OPTIMISATION OF CONTINUOUS DEVELOPMENT FROM FLATSAT TO 3D CUBESAT – A NOVEL OVER-THE-AIR SOLUTION

Ng Yu Heng[1], Rebecca Sim Zhi Ning[2], Lim Hui Min[3]
[1]Hwa Chong Institution (High School), 661 Bukit Timah Rd, Singapore 269734
[2]Raffles Institution (Junior College), 1 Raffles Institution Ln, Singapore 575954
[3]DSO National Laboratories, 12 Science Park Dr, Singapore 118225

1. Background and Research Purpose

Cube Satellites (CubeSats) are preferred for their low cost and short development cycles compared to large-scale satellites, and especially popular for Educational Institutions for research purposes. However, industry established development and practices can be improved for time efficiency.

This report validates the use of Over-The-Air (OTA) reprogramming and debugging to eliminate the need to deconstruct working hardware, thus eliminating points of failure, and significantly decreasing development time while keeping costs low.

2. Literature Review

To develop a CubeSat, a rigorous industry-standardised Assembly, Integration and Verification (AIV) process must be completed. This constitutes the development of various models of the CubeSat (thermal, engineering, qualification, flight models), which are put through tests that simulate the harsh space environment (thermal vacuum, vibration, acoustics tests).

Given that CubeSats can be as small as the form factor of 1U (10cm x 10cm x 10cm), the compact design renders testing of subsystems challenging once they are assembled into 3D form factors. Troubleshooting is especially an issue as there is little to no space for accessing components. The complexity of reassembling numerous miniature components of the mechanical, thermal structures, payload and subsystems may push CubeSat developers to overlook subsystem-level testing due to the difficulty and time wasted disintegrating and reintegrating the satellite [1]. This leads to either consequence: even more time is spent disintegrating the fully assembled CubeSat when issues are surfaced during tests; or mission failure. This presents an opportunity to optimise the CubeSat development process.

This study hypothesises that the integration of an ESP32-C3 development board for wireless debugging allows a more optimized workflow during CubeSat development with minimal hardware alterations.

3. Methods and Results

3.1. System Overview

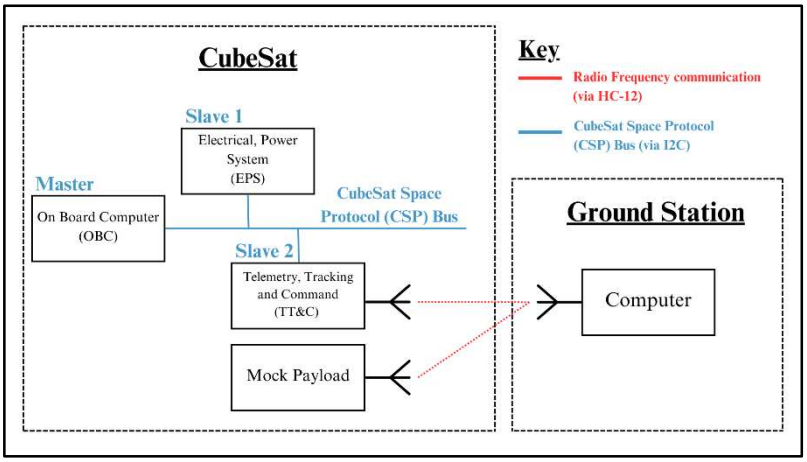

Figure 1. System block diagram

The system consists of the CubeSat and ground station. The CubeSat comprises four subsystems that will be detailed in the following section.

The ground station is connected to a computer and Graphical User Interface (GUI) for easy monitoring of telemetry and sending of telecommands.

3.2. Bill of Materials

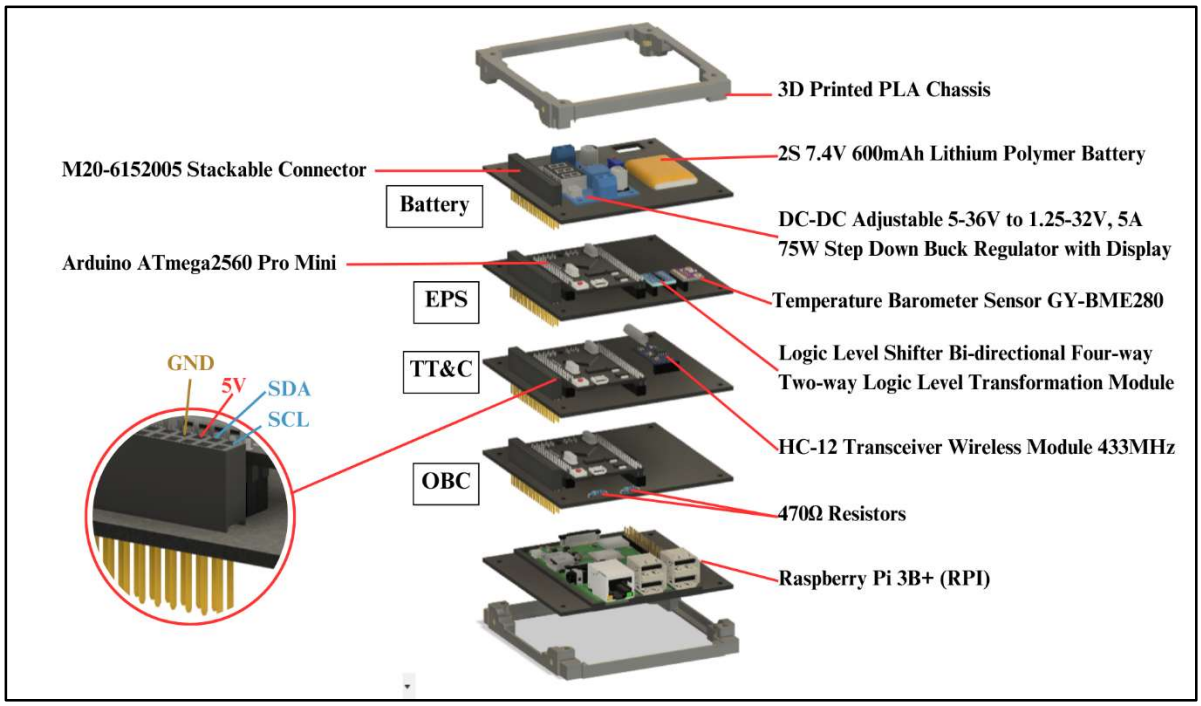

Figure 2. Bill of Materials on 3D Model of CubeSat
Note: Each subsystem is controlled by the Arduino ATmega2560 Pro development boards.

Battery: A 2S 7.4V Lithium Polymer (LiPo) battery is stepped down to 5V to power the CubeSat.

Electrical Power System (EPS): The GY-BME280 sensor measures the temperature of the LiPo battery and communicates this data to the On-Board Computer (OBC). A Logic Level Converter is used to step down the 5V supply to the 3.3V required by the GY-BME280.

Telemetry, Tracking, and Command (TT&C): The TT&C system employs an HC-12 module operating on the 433MHz band to handle communication between the CubeSat and the Ground Station for both downlink and uplink.

On-Board Computer (OBC): A CubeSat Space Protocol (CSP) bus was developed to enable subsystems to relay information via Inter-Integrated Circuit (I^2C) communication. Two 470Ω pull-up resistors are used to connect the Serial Clock Line (SCL) and Serial Data Line (SDA) to the 5V input. The SCL and SDA are common pins in the M20-6152005 connector for all subsystems to access.

Mock Payload: A Raspberry Pi (RPI) is used to store raw open-source data from hobbyist Coffee Can Radar sets [2]. A HC-12 module, tuned to 473MHz, is connected to the payload to ensure it does not interfere with the communications between the TT&C subsystem and Ground Station.

3.3. Development Process

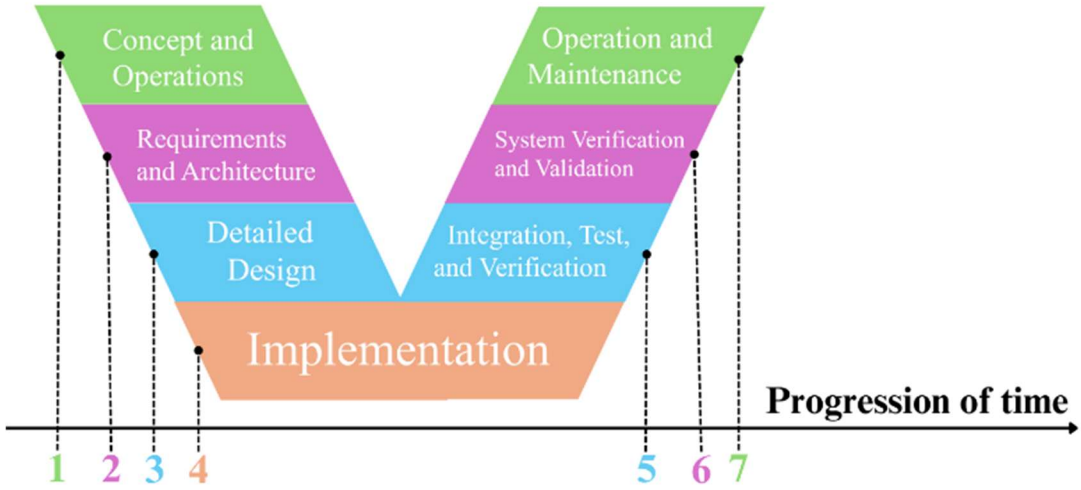

Figure 3. V Model for testing and development phases

The CubeSat was developed in 7 phases (Figure 1.) using wired debugging. Beyond stage 7, the ESP32-C3 microcontroller was integrated to compare between wired and wireless debugging and understand risks arising from constant alteration of hardware during testing. A proof-of-concept CubeSat system will be demonstrated using ESP32 to show the potential of wireless debugging.

1 System overview was established, and the project scope was discussed.

2 A mission mode diagram was created to define subsystem functions and code flow.

3 The Printed Circuit Boards (PCBs) were designed on EasyEDA and manufactured with JLCPCB. A 3D model was created using Fusion360 to plan the arrangement of individual

components. The Software design is based on the mission mode diagram in Phase 2 and was written in C, using the Standard C Library for AVR GCC (avr-libc).

4 Creation of Qualification Model (QM) 1 consisting of three Arduino ATmega2560 pro boards. Commercial Off the Shelf (COTS) parts were tested with preliminary code to ensure functionality.

5 Creation of QM2 consisting of three Arduino ATmega2560 pro mini boards mounted on the PCBs. This QM2 is known as the Flat Sat model, where we lay out each assembled PCB subsystem to perform functional tests.

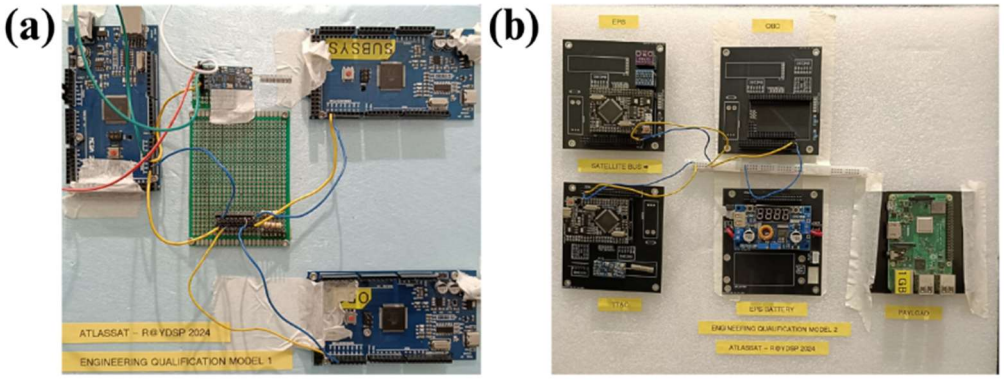

Figure 4. (a) Qualification Model 1 and (b) Qualification Model 2

6 Creation of QM3 by assembling PCB layers into 3D CubeSat. A 3D printed PLA chassis was attached. The full system software verified in phase 5 was used.

In phase 6, it was observed that numerous cables were needed. Like real-world CubeSats, the compact assembled model leaves minimal space between layers, causing the attaching of upload cables to be confusing as layers cannot be discerned directly. The arrangement of components is also limited by how close they are to one another.

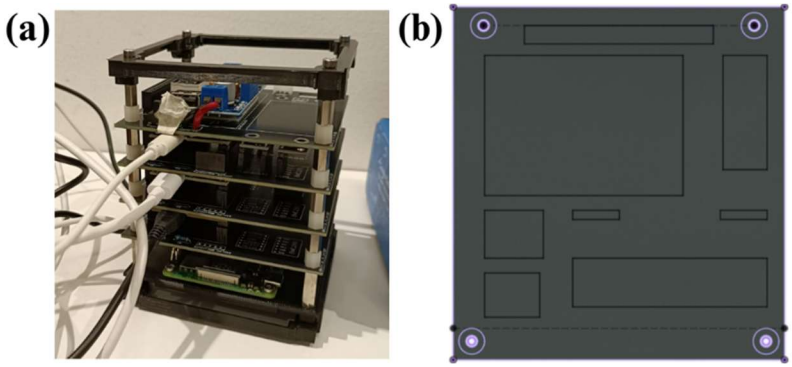

Figure 5. (a) Messy wires on QM3 and (b) Component arrangement in 3D model

7 The CubeSat was then powered by USB and left to run for five hours to stress test the system. A graph of the system current and temperature was plotted and monitored to ensure reliable performance. After which, the CubeSat powered by the LiPo battery for testing.

3.4. Ground Station

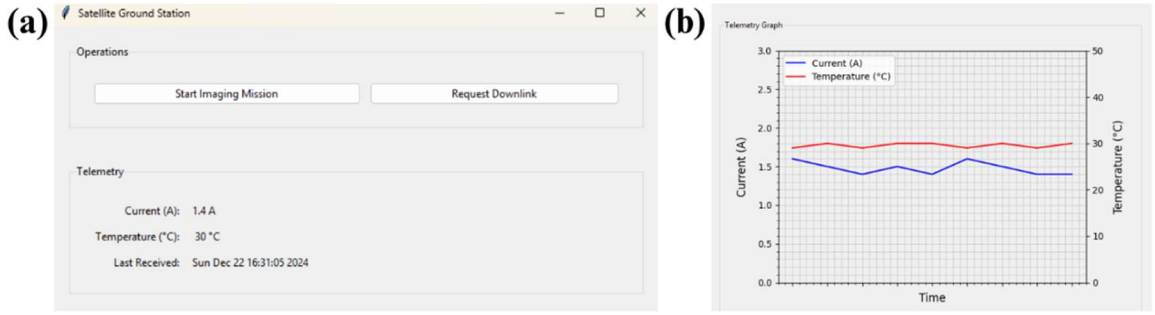

Figure 6. (a) Telemetry, telecommands and (b) Visual display for Ground Station GUI

Two HC-12s tuned to 433MHz and 473MHz allow for sending of telecommands/receiving telemetry and Payload data downlink respectively. The GUI provides a user-friendly way for telecommands to be sent to the CubeSat and for telemetry to be downlinked on demand. There is also a real-time visual display of telemetry such that the operations of the CubeSat can be monitored easily.

3.5. FreeRTOS

FreeRTOS is a lightweight Real-Time Operating System (RTOS) designed for embedded systems. By using FreeRTOS, functionality can be split into multiple independent tasks which can enable multitasking in an otherwise single threaded while loop. FreeRTOS also provides the ability to block or suspend tasks [3].

3.6. CubeSat Space Protocol (CSP)

CSP is an open-source lightweight protocol stack designed for communication between distributed Central Processing Units (CPUs) in memory-limited embedded systems. The protocol, following upon the TCP/IP model, contains a transport and routing protocol, as well as MAC interfaces [4].

CSP is structured to mimic the TCP/IP protocol stack and supports multiple communication interfaces such as I^2C [5], hence integrating neatly into the existing I^2C rail. The diagram below shows the CSP stack used in this project.

Layer 4	Unreliable Datagram Protocol		
Layer 3	Buffer Handling	Socket/Connection Table	Routing Table
Layer 2	I^2C Frame		
Layer 1	I^2C Driver		

Table 1. CubeSat Space Protocol Stack

Driver Layer (Layer 1): An interrupt driven I^2C driver was implemented on QMs 1 - 3, driving better stability and performance by removing the need to continuously poll for I^2C events [4].

Application Layer (Layer 5): By leveraging the transport layer (Layer 4) implemented in CSP, the OBC can send commands to various systems and services through sending requests to the correct addresses and ports [6]. This enables modularity in design, allowing subsystems and services to be added or removed without affecting the functionality of other subsystems.

The modular approach of services allows for seamless integration when adding new peripherals to the existing satellite bus. A new port just needs to be assigned to the new service.

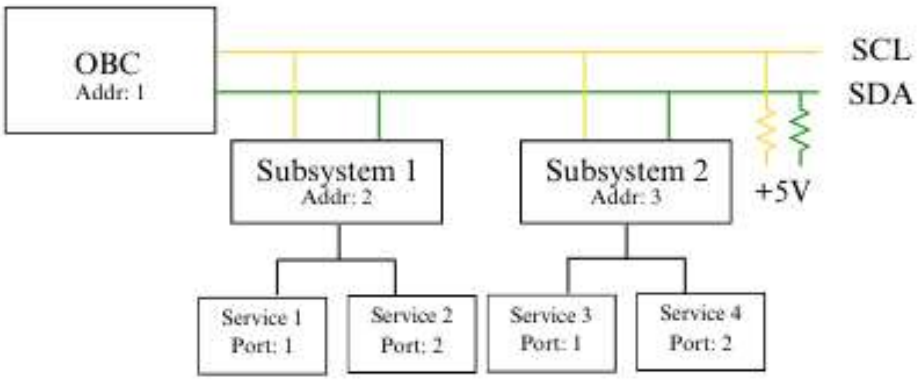

Figure 7. Example of addresses and ports in the satellite bus

3.7. Consultative Committee for Space Data Systems (CCSDS)

The Consultative Committee for Space Data Systems (CCSDS) is an international organization developing standards for spaceflight communication and data systems used by NASA, ESA, JAXA and other space organisations [7], [8].

CCSDS, being used by multiple international space agencies, provides a footprint for future interoperability with different ground stations. As data communication in space can be susceptible to bit-flips, the integrity of the message exchange is preserved by using a XOR checksum.

Packet Header (48 bits)							Packet Data Field (Variable)		
Packet ID				Packet Sequence Control		Packet Length	Data Field Header	Data	Packet Error Control
Version Number	Type	Data Field Header Type	APID	Sequence/ Groupings Flag	Sequence Count				
3	1	1	11	2	14				
16				16		16	24/64	Variable	16

Table 2. Telecommand/Telemetry Packet Structure

The CCSDS packet, sent via the programmed ground station, contains in it an Application ID (APID) as defined in the CCSDS Protocol [9]. The APID, when delivered to the OBC, is then processed into an address and port according to the CSP network topology. The payload from the

original CCSDS packet is then delivered via the I^{2}C bus, to the payload, where the payload can use it for further processing (if needed).

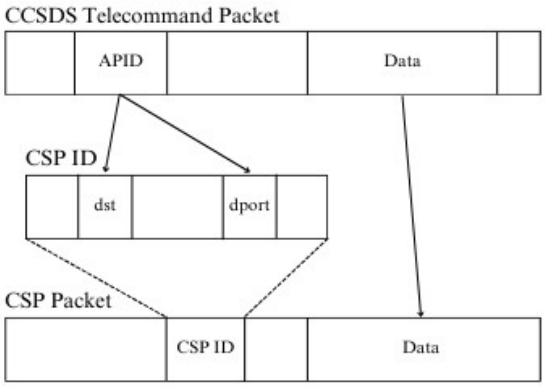

Figure 8. Creating CSP Packets from received CCSDS telecommand packets.

3.8. Program Flow

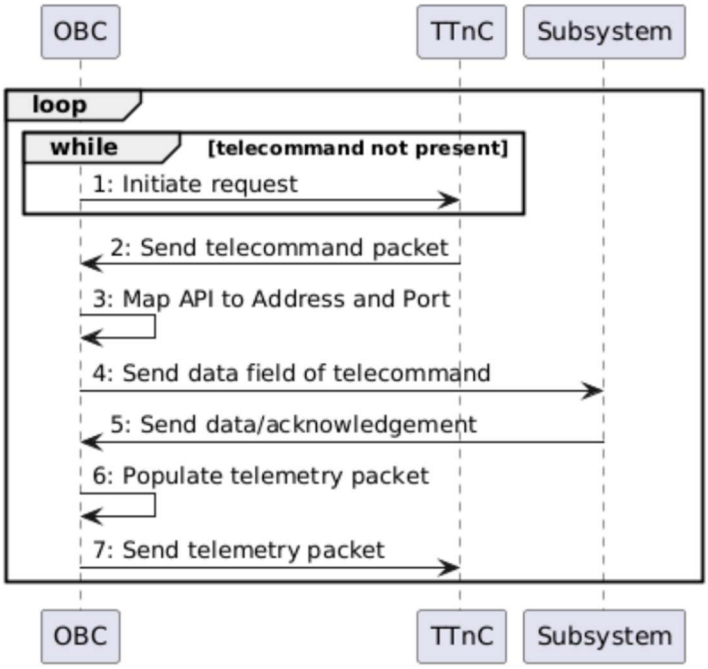

Figure 9. UML Sequence Diagram of OBC, TT&C, and Payload/EPS Interactions

On Board Computer (OBC): The OBC initiates an I^{2}C transaction to read from the Telemetry, Tracking and Command (TT&C) module. When data is available, the OBC receives from the TT&C a telecommand packet. The APID field of the packet is then mapped to the appropriate address and port. The OBC then sends the packet's data field to the determined address and port and waits for a response. The response is then sent to TT&C for downlink in a telemetry packet.

Telemetry, Tracking and Command (TT&C): The TT&C module executes three "concurrent" tasks: receive_from_ground(), send_to_ground(), and scheduled_telemetry().

receive_from_ground(): The TT&C module receives telecommands from the ground station using a UART driver. It calculates the checksum of the incoming telecommand, and checks against the checksum in the telecommand checksum field. It then waits for the OBC to request for the telecommand, where it responds with the telecommand.

send_to_ground(): The TT&C module waits for telemetry packets from the OBC. Upon receiving it, it transmits the packet to the ground.

scheduled_telemetry(): The TT&C module waits for a specified amount of time, before requesting the OBC for telemetry, which downlinks it via send_to_ground().

Electrical, Power System (EPS): The EPS module listens for data from the OBC. It checks the port the data was sent via, before calling the relevant function with the data as the argument.

3.9. Integration of ESP32

The diagram below shows how Over-The-Air transmissions are handled on the ESP32-C3-DevKitM-1 board. The ESP32 boards were used for an OBC and TT&C subsystem, where the CSP bus was verified for I2C communication between the two subsystems. OTA was verified by sending 'ping' to ground station and uploading second piece of code to transmit 'pong'.

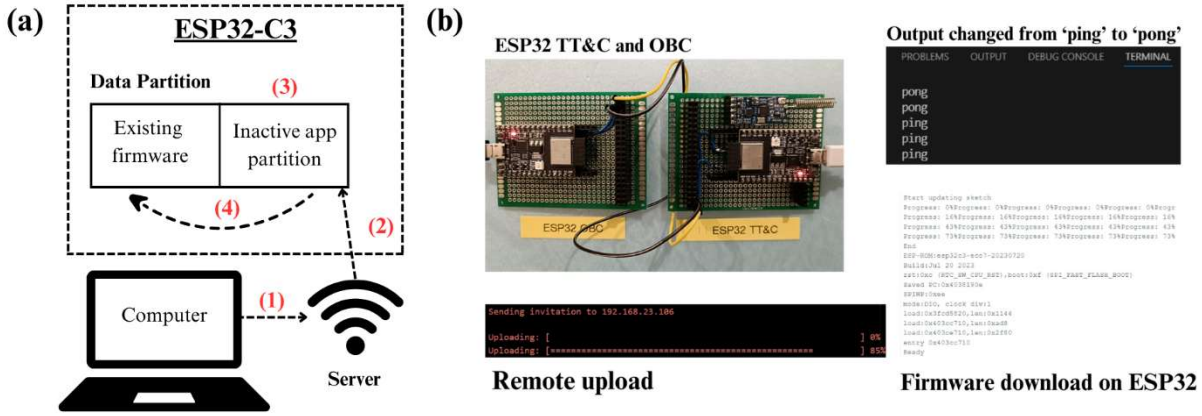

Figure 10. Diagrams of (a1) Uplink to server, (a2) send to ESP32, (a3) Firmware download and (a4) partition change; (b) ESP32 Setup and output screenshots

The ESP32 boards were flashed with code that transmits 'ping' to ground station. For the remote upload computer to access the ESP32 via OTA, both devices were connected to the Local Network.

To verify that the OTA was successful, a new piece of code was uploaded to the ESP32. Recompiled code on ESP32 transmits 'pong' to the ground station. On both the remote computer and the ESP32, the terminal indicated the progress of code upload. Upon successful download of the code, output from the ground station terminal changed from 'ping' to 'pong'.

4. Conclusion

Wireless debugging is possible through integrating the ESP32 into the CSP bus. Without having to reconnect wires during debugging, the development workflow is more optimized and error-free, compared the wired-debugging scenario.

4.1. Implications and Applications

As 3D CubeSat assembly and verification can be complex, the ESP32 chip will allow for a less error-prone debugging environment as it removes the need for uploading cables to be connected to problematic components and disassembling of any mechanical components in Flat Sat or the 3D assembled CubeSat. Thus, hardware remains untouched and working, while staff members are kept safe.

For efficient use of space on the CubeSat, the ESP32 chip can be embedded on the PCB. Wireless debugging can be similarly applied in industrial manufacturing of complex systems or in clean room environments, where handling of fragile hardware should be avoided.

4.2. Future Recommendations

The OTA solution can be adapted for post-launch usage as well. The verification of the chip's functionality in space fully justifies its integration into CubeSats designs as code updates can be performed on the same chip pre- and post-launch. Alternatively, a Field-Programmable Gate Array (FPGA) board can simulate OTA capabilities via logic gates and lookup tables, allowing existing CubeSats to make use of existing FPGA boards to wirelessly debug, without integrating a new component. Using the same parts for remote software updates and debugging on land and in space allows for CubeSat development to be further optimised.

4.3. Code Availability

Source code for all subsystems is available on GitHub, with DOIs assigned by Zenodo.

OBC: https://github.com/atlas-sat/obc (DOI: 10.5281/zenodo.14543597)

EPS: https://github.com/atlas-sat/eps (DOI: 10.5281/zenodo.14543595)

TTNC: https://github.com/atlas-sat/ttnc (DOI: 10.5281/zenodo.14543599)

Ground Station: https://github.com/atlas-sat/ground-station (DOI: 10.5281/zenodo.14543803)

The source code is currently not licensed under any specific terms.

4.4. Acknowledgement

We would like to express our deepest appreciation to Mr. Desmond Choong of DSO National Laboratories' Playground Lab, for his guidance during our prototyping of the CubeSat. Also, we would like to thank Ms. Emeline Sim and Ms. Joyce Yao from DSO National Laboratories' Talent Outreach team, for facilitating the project administration.

References

[1] C. Venturini, B. Braun, D. Hinkley, The Aerospace Corporation, and Greg Berg, “Improving Mission Success of CubeSats,” conference-proceeding. [Online]. Available: https://s3vi.ndc.nasa.gov/ssri-kb/static/resources/Improving%20Mission%20Success%20of%20CubeSats.pdf?utm_source=chatgpt.com

[2] “MIT OpenCourseWare,” MIT OpenCourseWare. https://ocw.mit.edu/courses/res-ll-003-build-a-small-radar-system-capable-of-sensing-range-doppler-and-synthetic-aperture-radar-imaging-january-iap-2011/

[3] "RTOS Fundamentals." FreeRTOS™. Accessed: Dec. 21, 2024. [Online]. Available: https://www.freertos.org/Documentation/01-FreeRTOS-quick-start/01-Beginners-guide/01-RTOS-fundamentals.

[4] "The Protocol Stack." Lib CSP. Accessed: Dec. 21, 2024. [Online]. Available: https://libcsp.github.io/libcsp/protocolstack.html.

[5] "History." Lib CSP. Accessed: Dec. 21, 2024. [Online]. Available: https://libcsp.github.io/libcsp/history.html.

[6] "Client and Server Example." Kubos. Accessed: Dec. 21, 2024. [Online]. Available: https://docs.kubos.com/1.2.0/apis/libcsp/csp_docs/example.html?highlight=port.

[7] “Blue Books: Recommended Standards.” CCSDS. Accessed: Dec. 21, 2024. [Online]. Available: https://public.ccsds.org/Publications/BlueBooks.aspx.

[8] “Member Agencies.” CCSDS. Accessed: Dec. 21, 2024. [Online]. Available: https://public.ccsds.org/participation/member_agencies.aspx.

[9] *Space Packet Protocol*, CCSDS 133.0-B-2, 2020. [Online]. Available: https://public.ccsds.org/Pubs/133x0b2e2.pdf. [Accessed: Dec. 21, 2024].

Appendices

Annex A

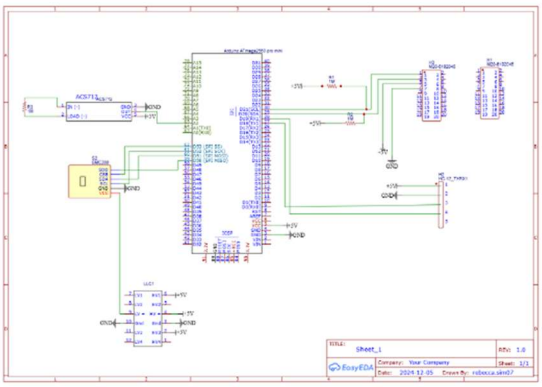

Figure A1. Schematic diagram of PCB