In-Flight Firmware Update of an On-Board Computer for Small Satellites

M. A. Mecha
ICIFI, UNSAM
Buenos Aires, Argentina
mmecha@estudiantes.unsam.edu.ar

L. L. Gagliardi
ICIFI, UNSAM-CONICET
Buenos Aires, Argentina
lgagliardi@unsam.edu.ar

L. Finazzi
ICIFI, UNSAM-CONICET
Buenos Aires, Argentina
lfinazzi@unsam.edu.ar

F. Di Nardo
ARSAT, UNSAM
Buenos Aires, Argentina
fdinardo@arsat.com.ar

N. Álvarez
ITECA, UNSAM
Buenos Aires, Argentina
nalvarez@unsam.edu.ar

G. A. Sanca
ICIFI, UNSAM-CONICET
Buenos Aires, Argentina
gsanca@unsam.edu.ar

F. Golmar
ICIFI, UNSAM-CONICET
Buenos Aires, Argentina
fgolmar@unsam.edu.ar

***Abstract*—In this work, we introduce an In-Flight Firmware Update technique, describing the firmware image preparation, transmission, uploading and error mitigation strategies. This technique was tested on the LabOSat-02 On-Board Computer with satisfactory results. The system was tested with a Bit Error Rate of up to 10^{-3} with transmission times of less than 3 times the base time (BER $= 0$) and without errors.**

***Index Terms*—On-Board Computer, Space Mission, Hardware, Firmware, Operation Modes, Small Satellites, Bootloader**

I. Introduction

In the modern era of satellite technology, nanosatellites have become a cost-effective platform for diverse scientific and technological missions. A key challenge is securely and efficiently updating flight firmware in orbit [1], [2], especially given limited communication and physical access. This work proposes an In-Flight Firmware Update mechanisms over LabOSat-02 [3] prioritizing the data reliability.

LabOSat (Laboratory-On-a-Satellite) is a research group focused on raising the Technology Readiness Level of electronic systems for space. Since 2014, it has participated in nine LEO missions, starting with LabOSat-01 [4], [5]. In recent years, the group developed LabOSat-02, an On-Board Computer (OBC) designed to control payloads, run experiments, and manage data via multiple communication protocols.

This work aims to design a Bootloader and in-flight safety mechanism to update or restore the OBC via Controller Area Network (CAN), chosen for its robustness and reliability. Unlike I2C, commonly used in CubeSats but prone to lockups and errors [6], CAN provides differential signaling, error handling, and automatic retransmission. RS-485 is another option offering noise immunity and throughput but lacks standardized protocols and error-checking [7]. According to ESA standards, the bit error rate threshold is 10^{-5} for the Ground Station (GS) Satellite uplink communication [8], [9]. For onboard CAN communication between the transceiver and the OBC, the Bit Error Rate (BER) is significantly lower due to the short distance and controlled electrical environment. Pessimistic BER values for CAN range between 10^{-11} and 10^{-7} [10], depending on noise levels and shielding quality, with a commonly assumed average around 10^{-9} in moderate conditions. These values will be taken in account to support the plotted results.

II. Procedure for Remote Firmware Updating

The main objective of this work is to transfer a binary image from the GS to the satellite transceiver and then, transfer it through CAN to the main OBC — both systems emulated with two OBCs in the laboratory — , which will update its own main program executing a secure sequence to avoid flashing errors. According to this, a general diagram of the complete communication system to be evaluated is shown in the Fig. 1.

Firstly, the firmware binary image is generated with a compiler tool chain and divided in pages, this enables the possibility of re-sending faulty pages which makes the data transmission process modular. To check the page validity, a Checksum (CS) is sent at the end of each page. Each page is divided into packets for transmission, and sent with its corresponding CS, if it does not match the one calculated by the OBC, the page is re-sent. In order to achieve a successful procedure, each page of the binary image and the Number of Pages (NP) received by the OBC is stored in an external Ferroelectric Random Access Memory (FRAM), selected for their high tolerance to ionizing radiation [11]. Finally, In case of GS sends a command to start it, the OBC will start the flashing process loading the binary image stored in the FRAM and into the program Flash. Fig. 2 illustrates the step-by-step mechanism that executes the OBC to update the Flash memory with a new firmware. It describes the structure of the internal and external memories and interactions between them. The numbers represent the order in which the steps are executed. In particular:

1) The Bootloader causes the program counter (PC) to jump to the main program.
2) Main Program stores the image and flag in the FRAM. **After making a power reset**:
3) The current flag value is evaluated in the Bootloader.

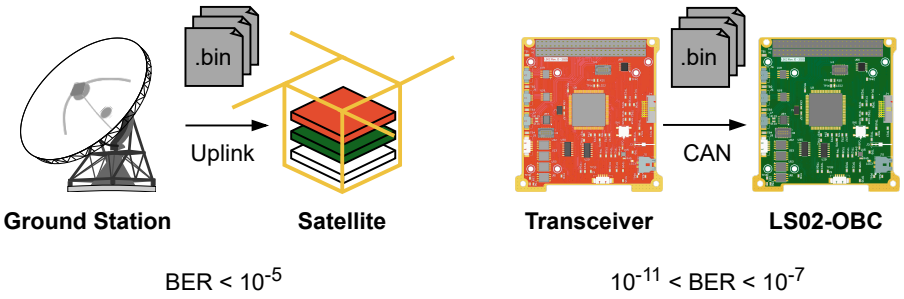

Fig. 1. Diagram that describes the main noise sources that can induce bit errors during the transmission. In this case, according to ESA standards [8] the uplink communication introduces a BER of 10^{-5} and the CAN bus communication adds, as a pessimistic value, a maximum BER of 10^{-7} [10]. The experiment was carried out on ground, where the transceiver was emulated by a secondary OBC, both units communicating via CAN.

4) Bootloader causes the PC to jump to Update Program.
5) The firmware image is read in the Update program and loaded in Flash.

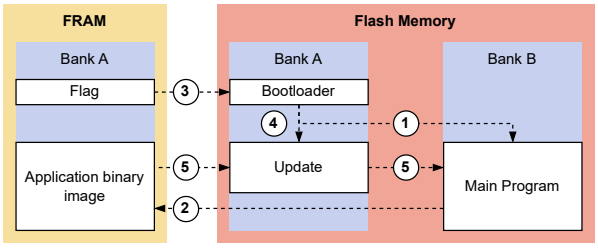

Fig. 2. Overview of the memory banks partitions. The Flash memory contains the Main, the Firmware Update and Bootloader programs. The FRAM stores the Binary Image and the flag that indicates if an update is in order.

A. Firmware Image Preparation

Once the firmware image is generated (using an Integrated Development Environment), the data is separated in 128 Byte pages, consisting of 16 packets of 8 Bytes each. This data partition guarantees that pages can be re-sent if a transmission error occurs (the optimum page size is dependent on the BER) [1]. To verify data, the process uses a CS for each page of the binary image. For the purposes of data communication, the goal of a CS algorithm is to balance the effectiveness at detecting errors against the cost of computing the check values. Furthermore, it is expected that a checksum will work in conjunction with other, stronger algorithms, such as a Cyclic Redundancy Check (CRC). In this case, the CAN data link layer uses a inherent CRC to check the data was not corrupted during transmission, and a CS is used by higher layers to ensure that data was not corrupted in the communication between GS and the satellite. A Fletcher CS is performed to each page, due to its lower computational effort (compared to a CRC) and its position dependency (unlike a standard CS) [12]. This ensures data safety while adding little computational overhead.

B. Data Transmission and Storage

To upload the data to the nanosatellite the Main program performs a communication sequence shown in Table I.

TABLE I
MESSAGING SEQUENCE DETAILING THE INTERACTION BETWEEN THE GS AND THE OBC DURING DATA TRANSMISSION.

N	GS		OBC
1	Start Transmission	→	
			Check for vital tasks
2		←	Ready
3	Data message 1 (Page 1)	→	
			Store in FRAM
4	Data message 2 (Page 1)	→	
			Store in FRAM
	...	...	...
18	Data message 16 (Page 1)	→	
			Store in FRAM
19	End of page (CS)	→	
			Verify Checksum
20		←	Status
Same sequence from step 3 to step 20 repeated for subsequent pages			
n	End Transmission (NP)	→	
			Verify NP
n+1		←	Status
n+2	Load to Flash command	→	
			Raise flag
n+3		←	Entering Bootloader
			Reset

To begin the process, a **Start transmission** is sent by the GS. Then, after waiting for vital tasks to end, the OBC responds with a **Ready** frame. The upload begins and each of the 16 **Data message** packets is sent and stored in FRAM. Subsequently, an **End of Page** message is transmitted, accompanied by the Fletcher CS of the entire page, which is calculated by the GS. Then, the OBC counts the number of packets received, computes the Fletcher CS and compares it to the one received. If both validations pass, the OBC transmits a correct **Status** frame; otherwise, it sends an status error message, prompting the page to be resent. After all the pages are transmitted, an **End Transmission** is sent, accompanied by the total NP. This is also verified by the OBC, which responds with a **Status**. If this validation is not correct, only this message is sent again three times. However, if the NP are

different at least two times, the full sequence is performed again. Finally, once the GS decides to start the Firmware Update, the **Load to Flash** command is sent. The OBC then raises the Flash Ready flag in the external memory, informs that is **Entering Bootloader**, by sending the corresponding message, and then performs a soft reset before the Bootloader program runs.

C. New Image Flashing

Once the messaging sequence is completed and the **Load to Flash** frame is received, the OBC Bootloader program executes. It begins by checking the flag, and then takes one of two routes, as described in Fig. 3. If this flag raised, the flashing process will begin. Otherwise, a memory jump to Main program will be performed. The Update Program begins by reading the external memory and loading each page in Flash. After all pages are loaded, the flag is turned down and a soft reset is performed. Finally, the Bootloader will make the jump to the Main program, due to the flag being down.

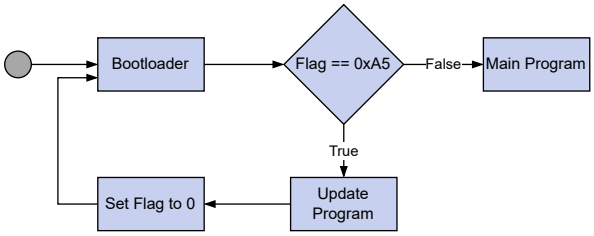

Fig. 3. Bootloader flowchart. The Bootloader checks the flag in FRAM and, according to the value, it will either perform a memory jump to the Main program or begin the flashing process.

D. Error Mitigation

Transmission errors can arise from environmental factors such as Free Space Path Loss (FSPL), Doppler shift, and atmospheric attenuation [9]. To address this, the system includes mitigation strategies:

- Connection loss: If packets or an entire page are lost, the NP check will fail, and the GS will not receive the "Correct Status" message. The affected page will be re-sent until acknowledged. Previously sent pages are stored in FRAM, avoiding their retransmission.
- Bit errors: Noise during transmission may cause incorrect bits. If the page CS at the OBC does not match the value coming from the GS, the page is re-sent.
- System Reset: The OBC implements an external watchdog timer and other protection circuits to handle the Single Event Effects (SEE) [7]. They can trigger a reset during a page transmission and therefore interrupt the complete firmware image transmission. After the reset, the Bootloader checks if the flag is not rised — meaning that the flashing process did not finish completely —, in which case the process is re-started to ensure completion.

III. Recovery mode

Due to the possibility of SEU-induced failures, the integration of fault-tolerance techniques that enable In-Flight firmware maintenance has significant utility [7]. According to this premise, a firmware replication in FRAM can be a robust solution against a malfunction over the operating program in Flash [2]. This method involves storing a backup copy of the firmware in the FRAM, whose tolerance to radiation makes it suitable for storing a secure copy of a safe mode program [11]. In case of a main program failure, the Bootloader uploads the safe mode binary image into the program memory.

Since each update writes the running image based on the information contained in the external FRAM memory, a backup image is always maintained and kept up to date.

IV. Experimental results

As it was previously discussed, the total BER of CAN is $\approx 10^{-}5$ but for the next experiments, we considered a worse scenario of 10^{-3}. The Fletcher CS algorithm detects bit errors and triggers a full page retransmission when one occurs. This ensures data integrity but introduces a trade-off in terms of overall transmission time, especially with a high BER. The total time required for a full firmware image transmission is inversely proportional to the probability of getting a full page without errors. The absolute time, as a function of BER, is modeled by Eq. 1 as:

$$T_{\text{total}} = \frac{T_{\text{base}}}{(1 - \text{BER})^{\text{P}\cdot\text{B}}} \tag{1}$$

Where:

- P: Number of packets per page.
- B: Number of bits per packet (8 bytes = 64 bits).
- T_{total}: Total transmission time.
- T_{base}: Transmission time when the BER = 0.

and:

$$T_{\text{base}} = (T_{\text{CS}} + (\text{P} + 1) \cdot T_{\text{packet}}) \cdot \text{NP} \tag{2}$$

Where:

- NP: Number of pages in the full firmware image.
- P+1: Packets transmitted per page (+1 for the CS)
- T_{CS}: Checksum calculation time.
- T_{packet}: Transmission time for each packet.

Fig. 4 illustrates the relationship between transmission time and BER. Both the experimental results and the theoretical predictions based on Eq. 1 are plotted to validate the model.

To establish a baseline, a full transmission without errors (BER $= 0$) was performed, measuring the time required under ideal conditions. All experiments described in this work were conducted on ground, using two OBC units to emulate the Ground Station and the satellite transceiver. Bit errors were injected in a controlled environment to simulate communication noise, allowing us to evaluate the firmware update system under harsh conditions without requiring in-orbit deployment.

As shown by Eq. 1, total transmission time grows with the page length. This arises from the need to retransmit an

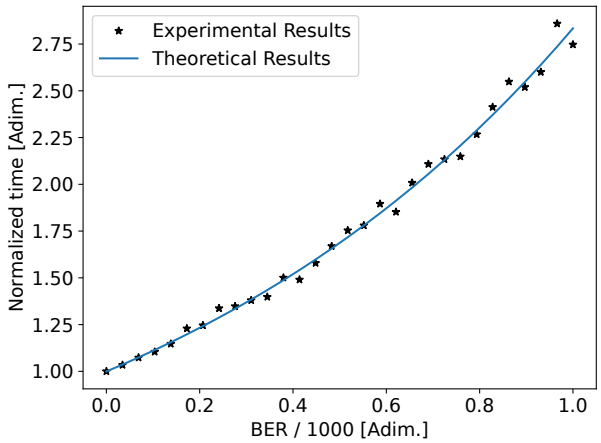

Fig. 4. Transmission Time for Different BER. Time is measured relative to the time of a full transmission with a BER = 0. The intention of this graph is to evaluate the time it takes to transfer the complete binary image under aggressive noise conditions (BER = 10^{-3}) beyond the BER = 10^{-5} set as a recommended limit by ESA [8].

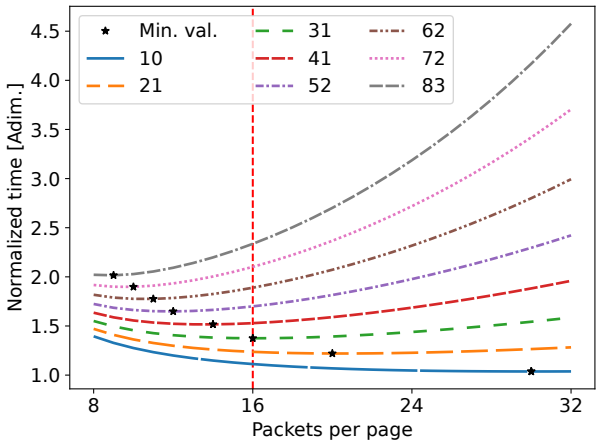

Fig. 5. Transmission Time for Different Page Lengths and BERs (expresed in 10^{-3}). Each page consists of packets of 8 bytes each. Each curve has an optimal page size shown with black stars in the plot. The vertical line represents the selected page size (16 packets), which is optimal for a BER of $31 \cdot 10^{-3}$.

entire page if any bit within the page is corrupted. Therefore, increasing the page length results in longer transmission times in noisy environments, as more data is at risk of corruption. This suggests that using shorter pages would reduce the impact of retransmissions in high-noise environments. However, shorter pages also mean that the CS must be calculated and transmitted more frequently, increasing overhead and time. Thus, the trade-off lies between having less retransmissions or having less overhead.

Fig. 5 demonstrates how the page length affects the relationship between transmission time and bit error probability. To establish a baseline, a full transmission with pages of size 16 and without errors (BER = 0) was performed, measuring the time required under ideal conditions. In low noise environments, using larger pages can reduce transmission time, as the CS is computed and transmitted less frequently. However, in high-noise environments, larger pages increase the likelihood of full-page retransmissions, leading to longer transmission times. Conversely, shorter pages are more suitable for high-noise environments but come with higher overhead due to frequent CS calculations and transmissions.

V. Conclusions

The Firmware Update procedure, including a recovery mode, was successfully developed and tested with satisfactory results. The steps for flashing were outlined, identifying potential error sources and proposing preemptive measures. The transmission protocol was evaluated with BERs of up to 10^{-3} for 16-packet pages, with transmission times of less than 3 times the base time (BER = 0) case, without errors in the final firmware image. Different page sizes were simulated and optimal page sizes were determined as a function of the BER.

Acknowledgments

The authors acknowledge financial support from ANPCyT PICT 2017-0984 “Componentes Electrónicos para Aplicaciones Satelitales (CEpAS)”, PICT-2019-2019-02993 “LabOSat: desarrollo de un Instrumento detector de fotones individuales para aplicaciones espaciales” and UNSAM-ECyT FP-001.

References

[1] I. Sünter *et al.*, “Firmware Updating Systems for Nanosatellites,” *IEEE Aerospace and Electronic Systems Magazine*, vol. 31, no. 5, pp. 36–44, 2016.

[2] P. Botma, A. Barnard, and W. Steyn, “Low cost fault tolerant techniques for nano/pico-satellite applications,” in *Proc. Africon*, 2013, pp. 1–5.

[3] L. Gagliardi *et al.*, “LabOSat-02: Hardware and Firmware Development of an On-Board Computer for Small Satellites,” *IEEE Embedded Systems Letters*, vol. 16, no. 1, pp. 37–40, 2024.

[4] G. A. Sanca *et al.*, “LabOSat-01: A Payload for In-Orbit Device Characterization,” *IEEE Embedded Systems Letters*, vol. 16, no. 1, pp. 45–48, 2024.

[5] L. Finazzi *et al.*, “Total ionizing dose measurements in small satellites in LEO using LabOSat-01,” *Nucl. Instrum. Methods Phys. Res. A*, vol. 1064, Art. no. 169344, 2024.

[6] J. Bouwmeester, M. Langer, and E. Gill, “Survey on the implementation and reliability of CubeSat electrical bus interfaces,” *CEAS Space Journal*, vol. 9, pp. 163–173, 2017.

[7] A. Cratere *et al.*, “On-Board Computer for CubeSats: State-of-the-Art and Future Trends,” *IEEE Access*, vol. 12, pp. 99537–99569, 2024.

[8] ESA, *ECSS-E-ST-50C Rev. 2 - Space Engineering - Communications*, European Space Agency, 2024.

[9] I. Latachi *et al.*, “Link budget analysis for a LEO cubesat communication subsystem,” in *Proc. Int. Conf. Advanced Technologies for Signal and Image Processing (ATSIP)*, 2017, pp. 1–6.

[10] Ferreira, J., Oliveira, A., Fonseca, P. & Fonseca, J. An experiment to assess bit error rate in CAN. pp. 15-18 (2004)

[11] G. Korkian *et al.*, “Single Event Upsets Under Proton, Thermal, and Fast Neutron Irradiation in Emerging Nonvolatile Memories,” *IEEE Access*, vol. 10, pp. 114566–114585, 2022.

[12] J. Stone, M. Greenwald, C. Partridge, and J. Hughes, “Performance of checksums and CRCs over real data,” *IEEE/ACM Trans. Netw.*, vol. 6, no. 5, pp. 529–543, 1998.