The term “On Board Computer” indicates, rather obviously, any unit flying on board a satellite which provides processing capability. However, the On Board Computer, or OBC, more commonly refers to as the computer of the satellite’s avionic sub-system, i.e. the unit where the On Board Software run. In turn, the “**On Board Software[in our case OS]**”, despite the rather generic label, is known as the software implementing the satellite’s vital functions such as: attitude and orbit control in both nominal and non-nominal cases, telecommands execution or dispatching, housekeeping telemetry gathering and formatting, on board time synchronisation and distribution, failure detection, isolation and recovery, etc.

Based on the above, the very essence of an OBC is the microprocessor board, consisting of microprocessor, non-volatile memories, volatile memories and the companion chip that connects the microprocessor to different peripherals.

However, in space as well as in the consumer market, as the shrinking of the electronic components enables it, the trend is towards more dense and integrated systems, aiming at continuous reduction of volume, mass, and power consumption.

A modern On Board Computers, in fact, not only provides the aforementioned processing resources, but it does include other functions in the same box such as:

* DC/DC Power conversion and regulation
* Ground Telecommand Decoding
* Packet Telemetry Formatting
* On Board time management
* Autonomous Reconfiguration
* Local Mass Memory function
* Housekeeping telemetry
* Interfacing with other Avionics subsystems

RTOS

Research 1:DANDE nano satellite

The Drag and Atmospheric Neutral Density Explorer (DANDE) started in 2007 as a student developed NanoSatellite project at the Colorado Space Grant Consortium.

DANDE’s mission is to study the effects of space weather and atmospheric drag on a small satellite platform.

A real-time operating system uses advanced task scheduling techniques and a preemptive kernel, which allows multi-threading of processes to occur. It depends very heavily on timing accuracy of task completion rather than serial functionality, or interrupt techniques, like some simpler operating systems.

DANDE’s real-time operating structure is unique for a small-satellite platform. In a large amount of CubeSats and NanoSats, a simpler operating system can be implemented to accomplish mission goals.

The enhancements in multithreaded capabilities and speed of execution allow for a more complicated software system, with more on-board processing capabilities. The processes themselves are also modeled as a real time system to handle multiple tasks and information sharing across the DANDE system, as will be described below. DANDE has 13 user processes that have been written to perform mission critical tasks. It contains an abbreviated list of about 50 Linux busybox commands and system diagnostic logging tools. It also has 10 operational modes, all of which perform run at least 9 of these processes at one time. The other 4 processes only run during specific mode operations, such as science collection, or attitude control.

1. Busmessenger and the Message Queue Handler
2. Datacollection
3. 3. NMS Datacollection and Commanding
4. Data Processing
5. Process and Subsystem Watchdogs
6. Attitude Control
7. Mode Scheduling
8. Com-CDH

The core benefit of real-time operating systems lies in how quickly and predictably it can respond to a large number of tasks. This is the more commonly used structure on large and complex spacecraft projects, such as Hubble. With the preemptive kernel structure, as implemented in DANDE, tasks are called specifically when needed, rather than in a serial loop like in a control loop based embedded operating system. This allows for better program flow and event response for more time critical tasks. The speed of the handling execution of task scheduling allows for multiple threaded applications to be run seemingly at once. This way, DANDE can be collecting data, housekeeping subsystem parameters, processing data, downlinking data to the ground and controlling attitude at one time with one shared resource.

Hurdles of DANDE’s RTOS Structure:-

1. Memory Leaks and Logging Implementation
2. Listing Command (ls)
3. Message Queue Concurrency
4. . Un-throttled Infinite Loops
5. The Zip Command

The implementation of a RTOS on a small satellite platform comes with an extensive series of design challenges, which if not considered, can cause major problems in system functionality and stability. Integration and final testing proved to take a significantly longer time than originally anticipated. The added complexity of the RTOS produced issues that were unexpected and extremely difficult to identify. In order for this implementation to be successful, the decision to run the RTOS must be made early and designed this way from the bottom up. If the problems we had seen during integration had been fully considered during the earlier development process, a lot of these issues could have potentially been more manageable during the final phases