Ardentis

0.0.1-alpha

Generated by Doxygen 1.13.2

Chapter 1

Class Index

1.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

Class Documentation

3.1 ardDataField Struct Reference

ardDataField structure to hold sensor data and timestamp

`#include <ard_sensor.h>`

Public Attributes

- int16_t **val**
- int64_t **timestamp**
- bool **updated**

3.1.1 Detailed Description

ardDataField structure to hold sensor data and timestamp

Parameters

val	Sensor value
timestamp	Timestamp of the last update measured in microseconds since boot
updated	Flag to indicate if the value has been updated

The documentation for this struct was generated from the following file:

- main/ard_sensor.h

3.2 ArdDataStruct Struct Reference

Public Member Functions

- uint16_t **getAltitude** (int sea_pressure) const

Public Attributes

- uint16_t **pressure**
- int32_t **latitude**
- int32_t **longitude**
- int16_t **horizAccuracy**
- int16_t **vertAccuracy**
- int16_t **temperature**
- int16_t **accelX**
- int16_t **accelY**
- int16_t **accelZ**
- int16_t **highAccelX**
- int16_t **highAccelY**
- int16_t **highAccelZ**
- int16_t **gyroX**
- int16_t **gyroY**
- int16_t **gyroZ**
- int16_t **altitude**
- bool **cont1**
- bool **cont2**
- bool **cont3**
- bool **cont4**
- bool **fired_1**
- bool **fired_2**
- bool **fired_3**
- bool **fired_4**

The documentation for this struct was generated from the following file:

- main/data_struct.h

3.3 ardDeployGpioConfig Struct Reference

This structure holds the configuration for GPIO pins used in deployment.

`#include <ard_deployment.h>`

Public Attributes

- gpio_num_t **fire** [4]
- gpio_num_t **cont** [4]

3.3.1 Detailed Description

This structure holds the configuration for GPIO pins used in deployment.

Parameters

fire	Array of 4 GPIO numbers for fire outputs.
cont	Array of 4 GPIO numbers for continuity detection inputs.

The documentation for this struct was generated from the following file:

- main/ard_deployment.h

3.4 ardOutputGPIO Class Reference

Class to manage GPIO pins for deployment.

`#include <ard_deployment.h>`

Public Member Functions

- ardOutputGPIO (ardDeployGpioConfig config)
  Construct a new ardOutputGPIO object.
- esp_err_t fire (FirePins pin)
  Fire the specified pin.
- std::array< bool, 4 > checkContinuity ()
  Check the continuity of the outputs.
- std::array< bool, 4 > fireStatus ()
  Check which pins have been fired.

3.4.1 Detailed Description

Class to manage GPIO pins for deployment.

This class provides methods to initialize GPIO pins, set their modes, and read/write values.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 ardOutputGPIO()

```
ardOutputGPIO::ardOutputGPIO (
            ardDeployGpioConfig config)  [inline]
```

Construct a new ardOutputGPIO object.

Parameters

config	Configuration for GPIO pins used in deployment

3.4.3 Member Function Documentation

3.4.3.1 checkContinuity()

```
std::array< bool, 4 > ardOutputGPIO::checkContinuity ()
```

Check the continuity of the outputs.

Returns

std::array<bool, 4> A array with the continuity status of each output

The array will contain true if there is continuity at the output, and false if there is not.

3.4.3.2 fire()

```
esp_err_t ardOutputGPIO::fire (
            FirePins pin)
```

Fire the specified pin.

Parameters

pin	The output number of the pin to fire.

Returns

esp_err_t Whether the operation succeeded or failed.

3.4.3.3 fireStatus()

```
std::array< bool, 4 > ardOutputGPIO::fireStatus ()
```

Check which pins have been fired.

Returns

std::array<bool, 4> A array with the staus of each output

The array will contain true if the pin has been fired since the instance was initialized, and false if it has not.

The documentation for this class was generated from the following file:

- main/ard_deployment.h

3.5 ardRadio Class Reference

Public Member Functions

- void **begin** ()
- void **sendData** (uint8_t *data, uint8_t length)
- void **receiveData** (uint8_t *data, uint8_t length)
- bool **isDataAvailable** ()
- void **setChannel** (uint8_t channel)
- void **setPowerLevel** (uint8_t level)
- void **setFrequency** (uint32_t frequency)

The documentation for this class was generated from the following file:

- main/ard_radio.h

3.6 ardSensors Class Reference

ardSensors class to manage sensor data acquisition and processing

```
#include <ard_sensor.h>
```

Public Member Functions

- **ardSensors** (gpio_num_t sda, gpio_num_t scl, int i2c, int gps_uart_num, gpio_num_t gps_tx, gpio_num_t gps_rx)
- esp_err_t initSensors (int LIS331_addr, int imu_adr, int baro_addr, int gps_baud)

  Initialize the sensors.
- esp_err_t **update** ()

Public Attributes

- ardDataField **accelX**
- ardDataField **accelY**
- ardDataField **accelZ**
- ardDataField **gyroX**
- ardDataField **gyroY**
- ardDataField **gyroZ**
- ardDataField **highAccelX**
- ardDataField **highAccelY**
- ardDataField **highAccelZ**
- ardDataField **pressure**
- ardDataField **altitude**
- ardDataField **temperature**
- ardDataField **latitude**
- ardDataField **longitude**
- ardDataField **horizAccuracy**
- ardDataField **vertAccuracy**
- ardDataField **gps_altitude**

3.6.1 Detailed Description

ardSensors class to manage sensor data acquisition and processing

Parameters

sda	GPIO pin for I2C SDA
scl	GPIO pin for I2C SCL
i2c	I2C bus number
gps_uart_num	UART number for GPS
gps_tx	GPIO pin for GPS TX
gps_rx	GPIO pin for GPS RX

3.6.2 Member Function Documentation

3.6.2.1 initSensors()

```
esp_err_t ardSensors::initSensors (
            int LIS331_addr,
            int imu_adr,
            int baro_addr,
            int gps_baud)
```

Initialize the sensors.

Parameters

LIS331_addr	I2C address for H3LIS331 accelerometer
imu_adr	I2C address for ICM42670 IMU
baro_addr	I2C address for MS5611 barometer
gps_baud	Baud rate for GPS UART

Returns

esp_err_t Error code indicating success or failure of the initialization

The documentation for this class was generated from the following files:

- main/ard_sensor.h
- main/ard_sensor.cpp

3.7 RadioPacket Struct Reference

Public Attributes

- int32_t **latitude**
- int32_t **longitude**
- uint16_t **altitude**

The documentation for this struct was generated from the following file:

- main/ard_radio_packet_spec.h

Chapter 4

File Documentation

4.1 main/ard_deployment.h File Reference

Header file for the ard_deployment class.

```
#include <driver/gpio.h>
#include <esp_err.h>
#include <esp_log.h>
```

Classes

- struct ardDeployGpioConfig

  This structure holds the configuration for GPIO pins used in deployment.
- class ardOutputGPIO

  Class to manage GPIO pins for deployment.

Enumerations

- enum class FirePins : uint8_t { **Zero** = 0 , **One** = 1 , **Two** = 2 , **Three** = 3 }

  Enum containint the valid fire indices.

4.1.1 Detailed Description

Header file for the ard_deployment class.

Author

your name (you@domain.com)

Version

0.1

Date

2025-04-25

Copyright

Copyright (c) 2025

This file contains the definition of the ard_deployment class, which is used to manage GPIO pins for deployment purposes.

4.2 ard_deployment.h

Go to the documentation of this file.

```
00001
00013
00014 #include <driver/gpio.h>
00015
00016 #include <esp_err.h>
00017 #include <esp_log.h>
00018
00026 struct ardDeployGpioConfig {
00027     gpio_num_t fire[4];
00028     gpio_num_t cont[4];
00029 };
00030
00035 enum class FirePins : uint8_t {
00036     Zero = 0,
00037     One = 1,
00038     Two = 2,
00039     Three = 3
00040 };
00041
00046 class ardOutputGPIO {
00047 public:
00053     ardOutputGPIO(ardDeployGpioConfig config) {
00054         config = config;
00055         for (int i = 0; i < 4; i++) {
00056             gpio_set_direction(config.fire[i], GPIO_MODE_OUTPUT); // Set fire pins as output
00057             gpio_set_level(config.fire[i], 0); // Initialize fire pins to low
00058             gpio_set_direction(config.cont[i], GPIO_MODE_INPUT);  // Set continuity pins as input
00059             gpio_set_pull_mode(config.cont[i], GPIO_PULLUP_ONLY); // Enable pull-up resistors on
      continuity pins
00060         }
00061     }
00068     esp_err_t fire(FirePins pin);
00069
00076     std::array<bool, 4> checkContinuity();
00077
00084     std::array<bool, 4> fireStatus();
00085 private:
00086     ardDeployGpioConfig config;
00087 };
```

4.3 ard_radio.h

```
00001 #pragma once
00002 /*
00003                           @@@@@@@@
00004                           @@@@@@@@@@
00005                         @@@@@@@@@@@@
00006                   @@    @@@@@@@@@@@@@@
00007                 @@@   @@@@@@@@@@@@@@@@
00008               @@@@  @@@@@@@@ @@@@@@@
00009             @@@@@ @@@@@@@@   @@@@@@@
00010             @@@@  @@@@@@@     @@@@@@@
00011           @@@@@ @@@@@@         @@@@@@
00012         @@@@@@@ @@@@@@@         @@@@@@@
00013       @@ @@@@@@@@@  @@    @    @@ @@@@@
00014          @@  @@      @  @@@@@ @       @
00015        @@  @@          @@@@         @@@
00016        @@  @@      @@@@@@@@@@@@@      @@
00017       @@@ @@@@@    @@@@@@@@@@@@@   @@@@@@
00018      @ @ @@@@@@@@    @@@@@@@@@   @@@@@@@@@
00019     @@@@ @@@@@@@@@@   @  @ @@   @@@@@@@@@@
00020    @@@ @ @@@@@@@@@@       @       @@@@@@@@@@
00021  @@@@@@ @@ @@@@@@@@@       @        @@@@@@@@@
00022  @@     @@@@ @@@@@@                  @@@@@@@@@@
00023  @@@@@@@@@@ @@@@@                    @@@@@@@@@@
00024 @@@@@@@    @@  @@@                   @@@@@@@@@@
00025  @@@@@@@@@@@@@@@@                     @@@@@@@@@@

00026 */
00027
00028
00029 #include "ard_sensor.h"
00030
00031 #include <stdint.h>
00032
00033 #include <esp_log.h>
00034
```

```
00035 class ardRadio
00036 {
00037 public:
00038     ardRadio();
00039     void begin();
00040     void sendData(uint8_t *data, uint8_t length);
00041     void receiveData(uint8_t *data, uint8_t length);
00042     bool isDataAvailable();
00043     void setChannel(uint8_t channel);
00044     void setPowerLevel(uint8_t level);
00045     void setFrequency(uint32_t frequency);
00046 };
```

4.4 ard_radio_packet_spec.h

```
00001 #pragma once
00002
00003 #include <Arduino.h>
00004 #include <stdint.h>
00005 #include <string.h>
00006
00007 struct RadioPacket {
00008     int32_t latitude;
00009     int32_t longitude;
00010     uint16_t altitude;
00011 };
00012
00013 String createRadioPacket(RadioPacket& packet, String& callsign, uint8_t version);
00014
00015 RadioPacket parseRadioPacket(const String& packetString);
```

4.5 ard_sensor.h

```
00001 #pragma once
00002
00003 #include <stdint.h>
00004
00005 #include <SparkFun_u-blox_GNSS_v3.h>
00006
00007 #include <Adafruit_Sensor.h> //Needed for adafruit sensor libraries
00008 #include <Adafruit_H3LIS331.h> // For H3LIS331 High G Accelerometer
00009
00010 #include <ICM42670P.h> // For ICM42670P IMU
00011
00012 #include <icm42670.h>
00013
00014 #include <MS5611.h> // For MS5611 Barometer
00015
00016 #include <Wire.h> // For I2C communication
00017
00018 #include <HardwareSerial.h>
00019
00020 #include <driver/i2c_master.h>
00027 struct ardDataField {
00028     int16_t val;
00029     int64_t timestamp;
00030     bool updated;
00031 };
00037 void updateArdDataField(ardDataField* field, int16_t val);
00038
00048 class ardSensors {
00049 public:
00050     ardSensors(gpio_num_t sda, gpio_num_t scl, int i2c, int gps_uart_num, gpio_num_t gps_tx,
      gpio_num_t gps_rx);
00051     ~ardSensors();
00052
00061     esp_err_t initSensors(int LIS331_addr, int imu_adr, int baro_addr, int gps_baud);
00062     esp_err_t update(); // Function to check if new sensor data is available and update the data
      fields accordingly.
00063     ardDataField accelX; // Accelerometer X-axis data
00064     ardDataField accelY; // Accelerometer Y-axis data
00065     ardDataField accelZ; // Accelerometer Z-axis data
00066     ardDataField gyroX; // Gyroscope X-axis data
00067     ardDataField gyroY; // Gyroscope Y-axis data
00068     ardDataField gyroZ; // Gyroscope Y-axis data
00069     ardDataField highAccelX; // High G Accelerometer X-axis data
00070     ardDataField highAccelY; // High G Accelerometer Y-axis data
00071     ardDataField highAccelZ; // High G Accelerometer Z-axis data
```

```
00072     ardDataField pressure; // Pressure data from barometer
00073     ardDataField altitude; // Altitude data from barometer
00074     ardDataField temperature; // Temperature data from barometer
00075     ardDataField latitude; // Latitude data from GPS
00076     ardDataField longitude; // Longitude data from GPS
00077     ardDataField horizAccuracy; // Horizontal accuracy from GPS
00078     ardDataField vertAccuracy; // Vertical accuracy from GPS
00079     ardDataField gps_altitude; // Altitude data from GPS
00080 private:
00081     TwoWire ard_sensor_i2c = NULL; // Arduino I2C bus object
00082     SFE_UBLOX_GNSS_SERIAL gps; // GPS object
00083     HardwareSerial gps_uart = NULL; // GPS UART bus object
00084     MS5611 barometer; // Barometer object
00085     Adafruit_H3LIS331 accel = Adafruit_H3LIS331(); // High G Accelerometer object
00086     icm42670_handle_t imu_handle = nullptr; // ICM42670 IMU object
00087     i2c_master_bus_handle_t i2c_handle = NULL; // ESP-IDF I2C bus object
00088     gpio_num_t sda; // SDA GPIO pin
00089     gpio_num_t scl; // SCL GPIO pin
00090     gpio_num_t uart_rx; // GPS RX GPIO pin
00091     gpio_num_t uart_tx; // GPS TX GPIO pin
00092     int gps_uart_num; // GPS UART bus number
00093     int i2c_num; // I2C bus number
00094     int16_t imu_update_time; // Timestamp of the last IMU update
00095     int16_t imu_update_interval; // Interval between IMU updates in microseconds
00096     int16_t high_g_accel_update_time; // Timestamp of the last High G Accelerometer update
00097     int16_t high_g_accel_update_interval; // Interval between High G Accelerometer updates in
      microseconds
00098     int16_t barometer_update_time; // Timestamp of the last Barometer update
00099     int16_t barometer_update_interval; // Interval between Barometer updates in microseconds
00100     int16_t gps_update_time; // Timestamp of the last GPS update
00101     int16_t gps_update_interval; // Interval between GPS updates in microseconds
00102 };
```

4.6 ard_ui.h

```
00001
```

4.7 data_struct.h

```
00001 #pragma once
00002
00003 #include <Arduino.h>
00004
00005 struct ArdDataStruct {
00006     uint16_t pressure;
00007     int32_t latitude;
00008     int32_t longitude;
00009     int16_t horizAccuracy;
00010     int16_t vertAccuracy;
00011     int16_t temperature;
00012     int16_t accelX;
00013     int16_t accelY;
00014     int16_t accelZ;
00015     int16_t highAccelX;
00016     int16_t highAccelY;
00017     int16_t highAccelZ;
00018     int16_t gyroX;
00019     int16_t gyroY;
00020     int16_t gyroZ;
00021     int16_t altitude;
00022     bool cont1;
00023     bool cont2;
00024     bool cont3;
00025     bool cont4;
00026     bool fired_1;
00027     bool fired_2;
00028     bool fired_3;
00029     bool fired_4;
00030
00031     uint16_t getAltitude(int sea_pressure) const {
00032         // Example calculation for altitude based on pressure
00033         // Replace with your actual formula
00034         return ((pow((sea_pressure / pressure), 1/5.257) - 1.0) * (temperature + 273.15)) / 0.0065;
00035     }
00036 };
```

4.8 fire_modes.hpp

```
00001 #define ARD_FIRE_INACTIVE 0 // Inactive Channel
00002 #define ARD_FIRE_APOGEE 1 // Will fire at apogee or soon afterwards
00003 #define ARD_FIRE_ALTITUDE_DESCENDING 2 // Will fire at or below set altitude on descent
00004 #define ARD_FIRE_ALTITUDE_ASCENDING 3 // Will fire at or above set altitude on ascent
00005 #define ARD_FIRE_TIME_ASCENDING 4 // Will fire at set time after liftoff
00006
00007 // For future use. Currently not implemented
00008 // #define ARD_FIRE_AIR_START 4 // Will fire at or below set time on descent
```

4.9 main/main.cpp File Reference

Main file for the Ardentis flight computer code.

```
#include "pins.h"
#include "fire_modes.hpp"
#include "RadioParameters.h"
#include "data_struct.h"
#include "ard_sensor.h"
#include "ard_deployment.h"
#include <esp_log.h>
#include <Arduino.h>
#include <Wire.h>
#include <FS.h>
#include <LittleFS.h>
#include <ICM42670P.h>
#include <SPI.h>
#include <SD.h>
#include <RadioLib.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_H3LIS331.h>
#include <Adafruit_LIS2MDL.h>
#include <MS5611.h>
#include <Preferences.h>
#include <SparkFun_u-blox_GNSS_v3.h>
```

Functions

- HardwareSerial **gpsUART** (ARD_GPS_UART)
- void **setFlag** (void)
- TwoWire **ard_i2c** (1)
- MS5611 **barometer** (0x77)
- void **setup** (void)
- void **loop** ()

Variables

- ardSensors **sensors** (ARD_SDA, ARD_SCL, 1, ARD_GPS_UART, ARD_GPS_TX, ARD_GPS_RX)
- SFE_UBLOX_GNSS_SERIAL **gps**
- File **flightLog**
- SX1262 **radio** = new Module(ARD_RADIO_NSS, ARD_RADIO_BUSY, ARD_RADIO_NRST, ARD_RADIO↩ _DIO1)
- int **transmissionState** = RADIOLIB_ERR_NONE

- volatile bool **transmittedFlag** = false
- volatile bool **radioBusy** = false
- bool **fire_signal_1** = false
- bool **fire_signal_2** = false
- bool **fire_signal_3** = false
- bool **fire_signal_4** = false
- bool **continuity_1** = false
- bool **continuity_2** = false
- bool **continuity_3** = false
- bool **continuity_4** = false
- bool **apogee** = false
- uint16_t **time_since_apogee** = 0
- uint16_t **altitude** = 0
- bool **descending** = false
- bool **ascending** = false
- uint16_t **time_since_liftoff** = 0
- int **ref_pressure** = 101325
- uint8_t **fire_1_mode** = ARD_FIRE_INACTIVE
- uint8_t **fire_1_param** = 0
- uint8_t **fire_2_mode** = ARD_FIRE_INACTIVE
- uint8_t **fire_2_param** = 0
- uint8_t **fire_3_mode** = ARD_FIRE_INACTIVE
- uint8_t **fire_3_param** = 0
- uint8_t **fire_4_mode** = ARD_FIRE_INACTIVE
- uint8_t **fire_4_param** = 0
- uint8_t **fire_1_altitude** = 0
- uint8_t **fire_2_altitude** = 0
- uint8_t **fire_3_altitude** = 0
- uint8_t **fire_4_altitude** = 0

4.9.1 Detailed Description

Main file for the Ardentis flight computer code.

Author

Roland Neill

Version

0.1

Date

2025-04-25

Copyright

4.10 pins.h

```
00001
00002 #define ARD_RADIO_SPI_SCK 7
00003 #define ARD_RADIO_SPI_MISO 8
00004 #define ARD_RADIO_SPI_MOSI 9
00005 #define ARD_RADIO_NSS 4
00006 #define ARD_RADIO_DIO1 1
00007 #define ARD_RADIO_NRST 10
00008 #define ARD_RADIO_BUSY 6
00009
00010 #define ARD_GPS_RX gpio_num_t(17)
00011 #define ARD_GPS_TX gpio_num_t(18)
00012 #define ARD_GPS_UART 1U
00013 #define ARD_GPS_BAUD 38400
00014
00015 #define ARD_SDA gpio_num_t(3)
00016 #define ARD_SCL gpio_num_t(2)
00017
00018 #define ARD_IS33_INT1 gpio_num_t(21)
00019 #define ARD_LIS33_INT2 gpio_num_t(16)
00020
00021 #define ARD_LIS2_INT gpio_num_t(12)
00022 #define ARD_ICM_INT1 gpio_num_t(39)
00023 #define ARD_ICM_INT2 gpio_num_t(40)
00024
00025 #define ARD_FIRE1 gpio_num_t(15)
00026 #define ARD_FIRE2 gpio_num_t(13)
00027 #define ARD_FIRE3 gpio_num_t(42)
00028 #define ARD_FIRE4 gpio_num_t(41)
00029
00030 #define ARD_CONT1 14
00031 #define ARD_CONT2 11
00032 #define ARD_CONT3 43
00033 #define ARD_CONT4 44
```

4.11 preference_shorthand.h

```
00001 #define WIFINAMESPACE "a"
00002
00003 #define WIFISSIDID "a"
00004 #define WIFIPASSWORDID "b"
00005
00006 #define RADIONAMESPACEID "b"
00007 #define RADIOFREQID "a"
00008 #define RADIOBANDWIDTHID "b"
00009 #define RADIOSFID "c"
00010 #define RADIOCRID "d"
00011 #define RADIOCALLSIGNID "e"
00012
00013
```

4.12 preference_values.h

```
00001 #ifdef PREFERENCES
00002
00003 // WiFi Parameters
00004 #define WiFiPassword "WiFiPassword"
00005 #define WiFiSSID "WiFiSSID"
00006
00007 // Radio Parameters
00008 #define FREQ 915
00009 #define BANDWIDTH 125
00010 #define SF 7
00011 #define CR 5
00012 #define CALLSIGN "NULL"
00013
00014 #endif
```

4.13 RadioParameters.h

```
00001 //#include <SX126x-Arduino.h>
00002 #include "pins.h"
```

```
00003
00004 #define ARD_RADIO_FREQ 915 // MHz
00005 #define ARD_LORA_BW 125 // kHz
00006 #define ARD_LORA_SF 7
00007 #define ARD_LORA_CR 4 // 4/5
00008
00009 #define LORA_PREAMBLE_LENGTH 8  // Same for Tx and Rx
00010 #define LORA_SYMBOL_TIMEOUT 0   // Symbols
00011 #define LORA_FIX_LENGTH_PAYLOAD_ON false
00012 #define LORA_IQ_INVERSION_ON false
00013 #define RX_TIMEOUT_VALUE 3000
00014 #define TX_TIMEOUT_VALUE 3000
00015
00016 #define ARD_LORA_SYNCWORD 0x12U
00017
00018 #define ARD_RADIO_TX_PWR 22
00019
00020 #define CALLSIGN "VE3FSZ"
00021
00022 #define BUFFER_SIZE 64 // Define the payload size here
00023
00024 //static RadioEvents_t RadioEvents;
00025 //hw_config ardRadioConfig;
00026
00027 // Define the HW configuration between MCU and SX126x
00028 /*void setupRadioConfig() {
00029     ardRadioConfig.CHIP_TYPE = SX1262_CHIP;         // Example uses an eByte E22 module with an SX1262
00030     ardRadioConfig.PIN_LORA_RESET = ARD_RADIO_NRST; // LORA RESET
00031     ardRadioConfig.PIN_LORA_NSS = ARD_RADIO_NSS;     // LORA SPI CS
00032     ardRadioConfig.PIN_LORA_SCLK = ARD_RADIO_SPI_SCK;   // LORA SPI CLK
00033     ardRadioConfig.PIN_LORA_MISO = ARD_RADIO_SPI_MISO;   // LORA SPI MISO
00034     ardRadioConfig.PIN_LORA_DIO_1 = ARD_RADIO_DIO1; // LORA DIO_1
00035     ardRadioConfig.PIN_LORA_BUSY = ARD_RADIO_BUSY;   // LORA SPI BUSY
00036     ardRadioConfig.PIN_LORA_MOSI = ARD_RADIO_SPI_MOSI;   // LORA SPI MOSI
00037     ardRadioConfig.USE_DIO3_ANT_SWITCH = false;     // Only Insight ISP4520 module uses DIO3 as
      antenna control
00038     ardRadioConfig.USE_LDO = false;                 // Set to true if SX126x uses LDO instead of DCDC
      converter
00039     ardRadioConfig.USE_RXEN_ANT_PWR = false;        // Antenna power is not controlled by a GPIO
00040 }*/
```

Index