Chapter Contents

Glossary

(ADC/DAC)	Analog to Digital/Digital to Analog
(API)	Application Programming Interfaces
(ASICs)	Application Specific Integrated Circuits
(ASIST)	Advanced Spacecraft Integration and System Test
(BSPs)	Broadband Service Providers
(C&DH)	Command and Data Handling
(CCD)	Charge Couple Devices
(CCSDS)	Consultative Council for Space Data Systems
(cFE)	core Flight Executive
(cFS)	core Flight System
(CI)	Continuous Integration
(CM)	Configuration Management
(CMOS)	Complementary Metal Oxide Semiconductors
(COTS)	Commercial-off-the-Shelf
(CRAM)	Chalcogenide Random Access Memory
(CRC)	Cyclic Redundancy Check
(DRAM)	Dynamic Random Access Memory
(ECC)	Error-Correcting Code
(EDAC)	Error Detection and Correction
(EPS)	Electrical Power System
(ES)	Executive Services
(ESSI)	Enhanced Synchronous Serial Interface
(FEC)	Forward Error Correction
(FERAM)	Ferro-Electric Random Access Memory
(FPGA)	Field Programmable Gate Arrays
(FSW)	Flight Software
(GPIO)	General Purpose Input/Output
(GPUs)	Graphics Processor Units
(HIL)	Hardware-in-the-Loop
(I/O)	Input & Output
(I&T)	Integration and Testing
(IMA)	Integrated Mission Architectures

(IoT)	Internet of Things
(ITOS)	Integrated Test and Operations System
(ITOS)	Integrated Test and Operations System
(LVDS)	Low-Voltage Differential Signaling
(MarCO)	Mars Cube One
(MBSE)	Model-Based Systems Engineering
(MRAM)	Magnetoresistive Random Access Memory
(NMF)	NanoSat MO Framework
(NMF)	NanoSat MO Framework
(OSAL)	Operating System Abstraction Layer
(PCM)	Phase Change Memory
(PIL)	Processor-in-the-loop
(PSA)	Payload and Subsystems Avionics
(PZT)	Lead-Zirconium-Titanium Oxide
(rad-hard)	radiation-hardened
(RAM)	Random Access Memory
(ROS)	Robot Operating System
(RTEMS)	Real-Time Executive for Multiprocessor Systems
(RTOS)	Real Time Operating System
(SCFW)	SpaceCloud Framework
(SDK)	Software Development Kit
(SDR)	Software Defined Radios
(SEEs)	Single Event Effects
(SEL)	Single Event Latch-up
(SEU)	Single Event Upsets
(SMP)	Symmetric Multiprocessing
(SRAM)	Static Random Access Memory
(SSA)	Small Spacecraft Avionics
(SWaP)	Size, Weight and Power
(TID)	Total Ionizing Dose
(TMR)	Triple Modular Redundancy
(USB)	Universal Serial Bus

8.0 Small Spacecraft Avionics

8.1 Introduction

Small Spacecraft Avionics (SSA) are described as all electronic subsystems, components, instruments, and functional elements included in the spacecraft platform. These include primarily flight sub-elements Command and Data Handling (C&DH), Flight Software (FSW), and other critical flight subsystems, including Payload and Subsystems Avionics (PSA). All must be configurable into specific mission platforms, architectures, and protocols, and be governed by appropriate operations concepts, development environments, standards, and tools. The C&DH and FSW are the brain and nervous system of the integrated avionics system, and generally provide command, control, communication, and data management interfaces with all other subsystems in some manner, whether in a direct point-to-point, distributed, integrated, or hybrid computing mode. The avionics system is essentially the foundation for all components and their functions integrated on the spacecraft. As the nature of the mission influences the avionics architecture design, there is a large degree of variability in avionics systems.

Traditional spacecraft avionics have been designed around centralized architectures where each subsystem relies on a single processor whereby if one element fails, then the entire architecture commonly fails. This design often results in heavy weight, high power consumption, large volume, complex interfaces, and weak system reconfiguration capabilities. An open, distributed, and integrated avionics architecture with modular capability in software and hardware design is becoming more appealing for complex spacecraft development needs. In anticipation of extended durations in low-Earth orbit and deep space missions, vendors are now incorporating radiation hardened or radiation-tolerant architecture designs in their small spacecraft avionics packages to further increase their overall reliability. Figure 8.1 illustrates the general functional construct and distribution of a centralized small spacecraft system.

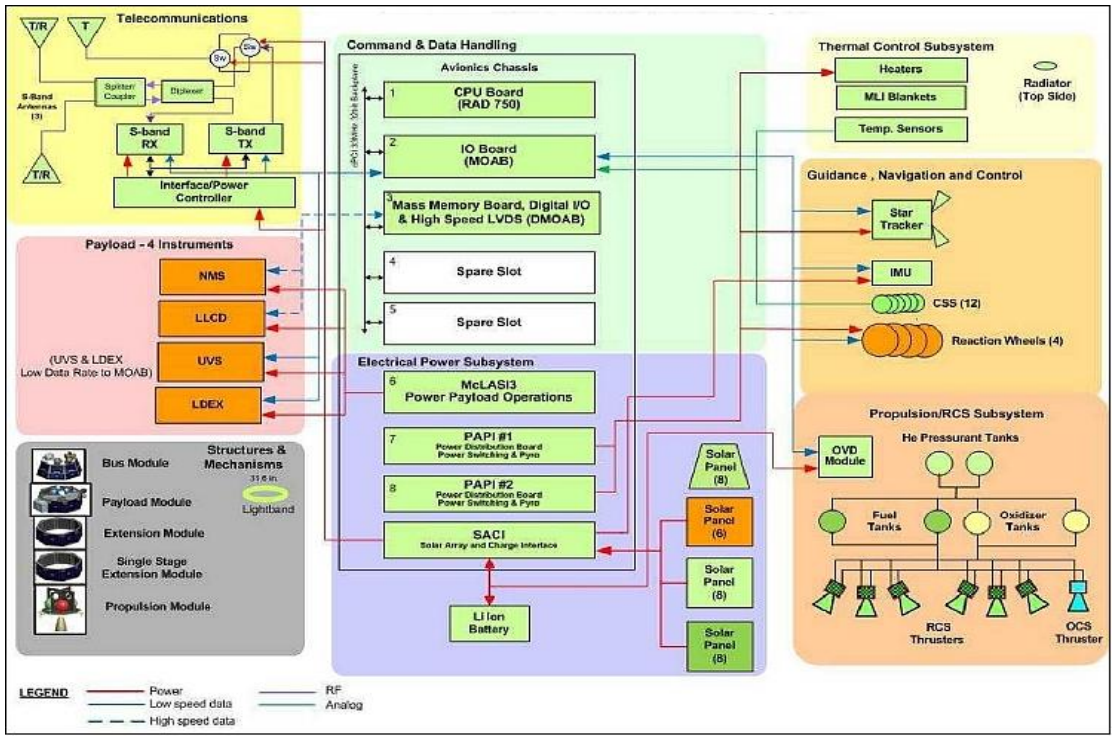

Figure 8.1: Functional block diagram of the LADEE spacecraft. Credit: NASA.

As new generation avionics systems will integrate most of the electronic equipment on the spacecraft, an avionics system designed with networked real-time multitasking distributed system software, which can also implement dynamic reconfiguration of functions and task scheduling and improves the failure tolerance may minimize the need for expensive radiation-hardened electronic components. The improved avionics composition can include high-performance computing hardware to handle the large amount of anticipated data generated by more complex small spacecraft; embedded system software networked for real-time multitasking distributed system software; and software partition protection mechanisms. Some systems now implement a heterogeneous architecture in mixed criticality configurations, meaning they contain multiple processors with varying levels of performance and capabilities.

An example of new generation SSA/PSA distributed avionics application is the integration of Field Programmable Gate Arrays (FPGA)-based software defined radios (SDR) on small spacecraft. A software defined radio can transmit and receive in widely different radio protocols based on a modifiable, reconfigurable architecture, and is a flexible technology that can "enable the design of an adaptive communications system." This can enable the small spacecraft to increase data throughput and provides the ability for software updates on-orbit, also known as re-programmability. Additional FPGA-based functional elements include imagers, AI/ML processors, and subsystem-integrated edge and cloud processors. The ability to reprogram sensors or instruments while on-orbit have benefited several CubeSat missions when instruments do not perform as anticipated, or they enter into an extended mission and subsystems or instruments need to be reprogrammed quickly. Figure 8.2 is a block diagram of an FPGA-based SDR system.

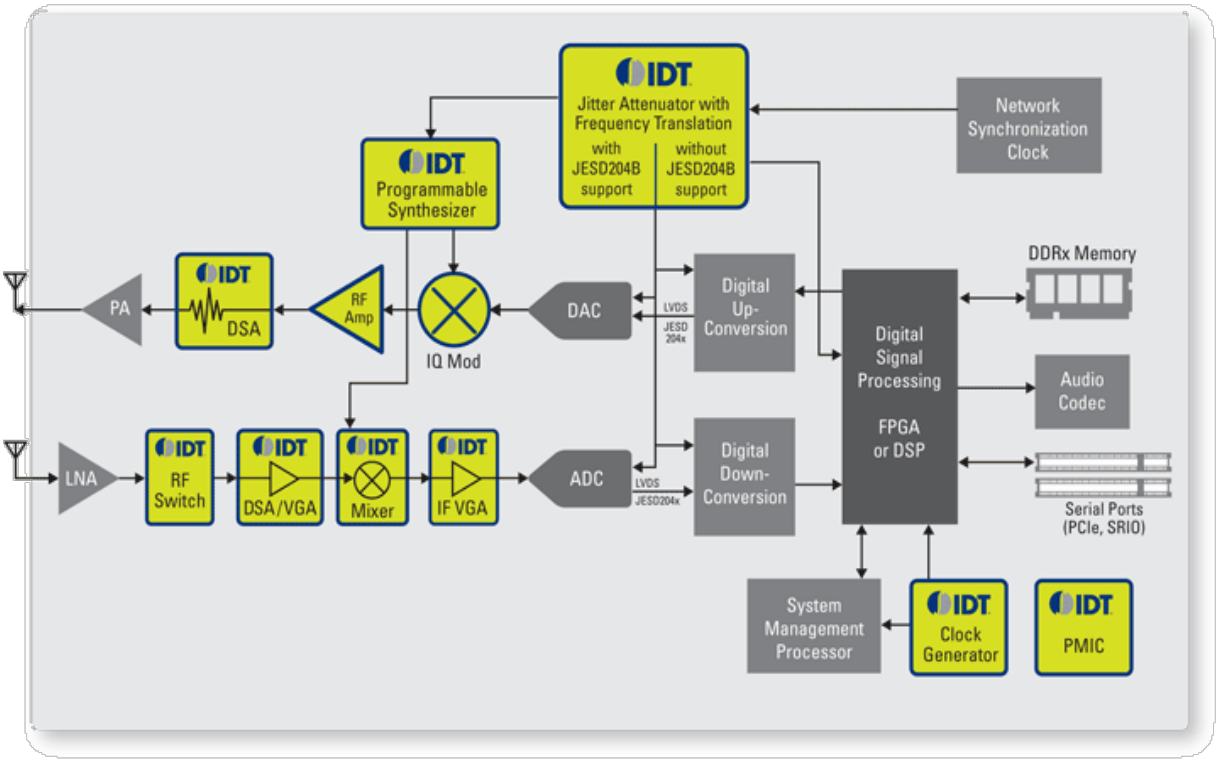

Figure 8.2: Functional block diagram example of a Software Defined radio. Credit: Renesas (IDT is now part Renesas).

8.2 Chapter Scope and Organization

This chapter updates and organizes state-of-the-art SSA by combining and integrating two previously separate chapters, C&DH and FSW. Given the distributed and integrated nature of

modern, small spacecraft avionics, flight payload and subsystems avionic elements are also included, but only describes the avionics (electronics) elements, while the more detailed small spacecraft subsystems specifics are discussed in their respective chapters.

The chapter organizes the state-of-the-art in small spacecraft avionics into C&DH (8.3) and FSW (8.4), where modern requirements of avionics systems are identified to meet the need of complex small spacecraft systems, and the technological progression of SSA systems avionics architecture and composition is expanded upon. Some of the challenges this technology may encounter are also identified, and on the horizon activities (TRL <5) (8.5) highlight the development of new generational small spacecraft avionics systems. Finally, a summary discussion of Avionics Systems Platform and Mission Development Considerations (8.6) is provided that discusses how these considerations are being addressed and/or mitigated by state-of-the-art advances in C&DH FSW, and Payload/Subsystems avionics products, and some projections for future Small Spacecraft Avionic systems (8.7).

The information described below is not intended to be exhaustive but provides an overview of current state-of-the-art technologies and their development status. It should be noted that Technology Readiness Level (TRL) designations may vary with changes specific to payload, mission requirements, reliability considerations, and/or the environment in which performance was demonstrated. Readers are highly encouraged to reach out to companies for further information regarding the performance and TRL of described technology. There is no intention of mentioning certain companies and omitting others based on their technologies or relationship with NASA.

8.3 State-of-the-Art (TRL 5-9): Command and Data Handling

Current trends in small spacecraft C&DH generally appear to be following those of previous, larger scale C&DH subsystems. The current generation of microprocessors can easily handle the processing requirements of most C&DH subsystems and will likely be sufficient for use in spacecraft bus designs for the foreseeable future. Cost and availability are likely primary factors for selecting a C&DH subsystem design from a given manufacturer. The ability to spread non-recurring engineering costs over multiple missions, and to reduce software development through reuse, are desirable factors in a competitive market. Heritage designs are desirable for customers looking to select components with proven reliability for their mission. This C&DH Section is organized as follows:

1. Avionics and on-board computing form factors
2. Highly Integrated On-Board Computing Products
3. Radiation-Hardened Processors and FPGAs
4. Memory, Electronic Function Blocks, and Components
5. Bus Electrical Command and Data Interfaces
6. Radiation Mitigation and Tolerance Schemes

As small satellites move from the early CubeSat designs with short-term mission lifetimes to potentially longer missions, radiation tolerance also comes into play when selecting parts. These distinguishing features, spaceflight heritage and radiation tolerance, are the primary differentiators in the parts selection process for long-term missions, verses those which rely heavily on commercial-off-the-shelf (COTS) parts. Experimental missions typically focused on low-cost, easy-to-develop systems that take advantage of open-source software and hardware provide an easy entry into space systems development, especially for hobbyists or those who lack specific spacecraft expertise.

Small spacecraft C&DH technologies and capabilities have been continuously evolving, enabling new opportunities for developing and deploying next-generation small spacecraft avionics. When small spacecraft were first introduced, a primary purpose was to observe and send information back to Earth. As awareness and utility has expanded, there is a need to improve the overall capability of collecting data in a specific mission environment. Small spacecraft, including nanosatellites and CubeSats, currently perform a wide variety of science in low-Earth orbit and these smaller platforms are emerging as platform candidates for more formidable missions beyond low-Earth orbit.

Since the publication of the earlier editions of this report, several CubeSats using COTS components and integrated systems have successfully flown in the low-Earth orbit environment with short mission durations of typically less than one year. As an example of open-source compilation resources, the Nanosatellite Database (https://www.nanosats.eu) describes itself as the “World’s largest database of nanosatellites, [with] over 3000 nanosats and CubeSats…”.

Significant differences in mission requirements between short-term experimental missions and long-term high reliability missions can impact how state-of-the-art is perceived for flight units. As CubeSats become larger and SmallSats become smaller, technology maturation and miniaturization will further increase capabilities. The Mars Cube One (MarCO) mission was the first CubeSat to operate in deep space, and in late 2021 Artemis I will release seven 6U spacecraft into lunar orbit, and five 6U spacecraft that will demonstrate a variety of technologies in deep space. Although not technically a spacecraft, the Mars Helicopter Ingenuity successfully integrated and demonstrated the use of COTS hardware and open-source software during its successful technology demonstration as a component of the NASA Perseverance Mars Rover mission currently in operation on the Mars surface.

As spacecraft manufacturers begin to use more space qualified parts, they find that those devices can often lag their COTS counterparts by several generations in performance but may be the only means to meet the radiation requirements placed on the system. Presently there are several commercial vendors who offer highly integrated systems that contain the on-board computer, memory, electrical power system (EPS), and the ability to support a variety of Input & Output (I/O) for the CubeSat class of small spacecraft. A variety of C&DH developments for CubeSats have occurred due to in-house development by new companies that specialize in CubeSat avionics, and the use of parts from established companies who provide spacecraft avionics for the space industry in general. While parallel developments are impacting the growth of CubeSats, vendors with ties to the more traditional spacecraft bus market are increasing C&DH processing capabilities within their product lines.

In-house designs for C&DH units are being developed by some spacecraft bus vendors to better accommodate small vehicle concepts. While these items generally exceed CubeSat form factors in size, they can achieve similar environmental performance and may be useful in small satellite systems that replicate more traditional spacecraft subsystem distribution. In anticipation of extended durations in low-Earth orbit and deep space missions, vendors are now incorporating radiation hardened or radiation tolerant designs in their CubeSat avionics packages to further increase the overall reliability of their products.

8.3.1 Avionics and On-board Computing Form Factors

The CompactPCI and PC/104 form factors continue generally to be the industry standard for CubeSat C&DH bus systems, with multiple vendors offering components that can be readily integrated into space rated systems. Overall form factors should fit within the standard CubeSat dimension of less than 10 x 10 cm. The PC/104 form factor was the original inspiration to define standard architecture and interface configurations for CubeSat processors. But with space at a premium, many vendors have been using all available space exceeding the formal PC/104 board

size. Although the PC/104 board dimension continues to inspire CubeSat configurations, some vendors have made modifications to stackable interface connectors to address reliability and throughput speed concerns. Many vendors have adopted the use of stackable "daughter" or "mezzanine" boards to simplify connections between subsystem elements and payloads, and to accommodate advances in technologies that maintain compatibility with existing designs. A few vendors provide a modular package which allows users to select from a variety of computational processors.

The form factors used in more traditional spacecraft designs frequently follow "plug into a backplane" VME standards. 3U boards offer a size (roughly 100 x 160 mm) and weight advantage over 6U boards (roughly 233 x 160 mm) if the design can be made to fit in the smaller form factor. It should be noted that CubeSats also use "U" designations, but these refer to the volume of the vehicle based on initial CubeSat standards of 1U (100 x 100 x 100 mm), 3U (100 x 100 x 300 mm), and 6U (100 x 200 x 300 mm). Some small spacecraft bus designers consider using just a single board C&DH unit as a means of saving weight.

A hybrid form factor configuration is that of the Qseven Computer-on-Module (Q7). “The Qseven concept is an off-the-shelf, multi-vendor, Computer-On-Module that integrates all the core components of a common PC and is mounted onto an application specific carrier board. Qseven modules have a standardized form factor of 70 x 70 mm or 40 x 70 mm and have specified pinouts based on the high speed MXM system connector that has a standardized pinout regardless of the vendor. The Qseven module provides the functional requirements for an embedded application. These functions include, but are not limited to, graphics, sound, mass storage, network and multiple USB ports. A single ruggedized MXM connector provides the carrier board interface to carry all the I/O signals to and from the Qseven module. This MXM connector is a well-known and proven high speed signal interface connector that is commonly used for high-speed PCI Express graphics cards in notebooks (https://sget.org/standards/qseven/).

8.3.2 Highly Integrated On-Board Computing Products

A variety of vendors are producing highly integrated, modular, on-board computing systems for small spacecraft. These C&DH packages combine microcontrollers and/or FPGAs with various memory banks, and with a variety of standard interfaces for use with the other subsystems on board. The use of FPGAs and software-defined architectures also gives designers a level of flexibility to integrate uploadable software modifications to adapt to new requirements and interfaces. Table 8-1 summarizes the current state-of-the-art of these components. Since traditional CubeSat designs are based primarily on COTS parts, spacecraft vendors often try to use parts that have radiation tolerance or have been radiation-hardened (rad-hard), as noted in the pedigree column in table 8-1. The vehicle column shows which spacecraft classification corresponds to each on-board unit; "general satellite" classification refers to larger SmallSat platforms (i.e., larger than CubeSats). It should be noted that while some products have achieved TRL 9 by virtue of a space-based demonstration, what is relevant in one application may not be relevant to another, and different space environments and/or reliability considerations may result in lower TRL assessments. Some larger, more sophisticated computing systems have significantly more processing capability than what is traditionally used in SmallSat C&DH systems, however the increase in processing power may be a useful tradeoff if payload processing and C&DH functions can be combined (note that overall throughput should be analyzed to assure proper functionality under the most stressful operating conditions).

Table 8-1: Sample of Highly Integrated On-board Computing Systems						
Manufacturer	**Product**	**Processor**	**Pedigree**	**Vehicle**	**TRL**	**Citation**
GomSpace	Nanomind A3200	Atmel AT32UC3C MCU	COTS	CubeSat	Ukn	(1)
ISISpace	iOBC	ARM 9	COTS	CubeSat	9	(2)
Pumpkin	PPM A1	TI MSP430F1612	COTS	CubeSat	9	(3)
	PPM A2	TI MSP430F1611	COTS	CubeSat	9	
	PPM A3	TI MSP430F2618	COTS	CubeSat	9	
	PPM B1	Silicon Labs C8051F120	COTS	CubeSat	9	
	PPM D1	Microchip PIC24FJ256GA110	COTS	CubeSat	9	
	PPM D2	Microchip PIC33FJ256GP710	COTS	CubeSat	9	
	PPM E1	Microchip PIC24FJ256GB210	COTS	CubeSat	9	
Xiphos	Q7S	Xilinx Zynq 7020 Arm 9	COTS w/SEE mitigation	Nano-, Micro- and SmallSats	9	(4)
	Q8S	Xilinx Ultrascale+ ARM Cortex-A53	COTS w/SEE mitigation	Nano- Micro- and SmallSats	8	(5)
BAE	RAD750	RAD750	rad-hard	General Satellite	9	(6)
	RAD5545	RAD5545	rad-hard by design	General Satellite	Ukn	(7)
AAC Clyde Space	Kryten-M3	SmartFusion Cortex-M3	COTS	CubeSat	Ukn	(8)
	Sirius OBC	SmartFusion Cortex-M3	COTS w/SEE mitigation	CubeSat	Ukn	(9)

Innoflight	cfc-300	Xilinx Zynq ARM Cortex A9	COTS	CubeSat	Ukn	(10)
	cfc-400	Xilinx Zynq Ultrascale+	COTS	CubeSat	Ukn	(11)
	cfc-500	Xilinx Kintex Ultrascale+ NVIDIA TK1	COTS	CubeSat	Ukn	(12)
Space Micro	CSP	Xilinx Zynq-7020 Dual ARM Core	COTS	CubeSat	Ukn	(13)
NanoAvionics	SatBus 3C2	STM32 ARM Cortex M7	COTS	CubeSat	9	(14)
MOOG	G-Series Steppe Eagle	AMD G-Series compatible	Rad Hard by design	General Satellite	Ukn	(15)
	V-Series Ryzen	AMD V-Series compatible	Rad Hard by design	General Satellite	Ukn	
SEAKR	Athena-3 SBC	PowerPC e500	Ukn	General Satellite	9	(16)
	Medusa SBC	PowerPC e500	Ukn	General Satellite	9	
	RCC5	Virtex 5 FX-130T	Ukn	General Satellite	9	
Unibap	iX10-100	Microchip PolarFire FPGA with RISC-V , AMD V1605b (Ryzen) CPU and GPU, and up to 3 Intel Movidius Myriad X VPUs and optional NVMe based compute storage (up to 8 TB)	COTS with SEE mitigation	Nano-, Micro- and SmallSats	5	(41)
	iX5-100	Microchip SmartFusion2 ARM Cortex-M3 and AMD G-Series SOC	COTS with SEE mitigation	Nano-, Micro- and SmallSats	8	(19)
	e2160	Microchip SmartFusion2 FPGA with ARM Cortex-M3 and AMD 2nd	COTS with SEE mitigation	Nano-, Micro- and SmallSats	9	(18)

		generation G-Series SOC CPU and GPU				
	e2155	Microchip SmartFusion2 FPGA with ARM Cortex-M3 and AMD 1st generation G-Series SOC CPU and GPU	COTS with SEE mitigation	Nano-, Micro- and SmallSats	9	(18)
Ibeos	EDGE Computer	Nvidia TK1	COTS with Single Event Effects and TID Characterization. Radiation-tolerant with single event effects mitigation	CubeSat, general SmallSat, ESPA-class satellite and larger	6	(42)
DDC	SCS750 series SBC	IBM 750FX	Rad-hard Sp-COTS	General SmallSats	9	(43)
	SCS3740	GR740 Rad-Hard quad-core LEON 4FT®	Rad-hard Sp-COTS	General SmallSats, NanoSats	6	(44)
EnduroSat	OBC	ARM Cortex-M7	COTS	CubeSats	9	

System developers are gravitating towards ready-to-use hardware and software development platforms that can provide seamless migration to higher performance architectures. As with non-space applications, there is a reluctance to change controller architectures due to the cost of retraining and code migration. Following the lead of microcontrollers and FPGA vendors, CubeSat avionics vendors are now providing simplified tool sets and basic, cost-effective evaluation boards.

Two such example units have been identified which may be able to support small satellite designs beyond the CubeSat form factors (see table 8-2). Spacecraft bus vendors may also have preferred sources for C&DH units, such as those developed in-house.

Table 8-2: Sample of Small C&DH Units

Vendor	Unit	Mass (kg)	Power (W)	Processor	MIPs	References
Moog	C&DH Avionics unit	< 3	25	BRE 440	266	(17)
SEAKR	C&DH Avionics unit	5.4	14	LEON	25	(16)

8.3.3 Radiation-Hardened Processors and FPGAs

The FPGA functions as the Main Control Unit, with interfaces to all functional subcomponents of a typical C&DH system. This then enables embedded, adaptive, and reprogrammable capabilities in modular, compact form factors, and provides inherent architectural capabilities for processor emulation, modular redundancies, and “software-defined-everything.”

Several radiation-hardened embedded processors have recently become available. These are being used as the core processors for a variety of purposes including C&DH. Some of these are the Vorago VA10820 (ARM M0) and the VA41620 and VA41630 (ARM M4); Cobham GR740 (quad core LEON4 SPARC V8) and the BAE 5545 quad core processor. These have all been radiation tested to at least 50 kRad total ionizing dose (TID).

Xilinx and Microchip (formerly Microsemi), leaders in the space-grade FPGA market, have both released new radiation-tolerant FPGA families in the past two years rated to 100 kRad TID. The Xilinx RT Kintex UltraScale, a 20 nm device, has 726 k logic cells and supports 12.5 Gbps serial data transmission. The Microchip RT PolarFire is a 28 nm device with 481k logic cells and up to 10.3125 Gbps data transmission. These both offer far more capability than either company's previous families of rad-tolerant FPGA (Xilinx Virtex-5 and Microchip RTG4) and may be adopted for more complex payload data processing needs than merely C&DH use. The Kintex UltraScale is integrated within the Innoflight CFC-500 and Moog Steppe Eagle and Ryzen, listed in the table above.

8.3.4 Memory, Electronic Function Blocks, and Components

The range of on-board memory for small spacecraft is wide, typically starting around 32 KB and increasing with available technology. For C&DH functions, on-board memory requires high reliability. A variety of different memory technologies have been developed for specific traits, including Static Random Access Memory (SRAM), Dynamic RAM (DRAM), flash memory (a type of electrically erasable, programmable, read-only memory), Magnetoresistive RAM (MRAM), Ferro-Electric RAM (FERAM), Chalcogenide RAM (CRAM) and Phase Change Memory (PCM). SRAM is typically used due to price and availability. A chart comparing the various memory types and their performance is shown in table 8-3.

Table 8-3. Comparison of Memory Types

Feature	SRAM	DRAM	Flash	MRAM	FERAM	CRAM/ PCM
Non-volatile	No	No	Yes	Yes	Yes	Yes
Operating Voltage, ±10%	3.3 – 5 V	3.3 V	3.3 & 5 V	3.3 V	3.3 V	3.3 V
Organization (bits/die)	512 k x 8	16 M x 8	16 M x 8; 32 M x 8	128 k x 8	16 k x 8	Unk
Data Retention (@ 70°C)	N/A	N/A	10 years	10 years	10 years	10 years
Endurance (Erase/Write cycles)	Unlimited	Unlimited	10^6	1013	1013	1013
Access Time	10 ns	25 ns	50 ns after page ready; 200 s write; 2 ms erase	300 ns	300 ns	100 ns
Radiation (TID)	1 Mrad	50 krad	30 krad	1 Mrad	1 Mrad	1 Mrad
SEU rate (relative)	Low-nil	High	Nil (cells); Low (device electronics)	Nil	Nil	Nil
Temperature Range	Mil-std	Industrial	Commercial	Mil-std	Mil-std	Mil-std
Power	500 mW	300 mW	30 mW	900 mW	270 mW	Unk
Package	4 MB	128 MB	128 – 256 MB	1 MB	1.5 MB (12 chip package)	Unk

There are many manufacturers that provide a variety of electronic components that have high reliability and are space rated (see table 8-4 for a noncomprehensive list). A visit to any of their respective websites will show their range of components and subsystems including processors, FPGAs, SRAM, MRAM, bus interfaces, Application Specific Integrated Circuits (ASICs), and Low-Voltage Differential Signaling (LVDS).

Table 8-4: Sample of Space-Rated Electronics Manufacturers		
Apogee Semiconductor (USA)	Honeywell (USA)	STMicroelectronics (Switzerland)
BAE Systems (UK)	Intel (USA)	Texas Instruments (USA)
Moog Broad Reach (USA)	Renesas (Japan)	3D Plus (USA)
Space Micro, Inc. (USA)	SEAKR (USA)	Xilinx (USA)
Cobham (Aeroflex, Gaisler) (Sweden)	Microchip (USA)	Vorago Technologies (USA)
Data Device Corporation (USA)		

8.3.5 Bus Electrical Interfaces

CubeSat class spacecraft continue to use interfaces that are common in the microcontroller or embedded systems world. Highly integrated systems, especially SoC, FPGA and ASICs, will typically provide several interfaces to accommodate a wide range of users and to ease the task of interfacing with peripheral devices and other controllers. FPGAs are commonly used for these interfaces because of their flexibility and ability to change interfaces if needed. Some of the most common bus electrical interfaces are listed below with a brief description of applicable interface standards:

- Serial Communication Interfaces (SCI): RS-232, RS-422, RS-485 etc.
- Synchronous Serial Communication Interface: I2C, SPI, SSC and ESSI (Enhanced Synchronous Serial Interface)
- Multimedia Cards (SD Cards, Compact Flash etc.)
- Networks: Ethernet, LonWorks, etc.
- Fieldbuses: CAN Bus, LIN-Bus, PROFIBUS, etc.
- Timers: PLL(s), Capture/Compare and Time Processing Units
- Discrete IO: General Purpose Input/Output (GPIO)
- Analog to Digital/Digital to Analog (ADC/DAC)
- Debugging: JTAG, ISP, ICSP, BDM Port, BITP, and DB9 ports
- SpaceWire: a standard for high-speed serial links and networks
- High-speed data: RapidIO, XAUI, SERDES protocols are common in routing large quantities of mission data in the gigabit per second speeds

8.3.6 Radiation Mitigation and Tolerance Schemes

Deep space and long-duration low-Earth orbit missions will require developers to incorporate radiation mitigation strategies into their respective designs. The CubeSat platform has traditionally used readily available COTS components. Use of COTS parts has allowed for low-cost C&DH development, while also allowing developers to take advantage of state-of-the-art technologies in their designs. Many of the component and system vendors also provide radiation hardened (rad-hard) equivalent devices as well. While there are many commercially available rad-hard components, using these components impacts the overall cost of spacecraft development. To

keep costs as reasonable as possible, C&DH developers will need to address appropriate use of rad-hard components, along with other radiation mitigation techniques for developing an overall radiation tolerant design as discussed in the following section.

For space applications, radiation can damage electronics in two ways. TID is the amount of cumulative radiation received, and single event effects (SEEs) are disturbances created by single particles hitting the electronics (20). Total dose is measured in kilorads and can affect transistor performance. Single Event Upsets (SEU) can affect the logic state of memory. A Single Event Latch-up (SEL) can affect the output transistors on Complementary Metal Oxide Semiconductors (CMOS) logic, potentially causing a high-current state.

This section summarizes techniques used to mitigate system failures caused by radiation effects. C&DH element areas of consideration include: memory, imaging, protection circuits (watchdog timers, communications watchdog timers, overcurrent protection, and power control), memory protection (error-correction code memory and software error detection and correction), communication protection (several components), and parallel processing and voting.

Memory

FRAM is a non-volatile random-access memory that is persistent like Flash memory. FRAM memory cells are latched using a Lead-Zirconium-Titanium oxide (PZT) film structure, which is more likely to maintain state during a single event effect than traditional capacitive latches found in RAM (21) (22).

MRAM is another type of non-volatile random-access memory that is persistent. It is different than FRAM and others in that it has virtually unlimited read and write cycle endurance. MRAM has been built into some processors (TI MSP430FR) as well as separate chips.

Imaging

Charge Couple Devices (CCD) and CMOS are image sensors that are useful in radiation environments. However, CCDs are preferred in space applications, while the CMOS detectors are a newer technology for rad hardened image sensors (23) (24) (25) (26).

Protection Circuits

Watchdog Timers

Watchdog timers are often used to monitor the state of a processor. A watchdog timer is a hardware circuit, external or internal to the processor, which resets the processor when the timer expires unless refreshed by the processor. If the processor jumps to an erroneous memory location through a SEU or a software exception, the watchdog timer resets the processor to restore operations.

Communication Watchdog Timer

A dedicated communication watchdog timer circuit can monitor commands and responses to determine if the system is locked up. Such a circuit resets power after a specific number of failed transmissions.

Overcurrent Protection

Single Event Latch-up (SEL) can cause device failure due to an elevated current state. Hardware and software overcurrent protection can be implemented to watch for elevated current levels and then issue a power reset to the offending circuit. The sampling frequency for software overcurrent protection must be sufficient to detect and reset the subsystem before the elevated current causes

permanent damage. For hardware protection, a shunt resistor and bypass diode can be used in conjunction to filter voltage and current spikes for rad hardened devices.

Power Control

Since many components are more prone to radiation effects when powered on, a candidate mitigation strategy is to power off devices when they are not operationally needed.

Memory Protection

Error-Correcting Code Memory

Error-Correcting Code (ECC) memory is capable of detecting and correcting bit errors in RAM and flash memory. In general, ECC works by storing a checksum for a portion of the memory. This checksum can be used to simply mark a portion of memory unstable. Additional processing can use the memory and checksums to correct single and sometimes multi-bit errors. The memory controller is responsible for managing the ECC memory during read and write operations (30).

Software Error Detection and Correction

Bit errors can be detected and corrected using software. In general, Error Detection and Correction (EDAC) algorithms use Hamming codes or three copies of the memory to detect and correct bit discrepancies. Software routinely "scrubs" the memory, compares each of the three stored memory values, selects the majority value, and corrects the erroneous memory location. Software EDAC can be performed at the bit or byte level. Memory lifetime needs to be considered for software EDAC implementations, since every correction increases the write count to a memory location.

Communication Protection

Shared Bus Switching

Another option is to decouple the clock and data lines so that each peripheral has its own pair. Additional data lines can be used on the master controller. Alternatively, an external FPGA could be used to assign a unique clock/data pair to each peripheral and, optionally, include a method to reconfigure those assignments in flight.

Cyclic Redundancy Check

Cyclic Redundancy Check (CRC) is a common method for detecting memory or communication errors. Parity is a single-bit implementation of a CRC where the bit of summary information is calculated by the XOR of the data to be communicated or stored to memory. For communication channels, a CRC is calculated prior to sending the message, and is appended to the message stream in a known location. When the message is received, the CRC is calculated again and compared to the previously generated CRC appended to the data stream. For memory, the CRC is calculated prior to writing the data to memory. When the data is read out, a new CRC is calculated and compared to the previously generated CRC. CRCs help detect data corruption but cannot be used to correct the defective data.

Forward Error Correction

Forward Error Correction (FEC) transmits redundant data to help the receiver recover corrupted data. In its simplest form, FEC could transmit three bits for every bit of data and then vote to

restore the original data. More efficient algorithms balance the data overhead with the correction accuracy (27).

Parallel Processing and Voting

Triple Modular Redundancy

SEU can interrupt discrete logic, including processing. Triple Modular Redundancy (TMR) is a fault mitigation technique where logic is replicated three times, and the output of the logic is determined by a majority vote.

Firmware Protection

Many spacecraft subsystems include a processor to handle and optimize operations. These processors require firmware which is written into onboard program memory. Like data memory, program memory is also susceptible to single-event upsets and device failure. To counter this issue, a bootloader may be used to check the validity of the firmware and provide a mechanism for uploading new versions. Additionally, multiple copies of the firmware may be stored in memory in case the primary version is corrupt.

8.4 State-of-the-Art (TRL 5-9): Flight Software

The FSW is, at a fundamental level, the instructions for the spacecraft to perform all operations necessary for the mission. These include all the science objectives as regular tasks (commands) to keep the spacecraft functioning and ensure the storage and communication of data (telemetry). The FSW is usually thought of as the programs that run on the C&DH avionics but should also include all software running on the various subsystems and payload(s).

There are many factors in the selection of a development environment and/or operating system used for a space mission. A major factor is the amount of memory and computational resources. There are always financial and schedule concerns. Another factor is what past software an organization may have used and their experiences with that software. Also, the maturity of the software as well as its availability on the target are additional factors to be considered in the final selection.

Flight Software complexity refers to the amount of operations to be performed and is not based on the size of the spacecraft, only the overall requirements and mission objectives. The more software is required to do, the bigger the task and cost. This complexity is what primarily drives the cost and schedule for the program or mission. Required reliability and fault management can also increase complexity and cost, regardless of the size of the spacecraft.

With the increase in processing capability with C&DH and other processors, more capabilities have been enabled with FSW. Previously, larger processors have only been in larger spacecraft and would not be possible in CubeSats and MicroSats. There have been several advances that make more processing capability now available for CubeSats. Low-power ARM-based processors, as well as advances in radiation hardened processors, have brought similar processing capabilities down to the small size of CubeSats. All of this has brought increased demands and requirements on FSW.

FSW must operate in a real-time environment. This definition can have numerous interpretations. Generally, C&DH and other subsystems need to be able to supervise several inputs and outputs as well as process and store data within a fixed time-period. These all need to be performed in a reliable and predictable fashion throughout the lifetime of the mission. The needs of each mission can vary greatly, but this basic deterministic and reliable processing is a fundamental requirement.

8.4.1 Implication of C&DH Processors on FSW

The processor and memory available on the C&DH can put significant limitations on the FSW. For some of the smaller jobs, or to reduce electronic complexity, smaller processors are used. These have typically been thought of as embedded processors, with many of them containing dedicated memory. Modern integrated space avionics, including heterogeneous and mixed criticality architectures, also impact the nature and operational constructs, and can contribute to advanced configurations such as Multiple Modular Redundant systems architectures which can allow advanced paradigms for radiation tolerance and system redundancies in critical small spacecraft missions.

Software code and programs are very integrated with the hardware, requiring careful implementation and integration. Software development environments for these kinds of processors usually come from the microprocessor themselves, or from third party vendors. Some of the past tools (and processors used) have been MPLAB (Microchip PIC family), and TI CCStudio (TI MSP430). On some of these types of processors a “bare bones” approach to the software design is usually implemented with limited to no operating system. This is primarily because of memory and processor limitations. These programs tend to be highly optimized. Part of the challenge with these systems is development and testing. Most interactions with the software must be done remotely through a secondary processor, usually a PC. This type of development usually requires unique skills and can involve a significant learning curve for developers. Efficient programmers need to have a good understanding of both the software and hardware and how they function together. Timing and performance matter greatly, so that they need to be able to write code in an efficient manner. Typically, these projects have up to 20,000 lines of code.

Larger processors have been increasing in popularity with current missions, especially CubeSats. With increases in power production as well as lower power processors, radiation tolerant processors have been available in both SmallSats and CubeSats. Several vendors have large processors that can run Real-Time Operating Systems (RTOS) such as VxWorks, RTEMS and FreeRTOS that were described earlier. Linux has been used, usually with real-time extensions. In other instances, functionality is distributed between a large capability processor and a smaller dedicated flight controller whereby the controller conducts and manages the real-time aspects, allowing efficient management of power and operational complexity. These give software developers a significant advantage with a software development environment and usually a base implementation on the processing target. RTOS have been designed to operate in minimal processor/memory environments with real-time needs. These projects typically have for small projects 50K to 70K lines of code, to larger projects that can exceed a million.

This FSW Section is organized as follows:

1. Frameworks: In the context of Small Spacecraft Avionics, a Flight Software Framework can be described as a hierarchal systems-of-systems architecture, sometimes described as a set of Lego-like building block constructs, partitions, and functions
2. Operating Systems (OS): System software that manages computer hardware, software resources, and provides common services for computer programming.
3. Software Languages: System programming involves designing and writing computer programs that allow the computer hardware to interface with the programmer and the user, leading to the effective execution of application software on the computer system (Techopedia).

4. Mission Operations Suites: Software and systems used to monitor, control, communicate, and display command, control, status, and data dissemination of all aspects of a space mission, include spacecraft performance and procedures, systems health, science and technology data handling and management, and telemetry tracking and control.
5. Development environment, standards, and tools: the collection of hardware and software systems tools to design, develop, validate, and operate small spacecraft missions, with adherence to accepted software and space mission standards.

8.4.2 Frameworks

cFS – https://cfs.gsfc.nasa.gov

The core Flight System (cFS) is a generic flight software architecture framework. cFS has been used in dozens of space missions ranging from flagship spacecraft to small satellite and CubeSats. cFS is actively being used in a number of missions both in flight and in development. The core Flight Executive (cFE) and Executive Services (ES) are a set of applications, application framework, and runtime environment developed by Goddard Space Flight Center. cFE includes core services like messaging, timekeeping, events, and table-driven commanding and configuration (18). cFS is built on an Operating System Abstraction Layer (OSAL) that leads to the same code base running on different operating systems. cFS provides most of the basic functionality to operate a spacecraft. The core Flight System, as well as supporting infrastructure, has been used by NASA on numerous missions and is being used by other organizations. cFS, as well as the supporting OSAL, are open-source and currently released under the Apache 2.0 license (29).

F' – https://github.com/nasa/fprime

F' is a software framework for rapid development and deployment of embedded systems and spaceflight applications. Originally developed at JPL, F´ is open-source software that has been successfully deployed for several space applications. It has been used for, but is not limited to, CubeSats, SmallSats, instruments, and deployables. F' is currently released under the Apache 2.0 license (33, 34). Most recently, F' was used to operate the Mars Helicopter, Ingenuity, along with other open-source SW and HW products.

NanoSat Mission Operations Framework –
https://en.wikipedia.org/wiki/NanoSat_MO_Framework

From the above webpage: "…The NanoSat MO Framework (NMF) is a software framework for nanosatellites based on [Consultative Council for Space Data Systems] (CCSDS) Mission Operations services. It facilitates not only the monitoring and control of the nanosatellite software applications, but also the interaction with the nanosatellite platform. This is achieved by using the latest CCSDS standards for monitoring and control, and by exposing services for common peripherals among nanosatellite platforms. Furthermore, it is capable of managing the software on-board by exposing a set of services for software management.[1]

In simple terms, it introduces the concept of apps in space that can be installed, and then simply started and stopped from the ground. Apps can retrieve data from the nanosatellite platform through a set of well-defined platform services. Additionally, it includes CCSDS standardized services for monitoring and control of apps. An NMF App can be easily developed, distributed, and deployed on a spacecraft. [2]

There is a Software Development Kit (SDK) to help develop software based on the NanoSat MO Framework. This SDK allows quick development of software that can run on ground and/or in

space. The reference implementation of the NanoSat MO Framework will be used in ESA's OPS-SAT mission."

SpaceCloud

The SpaceCloud Framework (SCFW) revolutionizes satellite software development, converting purpose-built, custom space hardware into flexible and reusable compute nodes. This allows for simplified space missions, providing an all-inclusive solution for on-orbit data processing and on the ground management software. Like cloud computing on ground, it also allows orchestration of operations each node (or satellite) will perform, including approving execution of applications and upgrading the SCFW software. SpaceCloud® offers common types of cloud resources such as processors, GPU or dedicated AI accelerators, and in some cases optimized resources in FPGA technology. For storage, the S3 application programming interfaces (API) is compatible with Amazon Web Services. Machine learning and inference can be done with TensorFlow, TVM, PlaidML, OpenVINO / OneAPI, among others.

ROS – https://en.wikipedia.org/wiki/Robot_Operating_System

Taken directly from webpage: "…Robot Operating System (ROS or ros) is an open-source robotics middleware suite. Although ROS is not an operating system but a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor data, control, state, planning, actuator, and other messages. Despite the importance of reactivity and low latency in robot control, ROS itself is not a Real Time Operating System (RTOS). It is possible, however, to integrate ROS with real-time code.[3] The lack of support for real-time systems has been addressed in the creation of ROS 2.0,[4][5][6] a major revision of the ROS API which will take advantage of modern libraries and technologies for core ROS functionality and add support for real-time code and embedded hardware."

8.4.3 Operating Systems

VxWorks

Windriver calls VxWorks the industry-leading RTOS. VxWorks is fully featured and has been used by the industry for many years, and by NASA for over 20 years since the Clementine mission. It is used in satellites as well as robotics such as Robonaut and MER. It has many features of a user operating system with tasks and processes, memory protection and separation. VxWorks has a commercial license, with several advanced development and diagnostic tools licensed separately. Due to the cost, VxWorks needs to be budgeted for the life of the mission.

VxWorks currently supports 32-, 64-bit, and multi-core processors including Intel, Arm, Power Architecture and RISC-V. Multi-core processors support both asymmetric and symmetric multiprocessing. There are numerous board support packages for enabling early prototyping and aiding software development (31).

RTEMS

From the RTEMS.org website: "the Real-Time Executive for Multiprocessor Systems (RTEMS) is an open-source RTOS that supports open standard API such as POSIX. It is used in space flight, medical, networking and many more embedded devices. RTEMS currently supports 18 processor architectures and approximately 200 [broadband service providers] (BSPs). These include ARM, PowerPC, Intel, SPARC, RISC-V, MIPS, and more. RTEMS includes multiple file systems, symmetric multiprocessing (SMP), embedded shell, and dynamic loading, as well as a high-

performance, full-featured IPV4/IPV6 TCP/IP stack from FreeBSD which also provides RTEMS with USB."

RTEMS is considered open-source, released under a modified GNU General Public License. Support is available through the primary manager OAR. It has been sponsored, deployed, and used widely on several NASA and ESA missions. RTEMS has been in development since the 1980s. RTEMS could be considered a simpler operating system with no provided memory or process management. Although build environments are provided, development tools are not as featured as commercial products (32).

FreeRTOS

FreeRTOS is a small, real-time operating system kernel designed for embedded devices. It is open-source and released under the MIT license. FreeRTOS is designed to be small and simple; however, it lacks some of the more advanced features found on larger operating systems. FreeRTOS has been used on several CubeSat projects where memory is limited.

Linux

Linux is another operating system that is being implemented on several spacecraft. Linux is deployed on PowerPC-, LEON-, and ARM-based processors. It is readily available and widely used in both government and commercial sectors. There are several distributions and guides that have been developed for embedded use that would be suitable for spacecraft use. Some of the distributions have been Yocto (Xilinx ZYNQ) and Debian (BeagleBone Black and PowerPC). There are real-time extensions, as well as additional extensions such as Xenomai, to improve critical real-time performance. Numerous development and diagnostic tools are available. Linux is a full featured operating system that has been used for desktop applications. Linux tends to be larger, requiring more memory and processing capability. It is popular on the ARM processors because those issues tend not to be a factor.

Debian – https://en.wikipedia.org/wiki/Debian

Taken directly from webpage: "Debian also known as Debian GNU/Linux, is a Linux distribution composed of free and open-source software, developed by the community-supported Debian. The Debian Stable branch is the most popular edition for personal computers and servers. Debian is also the basis for many other distributions, most notably Ubuntu. Debian is one of the oldest operating systems based on the Linux kernel."

8.4.4 Software Languages

Rather than list details for each listed software language, reference links to examples of relevant software languages are provided from Wikipedia and other sources listed below in table 8-5.

Table 8-5: Relevant Software Languages

Software Language	Wikipedia page
C	https://en.wikipedia.org/wiki/C_(programming_language
C++	https://en.wikipedia.org/wiki/C%2B%2B
Python	https://en.wikipedia.org/wiki/Python_(programming_language
Arduino	https://en.wikipedia.org/wiki/Arduino
Assembly Language	https://en.wikipedia.org/wiki/Assembly_language

8.4.5 Mission Operations and Ground Support Suites

Although not directly used on the spacecraft, operators need a way to talk to the spacecraft, and ground operations and testing need that same capability. For smaller spacecraft and missions, it is usually best to use the same ground support software for these three tasks: mission operations, integration and testing, and development and testing. There are numerous proprietary tools and programs, but a small set of tools that have been used at NASA are described below. For more information, please refer to the Ground Data System and Mission Operations chapter.

Integrated Test and Operations System (ITOS) is a space ground system developed for GSFC by the Hammers Company ITOS (37). It is a comprehensive command and telemetry solution for spacecraft, component, and instrument development, integration, testing, and mission operations. It is highly user configurable, and provides a scalable, cost-effective platform for small-budget projects to billion-dollar observatories. It includes multi-spacecraft control and closed-loop simulation capabilities.

Advanced Spacecraft Integration and System Test (ASIST) is also a space ground system developed for GSFC by Design America, Inc. ASIST provides satellite telemetry and command processing for integration and testing (I&T) and operations environments (38). ASIST is described as “an object-oriented, real-time command and control system for spacecraft development, integration, and operations. Mature and reliable, ASIST has logged hundreds of thousands of hours in component development, spacecraft integration, and validation labs.”

INCONTROL is a proprietary tool developed by L3HARRIS. Some of the features include providing support for single-mission, multi-mission, and constellation support. It also provides capabilities for automation, event logging, data distribution, procedure development, archiving, data displays, equipment monitor and control, data retrieval, report generation, and simulation (39).

COSMOS – https://www.ball.com/aerospace/programs/cosmos

COSMOS is a tool developed by Ball Aerospace that provides a framework for operating and testing an embedded system (40). COSMOS is open-source, licensed under the MIT license. The tool includes modules for telemetry display, plotting, scripting, logging, and configuration table management.

Yamcs – https://yamcs.org

Yamcs is an open-source software framework for command and control of spacecraft, satellites, payloads, ground stations and ground equipment. Yamcs */jæmz/* is open-source software developed by Space Applications Services, an independent Belgian company, with a subsidiary in Houston, USA. The aim of Space Applications Services is to research and develop innovative systems, solutions and products and provide services to the aerospace and security markets and related industries. Activities cover manned and unmanned spacecraft, launch/re-entry vehicles, control centers, robotics and a wide range of information systems. Yamcs is developed around the criteria of flexibility and open-source code to innovate, reduce MCS development, implementation and integration costs through easy expandability, scalability, and adaptability over time.

8.4.6 Development Environment, Standards, and Tools

Most software development tools that are used for FSW are also used in the overall software development industry. Common version control tools are Git and Subversion. More large projects are switching to the Git repository due to its distributed nature and merging features. The NASA cFS project uses Git and is sourced on https://github.com/.

Additional tools have been used with these version control tools to provide more process control and configuration management. The Atlassian tools are an example of these that interface directly to Git or Subversion and provide issue/bug tracking (Jira), documentation (Confluence), continuous integration (Bamboo) and others. The Atlassian tools are a licensed product that is free for trial and suitable for a small number of users. There are several other tools for each of these functions that are used. For instance, Trac is an open-source, web-based project management and bug tracking system.

Model-Based Systems Engineering (MBSE)

Excerpted from (https://www.omgwiki.org/MBSE/doku.php?id=start)

"The Object Management Group Standards Development Organization, in accordance with The INCOSE SE Vision 2020 (INCOSE-TP-2004-004-02 September, 2007), defines Model-based systems engineering (MBSE) as "the formalized application of modeling to support system requirements, design, analysis, verification and validation activities beginning in the conceptual design phase and continuing throughout development and later life cycle phases. MBSE is part of a long-term trend toward model-centric approaches adopted by other engineering disciplines, including mechanical, electrical and software. In particular, MBSE is expected to replace the document-centric approach that has been practiced by systems engineers in the past and to influence the future practice of systems engineering by being fully integrated into the definition of systems engineering processes." Applying MBSE is expected to provide significant benefits over the document centric approach by enhancing productivity and quality, reducing risk, and providing improved communications among the system development team.

Modeling has always been an important part of systems engineering to support functional, performance, and other types of engineering analysis. Wayne Wymore introduced a mathematical foundation for MBSE in his book entitled *Model-Based Systems Engineering* in 1993. However, the growth in computing technology and the introduction of modeling standards such as SysML, UPDM, Modelica, HLA, and others, are helping to enable MBSE as a standard practice, and provide a foundation to integrate diverse models needed to fully specify and analyze systems…."

Auto-Generation of Software

Automatic generation of source code from higher symbolic languages is being adopted by a wide number of missions. This technique is commonly being used by several NASA centers including ARC, GSFC and JSC. Key advantages of using this approach are rapid development and testing, and significant time and cost savings. There are a variety of tools that have been used in the past, but the most popular is MATLAB/Simulink. This allows an engineer to completely develop the algorithms in a graphical or higher-level language and have flight code automatically generated. Simulations and tests are also developed within MATLAB/Simulink. A common way that these kinds of tools are used are within the GNC development, but other missions such as LADEE have used it for almost the entire FSW with over 85% of the new code generated in this manner (35, 36).

Using these tools has advantages. They are designed for analysis and have built-in simulation tools. They are usually seen as being easier to understand due to their graphical nature. These tools are familiar to many engineers since they have been used by several colleges and universities. One thing to be aware when developing software with this method is that good modeling practices need to be adopted so that the resultant models produce good code. These include all the best practices performed with traditional software development. An example is to establish and use modeling guidelines so that the resultant code is consistent.

Simulations and Simulators

Simulations are needed to fully test software before release to verify and help validate the software. In a sense, unit tests are very simple simulations. Overall simulations need to be large enough to run all released flight software. The preferred method is to test all the FSW in an integrated fashion. If that cannot be performed, then partial tests may have to be performed. The testing should be designed to cover all executed code. The issues of not testing all the code is that total execution performance and possible interactions between modules may not be tested. Scenarios or a “day in the life” tests should be covered, as well as off-nominal fault recovery.

Simulators usually refer to the hardware and infrastructure needed to run the FSW and simulations. The main part of the simulation is the actual FSW. This should be run on a processing environment as close to the flight processor as possible. For some situations, that can be an actual spare flight unit. For some processors that are costly, such as the RAD750, either an engineering unit or a similar PowerPC processor that is binary compatible may be used. These processors are either connected to actual hardware interfaces that are connected to spacecraft subsystems, or subsystem simulators. These types of simulators are referred to as Hardware-in-the-Loop (HIL) simulators because they use actual hardware for testing. The other type of simulator is a processor-in-the-loop (PIL) simulator where a flight-like processor is tested against simulations of the hardware and subsystems. Depending on the environment and processing load, this is usually done in a separate processor, but can be done on a single flight-like processor. The simulation portion (non-flight software) is almost always preferred to be executed on a separate processor so that interference with the flight software is minimized or eliminated.

NASA Ames has created a development environment where the same flight executable can be executed on a flight-like processor in simulation. This is done by simulating each of the interfaces through a standard POSIX interface and having the flight executable talk to that interface. Lower-level interface communication can then occur either through a hardware interface (flight-like), or UDP Ethernet (simulation) based on the simulator configuration.

Software Best Practices and NPR7150

Software can be complex and overwhelming because of the large scope and unique nature of software development. Additionally, flight software can be costly and have reliability issues because it can be large in scope, complex, and there are significant difficulties with testing in a flight-like environment. To help address developmental challenges, software development has created best practices. These can be implemented several ways but encompass some basic elements. Some software best practices include:

- Create a plan*, schedule, and budget for software: a plan is needed to fully understand the scope of the software effort. Ideally, plans would be developed based on previous experiences, but there may not be a similar experience that can be used for a particular project, and the software manager must rely on instinct and best judgement. Usually, software will require multiple releases because incrementally developed features of the software are needed by the customer at various stages of the project (e.g. I&T, pre-launch, operations). Include a cost discussion and customer sign-off.
- Configuration management/revision control: this should be used for all software development not just FSW. There are many readily available tools, but two of the most popular are Git and Subversion. These tools provide an automatic history of the software development. Configuration Management (CM) allows coordination between multiple team members, assists in the overall software release, and tracks what changes are in that release. CM also allows back tracing to see when a software bug may have been introduced.

- Code reviews: all code should be inspected prior to being accepted by the project. These reviews can be performed in a variety of methods, from off-line informal peer reviews to more formal meetings such as perspective-based code inspections. Some developers believe that this is a poor use of time and are hesitant to have others look at their work. Code reviews lead to a higher quality product and better understanding of the software.
- Documentation: documentation can be both within the code or developed separately. Some documentation tools process the software code to produce formal documentation. The documentation should be consistent with the overall software effort.
- Testing: testing can come as three different parts.
    - Unit- and component-level tests: each software module should have a unit test (function level) and/or component test (module level) that is required to pass before that code is accepted for release. A record of these tests should be kept as part of the overall software release procedure. When fixing a discrepancy or bug the unit test should be modified to test those fixes.
    - Manual or interface testing: software is tested against the actual devices or a copy to ensure that both the hardware and software can successfully communicate and control each subsystem. Tests should be repeated, and edge cases should be tested whenever possible.
    - Integrated testing: integrated testing is the main time that all the FSW can be tested together. This ensures that the overall system operates in an expected and reliable manner. Ideally subsystems have actual hardware, but simulations can be used.
- Continuous integration: Continuous Integration (CI) works with the CM tools to know when changes have been committed. The CI tools automatically build executables and run configured tests (unit and integrated tests). This removes the burden of building and testing from the developers and finds any issues with new code much faster. CI does require setup time and an understanding of the tools.

*Software planning is a whole topic unto itself. There are several software development approaches. Currently agile software development is one of the most popular. The overall cost of the software development effort needs to be understood, and a detailed cost estimate should be performed. As the complexity of the FSW increases, so does the cost and the effort of estimating that cost. There are a number of different methods for estimating those costs, including analogy, parametric models such as Cocomo, and bottoms-up cost estimates (16) (17) (18). Typically, there is a lot of uncertainty in software cost estimates, so it is important to try to understand the bounds of that uncertainty and, if possible, to give confidence in the estimate.

In order to ensure that all NASA projects follow best software practices NASA Software Engineering Requirements standard NPR 7150.2 (currently NPR7150.2C – for updated NPR standards please see https://nodis3.gsfc.nasa.gov/) is mandated for all NASA Flight Software (and NASA developed software in general). It covers requirements for software management and planning, software engineering life cycle requirements, and supporting software life cycle requirements. Overall NPR 7150.2 addresses:

- Roles and responsibilities for tailoring requirements
- Software management
    - Software lifecycle planning
    - Cost estimates
    - Training
    - Classification assessments
    - Software assurance and software verification and validation
- Software engineering life cycle requirements
    - Requirements

    - Architecture
    - Design
    - Implementation
    - Testing
    - Operations, maintenance, and retirement
- Supporting software life cycle requirements
    - Configuration management
    - Risk management
    - Peer reviews/inspections
    - Measurements
    - Non-conformance or defect management
- Recommended software documentation

Digital Twin – https://en.wikipedia.org/wiki/Digital_twin

Taken directly from the webpage: "…A digital twin is a virtual representation that serves as the real-time digital counterpart of a physical object or process. …The first practical definition of digital twin originated from NASA in an attempt to improve physical model simulation of spacecraft in 2010. Digital twins are the result of continual improvement in the creation of product design and engineering activities. Product drawings and engineering specifications progressed from handmade drafting to computer aided drafting/computer aided design to model-based systems engineering.

The digital twin of a physical object is dependent on the digital thread—the lowest level design and specification for a digital twin—and the "twin" is dependent on the digital thread to maintain accuracy."

8.5 On the Horizon (TRL 1-4): Command and Data Handling

Many C&DH systems will continue to follow trends set for embedded systems. Short duration missions in low-Earth orbit will continue to take advantage of advances made by industry leaders who provide embedded systems, technologies, and components. In keeping with the low-cost, rapid development theme of CubeSat-based missions, many COTS solutions are available for spacecraft developers.

While traditional C&DH processing needs are relatively stagnant, as small satellites are being targeted for flying increasingly data-heavy payloads (i.e. imaging systems) there is new interest in advanced on-board processing for mission data. Typically, these higher performance functions would be added as a separate payload processing element outside of the C&DH function. Automotive and smartphone industries have pushed the energy efficiency of embedded Graphics Processor Units (GPUs) – processors optimized for matrix multiplication.

8.5.1 Open-Source Platforms

Several open-source hardware platforms hold promise for small spacecraft systems. Arduino boards consist of a microcontroller with complementary hardware circuits, called shields. The Arduino platform uses Atmel microcontrollers; therefore, developers can exploit Atmel's development environment to write software (https://www.arduino.cc). The ArduSat spacecraft used the Arduino platform and successfully engaged the public to raise funding on Kickstarter.

Raspberry Pi (https://www.raspberrypi.org) is another high-performance open-source hardware platform capable of handling imaging, and potentially, high-speed communication applications (27). Raspberry Pi microcontrollers have been shown to be able to accommodate NASA standard core Flight Software and are available in multiple, demonstrated embodiments (28).

BeagleBone (https://beagleboard.org/bone) has also emerged as a popular open-source hardware platform. BeagleBone contains an ARM processor and supports OpenCV, a powerful open-source machine vision software tool that could be used for imaging applications. BeagleSat is an open-source CubeSat platform based on the BeagleBone embedded development board. It provides a framework and tool set for designing a CubeSat from the ground up, while expanding the CubeSat community and bringing space to a broader audience.

Arduino has become known for being beginner friendly and making the world of microcontrollers more approachable for software designers. Though it presents a relatively familiar set of APIs to developers, it does not run its own operating system. On the other hand, the BeagleBone Black, Raspberry Pi, and Intel Edison are full-featured embedded Linux systems running Angstrom, Raspbian, and Yocto Linux kernels out of the box respectively. This broadens the range of developer tool options, from web-based interfaces to Android and Python environments. Not only does this further ease the learning curve for novice developers, but it allows the full power of a Linux system to be harnessed in computation tasks.

Several vendors have developed and implemented C&DH solutions using the Xilinx ZYNQ family of processors (https://www.xilinx.com/products/silicon-devices/soc/zynq-ultrascale-mpsoc.html). This processor offers single- to quad-core ARM processing at GHz speeds with built-in FPGA. Although not directly radiation hardened, several radiation mitigation factors have been implemented. These systems typically have been developed on open-source Linux OS.

8.6 On the Horizon (TRL 1-4): Flight Software

FSW is key to mission success. The field of software is a very dynamic environment and continuously evolving. The challenges with flight software usually remain the same regardless of the size of the spacecraft (CubeSat to SmallSat) and are related to the size and complexity of the endeavor. Overall, flight software can be known for scheduling issues and implementation issues especially during integration and test. Temptation of adding additional features is usually present. All these factors can drive up overall complexity and threaten success of FSW and the mission as a whole.

It is essential that FSW be as simple as possible. It is critical to survey the options and plan any FSW effort early. Wherever possible early development and testing should be exercised. Efforts to add additional features should be looked at very critically with strong efforts to stick to the existing plan. With good planning and careful execution, a favorable outcome can be achieved.

8.7 Avionics Systems Platform and Mission Development Considerations

There are many factors to be considered in the optimum selection, configuration and implementation of avionics subsystems, components, and elements for small spacecraft missions. Overall spacecraft concerns of size, weight and power (SWaP) always need to be considered. Some of the more pertinent issues and concerns that all small spacecraft missions must address include:

- Mission applicability and tailoring
- Element, module, and component modularity and interoperability
- Manufacturing and production efficiency, complexity, and scaling
- Mission environment, especially radiation and long-duration space exposure
- Standards and regulatory concerns

Small Spacecraft Avionic systems considerations of particular interest in determining the state-of-the-art for the C&DH, FSW, and subsystem/payload specific electronic systems include the following:

- Small spacecraft platform size ranges and configurations
- Integrated avionics platform architectures
- Mission avionics configurations
- Spacecraft and mission autonomy

8.7.1 Flight Payload and Subsystems Avionic elements examples

Below are some reference examples of representative flight payload and subsystems avionics products. These include onboard controllers, systems health avionics, payload processors, and cloud-based processors. Other examples can include use of FPGAs, single-board computers, compute modules, and open-source platforms as described elsewhere in this chapter.

Subsystem integrated OBC controllers:

ex. UNIBAP ix5-100 (www.unibap.com)

Integrated systems health avionics:

ex. IoT Mesh avionics **(https://ti.arc.nasa.gov/m/pub-archive/426h/0426%20(Alena).pdf)**

Onboard Payload Processors

https://www.esa.int/Enabling_Support/Space_Engineering_Technology/Onboard_Computers_and_Data_Handling/Architectures_of_Onboard_Data_Systems

Cloud-based processors

ex: Spacecloud (https://incubed.phi.esa.int/portfolio/uss/)

8.7.2 Platform Size Ranges and Configurations

The generally accepted and documented small spacecraft standard size definitions are described in incremental order of magnitude increments (subNano=0.1-10 kg; Nano=1-10 kg, and micro= 10-100 kg). With increased capability and miniaturization of space subsystems and components, and availability of standard launch vehicle deployers and interfaces, many SmallSat providers have begun to develop smaller spacecraft platforms based on containerized CubeSat ranges in increments of "U" described as 10 cm cubic volume, 2 kg max/cube. These units straddle the generic NanoSat/MicroSat size ranges, with 1-3U being considered nanosatellites, and 6-27U standard CubeSats now in the lower half of the microsatellite range.

This often-overlapping standardization impacts small spacecraft avionics in several ways:

- Spacecraft avionics components are performance driven, and not necessarily dependent on spacecraft platform sizes.
- In general, containerized CubeSat standard components are probably upward compatible with equivalent nanosatellite size ranges, but not necessarily the reverse.
- CubeSat subsystems, assemblies, and components are being developed in "U" compatible form factors and may in some cases limit the use and integration of products that do not fit within the U or multiple U dimension and mass constraints.
- Consideration for using available higher TRL avionics products may be constrained to those selected for containerized spacecraft platforms.

8.7.3 Integrated Avionics Platform Architectures

The aeronautics industry has defined and adapted the concept of Integrated Mission Architectures (IMA), where IMA are defined as real-time computer network airborne systems. Standard IMA avionics protocols exist (ARINC653 and ARINC 654) to define and inform the community, thus facilitating interfacing and interchangeability of systems components and software functionalities.

Following the trend now well established for aircraft, two embodiments and configurations for integrated modular avionics architectures for small spacecraft can be characterized as follows:

- Federated: Each subsystem of the spacecraft is considered an independent dedicated autonomous element, with the avionic component performing all functions independently, with data exchanged only over standardized communications protocols and interfaces.
- Integrated: Shared, distributed functionality, that can be configured as distributed, heterogeneous, and/or mixed criticality elements, with the capability for smart subsystems, and modular redundant fault tolerant radiation and anomaly mitigation procedures.

8.7.4 SS Mission Avionics Configurations

As the use and utility of lower cost small spacecraft for space missions gains acceptance beyond technology demonstration, risk reduction, and education, multi-satellite mission architectures are gaining increasing interest and acceptance. Such configuration architectures as distributed *ad hoc* constellation networks and swarms, synchronized formations, and other multi-satellite cluster formations are creating new opportunities and for small spacecraft avionics. Increased need for synchronization, intersatellite communications, controlled positioning for integrated C&DH functionality, coordination and conduct, operation of ConOps and autonomous operations impose new constraints on the avionics system, not only for single satellites, but now also as systems of systems, whereby overall mission performance is now dependent on all the platform elements acting in a co-dependent fashion.

8.7.5 Spacecraft and Mission Autonomy

The NASA 2020 Technology Roadmap defines autonomous systems (in the context of robotics, spacecraft, or aircraft) as a cross-domain capability that enables the system to operate in a dynamic environment independent of external control. (https://www.nasa.gov/sites/default/files/atoms/files/2020_nasa_technology_taxonomy.pdf)

Of particular interest for spacecraft autonomy are the topic areas listing the characteristics of autonomous systems:

- Situational and self-awareness
- Reasoning and acting
- Collaboration and interaction
- Engineering and integrity

Spacecraft autonomy can be considered a part of management, direction, and control for all subsystems and functions in a spacecraft. The C&DH is the brain and executive decision-making component and the nerve center. It takes input from and provides direction to all subsystems (ADCS, Power, Propulsion, Comm, vehicle health, etc.). Those subsystems may also have a degree of autonomy depending on the complexity if it's a local "smart subsystems" processor.

Some systems now implement a heterogeneous architecture, meaning they contain multiple processors with varying levels of performance and capabilities. For instance, the higher performance modules and components can be used for sophisticated number crunching and processing, AI and onboard computing for both spacecraft and mission performance optimization,

as well as science data real-time adaptive analysis (we used to call these co-processors, now heterogeneous architectures). This allows lower performance onboard processors and FPGAs etc. to conduct the routine spacecraft operations functions and interact with the subsystems which also may include distributed performance cascades.

8.7.6 Industry 4.0, Foundational and Enabling Technologies and Products

In keeping with the trends seen in other disciplines and industries, the Industry 4.0 and "digitally managed everything" is absolutely of critical importance for technological and programmatic efficiencies in small spacecraft avionic systems development and utilization. The following is just a short list of 21st century tools, technologies, and approaches that must be considered in development and deployment of next-generation small spacecraft avionic systems:

1. Artificial Intelligence, Machine Learning / Machine Vision
2. Robotics and Automation
3. Model-Based Systems Engineering
4. Embedded Systems / Edge Computing
5. Internet-of-Space-Things
6. Cloud Computing
7. Augmented Reality/ Virtual Reality / Mixed Reality
8. Software-Defined-Everything
9. Advanced manufacturing
10. Digital Twin

8.8 Summary

Space applications now require considerable autonomy, precision, and robustness, and are refining technologies for such operations as on-orbit servicing, relative and absolute navigation, inter-satellite communication, and formation flying. An exciting trend is that small spacecraft missions are becoming more complex in the anticipation of these platforms being used for lunar and deep space science and exploration missions. Small spacecraft technology must expand to meet the needs of this increasing small spacecraft mission complexity, to achieve the next generational goals of collecting important science in deep space using small spacecraft, as well as risk mitigation for larger more complex and mission-critical situations. In parallel, spacecraft electronic components have matured to have higher performance, higher reliability, and are being miniaturized to meet the growing needs of these now very capable spacecraft.

With the combination and integration of the previously separate C&DH and FSW chapters, the 2021 Small Spacecraft Avionics chapter has attempted to cast these elements in a broader, interrelated framework, and attempts to show that C&DH, FSW, and smart payloads are not just independent space platform subsystems but are part of an integrated avionics ecosystem of all electronic elements of a space platform, now primarily digitally based and or managed. Also, SSA should not be considered as an isolated spaceflight technology component, but rather as a core digital engineering technology emphasis area, capable of taking advantage of and integrating products, processes, and technologies from other disciplines. To continue to be relevant and efficient, the small spacecraft avionics communities must remain cognizant and receptive of the continuously evolving nature of the digital based Industry4.0 technology revolution now being evidenced in other related and/or associated vertical disciplines and solutions.

For feedback solicitation, please email: arc-sst-soa@mail.nasa.gov. Please include a business email for further contact.

References

(1) GomSpace. NanoMind A3200 - Datasheet. [Online] 2019. https://gomspace.com/shop/subsystems/command-and-data-handling/nanomind-a3200.aspx.

(2) ISIS. ISIS On board computer - Datasheet. [Online] 2019. https://www.isispace.nl/wp-content/uploads/2016/02/IOBC-Brochure-web-compressed.pdf.

(3) Pumpkin Space Systems. "Processor-specific CubeSat Kit™ Components." Pumpkin Products/Store. [Online] 2020. https://www.pumpkinspace.com/store/c19/Processor-specific_CubeSat_Kit%E2%84%A2_Components.html.

(4) Xiphos. Q7S Specifications - Datasheet. [Online] 2020. http://xiphos.com/wp-content/uploads/2015/06/XTI-2001-2020-e-Q7S-Spec-Sheet.pdf.

(5) Xiphis. Q8S SPECIFICATIONS - Datasheet. [Online] 2020. http://xiphos.com/wp-content/uploads/2020/06/XTI-2001-2025-f-Q8S-Rev-B-Spec-Sheet-1.pdf.

(6) BAE Systems. "RAD750 3U CompactPCI single-board computer." BAE Systems: Space product literature. [Online] 2008. https://www.baesystems.com/en-us/our-company/inc-businesses/electronic-systems/product-sites/space-products-and-processing/radiation-hardened-electronics.

(7) BAE Systems. "RAD5545 Space VPX single-board computer." BAE Systems: Space products literature. [Online] 2017. https://www.baesystems.com/en-us/our-company/inc-businesses/electronic-systems/product-sites/space-products-and-processing/radiation-hardened-electronics.

(8) AAC Clyde Space. Command & Data Handling KRYTEN-M3 - Datasheet. [Online] 2020. https://www.aac-clyde.space/assets/000/000/179/AAC_DataSheet_Kryten_original.pdf?1600342763.

(9) AAC Clyde Space. Command & Data Handling Sirius OBC LEON3FT - Datasheet. [Online] 2020. Available at: https://www.aac-clyde.space/assets/000/000/181/AAC_DataSheet_Sirius_OBC_-_updated_tables_original.pdf?1599046187.

(10) Innoflight. CFC-300: Compact Flight Computer - Datasheet. [Online] 2020. Available at: https://www.innoflight.com/product-overview/cfcs/cfc-300/.

(11) Innoflight. CFC-400: Compact Flight Computer - Datasheet. [Online] 2020. Available at: https://www.innoflight.com/product-overview/cfcs/cfc-400/.

(12) Innoflight. CFC-500: Compact On-Board Computer - Datasheet. [Online] 2020. Available at: https://www.innoflight.com/product-overview/cfcs/cfc-500/.

(13) Space Micro. CubeSat Space Processor (CSP). Datasheets. [Online] 2019. Available at: https://www.spacemicro.com/assets/datasheets/digital/slices/CSP.pdf.

(14) Nanoavionics. CubeSat On-Board Computer - Main Bus Unit SatBus 3C2 - Datasheet. [Online] 2020. https://nanoavionics.com/CubeSat-components/CubeSat-on-board-computer-main-bus-unit-satbus-3c2/.

(15) Moog. Radiation Tolerant, 75GLOP 3U SPaceVPX GPU Single Board Computer - Datasheet. Moog: Space Defense: Space Literature. [Online] 2020. Available at: https://www.moog.com/content/dam/moog/literature/Space_Defense/spaceliterature/avionics/Moog-Rad-Tolerant-75GFLOP-3U-SpaceVPX-GPU-Single-Board-Computer-Datasheet.pdf.

(16) SEAKR. Commercial Products. SEAKR: Catalog. [Online] 2020. Available at: https://www.seakr.com/catalog/.

(17) Moog. Integrated Avionics Unit - Datasheet. *Moog: Space: Avionics.* [Online] 2020. Available at: https://www.moog.com/content/dam/moog/literature/Space_Defense/spaceliterature/avionics/moog-integrated-avionics-unit-datasheet.pdf.

(18) Fitzsimmons, S.: *"Open-source Software for CubeSat Satellites."* AIAA/USU Conference on Small Satellites. 2012.

(19) Bruhn, F.C., Tsog, N., Kunkel, F. et al. Enabling radiation tolerant heterogeneous GPU-based onboard data processing in space. CEAS Space J 12, 551–564 (2020). https://doi.org/10.1007/s12567-020-00321-9

(20) Nguyen, M.: *"FPGA Advances Improve Radiation Mitigation for Remote-Sensing Satellites."* 2015, pp. vol. 17, no. 8.

(21) Ball Aerospace & Technologies Corp. "The User Interface for Command and Control of Embedded Systems." 2015.

(22) Henkel, H.: "Total Dose Radiation Tests at FRAM Non-Volatile Memories." 1996.

(23) NASA Goddard Space Flight Center. "GSFC Open-Source Software." 2015.

(24) Bardoux, A., et al., "Radiation Effects on Image Sensors." International Conference on Space Optics. 2012.

(25) Chapman, T.: "Radiation Tolerances." 2015.

(26) Holbert, K. E.: "Single Effect Events." 2015.

(27) Wooster, P, et al.: "Open-Source Software for Small Satellites." AIAA/USU Conference on Small Satellites. No. #SSC07-XII-3. 2007.

(28) Cudmore, Alan: "Pi-Sat: a Low-Cost Small Satellite and Distributed Spacecraft Mission System Test Platform." s.l.: NASA GSFC, 2015. GSFC-E-DAA-TN27347.

(29) NASA Goddard Space Flight Center. "Core Flight Software System." 2015.

(30) LaBel, K. A. et. al.: "Commercial Microelectronics Technologies for Applications in the Satellite radiation Environment." Aerospace Applications Conference. 1996.

(31) Windriver. VXWORKS. [Online] Available at: https://www.windriver.com/products/vxworks/.

(32) RTEMS. RTEMS Real Time Operating System (RTOS). [Online] 2020. Available at: https://www.rtems.org.

(33) NASA. A Flight Software and Embedded Systems Framework. [Online] 2020. https://nasa.github.io/fprime/.

(34) Bocchino, Robert, et al.: "F Prime: An Open-Source Framework for Small-Scale Flight Software Systems." Logan: 32nd Annual Small Satellite Conference, 2018.

(35) Multi-Purpose Spacecraft Simulator for the LADEE Mission. Benz, Nathaniel, Viazzo, Danilo and Gundy-Burlet, Karen. s.l.: IEEE, 2015.

(36) Gundy-Burlet, Karen: "Validation and Verification of LADEE Models and Software.". s.l. : American Institute of Aeronautics and Astronautics.

(37) NASA. ITOS Capabilities. *Integrated Test and Operations Systems (ITOS)* . [Online] 2020. Available at: https://itos.gsfc.nasa.gov/itos-capabilities.php.

(38) NASA. ASIST Public Web Site - NASA. [Online] 2012. Available at: https://nasa-asist.gsfc.nasa.gov.

(39) L3Harris Technologies. Space Software. Telemetry and RF products. [Online] 2020. https://www2.l3t.com/trf/incontrol/index.htm.

(40) Ball Aerospace. The User Interface for Command and Control of Embedded Systems. [Online] 2020. Available at: https://cosmosrb.com/.

(41) Bruhn, F.C., Tsog, N., Kunkel, F. et al. Enabling radiation tolerant heterogeneous GPU-based onboard data processing in space. CEAS Space J 12, 551–564 (2020). https://doi.org/10.1007/s12567-020-00321-9

(42) Ibeos: “Standard Products.” [Online] 2021. Accessed August 8, 2021. Available at: https://www.ibeos.com/standard-products

(43) eoPortal Directory, “DSX (Demonstration and Science Experiments) in MEO”, [Online] Accessed December 12, 2021. Available at: https://earth.esa.int/web/eoportal/satellite-missions/d/dsx

(44) Data Device Corporation, “High-Speed, SWaP Optimized 3U SpaceVPX SBC”, [Online] Available at: https://www.ddc-web.com/en/connectivity/processor-based-solutions/sbcsforspace-1/low-power-quad-core-3u-spacevpx-computer-for-space?partNumber=SCS3740