (SerialGPS) using the defined pins and baud rate.
  + Configures GPIOs for onboard LED, GPS power/reset (if defined).
  + **Calls** beginPower()**:** This is critical. It detects and initializes the AXP2101 PMU, configures all its power output rails to supply the correct voltages to the LoRa chip, GPS, sensors, SD card, display, etc., and sets up PMU interrupts.
  + **Calls** beginSDCard()**:** Initializes the SD card interface.
  + **Calls** beginDisplay()**:** Initializes the OLED display and shows a splash screen (unless disable\_u8g2 is true).
  + **Calls** beginGPS()**:** Attempts to detect and configure the connected GPS module (probing for L76K, then attempting UBlox recovery across multiple baud rates).
  + Configures the ESP32's RTC slow clock to use the 32kHz external crystal (enable\_slow\_clock) for better accuracy.
* beginPower()**:**
  + Detects if an AXP2101 or AXP192 PMU is present on Wire1.
  + **AXP2101 Configuration (T-Beam S3 Supreme):**
    - Enables/Disables specific DCDC and ALDO/BLDO power rails.
    - Sets voltages (typically 3.3V) for: ESP32 (DCDC1 - protected), LoRa (ALDO3), GPS (ALDO4), Sensors (ALDO1/ALDO2), SD Card (BLDO1). Note the code snippet has conflicting assignments for ALDO2/ALDO3/ALDO4 between different sections, the T-Beam S3 specific part likely takes precedence. *Careful review of the schematic vs code is needed for exact rail assignment.*
    - **Important Workaround:** If it's a cold boot (not waking from sleep), it temporarily disables then re-enables ALDO1, ALDO2, BLDO1 power rails with a 250ms delay. This is done to prevent potential bus conflicts between the SD card and QMC sensor during initialization.
    - Sets charging parameters (current limit 500mA, target voltage 4.2V).
    - Configures and enables specific PMU interrupts (Battery/VBUS insertion/removal, Power key short/long press, Charging start/done).
    - Enables PMU's internal ADC for monitoring system, VBUS, and battery voltages.
    - Prints a detailed report of enabled power rails and their voltages.
    - Sets the power key press time required to trigger a shutdown (4 seconds).
* scanDevices(TwoWire \*w)**:**
  + Iterates through I2C addresses (1-126) on the specified bus (w).
  + Uses w->beginTransmission() and w->endTransmission() to check for an ACK.
  + Prints the address of any found device.
  + Identifies known devices by address: BME280 (0x76/0x77), AXP PMU (0x34), OLED (0x3C), PCF8563 RTC (0x51), QMC6310 (0x1C). Sets corresponding bits in deviceOnline.
* beginDisplay()**,** beginSDCard()**,** beginGPS()**:** Implement initialization using standard libraries (U8g2, SD, Serial) and custom probing/configuration logic for the GPS.
* loopPMU(void (\*pressed\_cb)(void))**:** Checks the PMU interrupt flag, reads/clears the status, prints the interrupt source, and calls the provided callback on a short power key press.
* disablePeripherals()**:** Powers down specific components via PMU rails and disables PMU measurements, preparing for deep sleep.

**5. Application Logic (**TelemetryTx.ino **- Analysis of Code 1)**

This is the main Arduino sketch that defines the core behavior of the telemetry transmitter.

* **Includes:** Pulls in necessary libraries:
  + LoRaBoards.h: Board definitions (Code 2).
  + RadioLib.h: For LoRa communication.
  + Wire.h, SPI.h: For I2C and SPI communication.
  + Sensor Libraries: Adafruit\_BME280.h, SensorQMC6310.hpp, SensorQMI8658.hpp.
  + U8g2lib.h: For the display.
  + RS-FEC.h: For Reed-Solomon encoding.
  + TinyGPS++.h: For parsing GPS NMEA sentences.
  + Standard C libraries (math.h, stdio.h).
* **Global Variables & Objects:**
  + Sensor objects: bme, qmc, qmi, gps.
  + LoRa Radio object: radio (SX1262 instance from RadioLib).
  + Reed-Solomon encoder: rs instance (RS::ReedSolomon<223, 32>).
  + Telemetry Packet: tmPacket (instance of CCSDSPacket struct).
  + encodedData[]: Buffer to hold the RS-encoded packet (223 data + 32 parity = 255 bytes).
  + State variables: packetCounter, transmittedFlag (volatile bool for ISR), currentScreen, sensor reading variables.
* CCSDSPacket **Structure:**
  + Defines the data payload format.
  + Uses #pragma pack(push, 1) to ensure byte alignment without padding, critical for consistent transmission and decoding.
  + **Fields:**
    - version\_type\_apid (uint16\_t): Mimics CCSDS Primary Header fields (Version=0, Type=0 Telemetry, SecHdr=1, APID=0x7FF).
    - sequence\_flags\_count (uint16\_t): Mimics CCSDS Primary Header (SeqFlags=3, Packet Counter).
    - packet\_length (uint16\_t): Length of data field minus 1 (standard CCSDS definition). Calculated as sizeof(CCSDSPacket) - 7.
    - spacecraft\_id (char[12]): Identifier string.
    - mission\_time (uint32\_t): Milliseconds since boot.
    - Sensor Data Fields (float/double/int): BME280 (Temp, Pres, Hum, Alt), QMC6310 (MagX, Y, Z), QMI8658 (AccX, Y, Z, GyrX, Y, Z, Temp), GPS (Lat, Lon, Alt, Date, Time, Sats, HDOP), PMU (Batt Voltage/Percent, Charging, VBUS/System Voltage).
    - message (char[16]): Simple text message ("Packet: %d").
    - crc (uint16\_t): Calculated CRC-16 checksum.
* **Error Control:**
  + calculateCRC(): Implements CRC-16-CCITT-FALSE checksum calculation over the packet data (excluding the CRC field itself).
  + Reed-Solomon: Uses the rs.Encode() method from RS-FEC.h to encode the first rsMsgLen (223) bytes of the tmPacket into the encodedData buffer (255 bytes total).
* **LoRa Communication:**
  + Configured in setup(): Frequency (433.0 MHz), Bandwidth (125 kHz), Spreading Factor (12), Coding Rate (6), Sync Word (0x35), Tx Power (20 dBm), Preamble Length (16). Hardware CRC is *disabled* as software CRC is used within the packet.
  + Transmission is asynchronous: radio.startTransmit() begins sending the encodedData buffer.
  + setFlag(): A callback function attached via radio.setPacketSentAction(). It's called by an Interrupt Service Routine (ISR) when RadioLib finishes transmitting, setting transmittedFlag = true.
* setup() **Function:**
  + Initializes Serial and SerialGPS.
  + Calls setupBoards() (from Code 3) to initialize all hardware.
  + Initializes sensor objects (bme.begin(), qmc.begin(), qmi.begin()). Uses displayFatalError() if any sensor fails initialization.
  + Configures IMU (QMI8658) ranges, data rates, filters.
  + Configures Magnetometer (QMC6310) mode, range, data rate.
  + Initializes the LoRa radio parameters using radio.set...() methods.
  + Sets setFlag as the Tx complete callback.
  + Performs initial sensor readings (updateSensorReadings).
  + Builds the very first packet (buildTelemetryPacket).
  + Starts the first transmission (radio.startTransmit).
* loop() **Function:**
  + **Transmission Cycle:** Checks transmittedFlag. If true:
    - Resets transmittedFlag to false.
    - Increments packetCounter.
    - Calls updateSensorReadings().
    - Calls buildTelemetryPacket() to create the next packet with fresh data, CRC, and RS encoding.
    - Calls radio.startTransmit() to send the new encodedData buffer.
  + **Display Update:** Calls updateDisplay() (cycles screen index) and drawDisplay() (renders current screen).
  + **GPS Processing:** Continuously feeds incoming bytes from SerialGPS to the gps.encode() method of the TinyGPS++ library.
  + Includes delay(1) likely for system stability/yielding.
* buildTelemetryPacket()**:**
  + Constructs CCSDS header fields bitwise.
  + Copies sensor data from global variables into the tmPacket struct. Includes checks for GPS data validity.
  + Calculates the CRC using calculateCRC().
  + Copies the raw packet data (up to rsMsgLen) into the encodedData buffer.
  + Calls rs.Encode() to perform Reed-Solomon encoding *in-place* on the encodedData buffer (overwriting the original data section and adding parity). *Correction:* The code rs.Encode(reinterpret\_cast<char\*>(&tmPacket), reinterpret\_cast<char\*>(encodedData)) actually reads from tmPacket and writes the full encoded block to encodedData.
* **Sensor Update Functions (**update...Data**)**: Read latest values from sensors (BME280, QMC6310, QMI8658, PMU) and store them in global variables. The updateQMC6310Data includes heading calculation using atan2 and applies a hardcoded magnetic declination correction (-0.041 radians) suitable for **Galway, Ireland**.
* **Display Functions (**updateDisplay**,** drawDisplay**,** drawGPS...**)**: Manage cycling through different data screens on the OLED display using the U8g2 library, formatting sensor values appropriately.
* displayFatalError()**:** Prints error to Serial, shows a prominent error message on the display, and halts execution (blinking LED or infinite loop).

**6. Key Concepts & Technologies**

* **LoRa:** Low-Power Wide-Area Network radio technology used for long-range wireless communication. Managed by the RadioLib library.
* **CCSDS (Consultative Committee for Space Data Systems):** An organization that develops communication and data handling standards for space missions. This firmware uses a *simplified structure inspired by* the CCSDS Telemetry Packet Primary Header.
* **Reed-Solomon (RS) FEC:** A Forward Error Correction code that adds redundant parity bytes to data, allowing the receiver to detect and correct a certain number of errors introduced during transmission. Here, RS(255, 223) is used via the RS-FEC.h library, capable of correcting up to 16 byte errors per 255-byte block.
* **CRC (Cyclic Redundancy Check):** An error-detecting code used to verify data integrity. CRC-16-CCITT-FALSE is calculated and included in the packet.
* **I2C & SPI:** Serial communication protocols used to interface with sensors, display, and PMU (I2C) and the LoRa radio, IMU, and SD card (SPI).
* **PMU (Power Management Unit - AXP2101):** An integrated circuit that manages battery charging, power distribution (multiple voltage rails), and provides voltage/current monitoring. Managed by the XPowersLib library.
* **GPS (Global Positioning System):** Receives satellite signals to determine location, altitude, and time. Data is typically output in NMEA standard text sentences, parsed here by TinyGPS++. UBlox modules also use a binary UBX protocol for configuration.
* **U8g2:** A popular library for driving monochrome graphical displays (OLED, LCD).
* **RadioLib:** A comprehensive Arduino library for interfacing with various radio modules, including the SX1262 LoRa chip.

**7. Dependencies**

The complete system requires the following Arduino libraries:

* RadioLib by JGromes
* U8g2 by Olikraus
* Adafruit BME280 Library by Adafruit
* Adafruit Unified Sensor by Adafruit
* TinyGPSPlus by Mikal Hart
* XPowersLib by Lewis He (for AXP PMUs)
* RS-FEC Library (Specific source/library needs to be identified - RS-FEC.h)
* SensorQMC6310 library (Custom - SensorQMC6310.hpp)
* SensorQMI8658 library (Custom - SensorQMI8658.hpp)
* SD Library (Built-in for ESP32)
* SPI, Wire (Built-in)

**8. Operational Flow**

1. **Power On:** The ESP32-S3 boots.
2. **Board Setup (**setupBoards **in** boards.cpp**):**
   * Basic system clocks, Serial, SPI, I2C initialized.
   * PMU (AXP2101) detected and configured, power rails enabled.
   * I2C devices scanned.
   * Display initialized (shows splash screen).
   * SD Card initialized.
   * GPS detected and configured.
   * 32kHz RTC clock enabled.
3. **Application Setup (**setup **in** TelemetryTx.ino**):**
   * Sensor objects (BME280, QMC6310, QMI8658) initialized.
   * LoRa radio configured (frequency, SF, BW, power, etc.).
   * Tx Complete callback (setFlag) registered.
   * Initial sensor readings taken.
   * First telemetry packet built (buildTelemetryPacket) including CRC and RS encoding.
   * First LoRa transmission started (radio.startTransmit).
4. **Main Loop (**loop **in** TelemetryTx.ino**):**
   * Continuously checks if SerialGPS has data and feeds it to gps.encode().
   * Updates the display screen (updateDisplay, drawDisplay).
   * Waits for the transmittedFlag to become true (set by ISR upon LoRa Tx completion).
   * **Once Tx completes:**
     + Flag is reset.
     + Packet counter incremented.
     + Sensors are read (updateSensorReadings).
     + Next packet is built (buildTelemetryPacket) with new data, CRC, RS code.
     + Next transmission is started (radio.startTransmit).
   * The loop repeats, continuously sensing, packaging, encoding, transmitting, and displaying data.
   * (If loopPMU were called in the main loop, it would also check for and handle PMU interrupts).

**9. Potential Enhancements & Considerations**

* **Magnetic Declination:** The declination value (-0.041 radians) is hardcoded for Galway, Ireland. For deployment elsewhere, this needs to be adjusted or dynamically calculated/configured.
* **Error Handling:** Sensor read errors within the main loop are not explicitly handled (e.g., if bme.readTemperature() fails after initialization). The displayFatalError only covers initialization failures. Robust applications might need checks for invalid sensor readings. Radio transmission failures are also not explicitly handled beyond the callback mechanism.
* **Reed-Solomon Parameters:** The RS(255, 223) code is fixed. Depending on the channel conditions and desired data rate, different RS codes might be more optimal. The library used (RS-FEC.h) might support other configurations.
* **Power Consumption:** While disablePeripherals() exists, the main loop doesn't implement specific power-saving strategies (like sleeping between transmissions). For battery-powered operation, deep sleep cycles coordinated with sensor readings and transmissions would be essential.
* **GPS Lock:** The code transmits packets regardless of GPS lock status (it includes default 0 values if GPS is invalid). Depending on the application, waiting for a valid GPS fix before transmitting location data might be desirable.
* **Custom Libraries:** The system relies on custom sensor libraries (SensorQMC6310.hpp, SensorQMI8658.hpp) and the RS-FEC library. Ensuring these are available and correctly implemented is crucial.
* **Code Redundancy:** Some display drawing code (drawPMUScreen1, drawPMUScreen2 in Code 1) seems redundant with the switch statement in drawDisplay. The PMU power rail configuration in boards.cpp has potentially overlapping/conflicting assignments between the generic ESP32 block and the T-Beam S3 specific block – careful verification against the board schematic is recommended.

**10. Conclusion**

This firmware provides a comprehensive example of a telemetry transmitter system built on the capable LilyGO T-Beam S3 Supreme platform. It successfully integrates multiple sensors, implements robust data handling with CRC and Reed-Solomon FEC, utilizes LoRa for long-range communication via the RadioLib library, and leverages a detailed Board Support Package (boards.cpp, LoRaBoards.h) for hardware initialization, including sophisticated power management via the AXP2101 PMU. While primarily an educational demonstration, it forms a solid foundation for developing more complex, real-world LoRa-based monitoring and telemetry applications.