SSC22-WKP2-28

Co-Operating Systems: A Technical Overview of Multiple Onboard Operating Systems

Cameron Bonesteel, Evan Tichenor, Eric Miller, Andres Rodriguez
University of Georiga, Small Satellite Research Lab
1510 Cedar St. Room 107 Athens, GA 30602 ; (706) 284-4762
cameron.bonesteel@uga.edu

ABSTRACT

The Multiview Onboard Computational Imager (MOCI) built by the University of Georgia's Small Satellite Research Lab (SSRL) will be one of the first small satellites to include a small form factor graphics processing unit (GPU), the NVIDIA TX2i, in its design and therefore includes three onboard computers that will need to coordinate and interchange data to successfully complete the mission. The Onboard Computer (OBC), standard for most satellite systems, will coordinate most system controls. The GPU serves as the onboard payload manager and bulk data processor and coordinates with the attitude determination and control system (ADCS) to collect all necessary data for processing. These computers will communicate through hardware lines using two different serial protocols. In this paper, we propose a control system which synchronizes system clocks and transfers data efficiently through the differing serial protocols in order to process data without any bottlenecks in the data transfer or a decrease in accuracy between the data and its matching telemetry. The success of all these components working together will serve as a proof of concept for GPUs, like the TX2i, onboard small satellites and giving birth to future onboard computational imagers like MOCI.

INTRODUCTION

Traditionally, small satellites contain a primary onboard computer (OBC) that runs all system command and data handling (CDH).[1] This computer is responsible for all onboard systems, telling them when and how to complete tasks, or collecting important telemetry points such as temperatures and voltages. However, the University of Georgia's Small Satellite Research Laboratory (SSRL) is utilizing a second onboard computer on the Multiview Onboard Computational Imager (MOCI) mission. This onboard computer is the NVIDIA TX2i Graphics Processing Unit (GPU).[2] The TX2i will serve as the primary mission payload control and do onboard science data processing. As a result, the traditional CDH roles of the OBC are distributed to the TX2i in some respects, but the TX2i takes on its own new role of in-orbit data processing. With this new computer being integrated into the satellite design, certain challenges arise. Each computer is running a unique operating system, which functions slightly differently in how they manage files and transfer data. This becomes increasingly complex when data needs to be transferred between multiple systems, such as with data from the Attitude Determination Control System (ADCS). In this paper, we will explore the system architecture, looking at each main system and its operating system and challenges, the different boards and considerations for their design and the overall system design, and finally how the design all works together to make the mission a success.

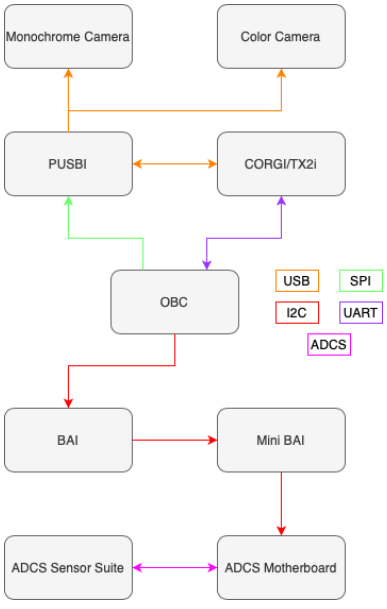

Figure 1: MOCI System Architecture

SYSTEM ARCHITECTURE

The system architecture is designed around the storage and movement of data onboard the satellite. Each section will go in depth to explain the main onboard computers, their operating systems, and important information about how their operating systems control and manage data. The general systems that will be covered include the NVIDIA TX2i, ADCS, and the OBC as seen in Figure 1.

NVIDIA TX2i

The NVIDIA TX2i is the primary payload control computer onboard MOCI. It controls the optical payload and manages science data including images, point clouds, 3D models, and object recognition results. It runs YOCTO, a custom-made, simplified Ubuntu operating system designed for the mission. This operating system eliminates unimportant features such as the user interface, as well as default packages that are not used, such as python. These improvements allow the TX2i to conserve space and boot quicker. Since YOCTO is built off Ubuntu, it manages files with a simple extended file system. This makes file management on the computer easy from an outside controller, such as the OBC. The general file structure contains a directory for raw images taken by the optical payload with each ground target having a unique identifier that can be used to create a variable path name inside the raw image directory using an integer. The unique identifiers make grouping target images simple for mission operations. Science output files such as point clouds and object recognition outputs can then have a separate directory in the root directory with the same variable numbering system to allow identification of output files for specific ground targets. This also allows for downlink preparation to be simplified since old downlinked data can be erased from the TX2i leaving only the new data to be downlinked. The TX2i is capable of using both a Universal Asynchronous Receiver/Transmitter (UART) and Inter-Integrated Circuit (I^2C) bus. For this mission, only the UART bus is utilized.

Attitude Determination Control System

The CubeSpace Attitude Determination Control System onboard MOCI not only controls the attitude of the satellite, but also logs important attitude information for science data processing. The ADCS runs a simple UNIX operating system. This UNIX operating system is less accessible than the other operating systems and acts as a sort of black box. It relies on a simple telecommand structure for sending and receiving data over an I^2C bus.[3]

Onboard Computer

The Onboard Computer is the main system control interface for MOCI. It is running on the FreeRTOS operating system. This operating system runs alongside the flight software which has its own file management system. The file management system is comparable to a series of buffers that store information. This starts with a storage channel which has a fixed capacity in terms of bytes. These storage channels then contain rows of data where each row is one byte. This system allows for a simple channel and row identification to access data. To access data that is larger than one row, a range of rows can be accessed and returned. The flight software also relies on the real time clock on the OBC to accurately schedule mission events for autonomous operation. The flight software specifies specific events for specific time frames and the operating system prioritizes these events based on how time sensitive they are so important mission events execute on time.

HOW THE SYSTEMS COOPERATE

While each operating system runs independently of each other, they each have information that the others may need. While routing the data directly between the hardware is the most straightforward approach, it becomes more difficult once many of the mission factors are considered; such as the interface boards and serial protocols, and mission lifetime. These factors will be considered in depth in the following sections.

Interface Boards and Serial Protocols

As seen in Figure 1, each of the two additional systems, the TX2i and the ADCS, require different interface boards. These interface boards serve a couple different purposes.

The Peripheral USB Interface (PUSBI) board is located adjacent to the OBC and the TX2i is mainly part of mission redundancy. This will be covered in the mission lifetime section. In addition to this board, the Core GPU Interface board (CORGI) is required to interface the TX2i with the rest of the system. The TX2i does not have the standard 104 pin header like the rest of the stack and therefore relies on CORGI to interface in this way. While the TX2i can use both a UART and I^2C protocol, CORGI is only designed for UART. We wanted to

interface with the TX2i through UART because it gives us more control of what data and how much data is sent across the bus compared to I^2C. Since a majority the data consists of science data which is large compared to most other system data, UART was the safer option.

The ADCS requires the Board ADCS Interface (BAI) and the Mini BAI boards since the 104 pin header is not laid out the same as on the rest of the stack. As mentioned before, the ADCS relies on I^2C for communication and commanding.

While the TX2i has I^2C communication capabilities which would allow it to communicate directly with the ADCS, this would require more time that we don't have to research and develop the TX2i to allow it to communicate with the ADCS seamlessly. Additionally, CORGI would have to be redesigned to add an I^2C bus line to the TX2i so it could communicate directly with the ADCS. CORGI has been a very challenging problem for both the software team and the hardware team, and adding another bus to the equation would add unnecessary complexity. In addition, having both the OBC and TX2i able to command the ADCS could lead to race conditions and simultaneous data access issues. Instead, the system structure as seen in Figure 1, continues to be the better solution to ensure operations are only happening one at a time.

To solve the problem of data needing to be shared and matched between systems, each system will have its system time updated from the OBC either on boot of that system or on boot of the OBC. This will allow the system times to match within less than a second, allowing for very precise matching of data.[4] Trying to match data in real time would have a longer delay due to the data being routed through multiple data lines.

Mission Lifetime

The mission lifetime and critical events are important to consider in the overall system design. The mission lifetime is expected to be no longer than 18 months. The TX2i's lifespan is noticeably shorter than the overall mission lifetime, with a maximum expected lifetime of around six months.[5] As a result a payload redundancy is needed to ensure we maintain control of the payload after the TX2i fails. PUSBI is the interface board that allows us to do this. This board sits adjacent to the OBC and the TX2i, and it interfaces directly with the two cameras via USB. It connects to the OBC using a Serial Peripheral Interface (SPI) and the TX2i using USB. In the case of a TX2i failure, PUSBI allows the OBC to maintain communication with the cameras via SPI, and the USB connection to the TX2i is cut off. In the event a TX2i boot fails, another boot attempt will be made automatically. After three failed boot attempts, the TX2i is considered dead and the OBC takes over. This failsafe further emphasizes why the TX2i cannot be directly interfaced with the ADCS. There are additional software failsafes to ensure the OBC can do what the TX2i was doing, but by making the OBC the primary controller of the ADCS, these failsafes do not need to extend to that relationship.

OPERATIONAL MODES

The MOCI mission uses operational modes to control onboard systems and manage mission resources. These operational modes allow the OBC to easily manage what systems are turned on and off, collect telemetry from them, and run automation workflows for collecting and processing science data. These operational modes are as follows:

- Deployment Mode
- Safe Mode
- Detumble Mode
- Cruise Mode
- Power Gen Mode
- Scan Mode
- Data Processing Mode
- Data Downlink Mode

The deployment mode is only used after satellite deployment and runs the automatic deployment sequence to set up mission systems like antenna deployments and health checks. The safe, detumble, cruise, and power generation modes are mostly basic housekeeping modes that allow it to be in low power states while it isn't doing mission tasks. These low power states are useful to protect the satellite after an error, stop it from tumbling, have it idle, or collect power respectively. The scan, data processing, and data downlink modes are more power hungry and are considered the main mission modes. These modes will be explored in depth in the following sections.

Scan Mode

Scan mode is the primary science collection process onboard MOCI. This mode requires the TX2i and the ADCS to work in conjunction to take pictures of the right target at the right time. Once the TX2i is booted, important timing and capture data is sent to program the pass. The TX2i controls the two onboard cameras and takes images based on the time intervals and capture data set after startup. The ADCS is then tasked with pointing the satellite at the correct target so these images are being taken correctly. The TX2i will need the pointing telemetry to do some of the processing calculations later on. In order to retain this data, the pointing telemetry is saved along with the time so the TX2i can match it to the image times later during processing. This shared data emphasizes the importance of accurate and synced system times as discussed earlier. In the case of the TX2i failure, this system time sync will be just as important as the ground systems will be processing the data in the same way, and the times will need to match the camera times.

Data Processing Mode

Data Processing mode is the primary demonstration of the MOCI mission. The TX2i is booted and processes the images gathered from the scan mode in orbit. Accurate ADCS data is required for Structure from Motion processing, as the 3D model can only be generated if certain information about the images angles are known. After data processing is finished, the data is stored on the OBC in a downlink-ready format. In the case of TX2i failure during the mission, images are stored directly on the OBC for on-Earth processing.

Data Downlink Mode

Data Downlink mode is the primary science data communication mode onboard the satellite. The ADCS is used for precise pointing to the ground station to downlink science data, whether it is processed or raw.

CONCLUSION

The MOCI mission is unique in its use of multiple onboard computers to handle commanding and science data. With the solutions proposed in this paper, the MOCI mission is on track to to be a successful implementation of the system and will hopefully be a model for future onboard computational imagers. It is understood that this system may fail upon launch, but hopefully with the failsafes mentioned, we can still fulfill MOCI's other mission objectives, beyond demonstrating this onboard system.

Future Work

Future work for this system includes final testing of all systems. The system will be tested for ease of use, data accuracy, and overall system function before launch. Once testing is complete on Earth, MOCI will be ready to prove itself in orbit.

Acknowledgments

The authors would like to thank the Air Force Research Laboratory's University Nanosatellite Program for giving us the opportunity to work on this mission through their funding and support. We would also like to thank Sydney Whilden, our lab manager, for supporting us through our research efforts and our principle investigator Dr. Deepak Mishra for supporting the lab and our research. Finally, we would like to give thanks to Caleb Adams, Godfrey Hendrix, Alex Lin, Allen Spain, Matthew Olson, and Cate Davis for laying the ground work for this mission, the interface boards, and much more that laid the groundwork for this paper.

References

[1] D. Schor, J. Scowcroft, C. Nichols, and W. Kinsner. A command and data handling unit for pico-satellite missions. In *2009 Canadian Conference on Electrical and Computer Engineering*, pages 874–879, 2009.

[2] Caleb Adams, Allen Spain, Jackson Parker, Matthew Hevert, James Roach, and David Cotten. Towards an integrated gpu accelerated soc as a flight computer for small satellites. In *2019 IEEE Aerospace Conference*, pages 1–7, 2019.

[3] Mike-Alec Kearney. *CubeSpace CubeADCS Firmware Reference Manual*. CubeSpace, 2019.

[4] Andres Rodriguez and Deepak Mishra. A simple synchronization process for imaging satellites. In *UGA CURO Symposium*, 2022.

[5] Bjorn Leicher, Paige Copenhaver, Graham Grable, Adam King, Caleb Adams, Sydney Whilden, and Jackson Parker. *MOCI Concept of Operations*. Internal Document, 2022.