# EXISTING LEO SATELLITE PROJECTS AND THE MICROCONTROLLERS USED

# OVERVIEW

To obtain the best functionality for our Leo satellite, we had to gather as much information as we could obtain from the documentation of different projects done by students across the world. This will help us make informed decisions.

The following are some of the documented Leo satellite projects we were able to gather some useful information on the different microcontrollers used.

# PROJECTS AND THEIR MICROCONTROLLER SELECTIONS

## NaSPUoN project

A project by NanoSatellite Platform for the University of Nairobi (NaSPUoN) Team.

The cameras investigated for this project are the Arducam 5MP Mini Camera, USB 3MP Camera and Raspberry Pi High Quality Camera. The camera chosen is the Raspberry Pi High Quality Camera.

The initial design choice was to use a **Raspberry Pi 4** for the **on-board computer**. However, mainly due to **size restrictions** in the 1U CubeSat specification, the **Raspberry Pi Zero** was the component that was settled upon.

Even though the Raspberry Pi Zero is less computationally powerful and has less interfaces compared to the Raspberry Pi 4.

The specifications of the Raspberry Pi Zero are as follows:

* Single-core BCM2835 at 1GHz
* 512MB RAM
* Small Size
* Typical Power draw of 0.75W – this is lower than that of the Raspberry Pi 4
* Can be powered using the GPIO header – the Raspberry Pi 4 cannot

Interfaces available:

* 1 UART \*{there are 2 UART’s but only one can be used at a time since they are on the same pins}
* 2 SPI buses (total of 5 CS pins)
* 1 I2C bus \*{2 I2C’s but only one is usable the other being a special-purpose I2C}
* 1 CSI header (only on Raspberry Pi v1.3)

For the **EPS**, an **STM 32** was used but the exact microcontroller is not specified in the documentation

The interfacing block diagram is shown below;

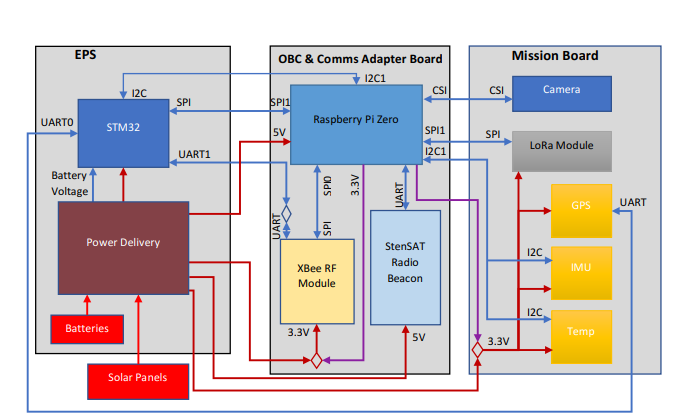


Fig 1. Subsystem interfacing block diagram

## Ardusat

**ArduSat** is an [Arduino](https://en.wikipedia.org/wiki/Arduino) based [nanosatellite](https://en.wikipedia.org/wiki/Nanosat), based on the [CubeSat](https://en.wikipedia.org/wiki/CubeSat) standard. It contains a set of Arduino boards and sensors. The general public will be allowed to use these Arduinos and sensors for their own creative purposes while they are in space.

The ArduSat project currently consists in two identical satellites: ArduSat-1 and ArduSat-X.

|  |  |
| --- | --- |
| **Category** | **Specifications** |
| General Architecture | **1U CubeSat** : the satellites implements the standard 10×10×10 cm basic [CubeSat](https://en.wikipedia.org/wiki/CubeSat) architecture. |
| Computing features | **Arduino-based** : The ArduSat is equipped with 16 processor nodes ([ATmega328](https://en.wikipedia.org/wiki/ATmega328)P) and 1 supervisor node (ATmega2561). The processor nodes are dedicated to the computing of the experiments (each on one node), the supervisor uploads the code to the processor nodes. |
| Sensors | The Arduino processors may sample data from the following sensors  :   * one digital 3-axis **magnetometer** (MAG3110) * one digital 3-axis **gyroscope** (ITG-3200) * one 3-axis **accelerometer** (ADXL345) * one **infrared temperature sensor** with a wide sensing range (MLX90614) * four digital **temperature sensors** (TMP102) : 2 in the payload, 2 on the bottomplate * two **luminosity sensor** (TSL2561) covering both infrared and visible light : 1 on the bottomplate camera, 1 on the bottomplate slit * two **geiger counter** tubes (LND 716) * one optical **spectrometer** (Spectruino) * one 1.3MP **camera** (C439) |
| Coding | The experiments for ArduSat are developed in **C/C++** for AVR/Arduino, using the [ArduSatSDK](https://github.com/ArduSat/ArduSatSDK). |
| Communication | ArduSat is equipped with a **half-duplex UHF transceiver**, operating in the 435–438 MHz [amateur radio satellite band](https://en.wikipedia.org/wiki/Amateur_radio_frequency_allocations). ArduSat-1 : 437.325 MHz 9k6 MSK CCSDS downlink   * ArduSat-X : 437.345 MHz 9k6 MSK CCSDS downlink   Both satellites have a Morse beacon (FM-modulated 800 Hz tones) that is transmitted at 20 WPM every two or three minutes on 437.000 MHz. The beacon will be structured in the following format.   * ArduSat-1 beacon: Battery voltage (uint16\_t), RX\_counter (number of received valid data packets, uint32\_t), TX\_counter (number of sent valid data packets, uint32\_t), "WG9XFC-1″ * ArduSat-X beacon: Battery voltage (uint16\_t), RX\_counter (number of received valid data packets, uint32\_t), TX\_counter (number of sent valid data packets, uint32\_t), "WG9XFC-X" |

Table 1. Ardusat specifications

## ESTCube-1 & -2 (Estonian Student Satellite-1 & -2)

ESTCube-1 & -2 were two Estonian student CubeSat projects of the University of Tartu, which started in the summer of 2008. The objective was to get students involved in space projects. Another goal was to foster the development of Estonian space and high-tech industry by training experts and disseminating knowledge about space technologies.

### EPS

The EPS was controlled through an **ATMega1280** 8 bit AVR microcontroller from Atmel (can be found in Arduino Mega 1280). The processor had been tested before and deemed suitable for conditions similar to the current mission.

### COMMAND AND DATA HANDLING SUBSYSTEM

The CDHS had to

1. store data from the CAM (Camera subsystem) in the form of multiple binary images of up to 600 KB each.

2. store housekeeping data of all subsystems.

3. compile the beacon and data packets for downlink.

The **CDHS** contained two **STM32F103** ARM processors, one of which was turned on. The two processors gave the possibility to activate the second one if the first was defective, to make sure that the satellite remained operational. The CDHS had three data interface types with other ESTCube-1 & -2 modules: SPI (Serial Peripheral Interface), I2C (Inter-Integrated Circuit), UART (Universal Asynchronous Receiver/Transmitter) which were organized into several logical buses. The FreeRTOS operating system was used to fulfil requirements of real-time operations, processing capacity and memory footprint.

### RF COMMUNICATIONS

Use of a VHF/UHF system for uplink (1.2 kbit/s) and downlink (9.6 kbit/s) data transmissions. Both uplink and downlink used AX.25 unnumbered information frames as a transport protocol. In addition, a CW beacon was used.

Quarter wave monopoles were used as antennas for both uplink and downlink. A power amplifier on the satellite provided up to 500 mW for data downlink, and a preamplifier provided 18 dB amplification for uplink. The transmit power of the CW beacon was 100 mW. Due to the shared RF chain the beacon and the primary downlink couldn't be transmitted simultaneously. The Texas Instruments **MSP430F2418** MCU was used on the COM.

### CAMERA PAYLOAD

 A robust independent camera module with on-board image processing, based on the **ARM Cortex-M3** microcontroller and fast static random access memory, had been developed and characterized for the requirements of the ESTCube-1 mission.

## GASPACS (Get Away Special Passive Attitude Control Satellite)

The **Get Away Special Passive Attitude Control Satellite (GASPACS)** was a 1U CubeSat technology demonstration mission designed to test inflatable structures in space.

For the first time ever, a Raspberry Pi was used as a satellite's on-board computer. GASPACS' Raspberry Pi Zero handles all of the satellite's computing, running on software written entirely by the Software Team.

 Attached to the **Raspberry Pi Zero W** is a **Raspberry Pi Camera**, which is used to capture an image of the deployed AeroBoom. The board also powers the burn wire system, which is used to deploy the AeroBoom. Also on the electrical board is a **DFRobot Beetle microcontroller** that acts as a **Watchdog,** ensuring that the Raspberry Pi is functioning nominally. The Pi sends a signal to the Watchdog at 0.25 Hz. If the Watchdog does not receive a signal for more than five seconds, the Watchdog will power cycle the Pi in hopes that the Pi will return to normality after reboot.

**Challenge**; Three days into the mission, GASPACS lost charging on the Y-axis solar panels. As a result of this, GASPACS became very power negative, meaning it used much more power than it could generate. This was reflected in GASPACS's ontime. GASPACS would typically be turned off and charging for around six hours, and then be on for about an hour.

## Swayam – College of Engineering, Pune (India)

**On Board Computer**

• **ARM7TDMI** based Microcontroller

• Foreground background interrupt driven system

• Responsible for data handling and housekeeping

• Custom network layer protocol ‘COEP Satellite Protocol’

• 2 GB On board SD Card Storage

## **ESTCube-2 – University of Tartu (Estonia)**

The EPS main board performs numerous functions related

to power management and is the core of the EPS. It is

responsible for power distribution, voltage conversion, and

monitoring different current, voltage, and temperature levels.

The system has a dedicated STMicroelectronics STM32L4

series microcontroller (MCU), external ferroelectric random-

access memory (FRAM) for storing data, external analog-to-

digital converters (ADCs) to perform analog measurements,

3.3 V housekeeping which is isolated from the rest of the

satellite, and a communication interface with other subsys-

tem

The EPS main board performs numerous functions related

to power management and is the core of the EPS. It is

responsible for power distribution, voltage conversion, and

monitoring different current, voltage, and temperature levels.

The system has a dedicated STMicroelectronics STM32L4

series microcontroller (MCU), external ferroelectric random-

access memory (FRAM) for storing data, external analog-to-

digital converters (ADCs) to perform analog measurements,

3.3 V housekeeping which is isolated from the rest of the

satellite, and a communication interface with other subsys-

tem

The EPS main board performs numerous functions related

to power management and is the core of the EPS. It is

responsible for power distribution, voltage conversion, and

monitoring different current, voltage, and temperature levels.

The system has a dedicated STMicroelectronics STM32L4

series microcontroller (MCU), external ferroelectric random-

access memory (FRAM) for storing data, external analog-to-

digital converters (ADCs) to perform analog measurements,

3.3 V housekeeping which is isolated from the rest of the

satellite, and a communication interface with other subsys-

tem

The EPS main board performs numerous functions related

to power management and is the core of the EPS. It is

responsible for power distribution, voltage conversion, and

monitoring different current, voltage, and temperature levels.

The system has a dedicated STMicroelectronics STM32L4

series microcontroller (MCU), external ferroelectric random-

access memory (FRAM) for storing data, external analog-to-

digital converters (ADCs) to perform analog measurements,

3.3 V housekeeping which is isolated from the rest of the

satellite, and a communication interface with other subsys-

tem

The **EPS** main board performs numerous functions related to power management and is the core of the EPS. It is responsible for power distribution, voltage conversion, and monitoring different current, voltage, and temperature levels. The system has a dedicated STMicroelectronics **STM32L4series** microcontroller (MCU), external ferroelectric random-access memory (FRAM) for storing data, external analog-to-digital converters (ADCs) to perform analog measurements,3.3 V housekeeping which is isolated from the rest of the satellite, and a communication interface with other subsystems.

The **primary communication (PCOM)** system is tuned at the radio amateur 70 cm band with a planned frequency of 435.8 MHz and 9600 to19200 baud variable data rate. Its processing unit is an STMi-croelectronics **STM32L4 series** MCU, and RF connectivity is achieved with a Silicon Labs Si4463 transceiver.

**SCOM** is primarily used as a secondary receiver subsystem and uses a radio amateur 2 m band. The data processing unit is a Silicon Labs **EZR32WG330 series MCU** with built-in RF transceiver module. For transmission, the subsystem can change its carrier frequency to the 70 cm band and use PCOM’s RF path to transmit the signal as a backup transmitter

The OBC handles the operations of

all subsystems and payloads, runs the AOCS algorithms and

stores housekeeping and telemetry data. The central com-

puter of the satellite is a STM32F7 series MCU. The MCU

features 512 kB of static random-access memory (SRAM)

and 2 MB ﬂash memory.

The **OBC** handles the operations of all subsystems and payloads, runs the AOCS algorithms and stores housekeeping and telemetry data. The central computer of the satellite is a **STM32F7** **series** MCU. The MCU features 512 kB of static random-access memory (SRAM)and 2 MB ﬂash memory.

There were issues with the STM MCU which are discussed in this document later in this document.

## E-st@r – Polytechnic University of Turin (Italy)

**OBC** (On-Board Computer): The OBC is based on an off-the-shelf processing unit developed by Pumpkin Inc. It consists of a microprocessor of Texas Instruments (**MSP430**, 16 bit), and works on the SALVO real-time operating system.

**Communication**; . A commercial transceiver (Radiometrix) is employed and integrated on the onboard shelf contained electronics, equipped also with a **PIC16** that accomplishes the modem function.

## AcubeSAT – Aristotle University of Thessaloniki (Greece)

The hosted subsystems are functionally isolated and feature an **ARM Cortex-M7**, radiation-tolerant microcontroller each.

## TJREVERB (Thomas Jefferson High School, USA)

The **Onboard microcontroller** used was a **Raspberry Pi Zero**.

## M-Cubed – University of Michigan (USA)

On-board control is provided by a Taskit Stamp9G20 [microcontroller](https://en.wikipedia.org/wiki/Microcontroller) running [RTLinux](https://en.wikipedia.org/wiki/RTLinux" \o "RTLinux).

| **Project** | **Subsystem** | **Microcontroller Used** | **Reasons** |
| --- | --- | --- | --- |
| **NaSPUoN** | On-Board Computer (OBC) | Raspberry Pi Zero (BCM2835, 1GHz, 512MB RAM) | Chosen over Pi 4 due to **size and power constraints**; suitable for camera processing |
|  | Camera / Payload | Raspberry Pi High Quality Camera | Connected via CSI to Pi Zero; higher resolution for imaging |
|  | Electrical Power System (EPS) | STM32 (specific model not given) | STM32 used for EPS—likely low-power series for energy efficiency |
| **GASPACS** | OBC | Raspberry Pi Zero | Simple Linux-based controller for student mission |
|  | Watchdog | ATmega32U4 (DFRobot Beetle) | Simple 8-bit MCU used for fault recovery and resets |
| **TJREVERB** | OBC | Raspberry Pi Zero | Used in student CubeSat for mission computing |
| **ESTCube-1** | CDHS | STM32F103 (Cortex-M3) | Dual-redundant STM32 for robust data handling with FreeRTOS |
|  | EPS | ATmega1280 | Power management using 8-bit AVR |
|  | Camera Payload | STM32F103 (Cortex-M3) | Used for interfacing with CMOS camera and data handling |
|  | COM | MSP430F2418 | Low-power controller for communication module |
| **ESTCube-2** | OBC | STM32F7 | Powerful ARM Cortex-M7 for central control |
|  | EPS | STM32L4 | Low-power microcontroller with advanced ADC features |
|  | SCOM | EZR32WG330 | MCU with built-in RF capabilities for simplified radio subsystem |
| **ArduSat** | Supervisor / Watchdog | ATmega2561 | Used for monitoring and fallback operations |
| **E-st@r** | COM (Modem) | PIC16 | Lightweight 8-bit MCU for simple modulation tasks |

Table 2. Summary of microcontrollers choices for different subsystems in Leo projects

# CHALLENGES

Some of the documented challenges with using different microcontrollers are listed below;

## RASPBERRY PI 4.

The NaSPUoN project had decided to use Raspberry Pi 4 but opted for **Raspberry Pi Zero** mainly due to **size restrictions** in the 1U CubeSat specification despite Raspberry Pi Zero being less computationally powerful and having less interfaces compared to the **Raspberry Pi 4.**

## STM32

The following were the limitations given from using STM32 MCU in the ESTCube-1 & -2 projects

The STM32 HAL is a high-level driver framework provided by STMicroelectronics to simplify hardware development across different STM32 microcontroller families. While useful for beginners and rapid development, it has several important limitations and potential pitfalls, especially in performance- and reliability-critical applications like satellite systems.

Below are key concerns explained clearly:

1. **No Programmatic Peripheral Access by Index**  
   The STM32 HAL does not support accessing peripherals (e.g., USART1, USART2, etc.) via index in a generic way.  
   Developers cannot write scalable or loop-based code to handle multiple peripherals dynamically. Each peripheral must be handled explicitly with its unique HAL handle (e.g., &huart1, &huart2).
2. **Model-Specific Handling is Manual**  
   The HAL does not completely abstract hardware differences across STM32 families (like F1, F3, F4, etc.).  
    Developers must account for family-specific quirks or register layouts manually, reducing code portability.
3. **Deadlocks in Interrupt Handlers**  
   Certain HAL functions (e.g., HAL\_UART\_Transmit) use blocking or polling behavior. If called within Interrupt Service Routines (ISRs), they may cause the system to hang.  
    Example: A HAL delay or wait in an ISR may never complete if interrupts are disabled or delayed, leading to deadlock.
4. **Incompatibility with Compiler Optimizations**  
   HAL code may rely on specific timing or memory access behavior. When compiler optimizations like -O2 or -O3 are used, this behavior can break due to reordering or inlining.  
   Developers may face unexpected bugs unless memory barriers or volatile variables are properly used.
5. **High Code and Memory Overhead**  
   The HAL generates a large amount of boilerplate and overhead code.  
   In resource-constrained systems, such as CubeSats or IoT devices, this can be problematic due to limited Flash/RAM.