ON-BOARD SOFTWARE DEVELOPMENT FOR A 3U CUBESAT

Emirhan Eser Gül[1], Boğaç Karabulut[1], Kaan Sarıca[1], Alim Rüstem Aslan[1]
[1] *Space Systems Design and Testing Laboratory, Istanbul Technical University, Istanbul, Turkey*
emirhaneser@hotmail.com, aslanr@itu.edu.tr, bogackarabulut@gmail.com, src.kaan@gmail.com

SharjahSat-1 is a 3U CubeSat developed by Istanbul Technical University (ITU) and Sharjah Center for Astronomy & Space Sciences (SCAASS). The satellite incorporates two payloads; an X-Ray detector and an optical camera system. A robust on-board software to control the payloads is critical for the mission in order to ensure the satellite performs its required tasks correctly and effectively. This paper imparts the development and implementation processes of the software architecture used in the On-Board Computer (OBC) of the nanosatellite.

This paper will concisely explain the determination of the software requirements, architecture of the on-board software itself, the complications encountered during development, and the test and verification of the implementation. The flight software is developed to efficiently use both payloads and ensure smooth operation of the mission.

Key Words: Mission Software, Cubesat, Microcontrollers, Real-Time Operating Systems

1. Overview

The developed software that controls all subsystems, including the payloads, runs on the OBC of the satellite. A Heterogeneous SoC-based OBC is used for ample performance and high-reliability. The mission software is developed using FreeRTOS embedded Real-Time Operating System for multi-threading functionality. Microsemi MSS HAL libraries and ARM CMSIS library are used for hardware abstraction of the peripherals and the ARM Cortex-M3 CPU, respectively.

The OBC performs various critical tasks on the satellite such as controlling accesses to peripherals, switching subsystems on or off for power efficiency, logging system parameters, processing telecommands and most importantly, commanding the payloads. Both the hardware and the software of the OBC must be as reliable as possible, as there is no redundant OBC and maintenance is not possible on-orbit. While a bootloader application can be developed to allow on-orbit software upgrades, it introduces complexity and requires additional code storage space. Since there wasn't enough time to finalize and test the software, this functionality was not implemented on Sharjahsat-1, but is recommended especially for projects with limited time.

The Space Systems Design and Testing Laboratory has designed and launched 6 CubeSats, and has plans for more in the future. Hence, one of the main goals of the software development was to achieve strong re-usability. The software uses multiple abstraction layers in order to reduce the necessity of developing base libraries from scratch and to provide a basis to be used in the future satellite missions. Similarly, a modular ground station software that can be re-used was also developed in accordance with on-board software utilizing an efficient space-ground data exchange protocol. SharjahSat-1 utilizes different operating modes in the mission, such as start-up, normal, safe, recovery; which are switched by a state machine. These various modes incorporate different mission critical parameters as well as attitude control specifications based on the status of other subsystems, especially battery.

1.1. SharjahSat-1 CubeSat

The main objectives of SharjahSat-1 can be inspected in two categories.

1. iXRD Payload (Primary Payload): An X-Ray detector to:
   - Extend knowledge on miniaturized X-Ray detectors and their capabilities to contribute to Space Weather research.
   - Detect hard X-rays from very bright galactic X-ray sources such as very bright black hole and neutron stars.
   - Observe and study the bright sun flares and development of solar coronal holes, responsible for driving the stellar wind at an early phase.
2. Optical Payload (Secondary Payload): A camera system with a GSD of at least 50 meter/pixel to:
   - Acquire the images of regions on Earth from low orbit.
   - Take photos of SAASST and its surroundings.

In order to achieve these goals, in addition to the OBC, the satellite is equipped with the following electronic subsystems which are interfaced to each other via PC-104 bus.

- Electrical Power System (EPS): to generate, regulate, and distribute the power to other subsystems as required.
- Battery: to store the generated power.
- Attitude Determination and Control System (ADCS): to detumble and point the satellite towards the requested location.
- Ultra-High Frequency (UHF) and Very-High Frequency (VHF) Modem: to uplink commands

(VHF) and downlink telemetry & housekeeping data (UHF). The VHF band is used for uplink with AFSK modulation at 1200bps, and the UHF band is used for downlink at 9600bps with GMSK modulation.
- S-Band Modem: to download the large payload files at 2Mbps.
- Interface Board: a custom board developed by SSDTL to deploy antennas, and accommodate the beacon and star tracker. A back-up RTC is also incorporated.

In terms of OBC, the Clyde Space CubeSat OBC is used as the main house-keeping computer for the mission, Figure 1.

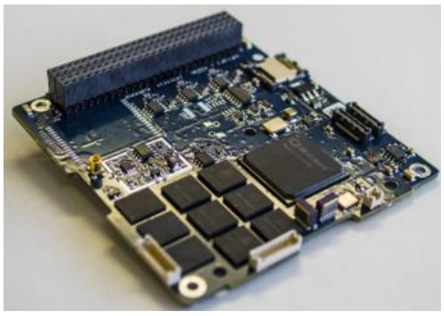
Fig. 1. ClydeSpace CubeSat OBC

Clyde Space CubeSat OBC uses a system-on-chip (SoC) device that includes a configurable FPGA fabric and an ARM Cortex-M3 processor. It is designed to be highly reliable and resistant to radiation effects, with features such as error-detection and correction (EDAC), safe state machines, and triple modular redundancy (TMR) registers implemented in the configurable logic. The OBC has a range of memory options, including embedded non-volatile storage (eNVM), embedded static random-access memory (eSRAM), magneto-resistive RAM (MRAM), and flash memory, all of which are interfaced with an EDAC to protect against radiation effects.

2. Software Requirements

2.1. Non-Functional Requirements

McConnell states that managing complexity is the most critical technical matter in software development [1], so emphasize was given to this topic.

According to a flight software report by NASA [2], the software gets complicated due to excessively strict requirements and a lack of consideration for testability can complicate verification; thus, the requirements weren't imposed for the sake of deciding matters but according to real constraints which results in a software that can be thoroughly tested.

The two most important requirements of the software were deemed as reliability and reusability. Since the software cannot be maintained after launch, a reliable software is a must have to ensure smooth operation during the flight. In order to assure reliability, three requirements were derived. All code should be documented, version control for the repository should be used, and software architecture should support debug tools.

It was also decided that there was a limit on the used memory and throughput. Since a RTOS is used, there are lots of interrupt routines which introduce a level of uncertainty in throughout. Thus, 30% of the available on-chip non-volatile memory and 30% of throughput is reserved for unforeseen expenses [3].

Another critical aspect of reliability is the selection of hardware, as it is known that 128 kB of Ram can experience up to 10 errors per day in orbit, so protection against single event effects (SEE) is essential [4]. The requirement "The OBC must be able to recover from transient errors such as SEL and SEU, no matter where or when they might occur." was considered during the selection of OBC.

Reusability is desired because most of the time only the payloads differ in CubeSat systems. Hence, the logic that operates the satellite and subsystems was separated from the one that operates payloads. The code was written in different modules that interact with each other and abstraction was used to achieve portability.

The same report also states that operations get complicated due to ill-conceived autonomy, and it was decided that the satellite would be operated manually with autonomy as a back-up option. This is further discussed in operation modes below.

2.2. Functional Requirements

These requirements were established from the operations that the satellite is expected to perform. The main goal is to operate the payloads and transmit the generated data to the ground station. In order to achieve this, several other critical requirements were derived.

- Development Requirements

The capability to attach the debugger and program the OBC during any stage of the development process is needed. An umbilical connector was designed and attached to the top panel of the satellite to both power the satellite externally and connect the debugger.

A debug mode has to be included to check the program flow and fix bugs and errors. One of the UART peripherals of the OBC was included in the umbilical connector for this purpose. When the DEBUG macro is defined in the software, statements required for debugging would be printed via this line and displayed on the PC.

- LEOP Requirements

During the LEOP phase, there are several tasks to be performed. After waiting for at least 30 minutes after P-POD ejection due to CubeSat standard regulations [5] and to allow the battery to be charged, the satellite should immediately attempt to deploy the antennas so that communication can be established. The ADCS will be commended in this stage for detumbling and stabilizing the satellite.

- Communication Requirements

The CW beacon should transmit critical information in regular time intervals. Housekeeping data of the subsystems should also be ready to transmit via UHF modem upon request.

- Telemetry and Telecommand Requirements

SharjahSat-1 should be able to receive, verify, and process telecommands issued from the ground station and respond with the status of execution. Telemetry for subsystems should be collected periodically and stored on the non-volatile space to download upon request.

- Payload Operation

Both payloads require pointing towards a location, so the ADCS must be used at least 15 minutes before the desired time to orient the satellite towards the point of interest (i.e., location on Earth for photos and direction of a star for XRD observations.) The exact time to perform the operation, and parameters of the operation such as the resolution and format of images should be provided from the ground and be processes on-board accurately. Large files such as raw images and iXRD measurements should be compressed to reduce storage size and time to download. The satellite should return to the orientation it had pre-operation after completion. In addition, a metadata file should be created in every operation to record the parameters used as well as circumambient information such as the temperature of the board.

- Power and Optimization Requirements

The satellite should incorporate different operation modes according to the state it's in. If battery is low to perform the desired operation, it should consume less power until the voltage crosses a certain threshold.

- Configurability Requirements

There are some critical system parameters that the satellite uses during its operation, such as the power levels of the transmitters, the intervals which the beacon is used and telemetry data is stored, and the battery voltage threshold levels to enter/exit safe mode. Although preliminary analyses were performed to simulate the power generation and consumption in-orbit, the real values remain unknown. Hence, all these parameters should be configurable from the ground station.

3. Software Architecture & Design

A CubeSat system has to control real-world entities such as switches, sensors, and reaction wheels that should be operated in real-time. Reflecting the parallel nature of the system in the structures of the program makes for a more maintainable, reliable, and readable application [6]. Therefore, a concurrent program that minimizes dependencies between modules would be an appropriate design for SharjahSat-1. In order to choose a real-time operating system (RTOS), several factors were considered. First, the board support package for the OBC was written in C language, which is a good choice for embedded systems programming as it allows direct hardware access and can be used to develop efficient programs [7]. Second, developing a custom RTOS from scratch would add a great deal of complexity [4]. In the end, considering the flight heritage (ESTCube, AAUSAT, GOMX-1) as well as the experience of the software team, FreeRTOS was chosen as the operating system of SharjahSat-1. Even though C does not support concurrency or has shared resource management, FreeRTOS is a robust library that uses preempting, mutexes and semaphores to enable the development of a real-time application. Hence, the driver for each subsystem makes use of mutexes to prevent concurrent access to peripherals. Inter-task communication is achieved by task notifications and global variables to retain simplicity and not over-complicate the system. In order to satisfy the debugging and version control requirements, an eclipse-based IDE provided by MSS was used along with an open-source version control system.

In real-time applications where guaranteed interaction response times are required, a master-slave architecture is recommended [8]. Therefore, communication between the subsystems is achieved via I2C bus. The OBC is used as the master and all other subsystems are slaves that are operated by commands from the OBC. Similar to other hardware peripherals, a mutex is used to lock the I2C bus as long as it's in use by a subsystem, so the bus is guaranteed to be used sequentially.

In order to achieve portability and re-usability, abstraction for all peripherals (I2C, SPI, UART, PDMA) were done. Bare-metal drivers for each individual subsystem were developed so that only the abstract peripheral functions can be replaced according to the hardware (e.g., I2C read/write) and the whole driver can be used in a different system. Separating both the OS and hardware layers also help with increasing maintainability.

In terms of power optimization, , the idle task hook was enabled in the RTOS, which is used to put the microcontroller into standby mode. However, it was observed that the time and energy consumed in order to enter and exit the low power state for every tick outweighs potential power saving gains unless FreeRTOS tickles idle mode is used, which increases complexity. Thus, power optimization was only achieved by turning off payloads and some subsystems until they are required.

3.1. Operation Modes

There are 8 fundamental operation modes of SharjahSat-1 on the highest level, implemented using 5 different tasks. Once the satellite boots, it checks the non-volatile memory space called "registry" to determine the latest mode to switch. The boot process is detailed in Figure 2.

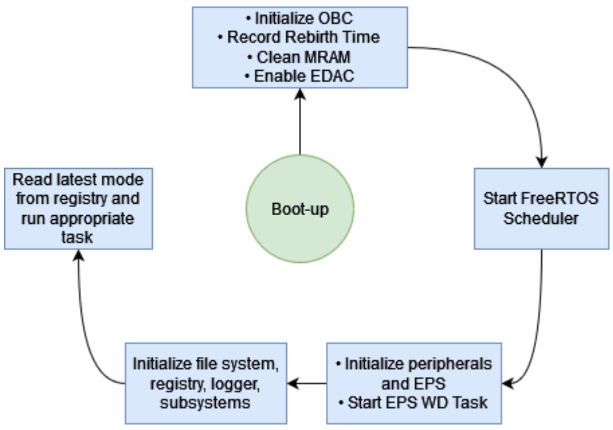
• Initialize OBC
• Record Rebirth Time
• Clean MRAM
• Enable EDAC
Boot-up
Start FreeRTOS Scheduler
• Initialize peripherals and EPS
• Start EPS WD Task
Initialize file system, registry, logger, subsystems
Read latest mode from registry and run appropriate task

Fig. 2. SharjahSat-1 Boot Sequence.

The details of the operation modes are as follows.

1. LEOP Mode

This is the first operation mode the satellite will use after launch. It records the time of first boot on the registry, then waits for 90 minutes. If the battery is depleted or the satellite is reset due to any other reason during this phase, the boot time is read from the registry to prevent unnecessary waiting. Afterwards, the antennas are deployed and the ADCS begins calibration and operating to detumble the satellite. This operation is expected to take a couple of days, and the ADCS can be commanded manually from the ground station to ensure everything goes as intended. The LEOP mode ends when a command to end it is received from the ground, or when a timeout of 3 weeks is reached. Then, the task is deleted and this mode will never be used again.

2. Nominal (Nadir-pointing) Mode

This is the default mode that the satellite will use. The ADCS is used to achieve nadir pointing. Beacon is operated and telemetry data is transmitted in regular intervals. Three periodic checks are performed in this mode.

- The registry is checked to see if a mission is scheduled from the ground station. If there's one in 15 minutes, the payload or transmit task is notified and the nominal task is suspended.
- The battery voltage is read to check if it fell under a configurable threshold. If that's the case, switch to safe mode.
- The time of last command received from the ground is read to check if a pre-determined time (3 weeks) have passed without a TC. In this case, it is assumed to be an issue with the communication link, and the autonomous mode is activated.

3. Sun-pointing Mode

Essentially the same as the Nominal mode, with the only difference being the ADCS pointing the satellite towards the sun for increased power generation. Whether the satellite should be in nadir-pointing or sun-pointing mode by default is set via a TC.

4. Safe Mode

Used when the battery voltage falls below a threshold that can be configured with a TC. The telemetry storage and beacon transmission intervals are greater in this mode, and no payload operation can be performed. The battery voltage is checked periodically and the last pointing mode is restored when the value exceeds another configurable threshold.

5. Camera Operation Mode

Used when there's 15 minutes or less remaining to an observation mission. The ADCS is used to track the target coordinates on Earth. When the time is reached, photos are taken and the last pointing mode is restored.

6. iXRD Operation Mode

Similar to Camera Operation Mode, but the ADCS tracks a given X-Ray resource and the iXRD is operated instead.

7. Transmit Mode

Used when a transmit operation is scheduled from the ground. The ADCS points the satellite towards the coordinates of the ground station. The beacon and telemetry logging are stopped. All of the payload data stored on memory that has not been sent before are transmitted using the S-Band once the target time is reached.

8. Autonomous Mode

Automatically switched if the satellite cannot be accessed by uplink for 3 weeks. In this mode, both payloads are operated daily, and then the obtained data is transmitted in regular intervals for one day. The telemetry logging and beacon operation continue periodically.

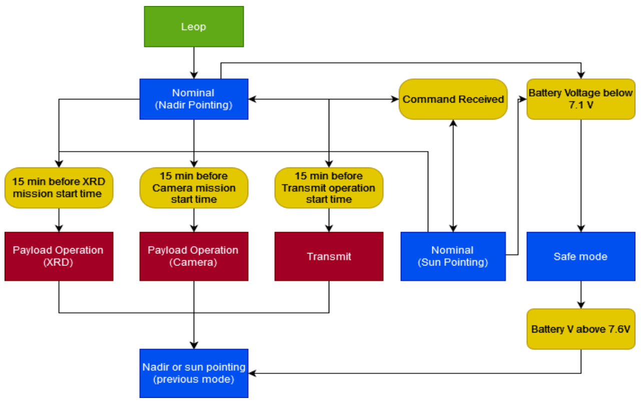
Leop
Nominal (Nadir Pointing)
Command Received
Battery Voltage below 7.1 V
15 min before XRD mission start time
15 min before Camera mission start time
15 min before Transmit operation start time
Payload Operation (XRD)
Payload Operation (Camera)
Transmit
Nominal (Sun Pointing)
Safe mode
Battery V above 7.6V
Nadir or sun pointing (previous mode)

Fig. 3. Operation Mode Relationships.

3.2. Tasks

The software functionalities and operation modes are performed in independent FreeRTOS tasks in order to allow a modular design, and because a task can tap into all the OS resources such as semaphores and timers [9].

- Watchdog Task

Periodically kicks OBC and EPS watchdogs. Has highest priority.

- Garbage Collector Task

Performs background file system garbage collection. Has lowest priority.

- Telemetry Task

Collects subsystem telemetry and stores on the file system periodically for housekeeping. The time interval is dependent on the current operation mode, and is stored on the registry. The collected housekeeping data is unformatted and the processing is done in the ground station to save memory, which results in a total size of 486 bytes saved on each execution. A new file is created every day so housekeeping data can be requested for time intervals. Has medium priority.

- Beacon Task

Operates beacon, then suspends itself until another task resumes it. The time interval is dependent on the current operation mode, and is stored on the registry along with beacon frequency and power level. Has highest priority.

- Command Handler Task

Checks whether there is new data available in the UHF/VHF modem. If there is, increases priority to maximum and determines if the received data has the correct command structure, then executes the received telecommand, sends acknowledge packet, and returns to original priority. If a mission is scheduled, writes its parameters to registry. Has medium priority.

- LEOP Task

This task runs only in LEOP mode. In first boot of the OBC, records the birth time, then waits until 90 minutes has passed. Then deploys antennas and starts ADCS LEOP flow. It terminates upon receiving command from the GS, or when a timeout of 3 weeks has passed. Has highest priority.

- Nominal Task

This task handles the nadir-pointing, sun-pointing, and safe modes. When it starts or resumes, checks the current operation mode and commands ADCS to either point to nadir or towards the Sun. Then, performs the necessary checks to switch to another operation mode. Has highest priority.

- Payload Operation Task

If there is a scheduled mission, the Nominal Task sends a notification to this task before suspending itself. Upon receiving the notification with the ID of the scheduled mission as parameter, commands ADCS to point to desired location and performs the camera or iXRD mission. After the mission is finished, resumes the Nominal Task with the latest mode (nadir or sun pointing). Has highest priority.

- Transmit Task

Similar to Payload Operation Task, switched from the Nominal Task via a notification. Commands ADCS to point to the target ground station, then transmits all the payload data obtained since last execution before resuming the Nominal Task. Has highest priority.

- Autonomous Task

Spawned or resumed from the Nominal Task if the last received telecommand is 3 weeks old. Handles the Autonomous Mode as described above. If a command is received, immediately resumes Nominal Task and suspends itself. Has highest priority.

The execution of tasks is constrained by the current operation mode. Telemetry Task does not run in Transmit Mode, and has a lower frequency in Safe and Camera/iXRD Operation modes. Beacon Task does not run in Transmit and Camera/iXRD Operation modes, and has a lower frequency in Safe Mode.

4. Implementation & Verification

4.1. Memory Management

The SoC has three storage components, an 8MB non-volatile Magnetoresistive random-access memory (MRAM), a 256 KB eNVM, and a 4 GB nand-flash memory. The eNVM is used to store code and global variables.

The variables that require fast access and robustness are stored in a portion of the MRAM reserved in the linker file, called “registry”. Most of the time, inter-task communication is also achieved using this region, so that the eNVM uses less space.

The remaining 7MB of the MRAM is used for the FreeRTOS heap. Payload and housekeeping data are stored on the file system that lives on the NAND-flash of the OBC. This memory includes hardware EDAC protection. Commands to download, upload, and delete files from the file system were implemented in the command handler, as well as commands to get list of files in a directory and format the file system. A command was implemented to delete all data on the file system older than specified amount of days.

When configuring FreeRTOS memory management, “heap4” was chosen for efficiency and to be able to directly use the portable layer memory allocation schemes. However, two important considerations were observed. The used C library implementation was newlib-nano, which internally uses default memory allocation routines. This would cause random hard faults to occur as it conflicts with the FreeRTOS memory allocation scheme. In order to circumvent this, three steps were taken. First, reentrancy was enabled in FreeRTOS configuration, which increased the memory usage considerably. Second, a memory management API written on top of the "C" standard (using newlib) was used [10]. Lastly, the malloc family functions were wrapped with functions using the FreeRTOS APIs. The third step is redundant but implemented as a back-up solution regardless.

4.2. Registry

The critical system parameters are stored on registry and for reliability these parameters have a checksum also in registry. On startup, the registry is read and the system parameters are read into a global struct. If the checksum is invalid, default parameters are used. These parameters can be changed via ground station commands, and the global struct is also immediately updated. A section of the registry table of the satellite looks like Table 1.

Table 1. Satellite Registry

Bytes [0:3]	Description
BIRTH_TIME	First time (1970 epoch) system boots after launch
DEATH_TIME	Last time satellite shuts down
REBIRTH_TIME	Time system reboots after last shutdown
DEATH_CAUSE	Cause for the last shutdown (undervoltage, hard fault, stack overflow)
LAST_MODE	Last operation mode
SYSTEM_PARAMS	Configurable system parameters. EPS watchdog time, UHF/VHF frequencies, modem power levels, beacon operation periods, telemetry storage periods.

4.3. File System

Since the largest non-volatile bulk storage memory on the OBC is a nand-flash, the filesystem was chosen as YAFFS due to its wide-usage, clear documentation, and flight heritage. YAFFS was configured to be concordant with the nand-flash by adjusting the block and page sizes.

A software ECC algorithm was not implemented as the memory is already protected via EDAC mechanism and the cost of using such an algorithm was found to be not worth the time and energy it consumes.

4.4 Telecommands

All commands sent to the satellite must be in a specific structure that includes a “magic number” that acts as a key to confirm that the command was really sent from the operator, a command id to identify the command, parameters to be sent along with the command, and the total length of the command packet. A state machine in Command Handler Task checks the received command and executes the appropriate function from the look-up table.

4.5. Telemetry

Similar to telecommands, all telemetry packets conform to a specific structure as given in Table 2. These packets are encoded in an AX.25 frame.

Table 2. Telemetry Packet Structure

Magic No (4 bytes)	TM ID (1 byte)	Data Length (1 byte)	Packet Counter (4 bytes)	Data (246 bytes)
Key to confirm the packet belongs to SharjahSat-1	Identifier of the packet.	Size of data section.	# of packet, counted from high to low.	Actual telemetry data.

4.6. Event Logging

To save space, only the errors are logged in a log file on filesystem. These may include subsystem and peripheral failures, and command processing errors such as unrecognized command format etc.

4.7. Beacon

Since the beacon uses a considerable amount of power, it is in our best interest to reduce the amount of characters transmitted to a minimum. The most critical values to be transmitted by the beacon were chosen and shown in Table 3.

Table 3. Beacon Data

Data	Description
Operation Mode & Health Status	Binary array of 16 bits in HEX format. Each bit represents the status of a subsystem, with the 4 MS bits representing the operation mode.
VBAT	Battery Voltage Level
IBAT	Battery Current Level
TBRD	Battery Board Temperature
IPCM3V3	Output Current level of 3.3V Bus
IPCM5V	Output Current level of 5V bus

Instead of transmitting the actual values, one of the 26 letters were assigned to represent an interval. For example, the letter "P" represents a voltage level between 7.6 and 7.7 V.

For the encoding of operation mode and health status, a different approach was used. In Morse Code, numbers always have 5 characters, and letters have a maximum of 4. Also, letters in hexadecimal go only to F, which means the rest of the alphabet is free. Thus, if we assign a letter for each number, we can reduce the number of characters transmitted even further. There are 10 such letters in Morse code (except for A, B, C, D, E, F) that are maximum 3 characters long. In this way, each number can be represented in up to 3 characters instead of 5. For example, the hexadecimal character of "1" is mapped to the letter "T". This way, the number of Morse Code characters the beacon transmits (excluding the callsign) can vary between 9 to 36 characters.

4.8. Tests

Most of the bugs and functional errors were fixed on-the-go as the software was being developed, as it is considered a better approach than waiting for the final code for such a large project. After the implementation was completed, numerous tests were performed to verify the integrity and performance of the software. Some of the test items are given in Table 4. The actual list consists of around 100 items.

Table 4. Software Test Items

Test	Expected Output
Demand telemetry packets from all subsystems for a determined time interval. Check time tag of the received data.	The received telemetry packet should be belong to the predetermined time interval.
Safe mode transition from other modes based on voltage level.	Operation mode should be changed when battery voltage decreases below 7.2V, and returns to previous mode when it increases to 7.6V.
Test all telecommands on-board	Command handler should execute the correct command and the function should give an expected output
Hard reset satellite sometime after start-up.	Satellite should detect for how long it was powered down, and deduce it from the amount of time it should wait.
Receive beacon message from ground station	Beacon should operate in configured time intervals and should transmit correct data.
Schedule multiple camera and iXRD missions from the GS.	The missions should start at the desired time and use the correct parameters.

5. Conclusion

This paper described the development and testing processes and the architecture of the on-board flight software of SharjahSat-1. The whole development process took about 2 and a half years, and more than 31,000 lines of code written on top of the OBC drivers, FreeRTOS, and file system source codes. A great deal of time and effort was spent on the reusability to lay the groundwork for future missions, and to make the system as flexible as possible to allow on-the-fly changes while ensuring the robustness of the software, as well as eliminating bugs and faults at every step. The satellite will be launched in the fourth quarter of 2022 and the reliability of the software will be proven then.

Acknowledgments

We would like to express our sincere appreciation to all members of the Space Systems Design and Testing Laboratory for their invaluable contributions to this project. Special thanks go to Eng. Aybüke Ağırbaş, MSc. Egemen Çatal, and Eng. Onur Öztekin for their dedicated support during the testing and verification process. We also extend our gratitude to ITUNOVA for generously providing the necessary resources and accommodations for the project.

References

1. McConnell, S.: Code Complete, Microsoft Press 2nd Edition, 2004, p. 159.
2. West, A.: NASA Study on Flight Software Complexity, NASA,

2009, pp. 1-3.
3. Larson, W.J., Wertz J. R.: Space Mission Analysis and Design, 3rd Edition, Microcosm Press, 2005, pp. 645-685.
4. Kiær, C.E., Arnesen, M.H.: Mission Event Planning & Error Recovery for CubeSat Applications, Institutt for elektronikk og telekommunikasjon, 2014.
5. The CubeSat Program: CubeSat Design Specification Rev. 13., California Polytechnic State University, 2014.
6. Burns, A., Wellings. A.: Real-Time Systems and Programming Languages 4th Edition, Pearson Education Limited, 2009.
7. Sommervile, I.: Software Engineering, International Computer Science Series. ed: Addison Wesley, 2004.
8. Koopman, P.: Better Embedded System Software, Drumnadrochit Education, 2010.
9. Askari, H.A., Eugene, E.W.H., Nikicio, A.N., Hiang, G.C., Sha, L. Choo, L.H.: Software Development for Galassia CubeSat – Design, Implementation and In-Orbit Validation, 2017.
10. Nadler & Associates: newlib and FreeRTOS, url: https://nadler.com/embedded/newlibAndFreeRTOS.html