Software Design of RHOK-SAT, a 1U CubeSat to Characterize Perovskites in Low Earth Orbit

Matar, Anas *, Pastrana, José R. †, Paul, Marouf Mohammad ‡, Wong, Zheng Yu §
Rhodes College Department of Physics, Memphis, TN, 38112
Rhodes College Department of Mathematics and Computer Science, Memphis, TN, 38112

Software is at the core of RHOK-SAT's functionality. The satellite's subsystems, mission experiment, and communications must operate autonomously and adaptively in low Earth orbit (LEO). Our top-level software design is constrained by a power budget, data budget, and hardware limitations. These constraints define our software requirements. The design's main components include command handling, payload measurement, data storage, and downlink scheduling. Our software runs on FreeRTOS using a cooperative scheduling scheme. This calls for a deterministic task execution pipeline to reduce the risk of undefined behaviors. Furthermore, we use statically allocated memory to prevent memory fragmentation over time. We are part of the SatNOGS open-access ground station network to increase data downlink opportunities. We are currently developing a second-stage bootloader on board to update our software from the ground.

I. Nomenclature

KiB, MiB, GiB	=	1024, 1024^2, 1024^3 bytes respectively
V_{OC}	=	open-circuit voltage
I_{SC}	=	short-circuit current

II. Introduction

RHOK-SAT is a 1U CubeSat collaborative project between Rhodes College and the University of Oklahoma's Photovoltaic Materials and Devices Group. The satellite is part of the 12^{th} round of NASA's CubeSat Launch Initiative (CSLI) and is planned for launch on an International Space Station (ISS) resupply mission in late 2023 or early 2024. The project's primary mission is to provide real-world engineering experience to students at Rhodes College, a liberal arts institution with no engineering program. RHOK-SAT's scientific mission is to characterize the performance and degradation of novel perovskite photovoltaic cells in low Earth orbit. The team at Rhodes College is responsible for designing the payload and top-level software of the satellite, while the team at the University of Oklahoma (OU) provides the experimental cells and analysis.

Flight software (FSW) is an umbrella term that refers to embedded software running on a spacecraft. Here, we use the term to describe the top-level software that drives the satellite's mission-specific operations, which in turn interfaces with subsystem firmware through the use of software libraries. It is our experience that software development receives less forethought than hardware development in the CubeSat field, partly because it requires a wider breadth of considerations to take into account [1]. These include both payload and subsystem requirements, using version control for effective collaboration, setting up the right debugging environment, and implementing continuous integration testing. Software plays a pivotal role in ensuring the reliable functioning of a satellite's hardware, proper execution of its payload experiments, and transmission of data back to the ground. The present paper focuses on the ongoing design of RHOK-SAT's FSW. We wish to provide an insight into the software development behind RHOK-SAT and help budding CubeSat teams consider its importance during their development process.

The paper will first outline the satellite's hardware and experiment. Their constraints and effects on software

*Undergraduate Student, Rhodes College Department of Physics, 2000 N Parkway, Memphis, TN 38112, AIAA Student Member 1421950.
†Faculty Advisor, Rhodes College Department of Physics, 2000 N Parkway, Memphis, TN 38112, AIAA Young Professional 773676.
‡Undergraduate Student, Rhodes College Department of Physics, 2000 N Parkway, Memphis, TN 38112, AIAA Student Member 1424490.
§Undergraduate Student, Rhodes College Department of Physics, 2000 N Parkway, Memphis, TN 38112, AIAA Student Member 1424492.

requirements are included next. Then, the FSW's code organization is discussed, along data collection and storage. Lastly, communications to and from our ground station are addressed.

III. Hardware Specification

Our 1U (10 cm × 10 cm × 11 cm) platform is provided by Innovative Solutions in Space (ISISpace) and contains the following subsystem boards: an on-board computer (iOBC) with a daughter-board (DB), an electrical power system (iEPS), a three-axis magnetorquer (iMTQ), a transceiver (TRXVU), antennas (AntS), and solar panels (iSPA). Our payload includes six perovskite solar cell samples, one control copper indium gallium selenide (CIGS) solar cell, eight dedicated measurement microcontrollers (called Aerospace Measurement Units (AMUs)), temperature sensors, and one sun sensor. Each perovskite sample contains six independent solar cells, termed pixels, and is connected to one AMU and multiplexer to allow for the measurement of each pixel. The CIGS is connected directly to an AMU. The temperature sensors are connected to each perovskite sample and the CIGS.

The iOBC runs on an ARM9-based AT91SAM9G20 microprocessor manufactured by Microchip Technology. It has four distinct memory spaces: a 1 MiB NOR flash, a 32 MiB SDRAM, a 256 KiB FRAM, and two 2 GiB SD cards. The iOBC also contains a real-time clock (RTC), a real-time timer (RTT), and an inertial measurement unit (IMU). Communication between subsystems is routed through an I^2C bus and the iOBC communicates with the DB via an SPI bus.

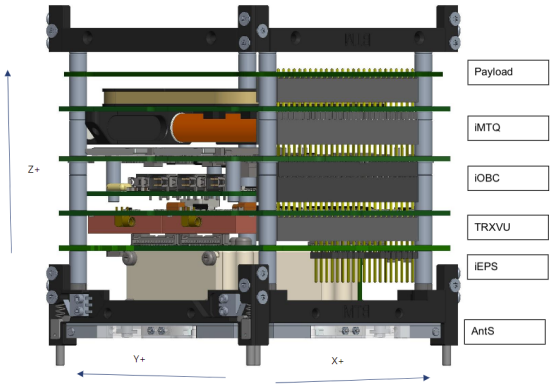

Fig. 1 Satellite subsystems

IV. Mission Description

RHOK-SAT's scientific mission is to characterize the performance and degradation of perovskite cells in orbit for a duration of nine to eighteen months. The instrumentation and hardware structure that enables this are detailed in [2]. Cell measurements consist of collecting voltage-current (IV) pairs from each cell's V_{OC} to I_{SC} by varying a resistive load. This process is called sweeping. Several prerequisites must be met before performing a sweep. First, the battery level must be sufficient to complete a full sweep. Second, the tumbling rate must be under 1°/s to keep a steady view at the Sun. Third, the incident angle of sunlight to the face of the cells must be within 35° for high quality measurements. Our software must meet these requirements in order to perform a satisfactory experiment and accomplish our scientific objective.

A. Cell Measurement Procedure

Counting each perovskite pixel and the CIGS, there are 37 solar cells to be measured. AMUs are connected to each one of these cells, and can perform simultaneous measurements of six pixels at a time. The AMUs also measure the temperature sensors and the sun sensor. The measurement procedure of the cells is as follows:

1) Sun angle measurement when the threshold is met
2) Temperature measurements of all cells
3) IV sweeps of each pixel group and the CIGS

4) Temperature measurements of all cells
5) Sun angle measurement

This measurement produces 42 IV sweeps, six per pixel group and six CIGS sweeps . The CIGS sweeps serve as relative measurements for the perovskite measurements, as the CIGS behavior at different temperatures is well-understood. By plotting the IV curves generated by the sweeps, information such as the efficiency and fill factor of the cell can be determined. Analyzing the data produced throughout the lifetime of the mission will allow us to assess the perovskites' degradation rate and behavior in space.

V. Hardware Constraints

Hardware constraints and scientific stipulations determine the main requirements of our software design. These constraints include the power budget, the data budget, the tumbling rate, the TRXVU's limitations, the payload instrumentation, and the different types of memory on board.

A. Power Budget Constraints

Our minimum operating voltage is 6.0 V. If the battery level drops under this threshold, the iEPS automatically disables all power lines, turning off all subsystems. Hence, a software requirement is to maintain the battery level above a specific voltage. To do so, power-expensive operations such as radio transmissions and detumbling are limited. If the specified voltage is crossed, necessary measures must be taken to save power until the battery is recharged so that the system is not suspended involuntarily.

B. Data Budget Constraints

As calculated in the data budget shown in Appendix A, which makes use of power and link budget information, a full cell measurement procedure will require 25420 bytes. Assuming a worst-day power generation rate, the system should be capable of performing 34 measurement procedures per orbit. Given that we use a worst-case assumption, this is the baseline requirement for the amount of data that the system should collect on a day-by-day basis. Further, assuming lossless transmissions, it should take 28 seconds to transmit these data. This is a loose downlink requirement since lossless transmissions are assumed, and packet loss is very common. However, the software can make use of techniques to reduce file size and make it more attainable. For example, data can be stored without identifiers since the location of each data point in the file is known and can be decoded on the ground.

C. TRXVU Constraints

The TRXVU buffer sizes constrain the amount of data that can be received and transmitted. Specifically, the input buffer is 200 bytes long and the output buffer is 235 bytes long. Hence, data must be structured such that they can be segmented to fit in the corresponding buffer.

D. Experimental Requirements

Although the satellite is expected to tumble in orbit, the iMTQ is assured by the vendor to reduce tumbling rate down to 1°/s overall. This is essential for the experiment to be conducted successfully, as the solar cells must be held at a constant illumination to be measured. This is a fundamental requirement, so the experiment is only performed when the condition is met.

Payload instrumentation includes the AMUs, temperature sensors, and a sun sensor. The measurement procedure is outlined in subsection IV.A. The only direct constraint for running a measurement procedure is meeting the necessary sun angle threshold. To meet this requirement, the software performs a simple periodic check.

E. Memory Constraints

The constraints of the memory types on board are as follows.

- NOR flash is a non-volatile memory space that stores our software images. Since it is only 1 MiB long, the default software image and any updated image must be small enough to fit in the NOR flash together. Additionally, NOR flash requires sectors to be erased before writing to them. This means that software updates, which are partitioned on the uplink, cannot be written to flash as they arrive, since the packets may reach the satellite out of order if

some are lost. Instead, they must be written to specific addresses or be reassembled in main memory and written at once.

- FRAM is a non-volatile memory space that stores critical variables and flags. These parameters are mapped to main memory upon system initialization, where they can be retrieved and modified as if in a database. The software design must account for data that should persist through reboots. An example is the most recent file number used to store experimental data, which, if reset, would cause new measurements to overwrite data after reboots.
- There are two commercial off-the-shelf SD cards on board that are not suited to withstand space radiation. One is a fallback for the other, and it is expected to fail mid-mission according to the vendor. Only one SD card is used at any time, since being used accelerates the damage. We use HCC Embedded's SafeFAT file system to store logs and experimental data. System events are retained through logs as a fault tolerance measure. Logging involves verbose traces throughout the FSW's execution for debugging purposes in case of errors. Radio transmissions are costly on the power budget, so experimental data should be formatted in a compact format to occupy as little storage as possible.

VI. Software Decisions

A. Cooperative Multitasking

Our FSW runs on FreeRTOS. FreeRTOS makes use of prioritized tasks to organize sequences of instructions. Our single-threaded processor uses a cooperative scheduler, which requires tasks to explicitly yield for others to execute. This greatly simplifies the FSW's flow of execution, which is shown in Fig. 2. If a task does not delay or suspend itself, it will run indefinitely and starve the other tasks. Thus, the flow of execution must be carefully planned to make it impossible to reach a state where a task does not eventually yield. If this were to happen, the iOBC's watchdog reboots the satellite.

B. Static Memory

We made the decision to avoid the use of dynamic memory allocation whenever possible. This is to prevent design mistakes that can lead to long-term memory fragmentation and memory leaks, which are common bugs in long-running embedded systems that utilize dynamic memory [3].

C. Software Updates

We are currently working on the implementation of a second-stage bootloader capable of updating the FSW from the ground in case of unforeseen error or behavior. Having this flexibility will not only increase the fault tolerance and adaptability of the design, but it will also extend the possibilities of the mission once its objective is attained. This decision has impacted our design by constraining software images to be under half the size of NOR flash. Likewise, the ability to receive partitioned software binaries, reassemble them, and verify them has been implemented. Next steps include the partial writing of updates to NOR flash to better utilize resources if an update takes multiple transfers to uplink, as well as writing the updates to, and booting from, an SD card.

D. Dependencies

In order to interface with the subsystems and AMUs, we must make use of a set of hardware abstraction layers and software libraries. The software must follow specific application programming interfaces (API) to adequately communicate with these hardware components.

VII. Code Flow and Organization

A. Deployment Mode

Upon launch from the deployment pod on the ISS, the satellite's battery-inhibit switches are released and the satellite turns on. In its first-ever boot, the satellite enters Deployment Mode, which initiates a countdown timer for 30 minutes. As per ISS protocol, the satellite is not allowed to perform any actions during this period. Once the timer ends, antenna

deployment is initiated, and the status of each antenna is continuously checked until they are properly deployed. A flag is then set in FRAM to signal that Deployment Mode has completed, ensuring it does not run again on subsequent boots. The FSW then reboots and begins nominal operations as outlined below.

B. Initialization

Immediately after boot, system components are initialized. These include the cache for the ARM9 core, the watchdog, the FreeRTOS scheduler, and the `INIT` task. `INIT` proceeds to initialize serial communication buses, FRAM, the SD card file system, the subsystems, peripherals, our payload, and the rest of our tasks. Events that happen in these preliminary processes before the file system is initialized are temporarily held on the stack until they can be written to the SD card. Finally, `INIT` begins the `NOM` task and deletes itself.

C. Nominal Loop

`NOM` is the main loop of the system, from where all other tasks are periodically resumed. It queries the iEPS at regular intervals to determine battery level. If a specified voltage threshold is crossed, `NOM` stops non-critical system operations to save power until the battery recharges. This avoids reaching the minimal operating voltage, at which forced system shutdowns occur. Additionally, `NOM` calculates tumbling rate using the IMU on board. If the satellite is tumbling faster than 1°/s, and there is enough power to do so, `NOM` activates the detumbling procedure of the iMTQ, decelerating the satellite and bringing it within an acceptable range where the experiment can be performed.

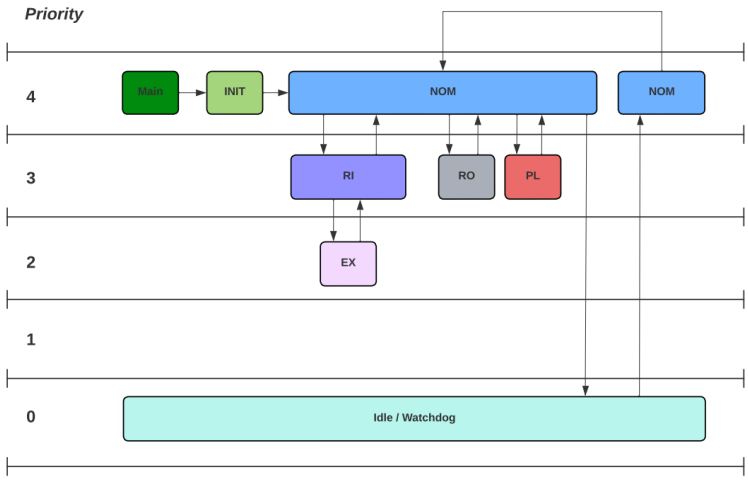

Fig. 2 Nominal code flow

D. Applications

Once the periodic nominal checks are completed, the main tasks of the system, termed applications, can begin. These include the procedures to handle incoming and outgoing communications and carrying out the experiment. For the tasks to run, `NOM` suspends itself and awakes the routines in the following order:

- Radio In Task (`RI`)

  `RI` handles any transmissions received by the satellite's transceiver. The task is responsible for extracting and parsing the data, which are encapsulated in AX.25 frames. It first checks a general purpose input/output (GPIO) pin associated with the receiver, which indicates whether data packets have been received, and extracts any into a struct. Character tags in the struct header indicate whether the received data are a command or file transfer. If it is a command, `RI` enqueues the instruction for execution. If the data are part of a file transfer, they are reassembled for a software update. `RI` then resumes `NOM` and suspends itself.

- Executor Task (EX)

  EX is resumed by RI. It uses a FreeRTOS queue with a limited length to receive commands from RI. Each command is a set of bytes with a predefined format that identifies its function. The first byte holds the subsystem ID. The second byte corresponds to the specific function to be performed. The third is the number of arguments. The subsequent bytes represent the length of each argument. The subsystem and function IDs are used to locate the desired API call. Upon processing all enqueued commands, EX resumes RI and suspends itself.

- Radio Out Task (RO)

  RO handles all transmissions made by the satellite. It uses a FreeRTOS queue to hold outgoing data. When data are dequeued, they are separated based on their type, whether experimental measurements or current satellite telemetry, and transmitted accordingly. Transmission operation is detailed in subsection VIII.B. There is a limit to the data RO can transmit every time it is resumed, as the power drain generated by transmissions may be significant. Once all data have been downlinked, RO resumes NOM and suspends itself. Experimental data on board is not deleted unless explicitly commanded to do so by the ground, as not all data may have been received.

- Payload Task (PL)

  PL handles the measurement procedure of the payload, as outlined in subsection IV.A. This task is only resumed if the satellite is tumbling under 1°/s and there is sufficient power to conduct a set of measurements. PL monitors the sun angle regularly. When the angle threshold is met, it instructs the AMUs to perform a full measurement procedure. The AMUs simultaneously gather data on each set of perovskite pixels as well as on temperature sensors and the CIGS. PL then collects the data points and writes them with a predefined format to a file. If the sun sensor threshold is not crossed for a specific amount of time, PL suspends and resumes NOM.

VIII. Data Transmission

The communication procedures between our ground station (GS) and the satellite make use of supplied hardware and software. The GS hardware consists of a very high frequency (VHF) and ultra high frequency (UHF) transceiver. The software utilizes the AX.25 and KISS protocols. AX.25 defines data frames with a cyclic redundancy check (CRC), causing corrupt frames to be dropped automatically. Since it is common for frames to get lost, we must validate their reception during critical data transmissions to the satellite as discussed below.

A. Ground Station to Satellite Communications

We transmit two types of data from the GS to the satellite: commands and software updates.

In order to implement resiliency, we constructed a command system that can index every subsystem API call on board from the ground. The commands are formatted as shown in Fig 3. Upon reception on board, EX handles the commands and executes them as specified in subsection VII.D. To simplify command transmission, we deploy a web server that handles command formatting and argument population automatically. This makes the process efficient and helps avoid human error.

The second type of data are software updates. This functionality is still under development. If our deployed software were to exhibit a critical bug, an update command is issued. This command informs the satellite that subsequent transmissions correspond to compiled software instead of commands. The update command contains the CRC of the software image and the number of frames that the GS will send. The software binary is split into frames and uplinked by our GS. Each frame is prepended with a file type indicator and a sequential ID number. If frames are missing upon reception, a bitmap of the missing packet IDs is sent down to the GS to re-request them. Once the satellite determines that all frames were received, it orders them according to their ID numbers and joins them back into binary form. It then calculates a CRC and compares it to the CRC received on the update command. If the CRC values match, the software image is written to NOR flash.

B. Satellite to Ground Station Communications

We predict the satellite will complete 16 orbits around the Earth in a day. However, only three of these orbits are expected to be good-quality passes over our GS. These windows last approximately 600 seconds. Based on our power

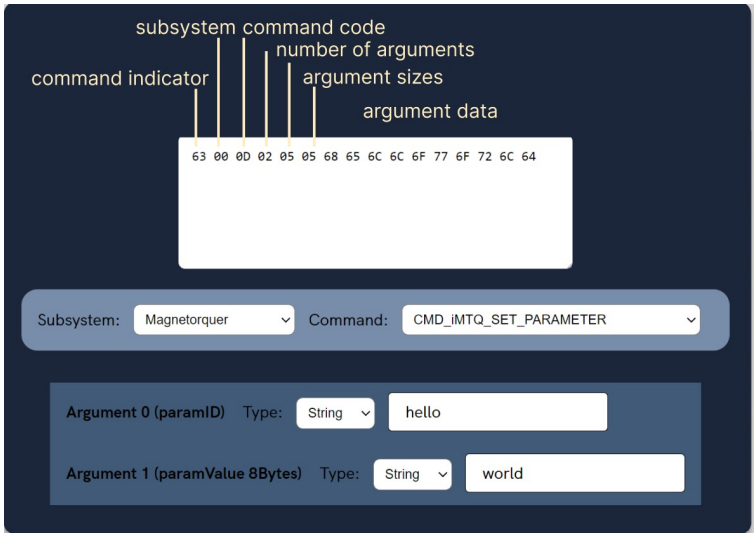

Fig. 3 Example command constructed using our web application

budget we expect the satellite to take 34 sweeps per orbit. The satellite can downlink up to 28 sweeps per pass as shown in Appendix A. To maximize the amount of downlinked data we joined the SatNOGS network, a free and open-access network of satellite ground stations. It increases our area of reception by allowing us to use any participating GS in the SatNOGS network. Our SatNOGS GS consists of an omnidirectional antenna that can be accessed by any SatNOGS member.

Two types of data are downlinked by the satellite: telemetry and files such as sweeps and logs.

Satellite telemetry is designed with the TRXVU's 235-byte limit in mind. We select essential subsystem telemetry to track the functionality of the satellite. We include data from the following subsystems in our telemetry beacons: iSPA, iEPS, AntS, and iMTQ. The satellite will transmit these beacons every 10 minutes. Telemetry constructs a view of the nominal state of the satellite over time, which creates a baseline of acceptable behavior. Anomalies detected may indicate a problem with the satellite's subsystems, and can help us identify and address issues before they cause significant problems. Because of our downlink data constraint, keys that explain the meaning of each value are not included in the beacons. We will parse these beacons using KaiTai on the ground. KaiTai is an API that maps raw bits to a human-readable data structure according to a predefined format. Afterwards, the telemetry is decoded into JSON objects that are uploaded to a SatNOGS dashboard known as Grafana for open-access visualization.

The downlink frame size restriction shapes our approach to transmitting files. We segment files on board into separate frames with ID numbers. The first frame includes the total number of frames to be transmitted and the CRC of the whole file. The GS validates the reception of all frames before sending an acknowledgement message with any missing IDs. The GS combines the frames to rebuild the file in a fashion similar to the procedure outlined in subsection VIII.A.

The satellite does not have systems on board to determine its position over Earth and downlink files automatically. Thus, a set of timers is kept on board counting down to the next pass over a GS. We calculate these timers when two-line elements (TLEs) are updated. We then uplink the timers from our GS on every pass. When the timers expire on board, the system is interrupted and `NOM` determines whether a transmission should be made based on the current battery level. If the system reboots before the timers expire, they are deleted to avoid useless transmissions.

C. Satellite as a Transponder

Our communications will utilize the ham radio bands. To give back to the amateur radio community, our satellite will function as a transponder on the weekends. In Transponder Mode, the system transmits back any audio signal that reaches the satellite's receiver. The transmitter is initiated when the received signal strength indicator (RSSI) exceeds a predetermined threshold, currently -77 dBm. When the RSSI falls below the threshold, the transmitter turns off after 3 seconds.

IX. Next Steps

Currently, the FSW is being tested to prepare for full integration with the payload. This entails unit, integration, and end-to-end tests. Certain features such as software updates and enhanced IV sweeps are still under development. We plan on extending the sweeping procedure by adding a reverse sweep from I_{SC} back to V_{OC} after the initial sweep to increase accuracy. We are also considering peak-power-tracking one pixel in each perovskite sample to increase the type of data we collect. Furthermore, measures for fault tolerance and resiliency must continue to be implemented. Lastly, all requirements must be validated before launch.

Appendix A: Data Budget

Datapoint size (bytes)	4
Data categories	3
Sweep datapoints	50
Solar cells swept	42
Metadata (bytes)	200
Temperature measurements	2
Sun angle measurements	2
Timestamp	1
Data per sweep (bytes)	**25420**

Data rate (bps)	9600
Sweep downlink time (s)	**21.18**
Orbit time (s)	5578
Sweep transmission duty cycle	0.38%
Transmission power (mW)	3300
Power to transmit one sweep (mW)	12.53
Available power per orbit (mW)	430
Possible sweeps per orbit	**34.31**
Pass window (s)	600
Possible sweeps transmitted per pass	**28.32**

Acknowledgments

RHOK-SAT is funded through a generous gift from Dr. Charles Robertson, NASA's CSLI program, the Mac Armour Fellowship, the Gene Haas Foundation, and the Society of Physics Students Chapter Research Award.

RHOK-SAT is advised by Dr. Bentley Burnham, Dr. Brent Hoffmeister, Dr. Ann Viano, Mr. Lanre Obadina, Dr. Phillip Kirlin, and Dr. Marion Lang, all of Rhodes College; Mr. Colin Mann from The Aerospace Corporation; Rhodes College alumnus Mr. Brad Hensley; and former project director Mr. Joseph McPherson.

The team is grateful for the support of the Photovoltaic Materials and Devices Group at the University of Oklahoma; Dr. Ian R. Sellers, Dr. Brandon K. Durant, Hadi Afshari, Sergio A. Chacon, and Megh N. Khanal. Dr. Bibhudutta Rout conducts solar cell irradiation at the University of North Texas.

The perovskite solar cells were fabricated and provided by Giles Eperon from Swift Solar in Colorado.

The CIGS solar cell material was provided by Dr. Dmitry Poplavskyy from MiaSolé Hi-Tech Corp. and processed by Hadi Afshari.

The AMUs were designed and provided by Colin J. Mann from The Aerospace Corporation in El Segundo, California.

The satellite structure and subsystems were provided by ISISpace in Delft, Netherlands.

The team is thankful to the SatNOGS community for their support, especially Kevin Croissant from Ohio University's Bobcat-1 for his communications expertise.

Other student contributors to the project include Jess Hamer, Olivia Kaufmann, Benjamin Wilson, and Dang Nguyen. Past contributors include Ryan Jones, Mohammed Hyder, and Hannah Cartier.

References

[1] Dr. John M. Bellardo, "Software: The Overlooked Glue that Holds CubeSats Together," `https://www.nasa.gov/sites/default/files/atoms/files/cubesat_software.pdf`, 2020. Accessed: 2023-02-27.

[2] Hamer, J. G., Kaufmann, O. A., Pastrana, J. R., and Wilson, B. T., "Payload Design of RHOK-SAT, a 1U CubeSat to Characterize Perovskites in Low Earth Orbit," , 2023. Pending review for publication in the AIAA's 2023 Region II Student Conference.

[3] Holzmann, G., "The power of 10: rules for developing safety-critical code," *Computer*, Vol. 39, No. 6, 2006, pp. 95–99. https://doi.org/10.1109/MC.2006.212.