# Satellite Onboard Software


- Overview
- Useful Resources
- Arduino-Pico Core
- Onboard Modules
- Functional Requirements
  - Functional Modes
  - Suspended Mode
  - Processing Mode
  - Phase A: initial setup
  - Phase B: waiting for the flight
  - Phase C: going up
  - Phase D: going down
  - Phase E: touching the ground
- Non-functional Requirements
- High-Level Design
- Low-Level Design
- Program Code


## Overview

Our Satellite is equipped with Raspberry Pi Pico board, based on RP2040 microcontroller:

- Arm Cortex-M0+ @ 133MHz - 32-bit, dual core
- 264kB on-chip SRAM
- has 40 pins (including 30 GPIO pins)
- supports 2 SPI controllers and 2 I2C controllers

RP2040 supports C/C++ SDK and Python SDK (MicroPython). We have chosen C/C++ SDK because the most of libraries provided by hardware modules in our setup are written for C++.

## Useful Resources

- Raspberry Pi Pico RP2040
- Raspberry Pi Pico C/C++ development guide
- Raspberry Pi Pico C/C++ SDK
- Arm Cortex-M0+ overview
- Arm Cortex-M0+ specification

## Arduino-Pico Core

Most of the libraries provided for our modules were done for Arduino and C/C++ SDK. This is why we decided to install Arduino-Core on our RP2040 microcontroller. More details:

- Arduino-Pico core by Earle F. Philhower
- How to install Arduino-Pico core
- Arduino-Pico core documentation

# Onboard Modules

todo

# Functional Requirements

Below we have listed the description of functions performed by Satellite.

## Functional Modes

We distinguish two functional modes of the Satellite during its mission performance:

- suspended mode - it’s ready to process the data, but does nothing to save the energy, this is the mode switched on right after Satellite is empowered and waits for the flight
- processing mode - it actually starts data processing, this is the mode switched on once it started the flight

## Suspended Mode

In suspended mode it:

- doesn’t query any sensor except of GPS module
- checks the altitude to understand its position

## Processing Mode

In processing mode it:

- begins new iteration every 5 seconds, during this iteration it performs one data processing cycle
- verifies the altitude and detects the moving direction

Data processing cycle includes:

- Collecting the current data from sensors:
  - temperature
  - pressure
  - humidity
  - location coordinates (latitude, longitude, altitude)
- Transmitting the collected data:
  - send new data package using radio 433 MHz frequency for transmission

The overall lifecycle of Satellite mission consists of the following phases.

## Phase A: initial setup

Satellite performs the following actions once it’s turned on:

- performs initial preparations like connection to every sensor
- switches suspended mode

## Phase B: waiting for the flight

While it’s not moving in terms of altitude (i.e. it doesn’t change significantly):

- this means it’s still positioned on the ground waiting for the flight, so it just remains in suspended mode (i.e. does nothing)

### Phase C: going up

Once it detects it started going up:

- this means it's being raised in a carriage, so it switches into processing mode

### Phase D: going down

Once it detects that it moves down, i.e. CanSat was dropped out from the carriage, it starts checking its current altitude position relative to the ground level:

- when it reaches the height of 500 meters over the ground, it opens the second parachute

### Phase E: touching the ground

Once it detects it went down but not moving anymore:

- it means it reached the ground level so it opens the feet

Even after mission is complete, Satellite still continues data transmission so it could be found by it's coordinates.

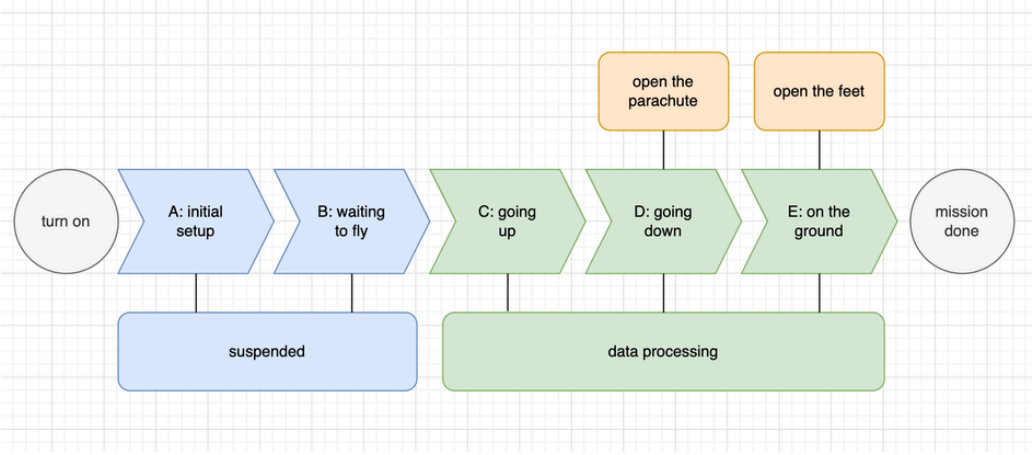
## Non-functional Requirements

Satellite should be capable to spend up to 4 hours in suspended mode before it is sent to flight, so it should save the battery during that time.

GNSS module requires up to 120 sec before it finds satellites and detects the proper coordinates, so it cannot be used at the very start of the application.

## High-Level Design

The below diagram outlines the main phases in Satellite program lifecycle:




## Low-Level Design

The below diagram describes the logic implemented in Satellite's onboard program.


## Program Code

The program is written in C++ and is aligned with Arduino-like flow, it includes these major sections

1. included libraries (#include statements) - both natively provided by Arduino-Pico core and those provided by the manufacturers of onboarded modules
2. declared global variables (used and modified during entire lifecycle of the application)
3. initial setup - setup() function, performs
4. main program code - loop() function

Most up-to-date version of the code can be seen in:

cansat/CanSat_2024 at main · LukaRozhok/cansat

```
#include <Wire.h> //Needed for I2C to GNSS

#include <Servo.h>
Servo servo_feet, servo_parachute;
#include <SPI.h>
#include <LoRa.h>
#include <SparkFun_u-blox_GNSS_v3.h> //http:///librarymanager/All#SparkFun_u-blox_GNSS_v3

#include "Adafruit_SHT4x.h"
#include "SparkFunBMP384.h"
```

```

BMP384 pressureSensor;
SFE_UBLOX_GNSS myGNSS;
Adafruit_SHT4x sht4 = Adafruit_SHT4x();

uint8_t i2cAddress = BMP384_I2C_ADDRESS_DEFAULT; // 0x77
#define myWire1 Wire1
const int csPin = 17;
const int resetPin = 1;
const int irqPin = 0;
int msgCount = 0;

const uint RED_PIN = 26;
const uint GREEN_PIN = PICO_DEFAULT_LED_PIN;
const uint YELLOW_PIN = 22;

const uint PARACHUTE_LOW_BOUNDARY = 1000;
const uint PARACHUTE_HIGH_BOUNDARY = 1500;

int gpsReadings = 0; // counter of GPS readings
bool gpsReady = false; // is GPS ready
int goingDownCounter = 0;
int notMovingCounter = 0;
bool parachuteOpen = false;
bool footsOpen = false;

int32_t ALT_PRECISION = 5; // meter(s)
int32_t altitude_current, altitude_prev, altitude_zero;

void init_pin(uint pin){
  gpio_init(pin);
  gpio_set_dir(pin, GPIO_OUT);
}

void setup()
{
  init_pin(GREEN_PIN);
  init_pin(YELLOW_PIN);
  init_pin(RED_PIN);
  blink_all();

  Wire.begin();
  Serial.begin(115200);
  delay(1000);
  Serial.println("Qwicc and LoRa test has started");

  servo_feet.attach(27);
  servo_parachute.attach(21);
  servo_parachute.write(0);
  Serial.println("Return servo parachute to position zero");

  SPI.setRX(16);
  SPI.setCS(17);
  SPI.setSCK(18);
  SPI.setTX(19);
  SPI.begin();
  delay(1000);

```

```
  LoRa.setPins(csPin, resetPin, irqPin);

  Serial.println("LoRa Sender Test");

  if (!LoRa.begin(433E6)) {
    Serial.println("Starting LoRa failed!");
    while (1);
  }
  Serial.println("LoRa Connected!");


  Serial.println("Adafruit SHT4x test");
  if (! sht4.begin()) {
    Serial.println("Couldn't find SHT4x");
    while (1) delay(1);
  }
  Serial.println("Found SHT4x sensor");
  Serial.print("Serial number 0x");
  Serial.println(sht4.readSerial(), HEX);

  sht4.setPrecision(SHT4X_HIGH_PRECISION);

  while(pressureSensor.beginI2C(i2cAddress) != BMP3_OK)
  {
      Serial.println("Error: BMP384 not connected, check wiring and I2C address!");
      delay(1000);
  }
  Serial.println("BMP384 connected!");

  while (myGNSS.begin() == false)
  {
    Serial.println(F("u-blox GNSS not detected. Retrying..."));
    delay (1000);
  }
  Serial.println("GNSS detected");
  //myGNSS.setI2COutput(COM_TYPE_UBX); //Set the I2C port to output UBX only (turn off NMEA noise)
  //myGNSS.saveConfigSelective(VAL_CFG_SUBSEC_IOPORT); //Optional: save (only) the communications port settings

  blink_multi();
}

void openParachute() {
  Serial.println("Opening parachute");
  Serial.println("servo parachute position 90"); // returns to the original position at start
  servo_parachute.write(90);
  parachuteOpen = true;
}

void openFeet() {
  Serial.println("Opening feet");
  Serial.println("servo feet position 0");
  servo_feet.write(0);
  delay(4500);
  Serial.println("servo feet position 90");
  servo_feet.write(90);
  footsOpen = true;
}


```

```
void blink(uint pin){
  gpio_put(pin, 1);
  sleep_ms(1000);
  gpio_put(pin, 0);
  sleep_ms(1000);
}

void blink_all(){
  gpio_put(GREEN_PIN, 1);
  gpio_put(YELLOW_PIN, 1);
  gpio_put(RED_PIN, 1);
  sleep_ms(1000);

  gpio_put(GREEN_PIN, 0);
  gpio_put(YELLOW_PIN, 0);
  gpio_put(RED_PIN, 0);
  sleep_ms(1000);
}

void blink_multi() {
  blink_all();
  blink_all();
  blink_all();
}

void blink_short(uint pin){
  gpio_put(pin, 1);
  sleep_ms(100);
  gpio_put(pin, 0);
  sleep_ms(100);
}

void loop()
{
  int32_t latitude;
  int32_t longitude;
  int32_t altitude ;
  Serial.print("started iteration ");
  Serial.println(msgCount);
  if (myGNSS.getPVT() == true)
  {
    latitude = myGNSS.getLatitude();
    Serial.print(F("Lat: "));
    Serial.print(latitude);

    longitude = myGNSS.getLongitude();
    Serial.print(F(" Long: "));
    Serial.print(longitude);
    Serial.print(F(" (degrees * 10^-7)"));

    altitude_current = myGNSS.getAltitudeMSL() / 1000; // Altitude above Mean Sea Level
    Serial.print(F(" Alt: "));
    Serial.print(altitude_current);
    Serial.print(F(" (mm)"));

    Serial.println();
  }
  else {
```

```
    Serial.println("no PVT response");
  }

  if (!gpsReady && gpsReadings >= 5) {
    gpsReady = true;
    blink_multi();
  }
  String gps_msg = "";
  if (gpsReady) {
    if (altitude_prev - altitude_current > ALT_PRECISION) {
      Serial.println("going down");
      gps_msg = "dir down";
      goingDownCounter++;
      notMovingCounter = 0;
      blink(RED_PIN);

      if (!parachuteOpen && goingDownCounter >= 3 &&
        altitude_current - altitude_zero < PARACHUTE_HIGH_BOUNDARY &&
        altitude_current - altitude_zero > PARACHUTE_LOW_BOUNDARY
        ) {
        Serial.println("we are at 1km above the ground level, open the parachute");
        gps_msg = gps_msg + " open parachute";
        openParachute();
      }
    } else if (altitude_current - altitude_prev > ALT_PRECISION) {
      Serial.println("going up");
      gps_msg = "dir up";
      goingDownCounter = 0;
      notMovingCounter = 0;
      blink(GREEN_PIN);
    } else {
      Serial.println("not moving");
      gps_msg = "not moving";
      goingDownCounter = 0;
      blink(YELLOW_PIN);
      if (!footsOpen && notMovingCounter >= 3) {
        gps_msg = gps_msg + " open feet";
        openFeet();
      }
      notMovingCounter++;
    }
  } else {
    Serial.println("GPS not ready");
    gps_msg = "not ready";
    blink_short(GREEN_PIN);
    blink_short(YELLOW_PIN);
    blink_short(RED_PIN);
  }

  if (altitude_current > 0 && !gpsReady) {
      Serial.println("positive altitude detected, getting ready...");
      gpsReadings++;
  }
  else {
      gpsReadings = 0;
  }

  bmp3_data data;
```

```
  int8_t err = pressureSensor.getSensorData(&data);
  if(err == BMP3_OK) {
    Serial.print("P "); Serial.print(data.pressure); Serial.println("hPa");
  }
  else {
      Serial.print("Error getting data from sensor! Error code: ");
      Serial.println(err);
  }

  sensors_event_t humidity, temp;
  uint32_t timestamp = millis();
  sht4.getEvent(&humidity, &temp);
  timestamp = millis() - timestamp;

  Serial.print("T "); Serial.print(temp.temperature); Serial.println(" C");

  Serial.print("Read duration (ms): ");
  Serial.println(timestamp);

  Serial.println("Lora sent started");
  LoRa.beginPacket();
  LoRa.print("MSG: ");
  LoRa.print(msgCount);
  LoRa.print(" T: ");
  LoRa.print(temp.temperature);
  LoRa.print(" P: ");
  LoRa.println(data.pressure);
  LoRa.print(" LA: ");
  LoRa.print(latitude);
  LoRa.print(" LO: ");
  LoRa.print(longitude);
  LoRa.print(" AL: ");
  LoRa.print(altitude_current);

  LoRa.print(" read: ");
  LoRa.print(gpsReadings);

  LoRa.print(" GPS: ");
  LoRa.print(gps_msg);
  LoRa.endPacket();
  Serial.println("Lora sent ended");

  Serial.println();
  msgCount++;

  if (footsOpen) {
    blink_multi();
  }
  delay(5000);
  altitude_prev = altitude_current;
}
```