Development of an adaptable On Board Computer for a Nanosatellite

P. M. Mulwa[1*], E. N. Orenge[2], A. N. Gichamba, B. M. Nzangi[4], P. M. Ndirangu[5]

[1,2] *Department of Electrical and Electronics Engineering, Jomo Kenyatta University of Agriculture and Technology, Nairobi, Kenya*

[3,5] *Department of Electrical and Electronics Engineering, University of Nairobi, Nairobi, Kenya*

[4]*Department of Geospatial and Space Technology, University of Nairobi, Nairobi, Kenya*

**Email: mosesmulwa123@gmail.com*

[1]

Abstract

With the vision of developing a 3-U Nanosatellite in mind, the named undergraduate students from the two universities embarked on a journey to develop an On-Board-Computer(OBC), a system that can be loosely termed as the heart of the satellite. This paper describes the process of conceptualization, design and testing of an affordable and adaptable Onboard Computer (OBC) for a 3U nanosatellite. The OBC of the satellite is responsible for recording and storage of telemetry and payload data, encoding and decoding of data packets to and from the ground station, processing commands from the ground station,controlling the satellite within its orbit, implementing watchdog functions and monitoring other subsystems. Affordability of the OBC was a key metric for the success of the mission. It also had to be adaptable and be able to be used for future missions and mission requirements.

***Keywords*—**On board computer, Nanosatellite, Computing System, Finite state machine.

.

I. INTRODUCTION

An alternate name for the OBC is command and data handling system(CDHS). This paper aims to cover the following areas regarding the OBC: section II describes the hardware architecture of the OBC, section III elaborates the software architecture and different modes in the OBC, section IV expounds on data transfer and handling and section V explains the various methods of fault tolerance and section VI gives a general conclusion based on the aforementioned areas of focus..

[1].

II. HARDWARE ARCHITECTURE

The OBC is designed to provide the full CDHS functionality while maintaining low power consumption and high reliability. This architecture is the result of the OBC's subsystem requirements. The team chose to build the OBC from discrete parts to maintain low cost, mass, power consumption and to have flexibility of the design. The architecture is as shown in Fig 1 below.

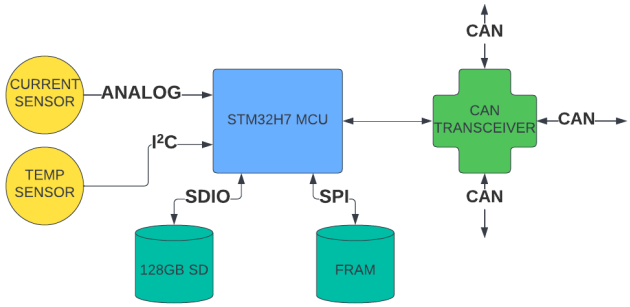

Fig 1: OBC architecture

At the heart is a STM32H743 microcontroller. It serves as the primary onboard processor, performing all the coordination and computation that is critical to supporting the mission.

The 128GB SD card stores the mission data which includes images captured by the optical payload and their metadata. The 2Mb FRAM stores critical configuration data and the latest stable version of the OBC firmware. This redundant structure improves the likelihood of the mission surviving despite corruption of the microcontroller's flash memory.

The (Controller Area Network) CAN transceiver allows the OBC to communicate with all the other subsystems through a

CAN bus. This bus allows any subsystems connected to the bus to communicate with one another, without the OBC as an intermediary. CAN, which employs differential signaling, was selected as an alternative to single-ended buses as the complementary signals in CAN offer double the common-mode noise immunity. Although CAN and RS-485 are similar at the electric level, CAN was preferred because of its relative ease of software development.

The current sensor and temperature sensor allow close monitoring of the OBC's state. Monitoring the temperature of the devices making up the OBC allows the devices' temperatures to be maintained within their optimal ranges to maximize their operational lifetime.

On an overall scale, the OBC is placed centrally between all other subsystems where it can manage data and commands from the subsystems. This can be seen in Fig 4.

III. Software Architecture

The functioning of the satellite can be represented as a finite state machine in which there are a set of states/modes and depending on certain conditions (such as satellite health, power level, or commands from other subsystems), the satellite will switch modes.
Finite state machine (FSM) is a term used by programmers, mathematicians, engineers and other professionals to describe a mathematical model for any system that has a limited number of conditional states of being. [2]
Switching between modes is interrupt based and polling for the metrics mentioned above will determine which determine to switch to. Each of modes has a set of tasks associated with it.

The software architecture is divided into four sections as shown in Fig 2. The top level is called the Flight Plan. This represents the mode that the satellite is currently in. It is tasked with determining what tasks are to be executed. It also handles task management and scheduling. It controls the tasks that are described next. The tasks are put in a list, ordered by next time of execution.
A task is forked and executed as a child process when it is at the front of the list and:
1. The current system time is greater than or equal to the time of next execution.
2. There is no child process running in the system that is doing the same task.
Once the process has been forked and executed, it is put back in the list as per the next time of execution. The list mentioned above also contains metadata about each child process including but not limited to its priority & frequency of execution

Each mode has mode specific tasks associated with it as shown in Fig 3. A task in this case is defined as a process that is carried under a particular mode. This includes processes such as watchdog timers, taking pictures, compressing data and so on.
A Real Time Operating System (RTOS) is used in the management of these tasks and actually implementing them. It is also useful in determining priorities for the tasks [4].
The lowemost layer of abstraction (User Space Functions) consists of a set of modules for device access on the specified buses.This provides the required abstraction from the device datasheet specifications while implementing the mode-wise subtasks that use the particular device.

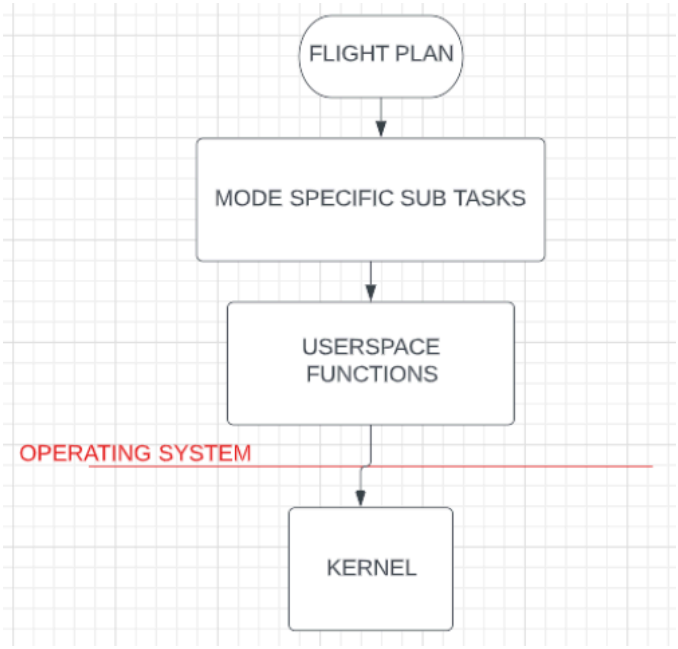

Fig. 2: Software Architecture of the OBC

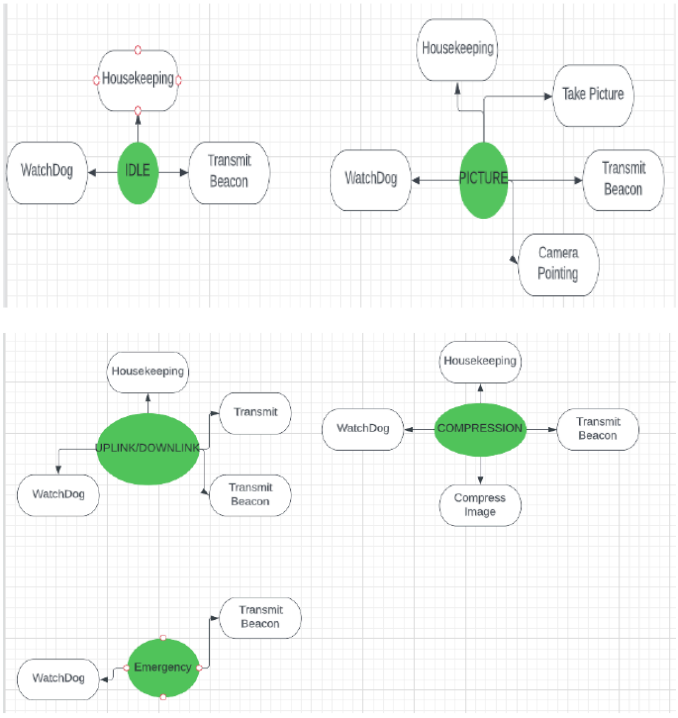

Fig. 3: OBC Modes and their associated tasks

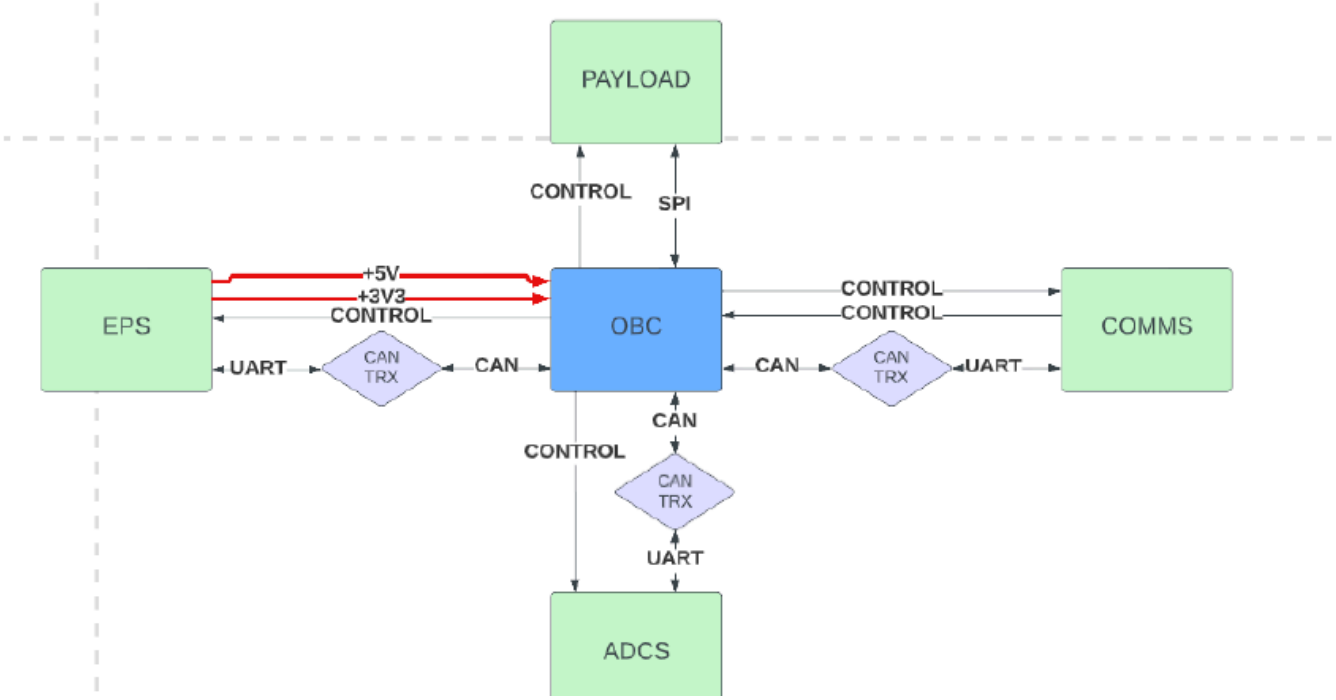

Fig. 4: Overall system architecture as related to the OBC

IV. Data Transfer And Handling

A. *Sensors and Telemetry Data*

Sensors and telemetry data is collected periodically. Every software mode has a sensor reading task. Sensor readings are used to switch modes if necessary as described in section 3. Typically sensor data consists of reading from the two sensors(current and temperature) which are part of the OBC and other readings from other subsystems. The Power subsystem sends data on power consumption and failures. The Attitude Determination and Control System(ADCS) sends data on the orientation of the satellite and its positioning. Since the two subsystems mentioned above have their own controllers and logging mechanisms, a 30-second interval was chosen for the logs. This is to reduce bus congestion due to constant transmission and also to reduce the amount of data stored by the Computing system. Three communication protocols are used for data transfer: I2C, SPI, and CAN. The areas for their use depend on the sensors used to capture the data.

Software modules for the communication protocols mentioned above have been developed to provide functionalities to the user space for communication over the I2C and SPI buses, respectively.

B. *Payload Data*

Payload data is generated by the camera used which is the Gecko Imager in this case. Data is generated on demand when the “take picture” task is executed. This data is then transferred to the OBC for storage using an SPI bus. This is due to its fast speeds [1].

Calculation of the data budget is shown below. It is noted that the imager generates 336 Mbps of data when used in its 8-bit mode.

Assuming the satellite is in view of the target for 10mins (600secs), to calculate the amount of data:

For a payload of 336Mbps, In 600s:

• Payload = 336 x 600 = 201600 Mb = 25200 MB = 24.61GB

Thus, onboard storage of 128GB is more than enough to handle the data generated in a single day. It is worth noting that this data is transferred to the ground station daily and thus

is only stored for a day.

V. Fault Tolerance Methods

Various methods for fault tolerance are included in our current model. The major points of vulnerability include: radiation tolerance of the Microcontroller and bus failure. In addition, the CDH unit may enter a non-responsive state or suffer memory corruption. The following methods are implemented for fault tolerance according to the above areas of focus. A watchdog timer will safeguard against any and all CDH failures, and bus failures.

A. Watchdog timer

This is a hardware timer that is reset at a predefined regular time interval [3]. If it is not reset before it expires, the default action is to reset the hardware that was supposed to reset it.
A watchdog timer exists on the Microcontroller unit(MCU) of the Electrical and Power Subsystem(EPS) which is interfaced with the OBC through a GPIO pin specifically meant for this. If the timer is not reset by the OBC, the EPS is supposed to perform a power cycle on the OBC.
A hierarchy of watchdog timers has been proposed to avoid system reboots for minor faults. This allows for fault isolation. The lowermost level of the hierarchy is the physical timer on the MCU of the EPS. It will be reset by sending a signal on the GPIO pin. This is very important while booting up the OBC. The timer reset for the EPS watchdog will be performed by a user space function call (which in turn calls the hardware function for the GPIO interface).
The software watchdog has exclusive access to this userspace function and hence it forms the second layer of the aforementioned hierarchy. The software watchdog will be implemented as a second thread within the Flightplan process. The following reasons justify this design:

a. The process being watched are children of this process. They can easily check in at specified frequencies by sending a signal to the parent which will be handled by a specific handler built for this purpose. The handler will write to a particular memory location corresponding to each child's process.
b. In case the process fails to check in, the Flightplan's watchdog thread can send a SIGTERM signal to the child and terminate execution. This is possible because the linked list node contains metadata about each process which includes its process ID. SIGCHILD signal will be received by the main thread of the Flightplan as soon as the child dies. It will update this information in the linked list node.

B. *Component selection for fault tolerance*.

Components used in the OBC have all been checked for flight heritage or technology readiness.
Technology Readiness Levels (TRL) are a type of measurement system used to assess the maturity level of a particular technology. Each technology project is evaluated against the parameters for each technology level and is then assigned a TRL rating based on the progress of the project [5]. A TRL rating of 9 shows proven flight heritage. Flight heritage is used to mean that the component has been used and has performed in a previous satellite mission. Satellite operators and manufacturers accept flight heritage when technology works in a commercially representative environment. Ideally, this means operated by a commercial company on a commercial satellite. [6]

Part	TRL level
Microcontroller	**9** – Proven heritage: M6P
Temperature sensor	**9** – Proven heritage: ArduSat
RS-485 transceiver	**9** - Proven heritage: MiRaTA
Current sensor	**5** - Component validation
FRAM	**5** - Component validation
SD card	**5** - Component validation

Fig. 5: TRL levels of OBC components

C. Bus Failures

Bus failures are mitigated by use of CAN as a communication protocol which has noise immunity, automatic error detection and retransmission protocols in the case of erros [7].
Priority levels are also assigned to all subsystems to prevent bus collisions.

VI. Conclusion

The aim of this project was to develop an affordable and adaptable onboard computing system for a nanosatellite. Affordability was achieved by using inexpensive components and layouts. Adaptability was achieved in two ways. For the hardware, a CAN bus interface was used to connect to other subsystems. This means that the addition of new subsystems is as easy as adding them to the bus. Removal of substems is also easy and does not break the existing layout.
Adaptability in software has been achieved by using abstracted layers in the architecture. This means that each layer can be modified independently of other layers.
Additionally, the software operation is in modes and thus the addition of new modes is easy. The addition of new tasks within existing or new modes is also easy and does not affect existing modes.
Thus both affordability and adaptability have been achieved as per the mission requirements.
While this computing system has been made for the primary use of onboard nanosatellites, it can be used in areas that require control of multiple systems or subsystems.

.

Acknowledgment

The authors would like to thank the Kenya Space Agency Tafiti project, Jomo Kenyatta University of Agriculture and

Technology, and the University of Nairobi for providing the resources and environment required to conduct this research

References

[1] Pei An, 4 - Expanding the Centronic, RS232 and game ports, Editor(s): Pei An, PC Interfacing, Newnes, 1998, Pages 109-147, ISBN 9780750636377, https://doi.org/10.1016/B978-075063637-7/50006-7. (https://www.sciencedirect.com/science/article/pii/B9780750636377500067)

[2] K. Moore, “Finite State Machines,” Brilliant Math & Science Wiki. [Online]. Available: https://brilliant.org/wiki/finite-state-machines/. [Accessed: 28-Aug-2022].

[3] : D. Neumann, “RF Wireless World,” What is watchdog timer | Applications of watchdog timer. [Online]. Available: https://www.rfwireless-world.com/Terminology/what-is-watchdog-timer-in-microcontroller.html. [Accessed: 01-Sep-2022].

[4] D. Gupta , “What is an RTOS - real time operating system information and training,” High Integrity Systems, 24-Jun-2021. [Online]. Available: https://www.highintegritysystems.com/rtos/what-is-an-rtos/. [Accessed: 28-Aug-2022].

[5] B. Dunbar, “Technology readiness level,” NASA, 06-May-2015. [Online]. Available: https://www.nasa.gov/directorates/heo/scan/engineering/technology/technology_readiness_level#:~:text=Technology%20Readiness%20Levels%20(TRL)%20are,based%20on%20the%20projects%20progress [Accessed: 28-Aug-2022].

[6] ESA, “Atlas lifts Satcom Heritage,” ESA, 2013. [Online]. Available: https://www.esa.int/Applications/Telecommunications_Integrated_Applications/Atlas_lifts_satcom_heritage#:~:text=What%20is%20flight%20heritage%3F,more%20valuable%20the%20flight%20heritage. [Accessed: 12-Sep-2022].

[7] D. West, “Can in automation (CIA),” CAN in Automation (CiA): History of the CAN technology. [Online]. Available: http://www.can-cia.org/can-knowledge/can/can-history/. [Accessed: 01-Sep-2022].