Wavefront - Design, Analysis, and Selected Testing of a Complete Martian Spacecraft System

a project presented to
The Faculty of the Department of Aerospace Engineering
San José State University

in partial fulfillment of the requirements for the degree
Master of Science in Aerospace Engineering

by

Stanley Marcus Krześniak

August 2023

approved by

Dr. Periklis E. Papadopoulos
Faculty Advisor

Marcus S. Murbach, L. Seth Schisler
NASA Ames Research Center

San José State
UNIVERSITY

ABSTRACT

Wavefront - Design, Analysis and Selected Testing of a Complete Martian Spacecraft System

Stanley Marcus Krześniak

In the coming years and decades, Mars will become a major destination, with several space agencies sending uncrewed spacecraft - and eventually human-rated missions. While humans will undoubtedly make the largest impact in terms of scientific output, uncrewed flagship-class spacecraft and smaller space probes will continue to have a substantial role in delivering science - and technology demonstrations - before, during, and after an initial human visit. Wavefront, a complete, NASA Flagship-class multiprobe mission, would deliver twelve rovers, twelve landers, and 120 nanoprobes to the surface of Mars, as well as one orbital relay to a sun-synchronous orbit. This will enable vast amounts of science returns and prepare humans for exploration on Mars.

In this thesis, a systems engineering and program management approach is taken to define, design, and develop a complete mission, as well as deliver certain components to demonstrate some concepts. An extensive literature review surveys notable successes and challenges from multiple world space agencies on missions to the Moon and Mars to avoid reinventing the wheel. The definition of the mission follows, starting from government-level regulatory and scientific requirements. With this baseline established, the top-level engineering requirements are then established within the purview of scientific and regulatory frameworks.

A design summary of hardware nanoprobe development, a universal electrical power system, and fluid dynamics analysis of entry, descent, and landing hardware follows. Due to a limited amount of time and a tremendous amount of detailed design work that went on up until the final months of the project, much of the specifics were

omitted. The code for multiple hardware pieces has been included in the Appendix to prove that full-stack engineering development occurred.

"Let us pick up our books and our pens. They are our most powerful weapons. One child, one teacher, one book, and one pen can change the world. Education is the only solution." - Malala Yousafzai

One person with the appropriate education and connections can change the world for the better. In this broken world full of *strife* and rapid climate change, humanity needs someone. Are you the *avalanche*?

Acknowledgements

I owe it to a lot of people on this long journey:

- First and foremost: my mom, dad, and sister for their advice, and unconditional and unwavering support for everything I've done;

- Marcus Murbach, although a reviewer, deserves additional acknowledgements for all the life advice and being able to stick with my occasional but necessary hardheadedness;

- L. Seth Schisler, also a reviewer, for the lucid advice, theory, and practice of political science, especially for my personal enrichment;

- Ian *(Sazh)*, Maisie *(Fang)*, Evan *(Setzer)*, Aysha *(Yuna)*, Steven *(4S)*, Javaneh, Joon, Joey, Andreana *(Serah)*, Jesse *(Snow)*, Svitlana *(Lightning)*, Asma *(Aerith)*, Hieu, Devin, and Derek for being some of my best friends;

- The SEEDS 1 team (many of whom are my good friends): Christian Espinoza *(Garlond)*, Dominic Campbell, Eduardo Marchan, Jay Mehta, Kobi Boateng, Louie Freitas, Maxime Carpentier, the SETI Institute, Dr. Nathalie Cabrol, and the Hines Family Foundation/John Hines, for whom I owe the seed - the idea and concept - of my masters project to bloom into something I totally did not expect;

- And finally, my senior design team. I am merely continuing the amazing work that Afrah *(A2)*, Kayla *(21O)*, and Nataliya *(4B)* put into SEEDS 1 and 2.

This masters project was supported by a Space Act Agreement between San José State University and the NASA Ames Research Center.

The SEEDS-A sensor network development project, detailed in Chapter 5, Section 1, was funded in part through John Hines Technology Associates LLC, the SETI Foundation, and NASA Grant NNA15BB01A.

Products and services mentioned as part of this report are not endorsements. Any perceived or implied endorsements do not imply favorability over any other product or company.

Table of Contents

List of Tables

List of Figures

1. Introduction

1.1 Motivation

Humanity's insatiable appetite to know more and to answer some of the most fundamental open scientific questions has continued to grow since the dawn of the Space Race. New fields of research and ones enabled by space exploration have opened up and florished. Space exploration has arguably accelerated technological advancement, despite the dynamic and complex nature of relations within humanity.

A prime and timeless example of this is computing in aerospace. To enable humans to navigate to - and land on - the Moon with incredibly limited computational resources with the Apollo Guidance Computer (AGC), two inventions had to occur: the real-time operating system (RTOS) [8], and the virtual machine [9]. The AGC is shown in Figure 1.1.

Today, personal computers, smartphones, and even some smartwatches are fast enough to perform hundreds, if not thousands of simultaneous tasks without the need for an RTOS. [10]. However, in requirement-constrained, mission-critical environments such as in aerospace and defense, RTOSes are still, and will always be, necessary. For example, an RTOS was foundational to the AGC because the processor had to perform multitasking between operating a landing radar, continuously computing the state vector of the spacecraft, controlling the reaction systems, among many other tasks. [8]. Today, in computer engineering for aerospace and defense applications, it is highly common to design physically separate processor and memory boards (and for extremely high-speed, highly time-sensitive tasks, application specific integrated circuits (ASICs)) to perform truly simultaneous, overlapping processing [11]. This was simply not possible in the Apollo program: the

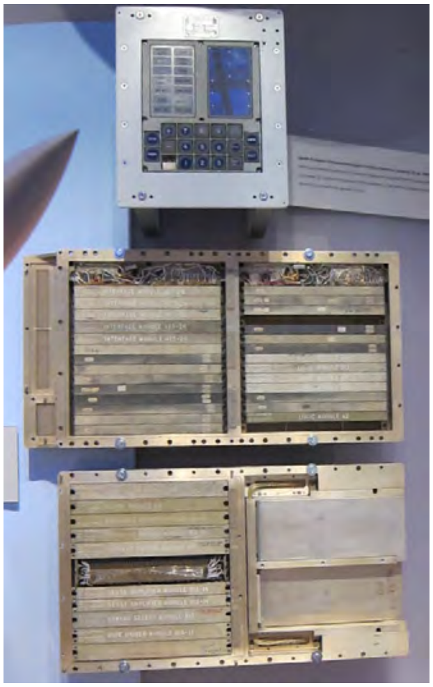


Figure 1.1: Apollo Guidance Computer shown on display at the Computer History Museum in Mountain View, CA.

AGC itself was 32 kg and consumed 70 watts [12] - compared to milligrams and milliwatts of similarly specified processors as of writing [13, 14].

The other major challenge of the first Space Race was random access memory (RAM) and program memory. RAM of the day was exceptionally expensive: it needed to be extremely fast, and be able to tolerate constant reading and writing without failure. On top of that, it needed to be robust against the rigors of spaceflight and the testing regimen: extreme G forces, heavy launch vibration, temperature extremes, and radiation. CMOS, or silicon-based memory, had not yet become small, power efficient, nor reliable enough in even the production phase [15]. Additionally, the effects of radiation on silicon-based memory were not yet well understood. This is apparent due to the lack of journal publications on the subject before 1972 [16], and to the fact that the first production CMOS memory was released in 1970 [17]. Therefore, core memory (core) filled this role. Core, shown in Figure 1.2, was the appropriate, tried-and-true solution of the time. Core was created out of microscopic ferrite donuts woven with thin magnet wire. By energizing and sensing the differences in current sent through the woven wires, bits could be read and written. It is highly resistant to radiation and strong magnetic fields.

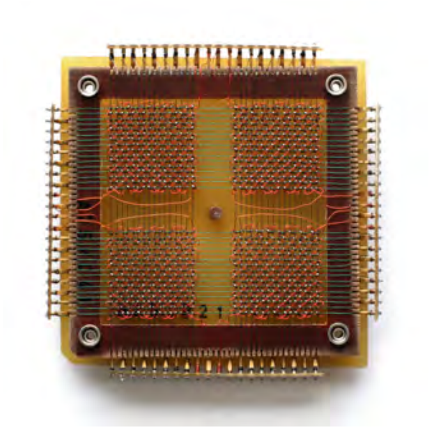


Figure 1.2: Core rope memory module storing 1024 bits. Credit: Wikimedia Foundation.

Despite the robustness, RAM and program memory were still physically large and extremely expensive - at the breakeven point of silicon-based RAM and core from 1970-1972, prices were $10 to $28 per kilobit [17]. Figure 1.2 shows how space-inefficient these modules are: the entire chip holds only 128 bytes. Additionally, the AGC had a limited number of processor instructions, due to the extraordinary size and mass of the computer compared to the vehicle. Adding native instructions to read-only program memory (ROM) increased cost and mass dramatically. To offset this, the MIT Instrumentation Laboratory developed the virtual machine: a piece of software that emulates a computer's native instruction set. Complex instructions, such as double or triple precision floating point matrix multiplication (which the AGC processor did not natively support), could then be implemented in an extremely limited ROM. By using an instruction interpreter that had its own language, a much more compact program could be stored and executed, at the expense of slower execution speed [9].

Whether it be from virtual machines and RTOSes of the Apollo project or the early advances in autonomy for interplanetary exploration from the Soviet space program, the desire to compete on the global, geopolitical stage allowed monumental advances in technology in the short span of a decade. To this day, the same VM and RTOS design pervade modern electrical systems of much higher (and lower) sophistication for products as niche as temperature tracking in foodstuffs [13].

Tracking food as it moves from farm to plate is important in a world of highly constrained resources and high demand. Characterizing the limitations of this global system is only possible with global cooperation, which, as of writing, is increasingly difficult to come by. The world is in the midst of what might be considered the Second Space Race. As in the 1950s and 1960s, where geopolitical tensions befell the world and nearly drove life as we know it to extinction[1] , the world is faced yet again with the same - if not greater - potential for a real-life rendition of Bethesda's *Fallout*. Add to that the threat of human timescale climate change, which has seemed to accelerate rapidly in the recent past.[2] In the big picture, the desire to answer fundamental open scientific questions includes such existential ones as "Where do we come from" or "Are we alone in this universe" is driven by advances in technology. As America applied technology from the Apollo program to every part of life, the world may do the same for the Second Space Race to the Moon and Mars. Technology itself will not solve the acute geopolitical issues at hand, but the correct use of

[1] The Cuban Missile Crisis

[2] Humans generally cannot comprehend change on geological timescales. Considering the overwhelming geological evidence for an Earth much warmer than today in the past, Mother Nature has tolerated things more severe than humans.

technological and political tools *tools* at hand just might.

It is therefore vital to demonstrate cooperation on a global scale to create and utilize such tools. They are borne of large-scale engineering and scientific endeavors. As discussed in the prior paragraphs, to this day, the same tools and methods for obtaining funding and building a space exploration mission are still being studied and utilized. With them, the largest space agencies today are investing heavily into lunar exploration partly because an incremental, stepping stone approach is necessary to send humans to Mars, which will be further discussed later. Part of this incremental approach is to send ever more complex robotic missions to Mars. This prepares the acting party (a governmental agency or a private corporation, or both) to the complex choreography of human deep spaceflight, and to site selection for resource utilization and further scientific value only realized by a human visit.

Indeed, on the scientific objectives, more time and *in-situ* data is needed from more parts of Mars, especially in identifying locations with large deposits of water in areas of thicker atmospheres. Once again, an internationally available body of scientific knowledge is required to advance this. On the stairstep of flying to the Moon, it is possible to bring everything with the exploration system, including the fuel to come back - as demonstrated by the Apollo program. However, when going to Mars, it is highly impractical to bring *everything* with the exploration system. According to the Tsiolkovsky equation (in Section x), only 0.2% of the mass of a launch vehicle launching from Earth is allowed as the lander mass and life support systems, if the vehicle were to return to Earth. To put this into perspective, if the Apollo Lunar Module were used to get to the surface of Mars and come back to Earth, the entire rocket getting it there would be 8,200 metric tons (mt). This is in comparison to the 2,965 mt that the Saturn V weighed in during the most science-heavy Apollo missions. This does not even include the additional mass required for life support, spare parts, and other important omissions for a multi-year journey, which would further exponentially increase the mass of such a launch vehicle.

While the oft-discussed Mars Sample Return (MSR) mission has very high complexity involving many separate missions and vehicles, MSR is a conceptual baseline for a human-rated mission. Figure 1.3 illustrates the mininal number of assets to achieve a successful mission. However, in a human-rated mission:

1. an appropriate site must be selected to maximize scientific returns for humans *in-situ*, which is the scope of this work,

2. the safety of the site must be guaranteed using robotic missions sent to the location, since, for example, the surface soil might be unpredictable as in [18] and spelled out as a requirement in [19],

3. the site or nearby sites must have adequate resources to provide water and fuel for in-situ resource utilization (ISRU), since current plans require ISRU to be a part of the mission [19],

4. habitats, ISRU plants, all-terrain vehicles (ATVs), and potentially an ascent stage must be sent on separate missions ahead of time, and of course,

5. an interplanetary transfer vehicle (ITV) with the capability of operating reliably and sustaining astronauts for years must be designed, built, and tested.

Therefore, while the MSR mission is tremendously complex, it is just a drop in the bucket compared to a human-rated mission. Thus, robotic exploration has a sizable role to play in paving the way for humans to walk the surface of Mars. In all phases of site selection, many robotic assets will be needed, most of which will need to act autonomously. For example, Mars 2020's Lander Vision Descent System (LVDS) performed a very complex site-selection, trajectory computation, and landing autonomously [20].

To some in computer and embedded systems engineering, the Internet of

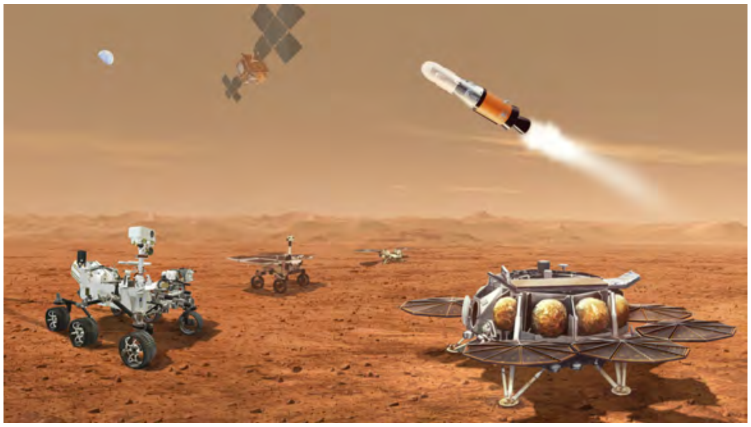


Figure 1.3: Concept of operations for NASA's Mars Sample Return mission, including all assets. From bottom left to top left counterclockwise: NASA Mars 2020/Perseverance, ESA Sample Fetch Rover, ESA Sample Fetch Lander, NASA Sample Retrieval Lander, NASA Mars Ascent Vehicle, and ESA Earth Return Orbiter. Credit: NASA/-Caltech Jet Propulsion Laboratory

Things (IoT) paradigm might be an overused phrase, but given appropriate adaptation for space-based operation, it may provide a robust framework for autonomous communication and information sharing on the large scales necessary. Examples of IoT-like systems and frameworks are NASA's Starling constellation [21], NASA's EDSN swarm [22], NASA's THEMIS constellation [23], space-based delay-tolerant networking [24], and compact atomic clocks for precise interplanetary references in base stations or gateways [25].

A major objective of this thesis is to design a mission architecture that is somewhere in between MSL/Mars 2020, and MSR. As previously mentioned, gathering more in-situ data for human lander site selection is an important objective. Page 549 of [19] spells this out in terms of "trafficability", or ability to drive and walk over a surface safely. There is even an Apollo-era precedent from the Soviet Union for surface safety: Luna XIII [26]. Meeting this objective will require significantly scaling up the number of spacecraft and lowering the development cost of such a network of fixed and mobile spacecraft. To date, no such distributed, IoT-like dedicated mission has launched to any interplanetary destination.

One main tradeoff study drives IoT design: power versus mass versus bandwidth [27]. In all spacecraft, this trade study becomes a very tightly coupled system due to launch vehicle mass limits within a set budget, power budgets when operating off of a nuclear power source or solar panels when far from the Sun, and the selection of relevant scientific data, which is itself heavily constrained by power availability and mass limits. This does not include all the minutiae of electrical engineering constraints explored in [27]. The coupling of this article and space-based constraints will be further discussed in Chapter 9.

The main reason the motivation section is partly a literature review on computer engineering is to illustrate the fact that much of the engineering that goes into spacecraft is of software, firmware, and computers. Indeed, much of the effort of this thesis went into those subjects. This thesis serves to demonstrate to an aerospace engineer that it is not practical to design one's own hardware without subject matter experts (SMEs) in computer, electrical, and software engineering. A specialist in aerospace engineering project management is also needed to dictate the appropriate requirements to said SMEs.

1.2 Literature Review

Due to the very broad nature of this thesis, the literature review is broken up into sections. Further, material from the literature that describes an individual system or method will fall under that respective system's chapter. Only full spacecraft systems for Moon and Mars are covered in the actual literature review. The majority of baseline information on lunar and Martian missions and systems draws from Asif Siddiqi's *Beyond Earth*, a comprehensive NASA Headquarters chronicle of all interplanetary space missions from 1958 to 2016, from all space agencies [28]. Additional references are included as necessary.

The depth of this literature review is intentionally and necessarily extensive: most selected spacecraft in this review represent "firsts" and "atttempted firsts". A systems-level approach to designing an entire mission requires the engineer and system architect to have a good idea of lessons learned for as many systems as possible, reaching as far back as possible. While Wavefront is a mission for Mars, system architectures from lunar systems are also very useful: except for an atmospheric package (heat shield and parachute), autonomous exploration systems for the Moon are the nearly the same as ones designed for Mars.

For example, the Soviet Mars 3 spacecraft, launched in May 1971, was the first spacecraft to make a soft landing on another planet. It included a tethered rover that was designed to "walk" with skids across the surface. However, within 20 seconds of landing and powering on its instruments, the spacecraft stopped transmitting. The cause of failure was determined to be static buildup due to the unfortunate timing of landing during a global dust storm: dust caused static buildup on the spacecraft, which eventually disabled it [28]. In designing a mission, being aware of this data point is important: operating during a dust storm requires static-mitigating operating procedures and engineering design, especially for nuclear-powered assets that can operate through heavy sky cover in a dust storm.

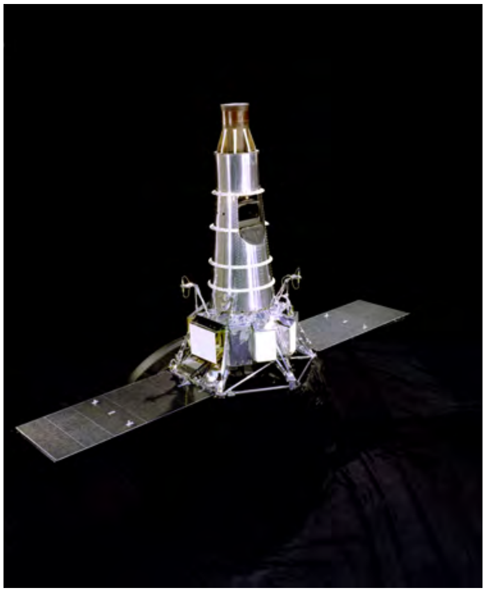


Figure 1.4: The Ranger VII Spacecraft. Credit: NASA/Jet Propulsion Laboratory.

1.2.1 Lunar Robotic Systems

1.2.1.1 Ranger VII

The first successful mission to return data about the surface of another celestial body, Ranger VII in Figure 1.4 was the product of 13 consecutive stinging failures by the Americans to return data about the Moon in close proximity. It offered a plethora of instruments, including cosmic radiation, dust, helium vector magnetometer, and of highest importance, an imaging system that used analog television cameras. Ranger VII weighed 365.6kg. 4,308 pictures were transmitted before impact, and scientists were able to conclude that the lunar surface was solid and smooth enough to land on [29, 28]. This paved the way for the Surveyor series of American spacecraft, which landed on the surface of the Moon.

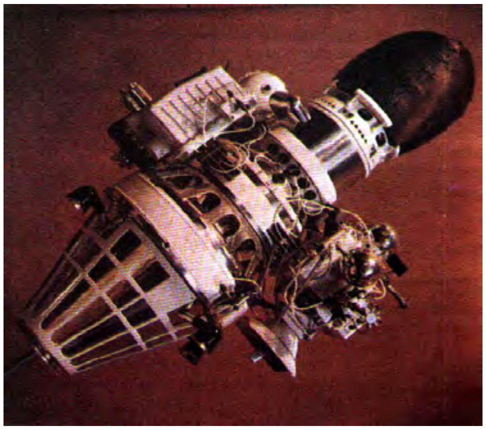


Figure 1.5: The Luna 9 spacecraft. Credit: NASA NSSDC.

1.2.1.2 Luna 9/Ye-6 No.13

But before the Americans could achieve a soft landing, the Soviets beat them to it. The 99kg Luna 9, in Figure 1.5, was designed and built by NPO Lavochkin. It landed on the Moon on February 3, 1966 using an airbag-based system, in addition to a novel, petal-based deployer that ensured the spacecraft would be deployed right side up [28]. This airbag, petal, and retropropulsion system was likely inspiration for the American Mars Exploration Rover and Pathfinder rover lander systems, covered in the next section. Owing to the fact that the Soviets had 12 successive failures in landing at the Moon, their scientific payload was minimal and consisted of a panoramic camera and a radiation sensor. The fact that they were able to land, not sink into the lunar regolith as some models predicted [30], and achieve scientific success before the Americans was a major milestone.

Luna 9 was battery powered and operated its radio for only 8 hours and 5 minutes. Radiation information was taken at the surface; the dosage was estimated at 30 mRad per day [31]. Nine images, including panoramas, were taken.

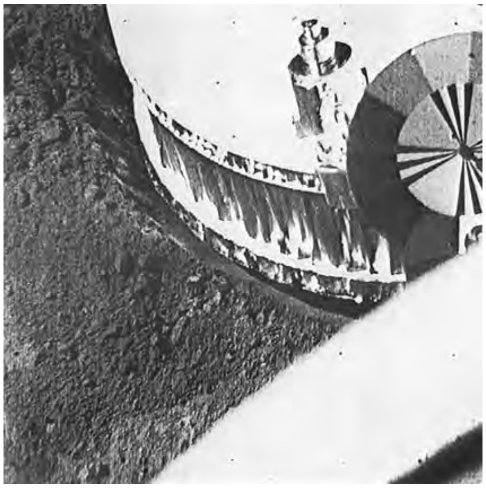


Figure 1.6: A picture of Surveyor 1's foot pad from it's zoomable television camera. The softness of the regolith is visible. Credit: NASA/Jet Propulsion Laboratory.

1.2.1.3 Surveyor 1

NASA finally had its day of glory when Surveyor 1 landed on the surface of the Moon on June 2, 1966 [28]. The 292kg probe achieved the first true soft landing on the Moon - a necessary step for humans to land and leave the Moon safely. Due to the string of failures to even crash a spacecraft into the lunar surface, JPL engineers were very surprised when Surveyor 1, the first spacecraft of a series of new spacecraft, landed flawlessly [32].

Surveyor 1, similar to Luna 9, had a minimal number of scientific instruments: it only had a camera. However, the camera boasted a zoomable lens, which enabled highly detailed panoramas and in-depth studies of soil mechanics (as shown in Figure 1.6). By the end of the extended mission (the spacecraft managed to survive several lunar day-night cycles), the spacecraft returned 11,440 images.

On the engineering side, Surveyor carried many more instruments:

- Strain gauges were mounted on the feet and mast to validate calculated loads and to infer soil mechanics,

- Temperature sensors to maintain control of temperature-regulated enclosures,

- Sun trackers and a Canopus (star) tracker for attitude knowledge,

- A radar for landing engine burn timing and surface reflectivity experiments, and

- Over 100 other unlisted engineering sensors [33].

The spacecraft was powered by an 85-watt solar array, and stored energy in a silver-zinc battery.

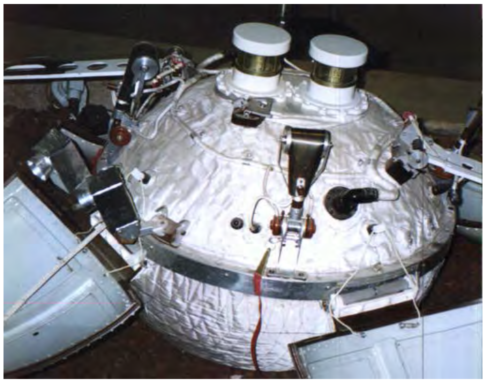


Figure 1.7: A full-scale model of the Luna XIII lander. Credit: NPO Lavochkin/Roscosmos.

1.2.1.4 Luna XIII

Luna 13 landed on the surface at *Oceanus Procellarum* on Christmas Eve, 1966, marking the last lunar mission of a very busy year in interplanetary space [28]. This 113kg probe leveraged the same design as its precedessors, including the landing system, but included a wide array of scientific instruments:

- Two television cameras,

- Four radiometers to measure heat and heat flow at the surface,

- A penetrometer to measure the softness of the lunar regolith,

- A densitometer with a ^{137}Cs gamma ray source to measure regolith density,

- A dosimeter measuring background radiation, and

- A high-resolution IMU to determine landing loads and further information on soil softness [26].

To simplify the design, the entire spacecraft electronics package was kept inside a hermetically sealed sphere at 1.2 atmospheres. This allowed the simplification of thermal management, and continued a long heritage of using spherical pressure vessels since Sputnik 1.

According to Siddiqi's research in Russian and other archives, the scientific results from Luna XIII proved to be valuable for future landers [28]:

- The penetrometer recorded a 4.5cm depth when a small solid rocket motor forced this instrument into the ground.

- The densitometer, after integrating reflected gamma rays from its ^{137}Cs source pointed at the surface, yielded a density of $0.8kg/cm^2$.

- The surface temperature, indicated by the radiometers, was found to be 117 ± 3^oC at noon.

- The IMU, recording landing forces, was able to determine regolith structure down to 20-30cm.

- The radiation sensor, in line with Luna 9 results, pointed towards a "less than hazardous" dosage in human terms.

1.2.1.5 Surveyor III

Surveyor III, in terms of instrumentation, was largely the same spacecraft as Surveyor I, but also carried a scoop and shovel to determine the bearing strength of the lunar regolith [28]. From Figure 1.8, this 296kg probe was visited by Apollo 12 - the only uncrewed mission that a crewed mission caught up to.

The major accomplishment of Surveyor 3 was in gathering more information on

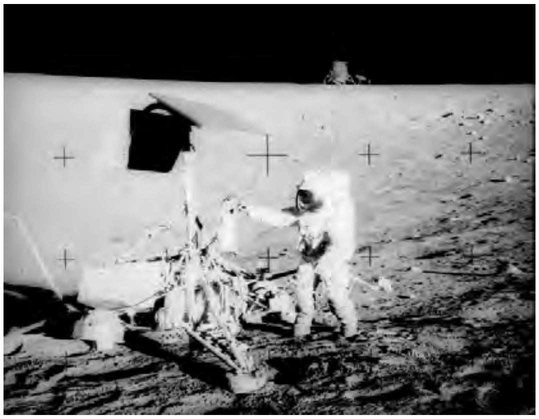


Figure 1.8: The Surveyor 3 spacecraft, Apollo XII lander, and astronaut Pete Conrad. Conrad is inspecting components on the inert Surveyor 3. This was the first, and only time as of writing where a crewed mission caught up to an uncrewed mission to investigate it. Perhaps, someday in the future, this feat might be replicated on Mars. Credit: NASA.

soil science, a continued theme from past lunar landers [34]. Through digging four trenches and performing 13 impact tests in the regolith, the strength of the soil was determined to be 0.7 kg/cm^2, similar to wet sand. This was sufficient strength to support a much heavier lunar lander, i.e. Apollo [28].

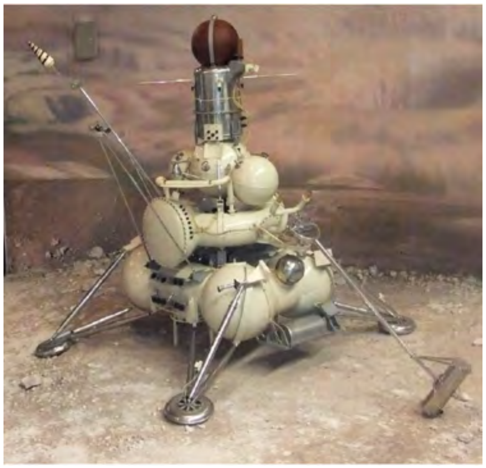


Figure 1.9: A model of the Roscosmos Luna 16 spacecraft. The scoop and spiral helical antenna are prominent. Credit: Asif Siddiqi.

1.2.1.6 Luna XVI

Luna XVI/16 was a monumental success for the Soviets. On 20 September 1970, a year and two months after Apollo 11, the 5,725kg Luna XVI descended to the surface of the Moon. Upon landing, the spacecraft deployed a drill, and drilled for 7 minutes. The 35cm depth sample was raised into the return capsule, dumping 101 grams of lunar regolith. After spending just over a day at the surface, the 512kg return stage powered up, firing a 42kg solid engine for 60 seconds. The probe landed in Kazakhstan on September 24, 1970, after returning at 10.95km/s and experiencing up to 350*g* of acceleration. [28]. This marked the first robotic sample return mission from another celestial body.

Aside from the samples returned, Luna XVI carried a stereo imager and radiation sensor, of which there is no publicly available data [35].

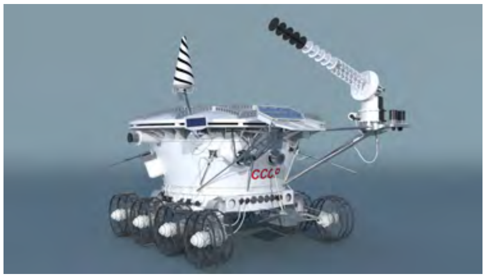


Figure 1.10: A high-resolution rendering of the Lunokhod 1 rover, part of the Luna 17 lander. To enable high-rate data return, a steerable Yagi was incorporated onto the rover. A conical helix antenna provided a low-gain link. Credit: NASA NSSDC.

1.2.1.7 Lunokhod 1 Rover/Luna 17

Luna 17 was yet another major success story for the Soviets. The 5,700 kg Luna 17 carried the Lunokhod 1 rover to the surface on November 17, 1970, the first wheeled vehicle to traverse another celestial body, as well as the first robotic - albeit teleoperated - vehicle to rove another celestial body [28]. They beat the Americans to this feat - the first American wheeled vehicle was the Lunar Roving Vehicle, deployed from Apollo 15 on July 30, 1971 [36].

Lunokhod 1 carried a generous number of scientific instruments:

- an imaging system with two low resolution TV cameras and four high resolution "photometers",

- an X-ray spectrometer, similar to the APXS found on a few American Martian rovers,

- a penetrometer to determine surface density and characteristics,

- a laser reflector to determine the precise location of the rover from Earth,

- a radiation detector,

- an X-ray telescope, and

- an odometer and speedometer [28].

A team of five cosmonauts were selected to drive the rover: commander, driver, flight engineer, navigator, and narrow-beam antenna-guidance operator [28]. While the rover was on the day side, the operators drove the rover for 322 Earth days over 9,930 meters. Tens of thousands of images were returned, several soil analyses were performed, and the penetrometer was used hundreds of times. The rover outlived its design life by almost 400% - it lasted 11 lunar days. The laser reflector continues to be used to determine distances to the Moon.

Several lunar Soviet rovers followed for some years to come, the final successful system being Luna 24.

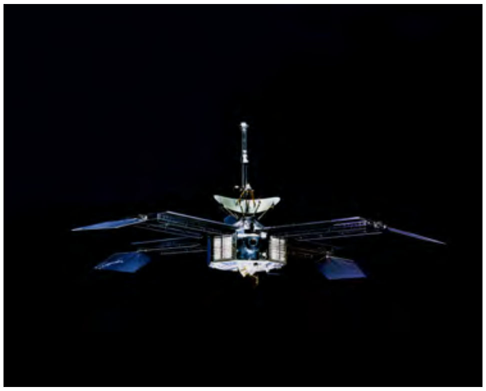


Figure 1.11: The Mariner IV Spacecraft. Credit: NASA/Jet Propulsion Laboratory.

1.2.2 Martian Robotic Systems

1.2.2.1 Mariner IV

The first successful mission to return data about another planet in close proximity, Mariner IV, followed the failure of Mariner III due to a launch vehicle failure [28]. The 260.8kg spacecraft carried similar instrumentation to Ranger VII, save for a trapped radiation detector for inferring the presence of a strong magnetic field. Data transmission rate for much of the mission was 8.5 bits per second. [37]. 22 TV images of the surface were returned and a very weak magnetic field was detected. At the time, it was still thought that there was intelligent life on Mars; these pictures and the data supporting surface temperatures of -100^{o}C put such theories to rest. Mariner IV continued to be tracked until its propellant was exhausted, which was markedly faster than anticipated. This was due to the spacecraft flying through a comet tail; dozens of micrometeroid strikes were recorded by the cosmic dust detector, which perturbed the spacecraft and caused loss of lock several times.

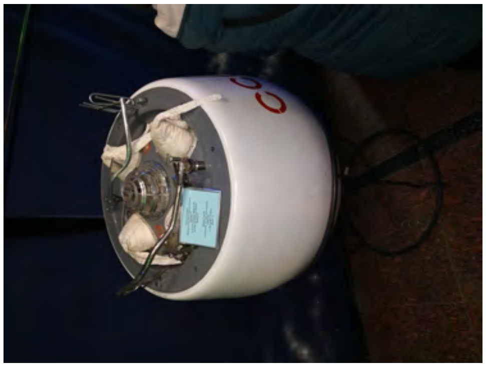


Figure 1.12: A model of the Roscosmos Venera 7 spacecraft. Credit: Emerezhko/Wikimedia Foundation.

1.2.2.2 Venera 7

While clearly not a Martian spacecraft, it is worth reviewing Venera 7. The 1,180kg spacecraft was one of a series of many, highly successful Soviet probes to Venus. On December 15, 1970, Venera 7 became the first spacecraft to land, and transmit data from, the surface of another planet [28]. Roscosmos engineers and scientists redesigned each successive Venera spacecraft to withstand higher and higher pressures and temperatures, as each prior version was destroyed at some point in the atmosphere. Venera 7 was designed to withstand 18MPa and 580^{o}C, an extreme amount of pressure that was not initially expected [38].

The probe entered the atmosphere at 11.5km/s and encountered over 200g of acceleration, similar to other Venera probes. The parachute opened to a 97% CO_2 atmosphere at 60km altitude. Data continued to be taken until perceived impact and the mission was declared complete. However, upon further review of the signal at a later date, an extremely weak and seemingly corrupt signal was recovered. After applying corrections, the data showed Venera 7 landed on the surface of Venus and continued to transmit pressure, temperature, and atmospheric composition data for another 23 minutes [39]. This fact conclusively proved, just as Mariner IV did for

Mars, that there was neither water nor intelligent life on the surface of Venus, as a pressure of 9.3MPa and 475^{o}C with a 2.5m/s wind was recorded at the surface.

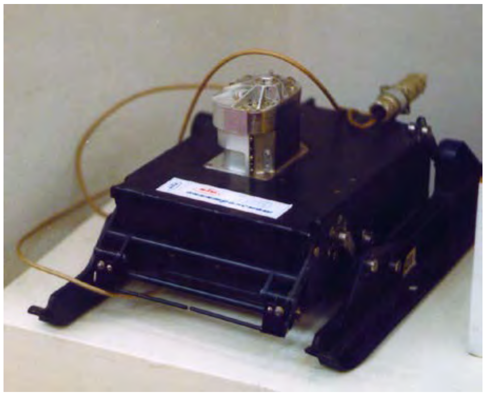


Figure 1.13: A model of the PrOP-M rover, a part of the Soviet Mars 2 mission. Credit: T. Varfolomeyev.

1.2.2.3 Mars 3

Mars 3 followed the failure of the Soviet Mars 2 to perform a soft landing. After an uneventful cruise, the entry, descent and landing of the 1,210kg lander on December 2, 1971 occurred smoothly. Immediately after landing, the Mars 3 orbital relay began to receive an image. However, this success was short-lived. The probe failed after only 20 seconds of transmission, possibly due to immense static buildup as a result of landing during a punishing global dust storm [28]. Despite this failure, the Soviets could once again gain its claim to fame for being the first to successfully soft-land on Mars [40].

Mars 3 also carried what would have been the first rover on Mars: the PrOP-M rover. Shown in Figure 1.13, the 4.5kg rover was a very small, CubeSat-sized rover that operated by "walking" on the surface with two skids while tethered to the lander [40]. It carried a ^{137}Cs gamma-ray source to perform surface density measurements and a penetrometer for soil hardness tests.

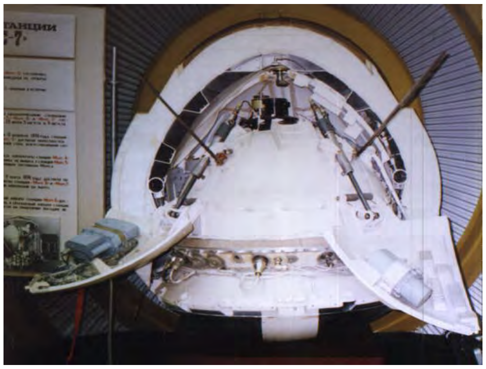


Figure 1.14: A cutaway of a full-scale model of the Mars 3 lander, on display at the NPO Lavochkin Museum. Credit: Alexander Chernov.

Mars 3 would not have been able to communicate directly to Earth, so it brought a relay along. This relay also included a very generous set of scientific equipment [28]:

- infrared (IR) radiometer to sense temperature, much like Mariner IV did,
- microwave radiometer to sense UHF radio frequency emissions from Mars,
- photometer, essentially a calibrated camera,
- 4-channel ultraviolet cameras,
- two high-resolution cameras delivering 480 images,
- triaxial fluxgate magnetometers,
- cosmic ray detectors,
- particle traps to detect certain low-energy particles,
- a French radio astronomy instrument,
- and a specific RF transmitter to determine atmospheric structure.

All these instruments operated successfully. The French instrument, the Stéréo-1 radio astronomy instrument, returned over 1 megabyte of data after operating for over 180 hours [40]. All in all, Mars 3 was a fairly successful mission.

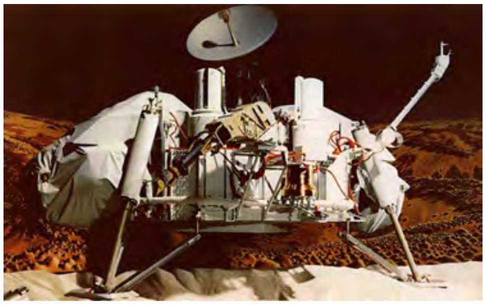


Figure 1.15: A model of the Viking 1 lander. Credit: NASA NSSDC.

1.2.2.4 Viking 1

Viking 1, at the time, was among the most complex robotic spacecraft launched. It was a high-stakes mission because it had a formidable compliment of instruments, including those that would explicitly determine if life was extant on Mars. The Viking landers were also the first to use nuclear power sources on an interplanetary mission. The 2,339kg orbiter had the following instruments:[28]

- imaging system using two vidicon tubes[1] ,

- IR spectrometers to map water vapor and surface thermal properties.

The 978kg lander had the following instruments:[28]

- two "fax" cameras, which scanned line-by-line providing high resolution digital images,

- gas chromatograph mass spectrometer, which measured the composition of heated gases,

- a seismometer, which unfortunately failed to deploy,

- an X-ray fluorescence spectrometer, which used X-ray radiation to illuminate and identify different compounds,

- a full, *wet* biological laboratory with the intention of determining if life was on

[1] Vidicon tubes, at the time, were starting to become dated due to the emergence of CMOS imagers. Reliability was a key reason they flew into the 70's and 80's.

Mars,

- a weather suite, measuring pressure, temperature, and wind, and
- a remote sampler arm.

To power all these instruments without having to be constrained by dust storms or inefficient and fragile solar panels of the time, two 13.6kg radioisotope thermoelectric generators (RTGs) were used. Each produced 30 watts electrical power at 4.4 volts, which charged a 28 volt, 8 amp-hour battery pack to handle peak loads [41]. The RTGs on their own were not able to support the load of individual experiments; however, the waste heat generated from the RTGs allowed systems to be safely powered off to allow recharging of batteries. Due to the RTG power provided, the guaranteed mission time was much longer than if the spacecraft were to have solar panels. A comparable, contemporary spacecraft with this problem is InSight, discussed in a later section.

The science data subsystem included a 40 megabit data tape recorder (DTR) and an S-band transmitter with redundant, 20-watt traveling wave tube amplifiers (TWTAs). The tape recorder was important since a constant link could not be maintained with Earth, like was done with missions to the Moon. All commands for science operations were auditioned on ground hardware before being sent to the spacecraft. Science data playback was at one of three rates: 250, 500, or 1000 bits/sec. To ensure the weak direct-to-Earth (DTE) link was stable and that bit errors could be readily corrected, biorthogonal block coding of 6 data bits for every 32 redundant bits were transmitted (32:6 in telecommunications notation) [42]. More efficient coding schemes emerged later; these are discussed further along in the thesis. Commands were sent up to the spacecraft at 4 bits/sec, and engineering data was received at 8.33 bits/sec.

Extensive efforts were made to decontaminate and sterilize the spacecraft. JPL went as far as purifying the poisonous hydrazine propellant [41]. This was critical because any contamination to the biological sampling payload could potentially produce a false positive.

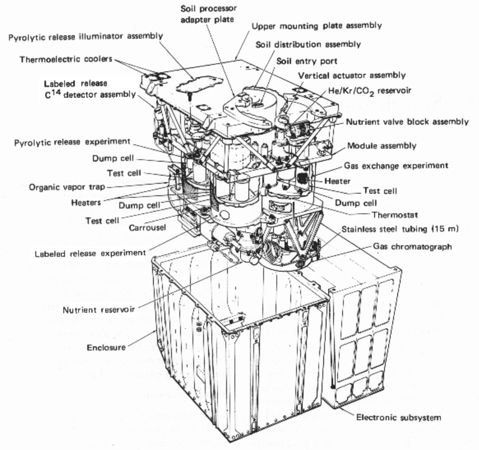

Soil processor adapter plate
Upper mounting plate assembly
Pyrolytic release illuminator assembly
Soil distribution assembly
Thermoelectric coolers
Soil entry port
Vertical actuator assembly
Labeled release C14 detector assembly
He/Kr/CO2 reservoir
Nutrient valve block assembly
Pyrolytic release experiment
Module assembly
Dump cell
Test cell
Gas exchange experiment
Organic vapor trap
Heater
Test cell
Heaters
Dump cell
Dump cell
Thermostat
Test cell
Carrousel
Stainless steel tubing (15 m)
Gas chromatograph
Labeled release experiment
Nutrient reservoir
Enclosure
Electronic subsystem

Figure 1.16: A detailed drawing of the Viking Biological Package, an experiment specifically designed to determine if life was present on Mars. Credit: NASA.

Figure 1.16 shows an engineering drawing of the central experiment to the Viking series. In one sub-experiment, called the Labeled Release (LR) experiment, soil from a sampler arm was deposited into this experiment's well. Radioactive ^{14}C tracer in a nutrient solution was injected into this soil, and the air above the sample was monitored for changes in radioactivity, which would determine if a biological process was metabolizing the nutrients [43]. The positive result from this experiment has continued to be highly contentious and have been represented by many as a false positive [44]. Journal articles from as recent as 2016 investigate data from the Viking Biological Package, incorporating data from recent Mars missions in an argument to continue to consider biological processes for the positive result [43].

Viking 1's mission was inadvertently ended in 1985 due to human error in command sequencing. Engineering commands meant to optimize the degrading battery's operation overwrote the antenna pointing firmware, resulting in an invalid antenna position. Over the next 4 months, JPL tried to reestablish communications based on the presumed direction the antenna was pointing, but were unsuccessful [42]. However by this point, Viking 1 had more than exceeded its' design life, and returned vast amounts of data.

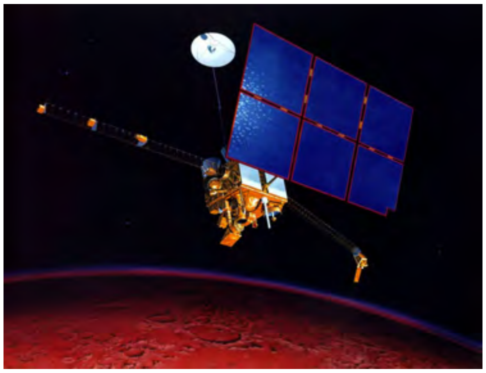


Figure 1.17: A rendering of the ill-fated Mars Observer. Featured prominently are the large solar panels, long instrument booms, and steerable dish antenna. Credit: NASA.

1.2.2.5 Mars Observer

After a very long lull of Mars missions, due to focusing of constrained agencies' resources towards Venus and the outer planets (Voyagers 1/2), Mars Observer was launched in 1992. This 1,018kg spacecraft was designed to image the entire surface of Mars over the course of a year. Mars Observer had an uneventful launch and cruise, but only three days prior to orbital insertion on August 21, 1993, the spacecraft suddenly ceased to transmit. The root cause of the failure was never conclusively determined, but an independent review from the Naval Research Laboratory stated that the most likely cause was a slow leakage of hypergolic propellant and oxidizer into the plumbing of the propulsion system. When the engine was reactivated, the engine exploded, potentially completely destroying the spacecraft [45].

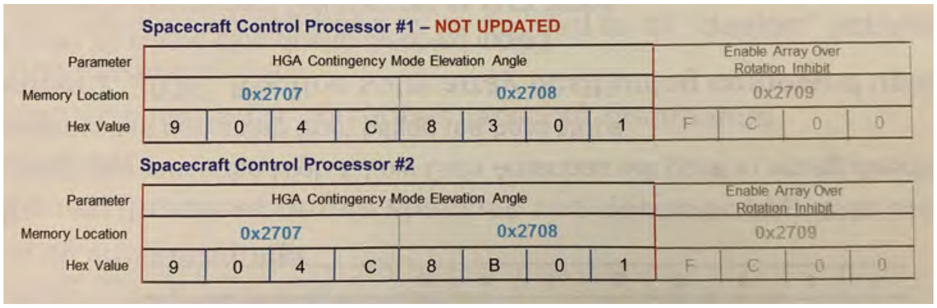
Figure 1.18: An artist's rendition of Mars Global Surveyor, the first spacecraft to deliver a global photographic and height map of Mars. Credit: JPL/Caltech.

1.2.2.6 Mars Global Surveyor

Following the investigation into the failure of Mars Observer, JPL, as the sole contractor of the "Faster, Better, Cheaper" Daniel Goldin-era of NASA, once again attempted a global mapping mission. Mars Global Surveyor (MGS), a 1,030.5kg spacecraft, launched on November 7, 1996. A procedure for aerobraking on arrival on September 11, 1997 was modified after one of the solar panels failed to completely deploy. Additionally, during the first phase of aerobraking, operators discovered that one solar panel had started bending backwards due to aerodynamic stresses [46] as a result of a 110 km periapsis [47]. Consequently, aerobraking took far longer than expected - it was completed in March 1999.

Most instruments on board MGS were carried over from Mars Observer, including an orbital relay, which would forward data from any surface asset back to Earth. This function was heavily utilized when the rovers Spirit and Opportunity arrived in 2003.

MGS was a highly successful mission: it returned over 83,000 images of the surface and produced a global height map of Mars consisting of 500 billion laser altimeter points [28]. Reflights over certain locations monitored geological evolution over its active period, such as erosion due to ice and wind. MGS was also the first spacecraft to image dust devils from orbit.



Spacecraft Control Processor #1 – NOT UPDATED

Parameter	HGA Contingency Mode Elevation Angle								Enable Array Over Rotation Inhibit			
Memory Location	0x2707				0x2708				0x2709			
Hex Value	9	0	4	C	8	3	0	1	F	C	0	0

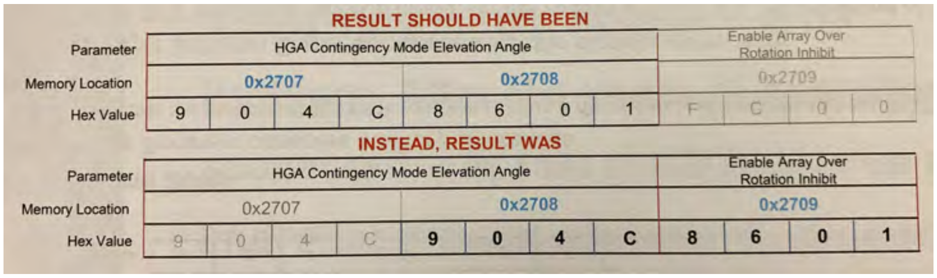
Spacecraft Control Processor #2

Parameter	HGA Contingency Mode Elevation Angle								Enable Array Over Rotation Inhibit			
Memory Location	0x2707				0x2708				0x2709			
Hex Value	9	0	4	C	8	B	0	1	F	C	0	0

Figure 1.19: The affected register location of a memory discrepancy between the primary and backup flight computers on Mars Global Surveyor. Credit: Aerospace Engineering Associates, LLC.

After a mission extension was granted in October 2006, the spacecraft operated for another month, until a solar array reposition command disabled the spacecraft. In Figure 1.19, two processor registers are shown containing three memory locations. In September 2005, issues with the spacecraft computers resulted in only the backup computer, Spacecraft Control Processor #2, receiving the update. Between this update and another command update scheduled in June 2006, the lead flight software engineer noticed a discrepancy at `0x2708` (`CMODE’HGA’ELE’ANGLE`): Computer #1 contained `0x8301` while Computer #2 contained `0x8B01`. This engineer retired shortly after documenting the discrepancy. The new lead flight software engineer *assumed* that the previous engineer’s note on a discrepancy at memory address `0x2708` was actually an error: they meant to write `0x2707`.

The rationale behind the new engineer’s assumption can be given credibility



RESULT SHOULD HAVE BEEN

Parameter	HGA Contingency Mode Elevation Angle								Enable Array Over Rotation Inhibit			
Memory Location	0x2707				0x2708				0x2709			
Hex Value	9	0	4	C	8	6	0	1	F	C	0	0

INSTEAD, RESULT WAS

Parameter	HGA Contingency Mode Elevation Angle								Enable Array Over Rotation Inhibit			
Memory Location	0x2707				0x2708				0x2709			
Hex Value	9	0	4	C	9	0	4	C	8	6	0	1

Figure 1.20: The intended processor registers to be updated versus what actually happened. Credit: Aerospace Engineering Associates, LLC.

because the processor in use at the time, an RCA 1750A, was a 16-bit CPU [48]: they would have thought the retired engineer was one byte off according to the diagram in Figure 1.19. Unfortunately, the new senior engineer's assumption was incorrect as shown in Figure 1.20. The unintentional offset had two fatal effects:

- it disabled over-rotation protection on one of the solar panel gimbals, and

- it moved the high-gain antenna to a totally incorrect position that pointed away from Earth *because* of going into safe mode due to perceived gimbal failure by the flight computer.

A very weak signal was detected briefly in the following days, but loss of attitude determination resulted in the batteries overheating, which subsequently resulted in loss of power [49].

1.2.2.7 Mars Pathfinder

Mars Pathfinder was the Americans' first attempt to launch a fully independent rover to another planet. The 870kg mission carried a 10.6kg [50] rover called *Sojourner*, 28 x 65 x 48cm in dimensions - similar to a microwave oven [28]. Pathfinder was launched on December 4, 1996, and landed on July 4, 1997 at 2:56:55 am local Mars time [51].

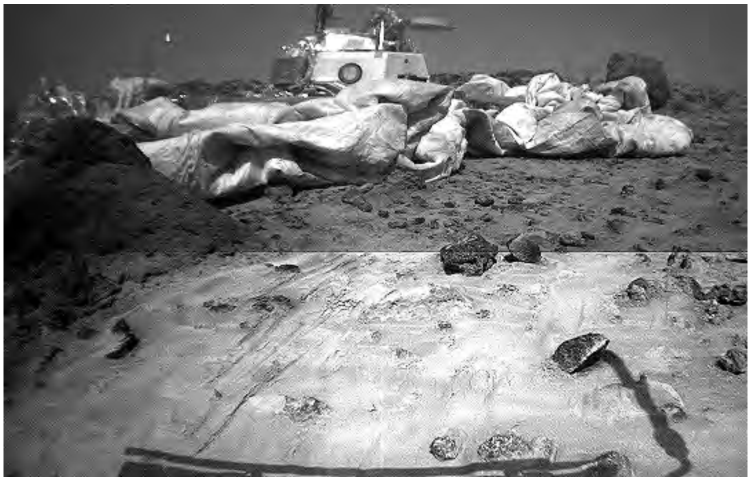


Figure 1.21: The Mars Pathfinder Lander, as imaged by the Sojourner Rover. Credit: NASA JPL/Caltech.

Interestingly enough, Pathfinder's primary objective was not science: it was to prove that the "faster, better, cheaper" mantra of Daniel Goldin's NASA was tenable, and that scientific instruments could be sent to the surface of Mars at only one-fifteenth the cost of the Viking program [51]. Indeed, the entire mission, including R&D, I&T, and launch and operations cost $440 million (in 1997 dollars). This was in contrast to the $3.5 billion 1997 dollars that the Viking Program cost [52]. The reference to "one-fifteenth" the cost and the clear price discrepancy is likely due to NSSDC citing the R&D cost cap of $150 million 1997 dollars.

MPF used a less than conventional approach to landing on the surface: after a conventional entry using a heatshield and supersonic parachute, an airbag-assisted landing was performed. Determined by landing radars, at 355 meters above the surface, three solid rockets fired from the backshell, still descending under a parachute. This slowed down the landing package to zero vertical velocity, where the bridle was then cut. Airbags cushioned the landing to a much more gentle 18g of acceleration. The spacecraft bounced at least 15 more times before coming to a rest.

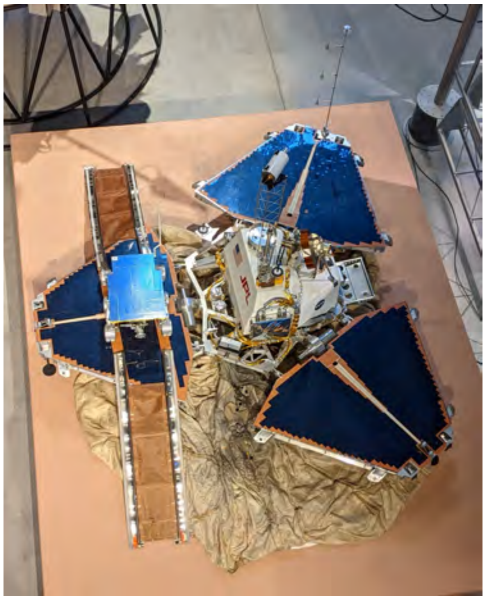


Figure 1.22: Mars Pathfinder and Sojourner model viewed from above in the Udvar-Hazy Center. Credit: NASA NSSDC.

In a similar vein to Soviet Lunar and Martian systems, the base station used petals to ensure the spacecraft was deployed the correct side up to allow the rover to roll off the base station. Figure 1.22 shows these petals, illustrating the tetrahedral design. By having only four sides and three petals, barring the failure of one of the petal actuators, the spacecraft would be guaranteed to fold out upright [28].

Originally designed to last only 7 and 30 days, the respective rover and lander ended up lasting 85 Earth days until the final communication. Being a technology demonstration, the mission had fulfilled a comprehensive success by their expected lifetimes. The rover transmitted 550 pictures and took several spectrometry samples of rocks, while the base station managed to take over 15,000 images and 8.5 million ASI weather station measurements [28]. From the images and spectrometry data, scientists determined that the system's landing site, Ares Vallis, hosted andesitic rock - similar to rocks found near volcanoes.

Table 1.1: Sojourner Rover Specifications [1].

Characteristic	**Specifications**
Total Mass	16 kg
Mobile Mass	11.5 kg
Lander-Mounted Rover Mass	4.5 kg - RF equipment
Autonomous Navigation	Laser striping to detect obstacles
Command and Telemetry	400 MHz UHF link to lander
Payload	Fore and Aft cameras, Alpha Particle X-ray Spectrometer (APXS)
Power	0.25m^2 solar array - 16Wh. Battery - 50Wh
Thermal	3x Department of Energy ^{238}Pu RHUs
Rover Computer	2 MHz Intel 80C85, 512kB RAM, 1.5W.
Imager Characteristics	5.2kg, 2.6W.
APXS Characteristics	0.74kg, 0.8W.

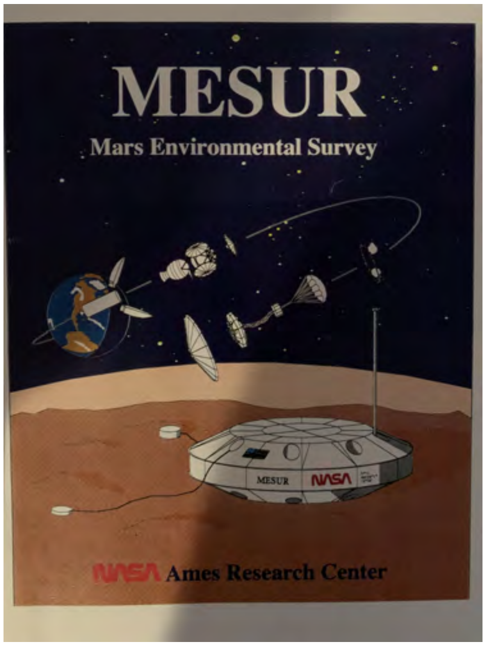


Figure 1.23: Figure from an unpublished report showing MESUR mission sequence from launch to landing. Photocopy credit: Stanley Krześniak.

1.2.2.8 MESUR

MESUR, or Mars Environmental Survey, was a proposal pitched by NASA Ames Research Center on July 19, 1991, in competition to Mars Pathfinder [52]. MESUR, in the original 1991 report, was designed to deploy a global Martian sensor network designed to measure weather. Although this proposal was never selected, it is worth including into this literature review due to the similarity of the mission described in this thesis.

Most information about MESUR is from a potentially unpublished copy found in an SJSU aerospace engineering lab, cited as [2]. MESUR's objectives were as follows:

> 1. Objectives that require the simultaneous operation of a number of globally-distributed [sic] surface stations. The primary examples are a global seismic network and a global network of meteorological stations.
> 2. Objectives that require sampling of a large number of globally-distributed [sic] sites. Examples include geological sampling, high-resolution surface imaging, and measurement of atmospheric structure along entry profiles. Particular emphasis would be placed on hard-to-reach sites (polar deposits, rugged volcano flanks, etc.) that would be difficult or impossible to investigate by other means.
> - MESUR Report, p.3 [2]

The MESUR program was intended to cost less than $1 billion to launch 16 probes over four launches, which would have carried four probes at a time. Figure 1.24 shows this configuration in a Delta launch vehicle.

Unlike the Mars Pathfinder mission, the primary objective of the mission was not for technology demonstration, but for scientific returns. The 16 MESUR probes would have carried the following instruments:

- Three-axis Seismometer with $10^{-10}g$ sensitivity,

- Meteorology Package measuring temperature, pressure, and wind speed and

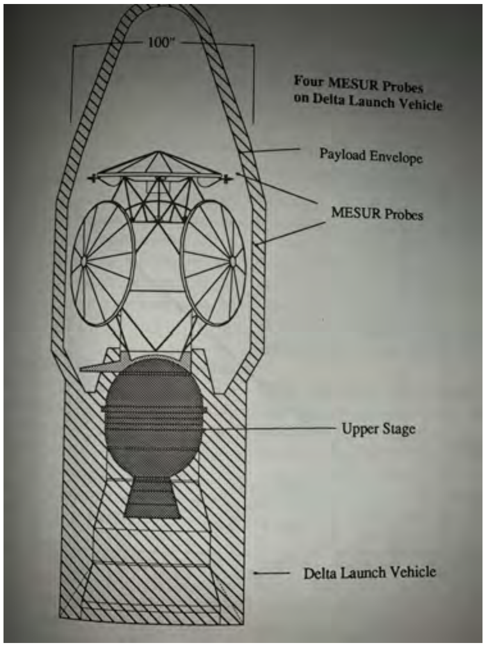


Figure 1.24: MESUR configuration inside a then-U.S. Air Force Delta II launch vehicle. Four MESUR probes are shown on a structure mounted to a solid rocket booster upper stage. The entire configuration would have fit within a 100-inch diameter. Credit: Stanley Krześniak.

direction,

- Elemental Composition Instrument, or APXS, which is exactly the instrument described on Pathfinder [52],

- Thermal Analyzer/Evolved Gas Analyzer, which was actively designed to sample the surface and heat it to determine the chemical composition of subsurface soil,

- Descent Imager to establish geological context of the probe,

- Surface Imager to determine day-to-day changes to the environment, and

- Atmospheric Structures Experiment measuring the atmospheric properties during descent, inferred from pressure, temperature, and acceleration measurements.

All the instruments described have since been flown on subsequent rover and lander missions to Mars.

To allow deployment at any latitude on Mars, and to be able to operate through dust storms, all sixteen MESUR probes would have contained a radioisotope thermoelectric generator (RTG). These would have been built around the U.S. Department of Energy's General Purpose Heat Source (GPHS), a standard that is still used to this day for most space missions that require RTGs [53]. One or two GPHS's would have been used in the design, sufficient to provide electrical energy while load sharing between instruments and thermal energy to keep all instruments within operating temperature.

A brief literature review [53] [54] [55] [56] shows that there is currently only one RTG available for deep-space usage, which is produced by the U.S. Department of Energy's Space and Defense Power Systems division. Previously produced RTGs include the odd-numbered SNAP series, which includes the SNAP-19. It has significant heritage - it powered the Pioneer deep-space series, Viking Mars landers, Galileo, and Cassini. The Multimission RTG (MMRTG), currently produced by the U.S. DOE, is NASA's only choice of RTG. It has been used on MSL, Mars 2020, the New Horizons Pluto mission, and will be used on Dragonfly, the Titan multi-rotor helicopter. This literature review within the MESUR mission serves to illustrate the contemporary difficulty of developing space-based nuclear power. Very few nuclear power sources have been designed since the 1960s; in fact, the entire SNAP series was developed between the 1950s and 60s. The MMRTG was a multi-year development that improved safety and efficiency, which was finalized by 2004. As necessary as an RTG might have been for MESUR, it may have also been its' major Achilles heel in the face of very limited space exploration budgets in the 1990s. As quoted from the MESUR report:

> "...because the GPHS [general purpose heat source] is already flight-qualified, repackaging the rest of the unit to produce a small MESUR RTG should not require a major development. Smaller RTG's are probably not practical because they would force the thermal brick design into flight requalification, which represents a major technology effort. The requalification process would also be required if the current GPHS brick was used and the number of fuel pellets was reduced" [2].

It is not clear that the MESUR team understood that creating a custom RTG for a low-cost mission, decades after the Space Race, was going to be financially and resource intensive - an oxymoron at best as shown in this quote. Repackaging a GPHS, as shown in Figures 1.25 and 1.26 to use less than four plutonium fuel pellets is a non-starter; since the GPHS design, every mission has used multiple GPHS bricks in the designed-as-intended configuration.

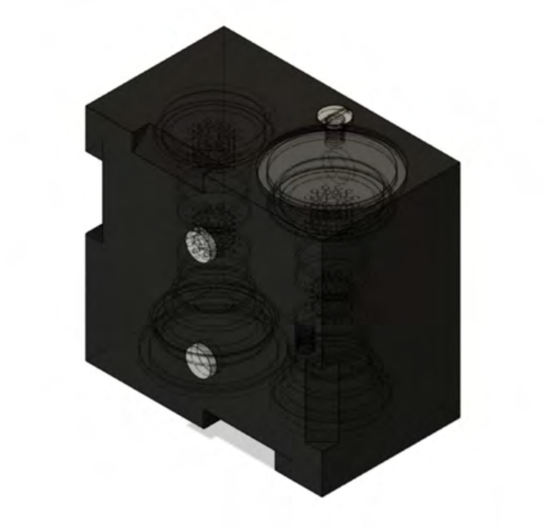


Figure 1.25: General purpose heat source CAD drawing, fully assembled and defined. Four plutonium fuel pellets are visible as silouhettes, with pairs encapsulated in their own silos contained in carbon-carbon sleeves. Credit: Stanley Krześniak.

Another reason MESUR likely was not selected is telecommunications: MESUR would have primarily relied on direct-to-Earth (DTE) communication via S-band,

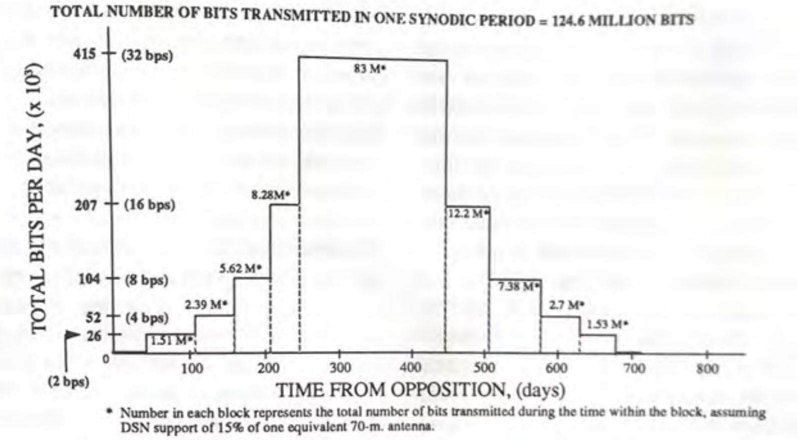
Figure 1.26: General purpose heat source CAD drawing, exploded. Credit: Stanley Krześniak.

2,200 MHz, for the first Martian year. For scientific instruments such as seismometers, returning 37 MBit per day [57], this is a red flag in the systems engineering trade study. According to an *Insight* mission literature review, the SEIS seismometer observed 20 events of M_w 3.0 to 4.0 over the timespan of 300 days [58]. Returning 37 Mbit per day, this represented 11.1 GBit over the period studied in the publication. Table 1.2 shows why this would have never been attainable - at near-maximum distance or opposition from Earth, the bit rate was estimated at 2 bits/second, or 1/4 character per second. Obtaining meaningful scientific data would simply take too long, the risk too high to wait for a relay, and continuous measurements would have been impossible.

All things considered, this assumed that JPL's Deep Space Network 70-meter antennas would be used, with 15% total utilization [2]. With JPL needing to support several other deep-space missions at the time, MESUR would have occupied too much antenna time for too little science return. Figure 1.27 illustrates this fact. It represents a best-case, or ideal scenario, which is never representative - few if any missions accomplish all objectives perfectly or as scientists intend them to. Even if the best-case were to be achieved, there would simply be too little data return. The mission was also predicated on there being a separate, follow-on orbital relay being sent the next Martian year. While the Viking landers lasted several Martian years, they were not constrained by solar power, a severely limited budget, and direct-to-Earth communications. As shown by Pathfinder, the mission only lasted 3 months with similar technology to MESUR, failing due to battery failure and inability to heat itself adequately at night.

Table 1.2: Maximum Mars/Earth Communications Distance (Mm) vs Data Rate and Transmitter Power [2]. For comparison, 1 astronomical unit (AU) is 150 Mm.

Bit Rate (bps)	**Transmitter Power (W)**				
	4	6	8	10	12
2	240	294	339	379	416
4	212	260	300	335	367
8	178	218	252	281	308
16	127	155	179	201	220
32	90	110	127	142	156
64	64	77	90	100	110
128	45	54	64	70	78
256	32	38	45	50	55



TOTAL NUMBER OF BITS TRANSMITTED IN ONE SYNODIC PERIOD = 124.6 MILLION BITS

* Number in each block represents the total number of bits transmitted during the time within the block, assuming DSN support of 15% of one equivalent 70-m. antenna.

Figure 1.27: MESUR network data rate capability using 15% of one DSN 70m antenna.

Despite not being selected, components and scientific objectives of the mission were reused for Mars Pathfinder and many other NASA/JPL missions going forward; most science requirements have since been fulfilled, up to Phoenix, InSight, the MER rovers, and Curiosity/Perseverance. In particular, InSight's weather instrumentation has made intriguing discoveries with respect to the Martian magnetic field and weather phenomenon. It has discovered very close similarities to Earth's atmosphere, particularly baroclinic and gravity waves, and convective vortices. Airglow has also been discovered through InSight's cameras, with similar processes to Earth's formation of airglow [59].

The only outstanding scientific requirement to be fulfilled is distributed, in-situ weather and geophysical monitoring of Mars. As of writing, there are no publically known comprehensive plans for such probes.

1.2.2.9 Deep Space 2 and Mars Polar Lander

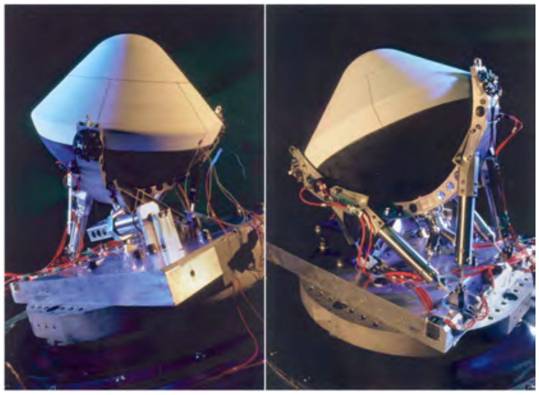


Figure 1.28: Deep Space 2 probes mounted to Mars Polar Lander bus. The heatshield was of a 45° design to accommodate its parachuteless crash landing.

The two 3.57 kg Deep Space 2 (DS2) microprobes shown in Figure 1.28, part of the 576 kg Mars Polar Lander mission, were launched on January 3, 1999. The trio of spacecraft were slated to explore and characterize polar ice near the Martian south pole. Controllers at NASA maintained control over the spacecraft up until entry, where they were expecting to regain contact with all spacecraft approximately 24 minutes after radio blackout [28]. The probes entered the atmosphere at 6.9 km/s.

Contact was never reestablished with Mars Polar Lander, and Mars Global Surveyor was never able to reach either of the DS2 probes.

According to a JPL Special Review Board commissioned to study why all three probes failed, the most probable cause for MPL's failure was a faulty indication of surface contact from the foot pad sensors during the deployment of the landing legs. Landing leg deployment would have occurred at 40 meters, at which the software logic for ignoring spurious foot pad signals was also deactivated. To add to this, a program management failure was the lack of entry telemetry and lack of DS2 system checkout capability prior to entry, which had previously been required to enable understanding of the EDL system's performance, as well as assisting in understanding failure mechanisms if one were to occur [60].

The programmatic failure at a NASA level was a very high pressure, understaffed, and severely underfunded mission due to JPL and Lockheed Martin Space's understanding that cost increases beyond the $165 million were not permissable. Additionally, according to a report to the U.S. House Science and Technology Committee in 2000, the combined MPL/DS2 mission cost less than the $200 million Pathfinder, and demanded "three times more science" [61]. Pathfinder was intended as a technology demonstration, which lends credence to this claim.

Just two and a half months prior, the ill-fated Mars Climate Orbiter disintegrated 57 km over Mars due to a unit conversion error from Metric to Imperial. It was also uncovered in the same report to the House Science and Technology Committee that similar unit conversion issues were uncovered in flight software. The lack of software testing and review was further evidence for the severe lack of funding, general project management inexperience, and lack of transparency [61]. The DS2, Mars Polar Lander, and Mars Climate Orbiter saga spelled the end of the "Faster, Better, Cheaper" mantra.

Had the DS2 probes successfully landed, they were to have endured up to 800,000m/s^2 (80,000g) of acceleration on hitting the ground, deploy a subsurface probe to sample soil conductivity and the soil itself, and to deploy a small weather sensor to determine ambient conditions [62]. The lithium thionyl chloride batteries and electronics were specifically designed to endure high acceleration and temperatures as low as -80°C. Unfortunately, these articles were never tested as a complete system, which could never guarantee survival at the very extreme peak acceleration on ground impact [60]. Additonally, blunt body entry vehicles tend to become less stable at lower Mach numbers. The phenomenon of "coning" increases as the vehicle slows down due to atmospheric friction. It is very likely that neither of the DS2 probes crash-landed with a zero angle of attack, thus limiting the effect of the crumple zone and preventing the release and deployment of the penetrator probe.

1.2.2.10 Perseverance

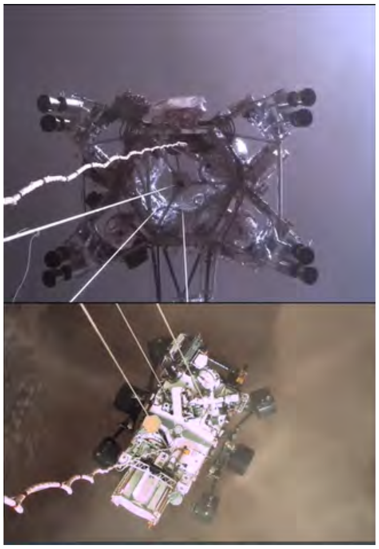


Figure 1.29: Screenshot of JPL video, replaying multiple angles of Mars 2020 while landing. The upper portion of the frame is a view from the rover looking up at the skycrane system, and the lower frame is the rover suspended from the skycrane.

A substantial upgrade to Curiosity, Mars 2020's Perseverance landed on February 18, 2021. The 3,645 kg system includes among the largest suite of instruments of any spacecraft surveyed in this literature review, which even includes an in-situ resource utilization demo, and showcased the high-precision "skycrane" landing system in multiple high resolution video camera views after heat shield deployment, both major firsts for an interplanetary mission. Most importantly, Perseverance carries a highly automated sample caching system, which is the first step of many in returning samples from the surface of Mars to Earth, which itself is a blueprint for NASA's current long-term priority of sending humans to the surface of Mars. Table 1.3 surveys the main scientific instruments and their capabilities, which does not include the large array of engineering instruments and systems.

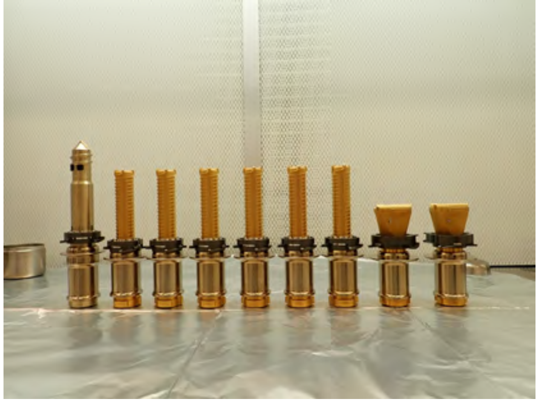


Figure 1.30: Perseverance's sample caching system drill bits. On the far left with a conical tip is a regolith drill, designed to drill into softer soil. The six drills in the middle are rock-coring drills, which have a cavity inside to allow rock to be deposited into a sample tube. On the right are two abrasion drills, which clean off a surface before measurements or a sample is taken. The gold color on all drills and abrasion tools are titanium nitride. Credit: NASA/JPL.

Of particular interest from Perseverance is the RIMFAX (Radar imager for subsurface experiment/Hrímfaxi) instrument. RIMFAX is an ultra-wideband (UWB) radar sounder designed to determine subsurface features at depths greater than 10 meters. Figure 1.31 shows the location of this instrument on the rover, and Figure 1.32 shows a detailed view of the signal processing electronics and enclosure.

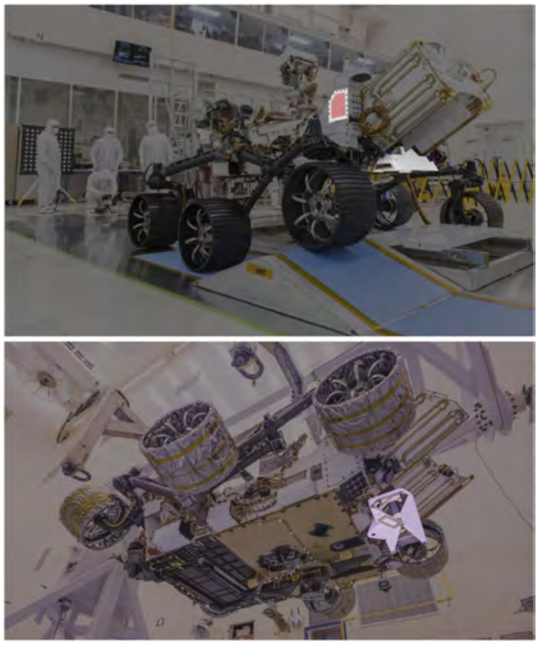


Figure 1.31: Location of RIMFAX data processing unit (top) and antenna (bottom). Credit: NASA/JPL/University of Oslo.

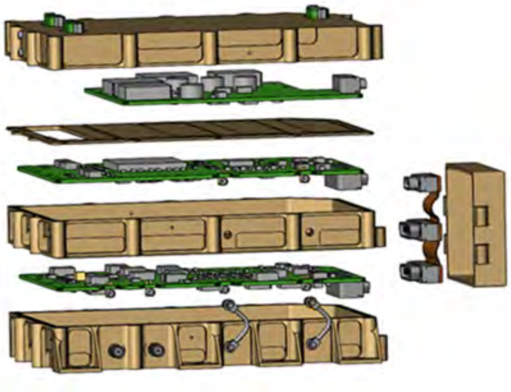


Figure 1.32: Internals and assembly of RIMFAX signal processing and conditioning. Credit: University of Oslo.

Ground-penetrating radars are ubiquitous in Earth geophysical sciences, and have extensive heritage in Martian remote observations - notably NASA Martian Reconaissance Orbiter's SHARAD [63], and ESA Mars Express's MARSIS [64]. Both radars have recently returned strong evidence of vast quantites of buried water ice and potentially liquid water deep below the surface. Figure 1.33 shows an example data product of a glacier on Earth, taken with a RIMFAX engineering unit. Planetary geologists continue to advocate for instruments that view terrain below the surface.

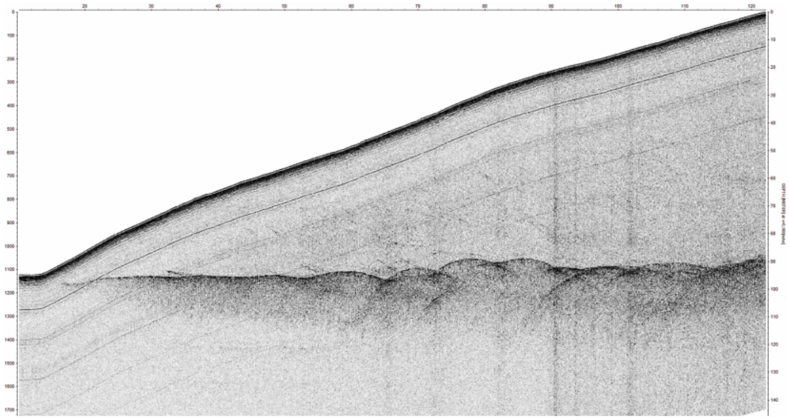


Figure 1.33: RIMFAX instrument test results at the Midtre Lovenbreen glacier in Svalbard, Norway. The solid, near-diagonal line is the surface, and features under it show different properties of the ice and water below the glacier. The strong, horizontal return is the "grounding line", where the ice meets rock. Credit: University of Oslo.

1.2.2.11 Ingenuity

Ingenuity is a landmark in aeronautics: it is the first aircraft to achieve controlled flight on another planet. The 1.8-kg helicopter was deployed from the bottom of Perseverance on April 3, 2021, and performed its first flight on April 17, 2021. As of writing, Ingenuity completed its 49^{th} flight, with over 11 km flown for nearly 1 hour and 30 minutes. Due to the novelty and extreme mass constraints, Ingenuity carries no scientific instruments [65]. Navigation is performed with off-the-shelf inertial measurement units, optical flow from a cellphone-grade camera,

Table 1.3: Perseverance Instruments [3]

Instrument	**Description**
Mars Oxygen ISRU Experiment (MOXIE)	Tech demonstration of oxygen generation from atmospheric CO_2
Planetary Instrument for X-Ray Lithochemistry (PIXL)	Detailed charictarization of surface minerals though X-ray exposure
Radar Imager for Mars Subsurface Experiment (RIMFAX)	Ground-penetrating radar to examine subsurface to 10m depth
Mars Environmental Dynamics Analyzer (MEDA)	Comprehensive weather station for in-situ weather conditions
SuperCam/AEGIS	Remote identification of biosignatures and chemical analysis
MastCam-Z	10x optical zoom camera
Scanning Habitable Environments with Raman and Luminescence for Organics and Chemicals (SHERLOC)	Ultraviolet Raman spectrometer and camera assembly to image detailed minerology and potential biosignatures
Stereo Microphones	Determine atmospheric structure and nature of Martian turbulence
Hazard Cameras (x16)	Hazard avoidance and additional spatial awareness for rover operators and artificial intelligence (AI)-based navigation

and a tilt sensor to initially calibrate the IMU. The most notable design feature of Ingenuity is that most parts are commercial, off-the-shelf. Aside from screening for higher single-event latchup (SEL) radiation immunity, the IMU, tilt sensor, and cameras are "cellphone-grade" [66]. Ingenuity already has faced a frigid Martian winter due to a lack of power required to heat itself from excessive dust accumulation on the solar cells [67]. Heaters kept Ingenuity to no lower than -15°C, but without them, internal temperatures dropped to lower than -80°C. Including the military-grade components, all parts were completely out of their operating temperature ranges [66] for an incredible 280 sols [68]. During this challenging winter,

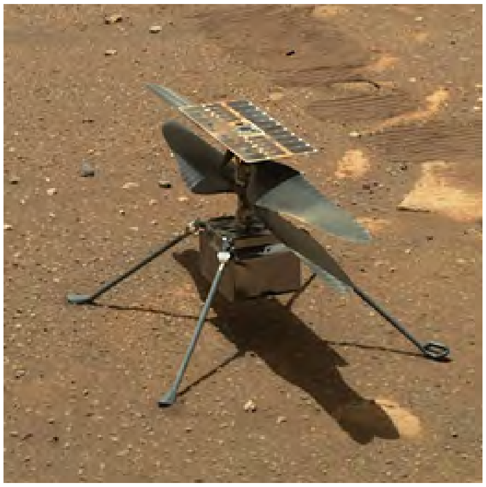


Figure 1.34: Ingenuity Helicopter sitting on the surface of Mars after being deployed by Perseverance. Credit: Jet Propulsion Laboratory.

dust storms blew through, further reducing solar output and the chance of survivability. Despite this, Ingenuity only managed to lose one component: the tilt sensor. A method for calibrating the IMU without the tilt sensor was developed, and was able to continue its unlimited mission extension. By April 2023, the Ingenuity chief engineer compared operating the helicopter to participating in a race with its base station, Perseverance [69], clearly demonstrating that special electronics are not always required.

Status update # 450 by the chief engineer shows a useful in-situ data point for local, point-to-point networks. Ingenuity uses a ZigBee(R) 900-MHz, IEEE 802.15.4 mesh network radio, similar to those used in smart utility meter networks [66]. Due to the high frequency used, nearly line-of-sight communication is required. In areas with steep canyons and narrow valleys, communication is limited to a few hundred meters, severely limiting the ability to downlink any useful scientific and operational information. During Flights 47 and 48, the team was unable to receive any information other than the helicopter successfully landed [69]. Although communications have been demonstrated at its maximum range of 1,000 meters, communication with ZigBee is slow. The helicopter generates about 700 megabits of data per flight - given the constraints, almost all the data has to be discarded [70]. For fixed mesh networks, robust link budget analyses should be conducted to determine instrument and power scoping.

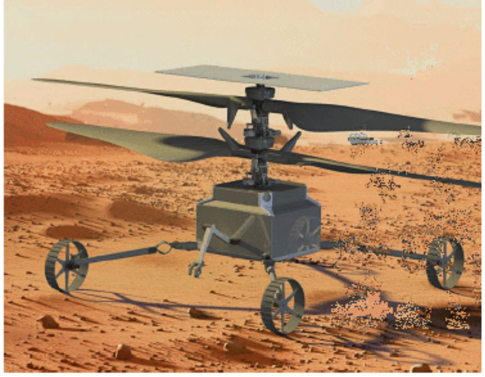


Figure 1.35: AeroVironment Sample Fetch Helicopter concept. Aside from the inclusion of a robotic arm and wheels, the helicopter design remains fairily similar.

Ingenuity has heralded a strategic sea change in scientific exploration: the trade space has opened up to helicopters with robotic arms that are also capable of carrying 5 kg of scientific instruments [70]. AeroVironment Inc's Sample Fetch Helicopter (SFH) is of Ingenuity heritage, and includes the ability to drive on wheels and pick up the sample tubes dropped off by Perseverance. A render is shown in Figure 1.35, showing the wheels and miniature robotic arm. It is currently in active development and is slated to be included on an upcoming Mars mission. This would support the Mars Sample Return mission as backup methods of delivering the cached samples to the ascent stage [71].

Another concept in active development is the Mars Science Helicopter, with a capability of carrying 5 kg of scientific instruments over 5 km range, with 3 minutes of hover time [70]. Such a helicopter would enable exploration of regions too dangerous for rovers or humans to land in, such as Valles Mareneris. It would continue to use the same technology and methods proved on Ingenuity, in particular, the power electronics and traditional helicopter-based collective and swashplate control.

While helicopters would be useful in their own right, for the Wavefront mission, stationary and buried assets are required. The success of Ingenuity came too far into the development of Wavefront to consider rotorcraft assets; that being said, it is the most important literature review piece for Wavefront. This is due to the success with off-the-shelf components and cold survivability. It greatly influenced the late design changes of the lander, nanoprobe, and rover, eliminating the need for extremely encumbering RTGs and RHUs.

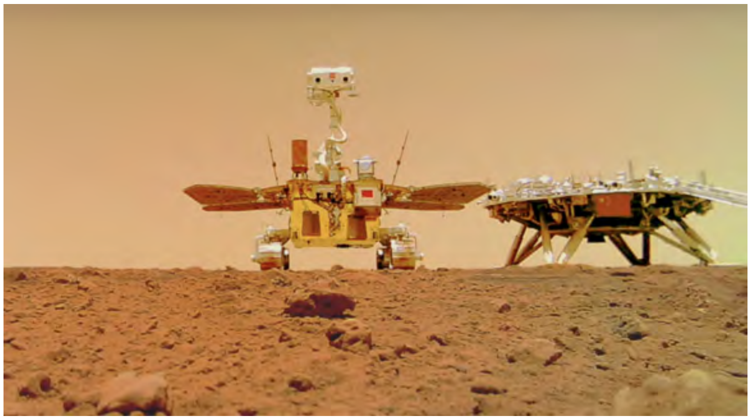


Figure 1.36: The Zhurong rover and lander system, imaged by a deployable remote imager. The imaging unit transferred data through a WiFi network. Credit: China News Service.

1.2.2.12 Zhurong/Tianwen 1

Zhurong and Tianwen 1 of China mark the fourth nation to soft land a vehicle on Mars [1] , and the second nation to land a rover on Mars. Launched on July 23,

[1] The United Kingdom's Beagle 2 was later determined by NASA's Mars Reconaissance Orbiter to have soft landed in 2003, but the probe was never heard from due to a solar panel actuator malfunctioning.

2020, Tianwen 1 and Zhurong flew in a Hohmann transfer window along with America's Perseverance and Ingenuity, and the United Arab Emirates's Hope. The Zhurong rover landed on 22 May 2021, after being deployed from the Tianwen 1 orbiter for a gentle, 4 km/s entry. Tianwen was formed as a direct result of the failure of the Russian Federation's Fobos-Grunt mission in 2012 [72], of which China provided the lander and rover.

A significant amount of engineering and scientific information is available on Tianwen and Zhurong in the scientific community. The 240kg and 1.85m tall rover with deployable boom [73] carried 7 instruments, tabulated in 1.4. The 3,715kg Tianwen 1 orbiter's instruments are tabulated in 1.5 [4]. Particular attention is paid to Tianwen due to the availability of this information. It provides a contemporary version of the American Viking 1/2 and Soviet Mars 3 architectures, which are instrumental in Wavefront's mission concept of operations development.

Table 1.4: Zhurong Instruments [4]

Instrument	**Description**
Mars Rover Penetrating Radar	Ground penetrating radar designed to image 100 meters below the surface
Mars Rover Magnetometer	Examines changes in crustal magnetic fields
Mars Meteorological Measurement Instrument	Weather instrument suite containing temperature, pressure, anemometer, and microphones
Mars Surface Compound Detector	Laser-induced breakdown spectrometer and infrared spectrometer
Multispectral Camera	Minerology characterization
Navigation and Topography Cameras	Automatic navigation camera system

Tianwen's mission concept of operations is similar to most NASA missions, with a notable exception. After launch and a speedy interplanetary cruise of just under 7 months, Tianwen 1 *and* Zhurong entered orbit on February 10, 2021. For

Table 1.5: Tianwen Instruments [4]

Instrument	**Description**
Mars Minerological Spectrometer	Identifies mineral distribution throughout Mars
Moderate Resolution Imaging Camera	Studies the characteristics of Martian topography and geological structure
High Resolution Imaging Camera	Studies the characteristics of Martian topography and geological structure
Mars Orbiter Magnetometer	Studies the Martian physical fields (electromagnetic, gravitational) and internal structure
Mars Ion and Neutral Spectrometer	Studies the ionosphere, surface climate and environmental characteristics of Mars

contemporary NASA missions, this is the point of departure. JPL has planned for and executed hyperbolic entries since Pathfinder due to several factors. Primarily, orbital insertion for such large missions is more of a mass penalty than carrying more thermal protection, since Curiosity data from the MEDLI instrument had shown less than 0.1-inch (2.54mm) of ablation [74] [75] at 5.8 km/s [76] - a nearly insignificant amount of mass. Tianwen 1 inserted into a $275x10,750$km by 86.3° orbit for imaging of the two candidate landing sites. In the literature, Zou, et. al. describe initial site selection through data obtained from MGS imagery and altimetry and MRO's SHARAD radar, and claim that the currently available spectral data is of low resolution, which necessitates the mission's investigation [4]. Ultimately, the site selection team chose Utopia Planitia based on orbital imagery from Tianwen 1 due to low crater densities, gentler terrain, and higher probabilities of finding evidence of an ancient ocean [77].

After site selection, Tianwen was programmed to release Zhurong at apoapsis [78]. The capsule entered at 4 km/s; peak acceleration values are not provided but could be reconstructed with significant error through the Mach number in [78]. Much like MSL and Mars 2020, Zhurong navigated to the target location through lift modulation at up to 50 deg angle of attack (AOA) during the hypersonic phase. After Max-Q and peak heating, a "trim wing" was deployed at Mach 2.8 to provide further target refinement, followed by disk-gap-band (DGB) deployment at Mach 1.8. Based on the literature review, this feature is novel. By published indications, the trim wing worked as intended. Upon reaching Mach 0.5, the heat shield was jettisoned and landing radars were activated to determine a solution. At Mach 0.25, the lander was ejected from the backshell, initiated a debris avoidance maneuver, and proceeded to the safest landing site determined by AI-driven hazard cameras. On landing, the lander deployed the ramps, unlocked the rover from the platform, and rolled off. A deployable WiFi camera was set on the surface to image the Zhurong lander and rover as shown in 1.36. Several publications are available detailing the scientific results, which appears to have met the CNSA's objectives.

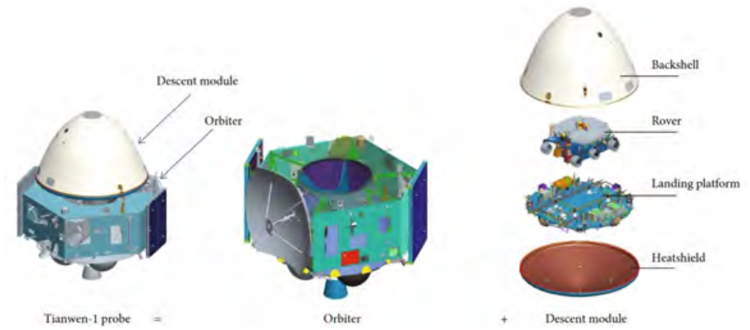


Figure 1.37: The Tianwen 1 spacecraft. Credit: China National Space Administration [78].

As of writing, Zhurong was supposed to have woken up from hibernation in December 2022 after the Martian winter, but the CNSA has been silent on the status of the rover. Researchers are beginning to speculate that the rover was not able to survive the winter [79]. Ingenuity status updates revealed that this winter was difficult; multiple dust storms greatly hindered power production [68]. Regardless,

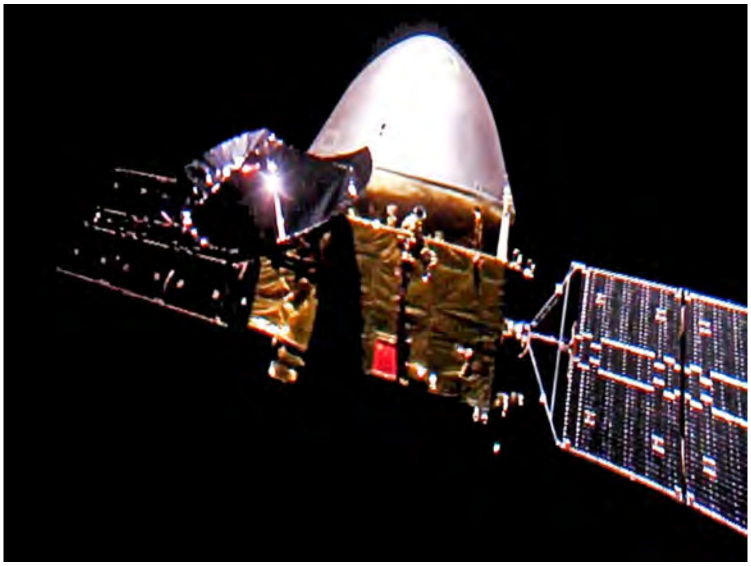


Figure 1.38: The Tianwen 1 cruise stage and orbiter. The upper part shows an oblong, white object - this is the Zhurong EDLS. It is apparent that the EDLS backshell is taller than U.S designs. The reflective object off the left side is the high-gain dish antenna, and the gold rectangle is the cruise stage and orbiter. Credit: China National Space Administration.

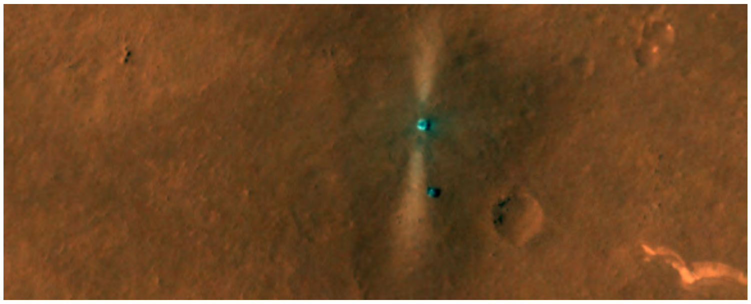


Figure 1.39: The Zhurong rover and lander system on the surface, imaged by NASA JPL's Mars Reconnaissance Orbiter. The blast pattern from the descent engines of the lander are shown. To the right of the lower blast pattern is the Zhurong rover. Credit: NASA/JPL-Caltech/Arizona LPL.

Zhurong greatly exceeeded its' 90-sol lifetime requirement, achieved its scientific objectives, and landed successfully on Mars on the CNSA's first attempt. More missions of much greater ambition are expected from China in the coming years.

1.3 Project Proposal

Work on this project has been ongoing since 2017. From 2017 to late 2019, much of the work was devoted to substantially strengthening computer engineering and electrical instrumentation design skills, specifically for spacecraft systems. This took the form of multiple projects, such as [80], [81], and numerous, yet-to-be published works. Work for SEEDS and Nines nodes were performed between 2019 and 2022, and perfected embedded system design to specific requirements and demonstrated the necessary miniaturization for some scientific instruments and subsystems. Between late 2021 and early 2023, work was performed in conjunction with NASA Ames under a Space Act Agreement with San Jose State - power, guidance, and other selected systems were developed to fly on the TechEdSat platform. The TechEdSat work also significantly focused the development of spacecraft system, scientific, and mission requirements. The remainder of work in the final months is therefore focused on compiling information and performing basic mission analysis.

1.4 Methodology

This publication makes an attempt to be as scope-complete as possible in a given amount of time. First, the problem is defined in programmatic and regulatory terms: in addition to fulfilling *diplomatic* requirements and formulating contracts with the right companies and contacts, engaging in safety and cost reduction measures through government regulation, prior experience, and existing infrastructure is a hard requirement. Science and engineering cannot move forward without such frameworks. A cursory exploration of some of these programmatic and regulatory requirements is outlined in the first main chapter.

The basis of scientific requirements are then outlined in the next chapter.

Through the Scientific Method, scientists pose hypotheses and theories. In the case of Mars, these are followed up with remote and in-situ observations and experiments. Science, and the ability to physically return it, is constrained by regulatory and financial priorities, as well as progress in engineering and instrumentation to make scientific measurements. Engineering requirements informed by the type of science under investigation therefore move the design space forward.

With this design space now fully defined, the engineering design work can then proceed. Several iterations of a mission to Mars are explored, each with increasing levels of maturity, complexity, and budget. The final iteration is assumed to be a NASA *Flagship*-class mission, with a dedicated launch vehicle and budget running into the US$3- to $5-billion (CY2023 dollars) range for the life of the project.

Due to the very long timespan of this project and resulting large quantity of prototypes, systems, and data produced, each chapter chronicles the design process of each component of the mission. The majority of this effort was placed into nanoprobe and lander telecommunication development; this is wrapped up into one chapter with many sections detailing the design process and resulting data. The final revision is a purely theoretical design, but one informed by research, development, and testing with hardware. A lengthy section is finally spent describing the firmware and custom RTOS for the nanoprobes, which form the basis for the rest of the firmware and software for all other assets.

The next chapter focuses on development of the rover. The rover is instrumental in deploying the nanoprobes - the main payload is a robotic arm with an impact driver designed to drive the nanoprobe spike into the ground. A few design iterations were considered, before converging on a standard, 6-wheel, JPL rocker-bogie suspension. Detailed design down to off-the-shelf component level of rover internals was not acheived due to time constraints. However, the nanoprobe design process greatly informed software, firmware, and computer design of the rover, which goes into low-level details.

A design of the lander follows. Not including the lander's primary job of landing the rover and nanoprobes safely, it is the most important component of scientific operations. In IoT terms, it is a "concentrator" for the "edge" devices - in other words, the lander collects and stores the data collected from the rover and probes for later uplink to an orbiting asset. It also functions as a weather station and time standard for the rest of the local network of landed assets. Particular attention was paid to power budgeting: an option to have an RTG-powered lander was assumed for nearly the entire duration of this study. However, near the end, solar power was ultimately selected. This was primarily due to the perceived regulatory and financial difficulty of developing and qualifying a new RTG, despite using the same Department of Energy standard heat source. Regardless, a section of this chapter studies a complete mechanical design of a new, *Discovery*-class RTG.

Although the lander has a component of this system, a separate chapter is warranted for the entry, descent, and landing system (EDLS). The EDLS ensures the scientific instruments make it to the surface of Mars as intended, but levies a severe constraint on virtually all aspects of mission design. The literature review guides the assumption of using a twice-proven capsule shape from the Curiosity and Perseverance missions. A basic aerodynamic study is performed on this shape, with a 2-D hypersonic, full reacting flow simulation for a modest, 4.3 km/sec EI velocity.

The final asset of the mission is the reason for a very slow entry: the orbiter. Much like NASA's Viking landers, and China's recent Tianwen 1 lander and orbiter, this orbiter will carry all four lander sets to a low Mars orbit. While having a considerable mass penalty in terms of fuel required for an entry burn, an orbiter is critical for returning large amounts of data through acting as an orbital relay.

Further, by designing an orbiter to have very large fuel margins, the orbital relay can serve other missions beyond the lifespan of the landers. A significant portion of this chapter describes launch vehicle sizing. At the beginning of the study, an Atlas V 421 was considered. However, as of writing, the Atlas V is slated to go out of production soon. Later on, the mission was then sized for a "New-Space Space Launcher" (NSSL)-class vehicle - specifically a Firefly Aerospace Alpha; but bringing an orbiter with sufficient fuel margins for an entry burn became likely infeasible. Finally, due to its proven track record and competitive costs, a SpaceX Falcon 9 Block 5 or Falcon Heavy, which would give generous margins and the same fairing inner mold line (IML) for both rockets, was selected.

Finally, a high- to mid-level analysis is performed for the entire mission span, from launch to landing. A Monte Carlo-type approach is taken for statistical analysis of mission outcomes, which validates whether the mission would achieve comprehensive success or not. All variables and probabilities of failures are tabulated and scenarios are computed. Given enough time, additional validation of launch to landing maneuvers would be conducted in the video game Kerbal Space Program (KSP), with heavy modifications (mods). The primary mod is Principia, based on Quinlan and Tremaine's 12^{th}-order, n-body planetary integrator [82] among others, written and maintained in C++ by Pascal Leroy and Robin Leroy [83]. This drives another mod called RSS (Real Solar System), which converts the default solar system into the real-world solar system simulating 30 of the solar system's largest bodies, including asteroids and moons. Solar system initial conditions are provided by the JPL HORIZONS service [84]. Details such as planetary oblateness J_2 are implemented in Leroy's methods for most celestial bodies.

2. Problem Scope

This chapter discusses regulatory considerations and requirements for Wavefront. Requirements, in the scientific and engineering sense, are *quantitative.* While *regulatory* requirements are easier to put a number to, *political* requirements are less intuitive. Statements that specify who or what to work for translate into a single quantitative term: a budget.

2.1 A Word on Management and Expectations

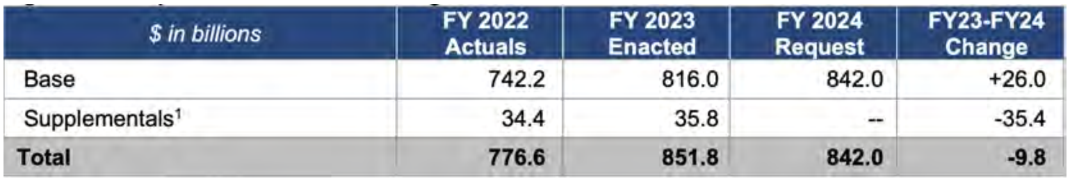
For a project of the *Flagship*-class, rigorous systems engineering, project management, and a very good working relationship between both disciplines is required. Cost overruns typically occur because of poor management, poorly defined requirements, and/or poor understanding of the problem [49] [60] [85] [86]. To set the stage, a top-down, bottom-up approach is the first hard requirement. In other words, the applying the "V-model" of systems engineering is the first management step. Before reaching the Authority to Proceed (ATP) stage, program managers and talent acquisition departments must evaluate the talent pool available - especially through university outreach and engagement - to strike a balance between talent margin and overhiring. The project managers and principal investigators can then make an informed decision on scoping the most expensive portion of their budget: the people.

Successfully appeasing stakeholders by delivering the intended result (and shareholders for publicly traded contractors on large capital expenditure projects) plays into managements' forward thinking. The political game played is one of underpromising and overdelivering. Upper management, PI's, and PM's must develop program management plans that fulfill requirements for maintaining financial and capital solvency, as well as margins on deliverables. For example, the twin Mars Exploration Rovers (MER) Spirit and Opportunity were designed to last only 90 days, yet Opportunity lasted over 5,000. While an extreme and arguably an unintended margin, NASA and JPL were able to leverage part of this in their public relations to successfully recover from the reputational damage incurred by the loss of three Mars missions and Space Shuttle *Columbia* with its' seven astronauts. Therefore, management of expectations should be baked into some of the highest level requirements, even when scientists and engineers might be able to do more. Stretch goals can be achieved, as long as they are at no additional cost or risk to the main objectives and deliverables to the mission. This is called the "do-no-harm" clause, per NASA Policy Directive 7120.8 [87].

A mission to Mars not only has to account for lessons learned from previous missions, but take into consideration certain *diplomatic* and *scientific* objectives and requirements. More specifically, how would a Mars mission serve the general public in the short- and long-term? Interplanetary space missions are taxpayer-funded because for-profit ventures are still not financially viable, and returns on investment occur in the span of decades [88]. This brief chapter constructs the scientific objectives, derived from government policy, so that *Wavefront* and *Aerith* would make the best use of taxpayer dollars and any other corporate sources of funding.

2.2 Political Requirements

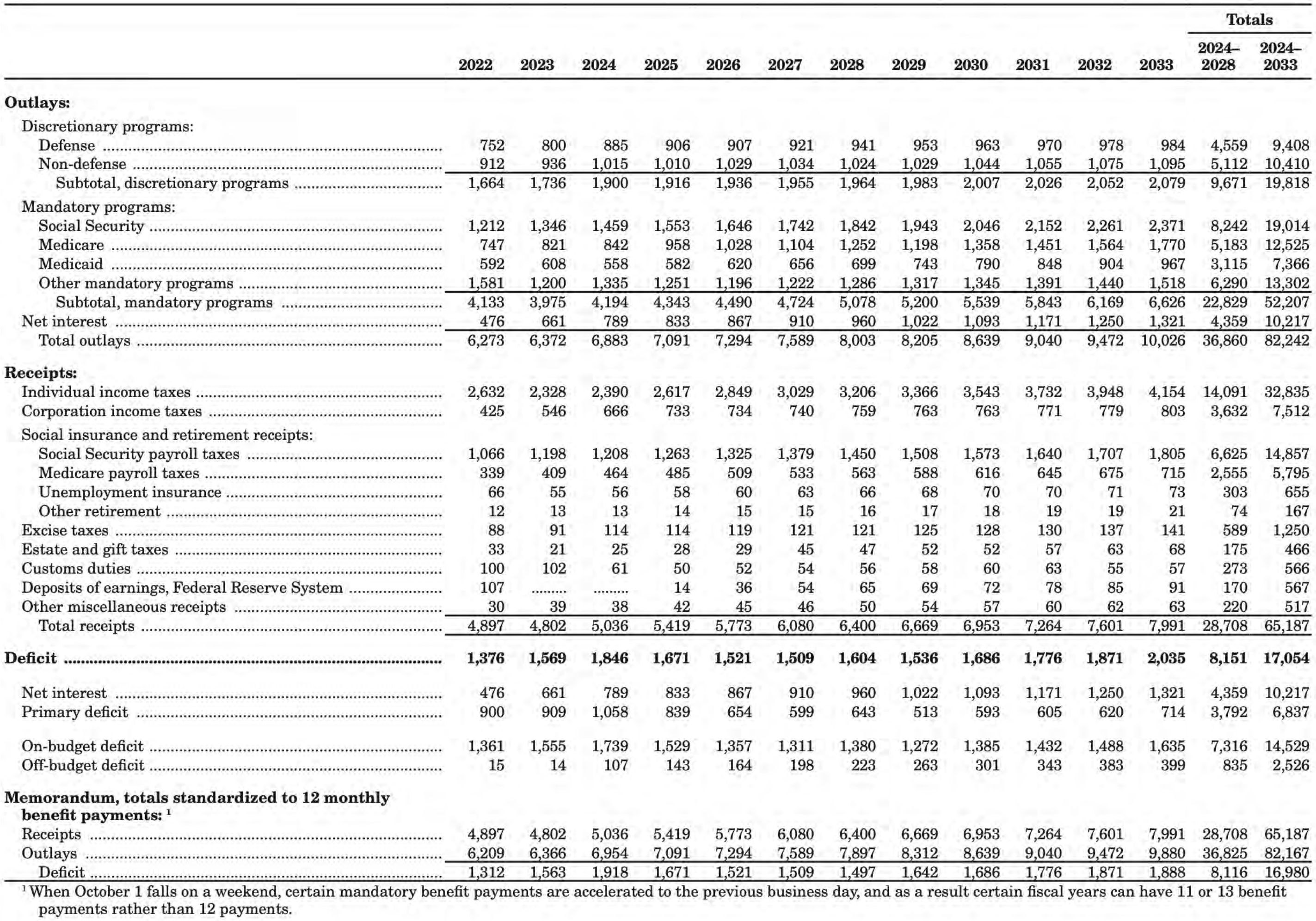
Today, in America, policy is typically controlled by the governing political party. In Europe and European Union-aligned states, coalition governments composed of several political parties form the governing party. And in authoritarian states, a single party, if not a single individual, drives all decisions. In 2023, for any of these types of described governments, opposition to the majority is not usually taken into consideration.



$ in billions	FY 2022 Actuals	FY 2023 Enacted	FY 2024 Request	FY23-FY24 Change
Base	742.2	816.0	842.0	+26.0
Supplementals[1]	34.4	35.8	--	-35.4
Total	**776.6**	**851.8**	**842.0**	**-9.8**

Figure 2.1: U.S. Department of Defense Budget. While a decrease from the previous year, the Fiscal Year 2024 requested budget represents a very large financial commitment for any nation.

However, one thing is certain: aerospace and defense policy, especially in America, enjoys broad support amongst any political affiliation, especially during this new era of global conflict. In Europe and Asia, similar alignments are occurring, regardless of type of government. This is due to a number of factors: in America, this describes the "military-industrial complex", as indirectly shown in the FY 2024 Department of Defense budget [89]. The overall FY2024 budget, shown in 2.1, is U.S.$842.0 billion. The majority of this money, directly or indirectly, goes to defense contractors such as Lockheed Martin, General Dynamics, Northrop Grumman, Boeing, Raytheon, and Newport News Shipbuilding. Unsurprisingly, many of these contractors also play a role in civil aerospace, such as the building of airliners and spacecraft systems. It is this connection that renders space and defense inseperable in the highest levels of politics regardless of whether the state is a democracy or dictatorship. The U.S. President's budget for FY 2024 further shows this connection - in Figure 2.2, FY2024's discretionary spending under "Outlays" (expenses) has a line item specifically for defense. This amounts to 12.8 percent of total federal expenses [90]. Therefore, the highest level requirements must align with the economic, and by association, the political and executive, objectives of the state.



	2022	2023	2024	2025	2026	2027	2028	2029	2030	2031	2032	2033	Totals	
													2024–2028	**2024–2033**
Outlays:														
Discretionary programs:														
Defense	752	800	885	906	907	921	941	953	963	970	978	984	4,559	9,408
Non-defense	912	936	1,015	1,010	1,029	1,034	1,024	1,029	1,044	1,055	1,075	1,095	5,112	10,410
Subtotal, discretionary programs	1,664	1,736	1,900	1,916	1,936	1,955	1,964	1,983	2,007	2,026	2,052	2,079	9,671	19,818
Mandatory programs:														
Social Security	1,212	1,346	1,459	1,553	1,646	1,742	1,842	1,943	2,046	2,152	2,261	2,371	8,242	19,014
Medicare	747	821	842	958	1,028	1,104	1,252	1,198	1,358	1,451	1,564	1,770	5,183	12,525
Medicaid	592	608	558	582	620	656	699	743	790	848	904	967	3,115	7,366
Other mandatory programs	1,581	1,200	1,335	1,251	1,196	1,222	1,286	1,317	1,345	1,391	1,440	1,518	6,290	13,302
Subtotal, mandatory programs	4,133	3,975	4,194	4,343	4,490	4,724	5,078	5,200	5,539	5,843	6,169	6,626	22,829	52,207
Net interest	476	661	789	833	867	910	960	1,022	1,093	1,171	1,250	1,321	4,359	10,217
Total outlays	6,273	6,372	6,883	7,091	7,294	7,589	8,003	8,205	8,639	9,040	9,472	10,026	36,860	82,242
Receipts:														
Individual income taxes	2,632	2,328	2,390	2,617	2,849	3,029	3,206	3,366	3,543	3,732	3,948	4,154	14,091	32,835
Corporation income taxes	425	546	666	733	734	740	759	763	763	771	779	803	3,632	7,512
Social insurance and retirement receipts:														
Social Security payroll taxes	1,066	1,198	1,208	1,263	1,325	1,379	1,450	1,508	1,573	1,640	1,707	1,805	6,625	14,857
Medicare payroll taxes	339	409	464	485	509	533	563	588	616	645	675	715	2,555	5,795
Unemployment insurance	66	55	56	58	60	63	66	68	70	70	71	73	303	655
Other retirement	12	13	13	14	15	15	16	17	18	19	19	21	74	167
Excise taxes	88	91	114	114	119	121	121	125	128	130	137	141	589	1,250
Estate and gift taxes	33	21	25	28	29	45	47	52	52	57	63	68	175	466
Customs duties	100	102	61	50	52	54	56	58	60	63	55	57	273	566
Deposits of earnings, Federal Reserve System	107	....	....	14	36	54	65	69	72	78	85	91	170	567
Other miscellaneous receipts	30	39	38	42	45	46	50	54	57	60	62	63	220	517
Total receipts	4,897	4,802	5,036	5,419	5,773	6,080	6,400	6,669	6,953	7,264	7,601	7,991	28,708	65,187
Deficit	**1,376**	**1,569**	**1,846**	**1,671**	**1,521**	**1,509**	**1,604**	**1,536**	**1,686**	**1,776**	**1,871**	**2,035**	**8,151**	**17,054**
Net interest	476	661	789	833	867	910	960	1,022	1,093	1,171	1,250	1,321	4,359	10,217
Primary deficit	900	909	1,058	839	654	599	643	513	593	605	620	714	3,792	6,837
On-budget deficit	1,361	1,555	1,739	1,529	1,357	1,311	1,380	1,272	1,385	1,432	1,488	1,635	7,316	14,529
Off-budget deficit	15	14	107	143	164	198	223	263	301	343	383	399	835	2,526
Memorandum, totals standardized to 12 monthly benefit payments: [1]														
Receipts	4,897	4,802	5,036	5,419	5,773	6,080	6,400	6,669	6,953	7,264	7,601	7,991	28,708	65,187
Outlays	6,209	6,366	6,954	7,091	7,294	7,589	7,897	8,312	8,639	9,040	9,472	9,880	36,825	82,167
Deficit	1,312	1,563	1,918	1,671	1,521	1,509	1,497	1,642	1,686	1,776	1,871	1,888	8,116	16,980

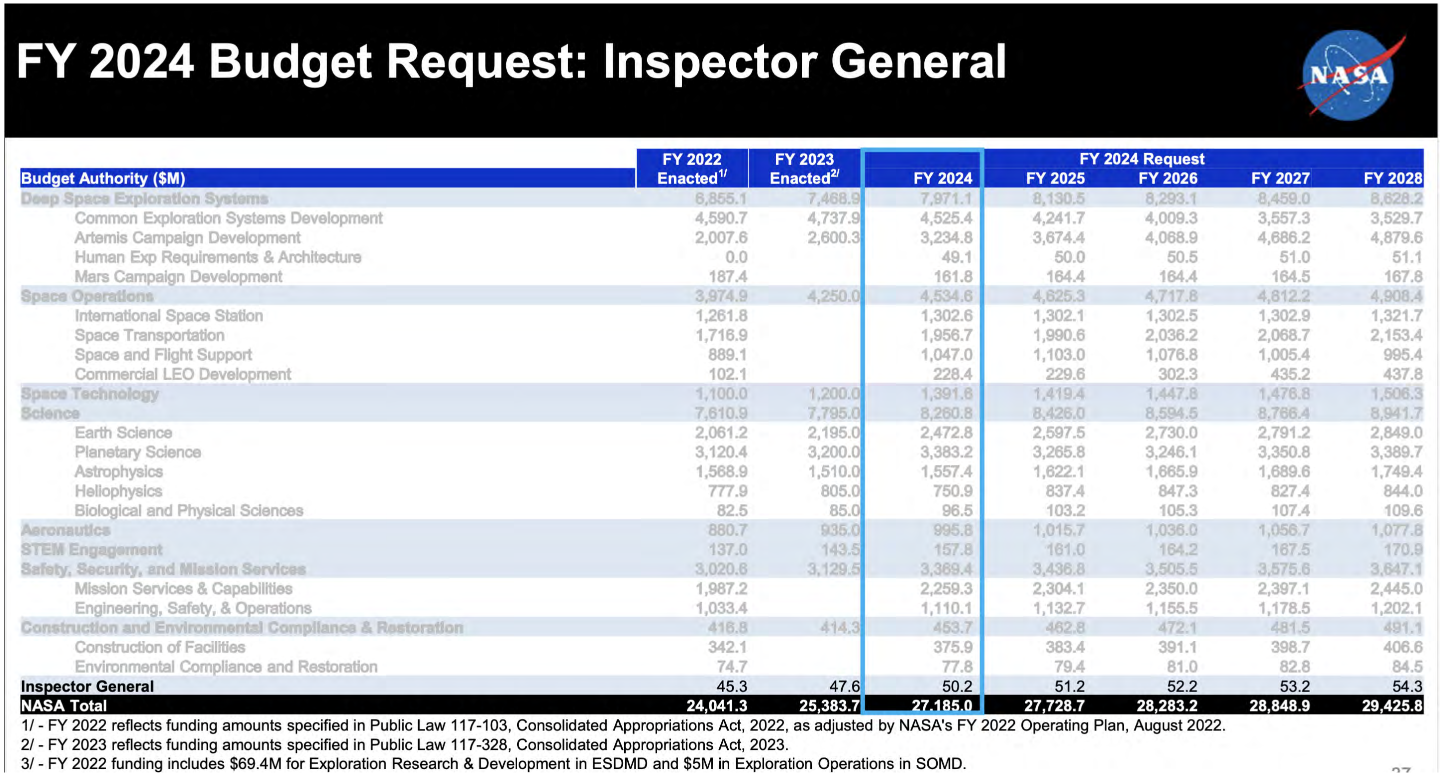
[1] When October 1 falls on a weekend, certain mandatory benefit payments are accelerated to the previous business day, and as a result certain fiscal years can have 11 or 13 benefit payments rather than 12 payments.

Figure 2.2: U.S. Federal Budget, otherwise known as the Presidential Budget [90].

While planetary sciences would not ideally have anything to do with the defense industry, many of the contractors listed above have leading roles in the design and fabrication of interplanetary spacecraft. For example, per the literature review, nearly all U.S. Mars missions after Pathfinder list Lockheed Martin as a contractor. Ingenuity's deployment mechanism was designed and built by Lockheed Martin Space Systems [91]. Space exploration and defense have been inextricably tied for decades because space exploration demands the most cutting-edge technology to function with high reliability and high fault tolerance, required of any defense hardware.

The NASA requested FY2024 budget of US$27,185 million is 7.09% larger than the enacted FY2023 budget, shown in Figure 2.3 [92]. When studying the breakdown, about US$12 billion is spent on human spaceflight development. In terms of the defense budget studied before, and the fact that during the Apollo era, approximately 4% of total federal spending, or roughly US$0.25 trillion in 2023 dollars was spent in FY1964 [93], it is a drop in the bucket. Yet, R&D in support of scientific exploration, human or robotic, has been forced to become more economically lean due to many domestic political and geopolitical circumstances in the past four decades [94] [95].

Understandably, the financial and capital costs of R&D for space and defense technology is high; in government terms, there must be a very good reason for billions of *taxpayer* dollars to be spent on any R&D for abstract or unseen investments. In fields other than defense, startups dedicated solely to commercial space usually must bear R&D costs. The recent Chapter 11 debt restructuring of Virgin Orbit National Systems (VO), a startup dedicated to launching small spacecraft to any orbital inclination anywhere in the world, is a contemporary example. While VO obtained venture capital (VC) funding from lenders like Silicon Valley Bank (SVB) and had four successful flights, the failure of VO's 5th launch and the lack of liquidity in SVB shows how thin the financial margins are. Despite a relatively high success rate and



FY 2024 Budget Request: Inspector General

Budget Authority ($M)	FY 2022 Enacted[1/]	FY 2023 Enacted[2/]	FY 2024 Request FY 2024	FY 2025	FY 2026	FY 2027	FY 2028
Deep Space Exploration Systems	6,855.1	7,468.9	7,971.1	8,130.5	8,293.1	8,459.0	8,628.2
Common Exploration Systems Development	4,590.7	4,737.9	4,525.4	4,241.7	4,009.3	3,557.3	3,529.7
Artemis Campaign Development	2,007.6	2,600.3	3,234.8	3,674.4	4,068.9	4,686.2	4,879.6
Human Exp Requirements & Architecture	0.0		49.1	50.0	50.5	51.0	51.1
Mars Campaign Development	187.4		161.8	164.4	164.4	164.5	167.8
Space Operations	3,974.9	4,250.0	4,534.6	4,625.3	4,717.8	4,812.2	4,908.4
International Space Station	1,261.8		1,302.6	1,302.1	1,302.5	1,302.9	1,321.7
Space Transportation	1,716.9		1,956.7	1,990.6	2,036.2	2,068.7	2,153.4
Space and Flight Support	889.1		1,047.0	1,103.0	1,076.8	1,005.4	995.4
Commercial LEO Development	102.1		228.4	229.6	302.3	435.2	437.8
Space Technology	1,100.0	1,200.0	1,391.6	1,419.4	1,447.8	1,476.8	1,506.3
Science	7,610.9	7,795.0	8,260.8	8,426.0	8,594.5	8,766.4	8,941.7
Earth Science	2,061.2	2,195.0	2,472.8	2,597.5	2,730.0	2,791.2	2,849.0
Planetary Science	3,120.4	3,200.0	3,383.2	3,265.8	3,246.1	3,350.8	3,389.7
Astrophysics	1,568.9	1,510.0	1,557.4	1,622.1	1,665.9	1,689.6	1,749.4
Heliophysics	777.9	805.0	750.9	837.4	847.3	827.4	844.0
Biological and Physical Sciences	82.5	85.0	96.5	103.2	105.3	107.4	109.6
Aeronautics	880.7	935.0	995.8	1,015.7	1,036.0	1,056.7	1,077.8
STEM Engagement	137.0	143.5	157.8	161.0	164.2	167.5	170.9
Safety, Security, and Mission Services	3,020.6	3,129.5	3,369.4	3,436.8	3,505.5	3,575.6	3,647.1
Mission Services & Capabilities	1,987.2		2,259.3	2,304.1	2,350.0	2,397.1	2,445.0
Engineering, Safety, & Operations	1,033.4		1,110.1	1,132.7	1,155.5	1,178.5	1,202.1
Construction and Environmental Compliance & Restoration	416.8	414.3	453.7	462.8	472.1	481.5	491.1
Construction of Facilities	342.1		375.9	383.4	391.1	398.7	406.6
Environmental Compliance and Restoration	74.7		77.8	79.4	81.0	82.8	84.5
Inspector General	45.3	47.6	50.2	51.2	52.2	53.2	54.3
NASA Total	**24,041.3**	**25,383.7**	**27,185.0**	**27,728.7**	**28,283.2**	**28,848.9**	**29,425.8**

1/ - FY 2022 reflects funding amounts specified in Public Law 117-103, Consolidated Appropriations Act, 2022, as adjusted by NASA's FY 2022 Operating Plan, August 2022.

2/ - FY 2023 reflects funding amounts specified in Public Law 117-328, Consolidated Appropriations Act, 2023.

3/ - FY 2022 funding includes $69.4M for Exploration Research & Development in ESDMD and $5M in Exploration Operations in SOMD.

Figure 2.3: NASA's FY2024 proposed budget, with major program breakdowns [92].

accurate deliveries of customer spacecraft to their intended orbits, VO still had to file for Chapter 11 bankruptcy. The case of VO illustrates the nascent difficulty and high risk of operating in commercial space without the same government funding that R&D in defense gets.

All things considered, planetary exploration (even at the Moon), cannot be performed without funding from a state entity. As of writing, the human race has neither poured enough capital investment nor visited enough celestial bodies to make any aspect of planetary exploration or exploitation commercially viable. Therefore, the first requirement is to leverage any and all existing state (U.S. federal) frameworks.

To fit Wavefront into the purview of a proposed American NASA flagship-class mission, Wavefront must, at minimum, follow a management framework that landed Curiosity and Perseverance to the surface of Mars. In other words, the mission must follow legacy missions of similar scale and scope. Quantitatively, this means the budget shall not exceed US$3 billion, with 50% margin. These are commonly called fixed-price contracts; any cost overrun shall be borne by the contractor. There are several nuances missing, such as "too-big-to-fail" missions that fall under U.S. Congressional budget line items; for the sake of containing this thesis to an engineering and scientific exercise, these are omitted.

However, such a large budget presents a chicken-before-the-egg problem: the return on investment (ROI), or the "so-what" factor. Without a justification to ROI, the first requirement cannot possibly exist. To the American taxpayer, is it *really* worth it to send scientific instruments to the surface of Mars to learn more about geophysical past and the current geological and meteorological conditions? As a selfish individual, how would the average individual directly benefit from this? This plays into the highest levels of Washington politics, where congresspeople would (rightfully) ask, "Are my consitituents of *X* district affected by this project? How will it affect my unemployment numbers, tax revenues to the cities and counties that fall under my district, and how will that affect my chances of getting reelected? Am I appeasing the business interests in my district?" Most engineers and some scientists would never need to answer to these questions. But from a programmatic standpoint, the message starts with the stated goal of NASA - to advance the knowledge of humanity for all. The justification must start from all the way at the top of governance. As covered in Chapter 1.1, Apollo technologies, knowledge, and methods were widely distributed throughout humanity. What is learned from a Mars mission dedicated to geological and meteorological sensing sets the stage for astronauts to walk and live on Mars, safely. It allows for the potential expansion of commerce to beyond Earth's orbit - cautious exploration and characterization of other worlds must occur.

All of this is to say that the mission shall comply and align fully with NASA's vision, because a failure to do so will not permit any money to flow under such limited Agency budgets. Table 2.1 summarizes the requirements in this section.

2.3 Peer-Reviewed Science Requirements

Higher-level requirements must be evaluated for compliance continuously, but foundational requirements such as scientific definition must be *rigidly defined* at the outset. Science and engineering R&D can only be as focused as the agency, political, and scientific requirements are. Wavefront intends to tackle many scientific interests, including geophysics, meteorological models of Mars, and safety of astronauts for future visits to Mars. With these interests in mind, the principal investigator and program manager must now tailor those broad scientific endeavors to specific science

Table 2.1: Political Alignment Requirements

Requirement	**Rationale**
0-0. **Shall** follow the 7000- and 8000-series NASA directives	The 7000-series represent program formulation and related systems engineering practices - in particular, the "V-model" of systems engineering; and the 8000-series represent program management [96] [97].
0-1. **Must** follow a management framework that follows legacy missions of similar scale and scope	In particular, the missions **should** be current, within the past 10 to 20 years. While following NASA program directives are required and provide all the necessary frameworks, following a recent precedent gives politicians, taxpayers, and engaged contractors confidence that the project will proceed efficiently, and that R&D costs are minimized.
0-2. **Shall** evaluate compliance and alignment with NASA's vision.	See Section 1.1, Motivation for further analysis of rationale. Compliance failure will result in missed funding. This is technically a "Level-1" requirement, peer-reviewed scientific definitions are dependent on (flow down to) this requirement.

needed from the scientific community.

The Mars Exploration Program Analysis Working Group (MEPAG) is a science definition committee representing the views and research of many scientists. They work to sharpen research focuses and prioritize Mars missions [98]. While a committee based at JPL, MEPAG takes inputs from scientists around the world, peer-reviewed research, and from the Planetary Science Decadal Survey, itself commissioned by the U.S. National Academy of Sciences, a congressionally chartered organization providing science policy advice [99]. To be in line with political and policy requirements, the latest MEPAG committee document [100] will provide the legal basis for all other all scientific and engineering requirements here forward.

The MEPAG 2020 Goals and Objectives document specifies four major categories of research on Mars:

1. To determine if Mars ever supported, or still supports, life,
2. To understand the processes and history of climate on Mars,
3. To understand the origin and evolution of Mars as a geological system, and
4. To prepare for human exploration.

Per the literature review, Goal 1 requires highly complex instruments that take years to develop and rigorously validate. Viking 1/2 are an example of this, the inclusion of life detection instruments required extreme planetary protection requirements such as whole-spacecraft bakeout to nearly electrically intolerable temperatures for hours [93] and extensive testing and validation, yet the results from their Labeled Release experiments are still contentious to this day [44]. Due to this complexity and large size of these instruments, Wavefront cannot be scoped into life science requirements.

Goals 2 and 3 respectively relate to meteorology and geophysics, which are the most desirable types of science Wavefront is scoped for. Table 2.2 lists the three major objectives for attaining Goal II.

2.3.1 Goal II

Completing Goal II requires observations, computational models, and laboratory experiments. Over the past several decades, the U.S., Soviet Union, the European Union, India, China, and Emirates have contributed both remote and in-situ observations. They have fed into global climatological models to attempt to fill in the gaps that recreate the data points observed by probes. One particular

Table 2.2: MEPAG Goal II Objectives

Objective
Objective A: Characterize the state and controlling processes of the present-day climate of Mars under the current orbital configuration
Objective B: Characterize the history and controlling processes of Mars' climate in the recent past, under different orbital configurations.
Objective C: Characterize Mars' ancient climate and underlying processes.

investigation of high priority is how dust lifting occurs on the surface of Mars, and why and how turbulence forms. As many probes and staging for human exploration will use solar power, dust accumulation is of concern. The serendipitous occurrence of dust devils in Spirit and Opportunity's landing zones in part allowed them to last for many years beyond their intended lifespans; the same dust devils did not occur in InSight's landing zone, despite the similar geology. Recording dust loading and turbulence profiles requires purpose-designed weather stations. Specifically, hot-wire anemometers and saltation (dust classification) sensors fit the scientific requirements for Wavefront.

Key to Goal II is the wide distribution of sensors. Remote sensing data, such as from MAVEN and ExoMars TGO, can characterize the state of volatile gases escaping Mars, such as hydrogen from water, but pinning down the exact methods of sublimation or generation of other gases like methane requires measurements to be made at the source. Wide but sparse networks were proposed in the 1990's with MESUR, but never took off for a multitude of reasons examined in the literature review.

Measuring gas quantities is difficult with small instruments - many of the most sensitive spectrometers designed for analyzing evolved gas or gas concentrations are very large and consume large amounts of power. Methane is regarded as a biosignature, and the highly transient detections of it by Curiosity has spurred extreme interest. The Trace Gas Orbiter, provided by ESA, failed to detect any methane as of their 2019 datasets [101]. These discrepancies have proved to be extremely scientifically controversial, as some [102] dispute the accuracy of Curiosity's spectrometer, while others point out the need for more in-situ point measurements. Others [101] [103] state that assuming the instruments are accurate, methane might be so transient that there are still yet unknown geophysical, geochemical, or potentially biological sinks that can quickly disperse the methane. While passed off as out-of-scope, the inclusion of gas chromatographs and spectrometers for Goal II may address Goal I as a stretch goal given enough time and effort to characterize and calibrate the instruments.

In summary, Goal II will require weather stations that can sample at kilohertz rates to resolve turbulence, saltation sensors that can determine airborne, suspended sand grain inertia, gas chromatographs or nondispersive optical sensing to determine if certain trace gases are present at the measurement source, and spectrometers to supplement bulk gas sensing.

Table 2.3: MEPAG Goal III Objectives

Objective
Objective A: Document the geologic record preserved in the **crust** and investigate the processes that have created and modified that record.
Objective B: Determine the structure, composition, and dynamics of the **interior** and how it has evolved.
Objective C: Determine origin and geologic history of **Mars' moons** and the implications for the evolution of Mars.

2.3.2 Goal III

Tackling Goal III can similarly be accomplished with orbital and in-situ assets. The placement of in-situ assets is best informed through orbital imaging, but require site revisits after a set period of time. Geological (areological) changes take place at a much slower rate than on Earth, but are observable on human timescales. That being said, Goal III objectives would benefit from distributed probes that have a very long period of collection. This, of course, is limited by engineering due to Martian factors, such as unpredictable dust storms, lack of "cleaning" events like dust devils, unanticipated terrain and subsurface, the extreme cold, and lower solar insolation.

These engineering challenges can best be approached with highly distributed networks of in-situ sensors, given a design tolerant to dust storms and/or loss of power. This architecture also has the capability of determining the extent of water, sulfur, and carbon in the subsoil, the highest priority scientific objectives in Goal III. One approach employed in geophysical and civil engineering site analysis is electical resistivity tomography (ERT). Stakes are driven into the ground and pulses from a base station are applied into the ground. By detecting a very slight change in voltage relative to the base station's ground potential, the presence of large areas of moisture, clays, or softer ground can be inferred [104] [105]. Porting this to Mars would require wireless ERT receiver nodes to perform tomography over a scientifically useful volume below ground, with a method of driving conductive stakes into the ground. The transmitter requires two stakes with a common frontend; ideally a larger stationary lander with adequate power. By using similar but related methods to medical electrical impedance tomography [106] on much longer time scales, geological changes in underground and near-surface liquids and brines can be monitored in multiple locations. This is of particular interest; scientists on the MRO mission have observed so-called "recurring slope lineae", which are theorized to be liquid brine flow. ERT would be able to characterize this in real-time and pinpoint exactly when and under what conditions they occur, especially if the source is below ground.

ERT would not be the only way to determine subsurface changes in geology. The RIMFAX radar on Perseverance and Zhurong's radar are both designed to analyze subsurface structure by listening for reflected chirps from the radar. Per the literature review, the radars had great success in testing. As of writing though, findings and effectiveness of the radars have yet to be published. Including a radar on Wavefront would be a substantial challenge, however. Both radars are broadband, from UHF to S-band, which necessitate clever RF frontend and antenna design to fit such a unit in a very small lander package.

In summary, Goal III can be tackled with below-ground electrical resistance tomography, radars if able to be accommodated on a small lander, and cameras observing changes to the surroundings. All methods are contingent on a spacecraft design and architecture that is resiliant and can withstand severe dust storms and Martian winters.

2.3.3 Goal IV

While Wavefront is not explicitly scoped for human exploration, the mission has a wealth of capabilities to advance Goal IV. Regardless of how long astronauts stay, radiation dosimetry is of utmost importance. Several missions to Mars, including the Curiosity rover, carry or carried radiation dosage sensors. Figure 2.4 shows a comparison of human dose equivalent for different scenarios. Of greatest interest is the expected dose for a 180-day transit to Mars and an extended stay on Mars. A person on a full, round-trip flight to Mars would accumulate over an order of magnitude more radiation than the average dose for an astronaut on the International

Table 2.4: MEPAG Goal IV Objectives

Objective
Objective A: Obtain knowledge of Mars sufficient to design and implement human landing at the designated human landing site with acceptable cost, risk and performance.
Objective B: Obtain knowledge of Mars sufficient to design and implement human surface exploration and E-VA on Mars with acceptable cost, risk and performance.
Objective C: Obtain knowledge of Mars sufficient to design and implement In Situ Resource Utilization of atmosphere and/or water on Mars with acceptable cost, risk and performance.
Objective D: Obtain knowledge of Mars sufficient to design and implement biological contamination and planetary protection protocols to enable human exploration of Mars with acceptable cost, risk and performance.
Objective E: Obtain knowledge of Mars sufficient to design and implement a human mission to the surface of either Phobos or Deimos with acceptable cost, risk, and performance.

Space Station. Preliminary results from Curiosity's Radiation Assessment Detector (RAD) show a human dosage rate of $0.21 \pm 0.04 mGy/d$ [107], which back up the estimates in Figure 2.4. These results are applicable for all Goal IV objectives. For future missions, especially of a distributed format like Wavefront, gathering more radiation data during the cruise stage and during in-situ operations, especially from unexplored locations on Mars, would further inform human safety requirements for operations. Data from rover missions suggest localized magnetic fields that are much stronger than modeled or expected - as the magnetosphere does for Earth, localized high-intensity magnetic domains on Mars would provide additional shielding from radiation.

Determining specific regions that provide more shielding is signficantly

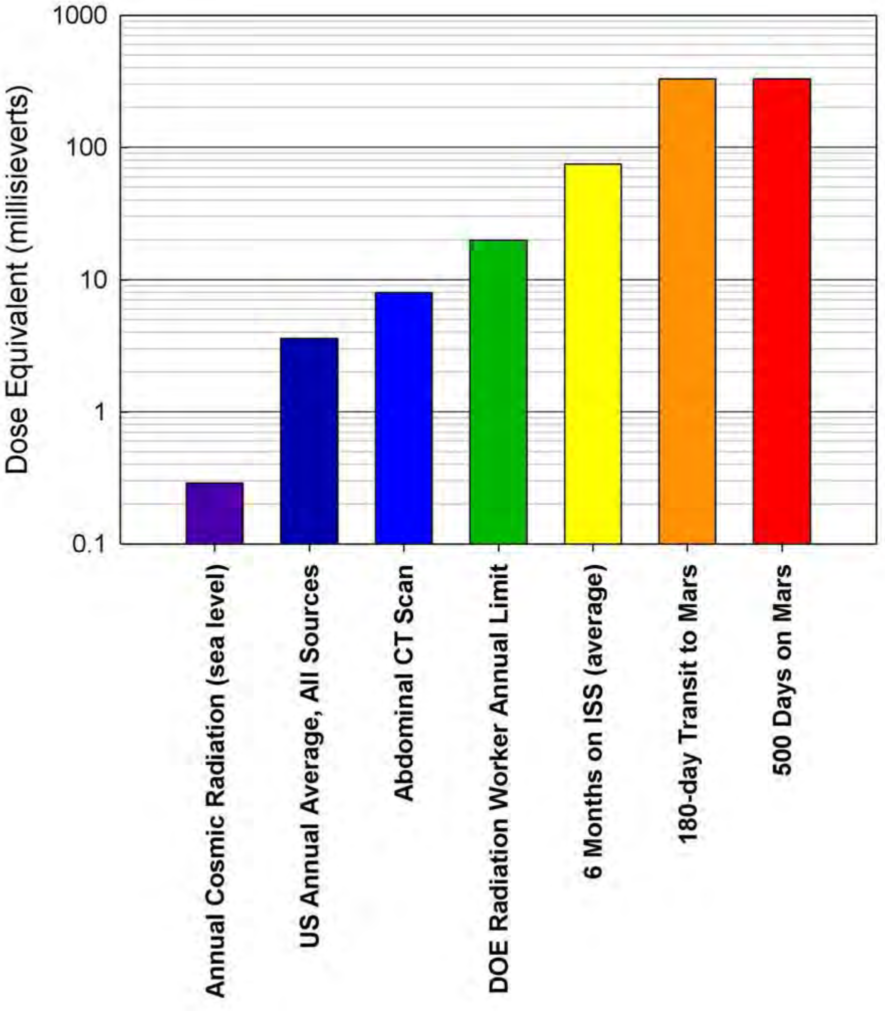


Figure 2.4: Equivalent human dosages for seven different scenarios, logarithmic dosing scale. Of particular interest is the 180-day transit to Mars and the 500 days on Mars. Credit: NASA/JPL/Caltech [107].

augmented by more in-situ measurements. Another objective achieved by virtue of visiting more locations is IV.A. Experimental and in-situ atmospheric entry, descent, and landing data is obtained through a distributed mission, as long as all probes are instrumented and are able to transmit some of the atmospheric data, regardless of landing success. The MEPAG report states that retropropulsion techniques alone might be able to land human-rated missions to the surface, but that there are large error bars due to many unknowns during different seasons, different altitudes, dust loading, among other factors.

Mentioned previously in the literature review is the point of "trafficability", stated explicitly as an objective in the MEPAG report as a sub-objective. In general, higher-value scientific regions are very risky in terms of landing - obstacles such as boulders and quicksand-like regolith are examples. Missions that profile the regolith through physical probing would be desirable, as demonstrated in the InSight HP^3 instrument - which demonstrated the regolith had unexpected features due to its inability to self-hammer into the surface.

In summary, many technologies and scientific investigations relating to human exploration can be tested, much of which is not covered here; but in particular, radiation dosimetry and determining traversability and trafficability is desirable.

2.4 Regulatory Requirements

Last, but certainly not least, are the regulatory requirements for such a large-scale project. The NASA Policy Directives (NPD) are a series of documents that provide a legal framework for governance models, project management, engineering, and scientific guidance [108]. However, as stated in the directives, they are not legally binding unless stipulated in a contract. If Wavefront were to be a formalized mission, these NPD's would be legally binding. There are 10 volumes of NPD's; the most relevant volumes are the 7000 and 8000 series, which respectively cover program formulation and program management.

2.4.1 Program Formulation

The first document that must be followed, especially for a project of this size and scope, is NPD 7120.5F - NASA Space Flight Program and Project Management Requirements. Between recently released Version F and the prior version, projects with lifecycle costs (LCC's) of greater than US$1 billion are required to perform a "Joint Cost and Schedule Confidence Level" (JCL) analysis, especially if there is an open-ended closeout date (mission of potentially indefinite length due to use of nuclear power systems, for example).

One of the first and most influential determining factors in the usage of 7120.5F and other NPDs are the flight of nuclear material. "Significant" amounts of nuclear material is determined by NPD 8715.26, which would ultimately require sign-off by the NASA Administrator and the President of the US. Up until late in the design phase of the mission, flight of up to 13 nuclear power sources were considered and designed for, due to the length of time required for in-situ weather and geologic measurements. Due to in part the extremely stringent regulations required, heightened risk of accidental or intentional nuclear proliferation, and the overwhelming engineering constraints of carrying radioisotope thermoelectric generators (RTGs) on very small probes, the nuclear option was completely ignored by the end of the initial design phase. Substituting energy sources for solar power would still require a large, but more attainable and realistic engineering effort.

As an LCC mission of US$3 to 5 billion, the program would be considered a "Category 1" mission, and would therefore require approval and concurrence of the associate administrator, the NASA chief engineer, and NASA center directors. Category 1 missions are expected to have continuous management, systems and program management audits, and robust quality assurance.

For the rest of the document, typical systems engineering practices and work breakdown shall be implemented. Figure (fig) shows a NASA standard Level 2 work breakdown structure (WBS) for projects - an example of how work flows down into the rest of the program elements. For missions with Flagship-level budgets, substantial capital investment and procurement is required - the testing, data storage and archival, and ground support equipment to name a few would likely exceed the total cost of the spacecraft. The design of Wavefront benefits from serial manufacturing: 12 landers, 12 rovers, and 120 nanoprobes would be developed for flight. Once the design is finalized, the unit cost of each probe is expected to be lower than if they were independently developed for different missions. The capital used to produce, test, and validate operation of each unit would be used for every probe. In this situation, the ground and mission support equipment costs would have the potential to be higher. The orbital relay element, Aerith, would relay tens of terabytes of information per Earth year, as long as all assets continue to function. For scientists inside and outside of the agency, datacenters and supercomputing facilities would be required for storage and processing of this data.

2.5 Work and Cost Breakdown

On designing a full systems engineering process for a flagship mission, a rigorous work breakdown is needed before any work starts. All NASA flagship missions are usually single, large vehicles, which sometimes carry a tech demonstration or secondary payload, such as that of Mars 2020's Ingenuity helicopter, or Cassini's Huygens Titan probe. Wavefront would be a complete paradigm shift, as multiple, serially produced probes would form the primary payload. While critical to the mission in terms of returning scientific data and carrying the probes to the destination, the Aerith orbiter would be the secondary payload.

2.5.1 Projected Program Lifecycle

Once the program has reached the Authority to Proceed phase, initial R&D work begins for the design process to advance the program to the Preliminary Design Review phase. Initial drawings and designs would be refined by larger teams, and more detailed interface requirements would be developed. Further technological feasibility is studied as well, and any shortcomings are either addressed if enough time and budget is avaialble from the financial margins pool, or downscoped if it is determined an engineering solution cannot be developed for the specific challenge. R&D also includes development of the manufacturing, testing, and validation process. Especially for the Wavefront program, if they are not streamlined early in the program lifecycle, cost overruns can quickly mount and grow the program out of control to cancellation. This is why it is important to leverage appropriate organizations with plenty of experience in spacecraft development.

As previously mentioned, in the systems engineering and project management aspect, the ATP is the general starting point for a project as a major program review. Figure 2.5 outlines this schedule, and Table 2.5 lists out the acronyms for KDPs and program reviews. In particular, the figure will refer to the "Single Project and Tightly Coupled Programs" row, since this is the expected mission classification sized to the class of budget. Most that work under the project, however, whether that be technicians or entry-level engineers, will only need to be cognizant of Figure 2.6 - the *project* lifecycle. Since Wavefront is considered a robotic mission, fewer reviews need to be conducted since human spaceflight is necessarily more rigorous.

Assuming an ATP date of June 2025, Figure 2.6’s beginning phase of formulation under the NASA Lifecycle Phases row, development would stretch several

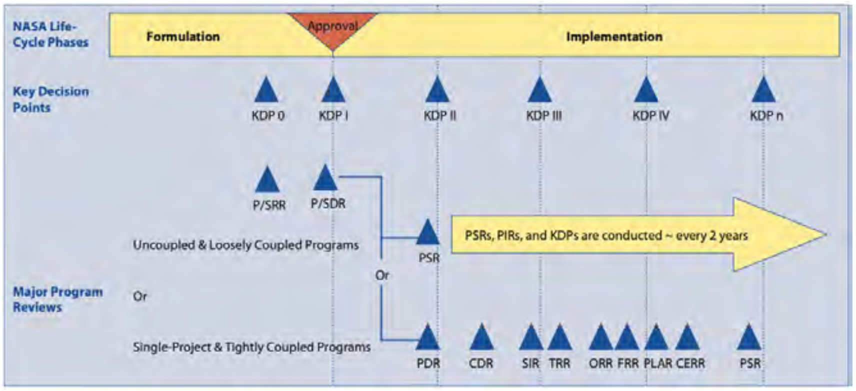


Figure 2.5: Systems engineering key decision points and major program reviews.[5]

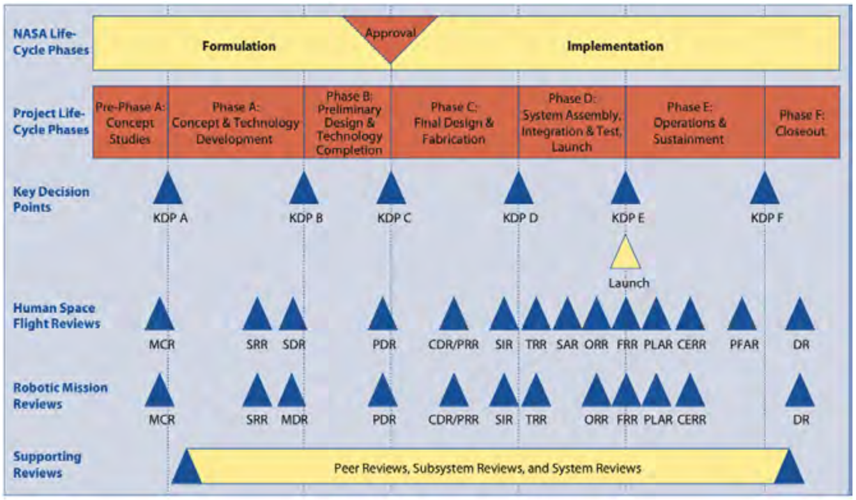

NASA Life-Cycle Phases
Formulation
Approval
Implementation
Project Life-Cycle Phases
Pre-Phase A: Concept Studies
Phase A: Concept & Technology Development
Phase B: Preliminary Design & Technology Completion
Phase C: Final Design & Fabrication
Phase D: System Assembly, Integration & Test, Launch
Phase E: Operations & Sustainment
Phase F: Closeout
Key Decision Points
KDP A
KDP B
KDP C
KDP D
KDP E
KDP F
Launch
Human Space Flight Reviews
MCR
SRR
SDR
PDR
CDR/PRR
SIR
TRR
SAR
ORR
FRR
PLAR
CERR
PFAR
DR
Robotic Mission Reviews
MCR
SRR
MDR
PDR
CDR/PRR
SIR
TRR
ORR
FRR
PLAR
CERR
DR
Supporting Reviews
Peer Reviews, Subsystem Reviews, and System Reviews

Figure 2.6: Systems engineering project lifecycle required reviews. For large projects, especially flagship-class, every review is required to be completed to ensure a continuation of the project [5].

years. Since the U.S. federal budget is allocated on an annual basis starting on October 1, Wavefront's budget, activities, and lifecycle milestones will need to align with the fiscal year (FY). See Figure (figure) for a visual guide to the following explanatory paragraphs.

Phase A, spanning from June 1, 2025 to Q1 FY2028 (October 1, 2027) would represent refinement of Wavefront's conceptual studies and development. During this

Table 2.5: Major Program Review Acronyms in Systems Engineering [5]

Acronym	**Meaning**
ATP	Authority to Proceed
CDR	Critical Design Review
CERR	Critical Events Readiness Review
DR	Decommissioning Review
FRR	Flight Readiness Review
KDP	Key Decision Point
MCR	Mission Concept Review
MDR	Mission Definition Review
ORR	Operational Readiness Review
PDR	Preliminary Design Review
PFAR	Post-Flight Assessment Review
PIR	Program Implementation Review
PLAR	Post-Launch Assessment Review
PRR	Production Readiness Review
P/SDR	Program/System Definition Review
P/SRR	Program/System Requirements Review
PSR	Program Status Review or Pre-Ship Review
SAR	System Acceptance Review
SDR	System Definition Review
SIR	System Integration Review
SRR	System Requirements Review
TRR	Test Readiness Review

time, final changes to the overall concept during the entire operations phase would be completed. This not only includes the spacecraft, but the development of the ground segment, science data distribution plans and archival, integration logistics, facilities usage, among many other higher-level, programmatic aspects that support the devleopment and operations of Wavefront. Due to the sheer number of operational assets, a heavier focus on mission support, operations, and sustainment will be required - potentially similar to supporting a human-rated mission. This is the rationale to having a very long conceptual development arc. One of the primary requirements in the ground support (GS) segment must specify that any GS and operational components shall be repurposeable to human-rated missions to Mars at little to no cost. Without this requirement fulfilled, the program cannot move past SRR to MDR.

Once the MDR has been cleared with an adequate mission definition, the project can then move onto Phase B - preliminary design and technology completion, ending Q4 FY2028 (June 30, 2028). This phase will also be relatively long due to the need for the initial design of the spacecraft to be completely defined and methods for collecting scientific data will need to be validated and brought to completion. Of particular concern would be the impedance-measuring nanoprobes, which requires precision timing, network synchronization with very high sensitivity, and high robustness needed to survive dust storms and long Martian winters. This detail is covered in a separate design chapter. Other details, such as robotic arms with impact drivers on the rover, are critical but do not require very much study effort because there is a wealth of experience with mechatronic systems. Concerning the most critical element of the mission - the Aerith orbiter, most of the design work would center around integration of existing spacecraft buses into a tube form factor and development of high aerodynamic pressure capable solar wings. In data flow terms, Aerith could be considered a single-point failure. However, the high-speed communications relay is a relatively trivial component despite being very critical; Maxar Technologies has decades of experience of building highly reliable, high-throughput relays in buses such as the SSL 1300 [109]. JPL would be responsible for developing the rover during this time, since their decades of experience with rovers would pay dividends into the miniaturized build of the Pascal rover, despite their recent forays into helicopter exploration of Mars.

By this point in time, only a small portion of the budget should be spent, since the Formulation phase of the mission can also be considered the research and development phase. By PDR, the engineering work to accommodate scientific instruments should be complete, and only minor changes should be necessary up until CDR. The time between PDR and CDR in this case, shall only be reserved for finalizing engineering work for science, in order to meet the core scientific requirements developed at the beginning of the program, defined by MEPAG. In all, KDP C is arguably the most important decision point or review because requirement scoping and "last-minute" revisions can still be made.

A successful KDP C leads up to Phase C, where the engineering work and R&D work for science comes to an end. Reports on instrument performance, integration activities, test and integration procedures, contingencies, projected schedule and budget margins, and other related documentation are finalized for reporting in CDR. For Wavefront, the CDR process will take weeks, since the chief engineer, principal investigator, instrument PI's, and the program manager will need to personally delegate shortcomings or challenges and ultimately sign off on every single component of the program. If all goes to plan and the CDR is passed, fabrication of all consitituent systems can finally begin. Since contractors would be accomplishing this work, forward-funding mechanisms and fixed-price contracts would dictate the schedule. In other words, contractor schedules can be decoupled from the FY to an extent. Since much of the time was spent on R&D, fabrication of all components with several contractors would take Phase C up to December 31, 2029. The spacecraft will be the easiest, since serial manufacturing is assumed to be easily accomplishable given thorough procedures and experienced spacecraft instrument manufacturers.

KDP D is more of a soft decision point, and is the most likely point at which delays would occur. This is to say that evaluation of engineering samples and partly integrated spacecraft from serial production is required. The first produced landers and nanoprobes will almost certainly have defects or manufacturing difficulties;

mid-production corrections can be made. However, if a chronic issue does occur that cannot be fixed easily, a program stand-down would then be required. KDP D then becomes the most significant milestone. Delays would have to be aligned to Earth-Mars Hohmann transfer windows, since these occur roughly every two Earth years. Any delays with large numbers of contractors would burn through substantial sums of money and capital to stand down an entire program to review requirements. Thus, a fine balance between required workforce, risk of program-jeopardizing delays, and allowable budget with defined maximum-allowable stand down periods. Figure (figure) illustrates this programmatic engineering challenge.

Assuming there are no program standdowns, Phase D is a smooth transition from Phase C, with engineering samples and demonstration units and spares being built immediately following CDR. Assembly, integration, and testing (AI&T) is performed with all assets in parallel. This massively parallel AI&T process MUST be defined rigidly in systems engineering terms and with information sharing systems and procedures. Product lifecycle management (PLM) software is a required component of this process, especially for final validation of requirements through measurement and characterization. Of course, these processes must occur in a nondestructive manner, and must be designed and built to withstand testing. While testing and system characterization is being finalized, the ORR will then be conducted with selected mission operators, who have, by this point, been walked through the as-built mission hardware. This is necessary to allow operators to rehearse operations through simulations and use of actual hardware/engineering demonstration units (EDUs). The end of Phase D occurs when launch occurs; this is during the point at which C_3 is lowest in the Hohmann transfer window. This date is assumed to be approximately June 2032. Integration of the final payload onto the Falcon 9 occurs T-90 days before launch, according to SpaceX F9 user guides, or approximately February to March 2032.

Assuming the launch occurs and Wavefront is injected into the correct transfer orbit, cruise will take 9 months. Wavefront officially arrives when Aerith fires the main engine to insert the entire stack into a highly elliptical orbit. A CERR is required before this event for mission operators, because operations will very quickly begin to unfold. Before aerobraking to circularization, and after orbit characterization, Aerith will drop one lander per apoapsis to reduce the amount of fuel spent by the Wavefront aeroshell GNC to deorbit. After all payloads are dropped, and Aerith completes circularization, the operations phase kicks into very high gear. This phase ends once all assets are deployed to their correct locations and configurations. The mission is sustained for as long as there are assets surviving on the surface; this would continue for at least two Martian years.

Finally, at the conclusion of the last transmission of the last surviving Midgar lander, a decommissioning review and mission closeout is held, reviewing the mission, engineering and program data, as well as compiling a lessons learned document. The expected data volume from Wavefront will likely be in the thousands of terabytes (TB); data analysis plans to more effectively use and classify science would be required as part of the program plan beyond the decommissioning review. Artificial intelligence and machine classification will almost certainly play a role in this plan.

2.5.2 Project and Work Breakdown

In terms of project and work breakdowns, both must be combined and defined before PDR. However, a project breakdown structure (PBS) must come before a work breakdown structure (WBS) because the elements/subsystems of a system must be defined before any work can be assigned. That is, the scope and variables of the entire engineering problem or project must be known, fully observable, and fully controllable[1] especially for a flagship-sized mission. Figure 2.7 illustrates this in a more engineering-friendly manner.

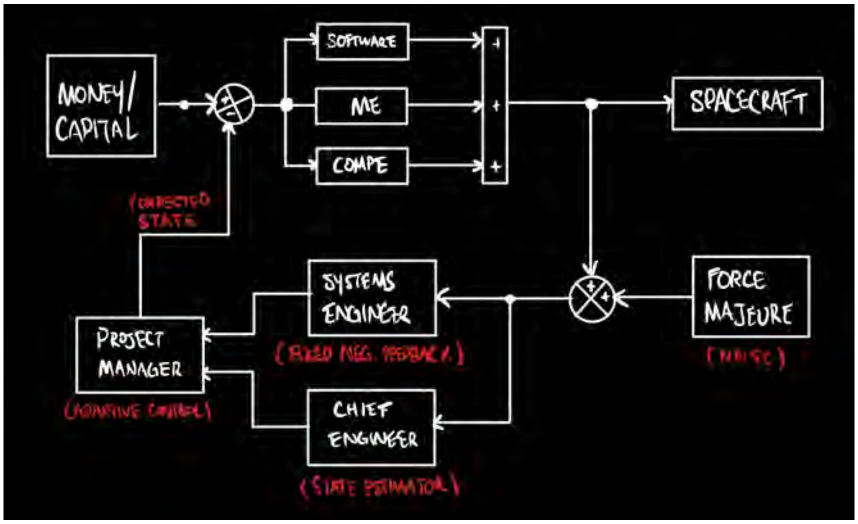


Figure 2.7: A representation of project management as a control system. The highest level input required is money and capital - this flows into engineering work. The sum of this work is included in the output goal of producing a spacecraft. However, things might not go according to plan, such as engineering work not proceeding at the correct pace or unforeseen circumstances interfering with the goal. The systems engineer would analyze the effect on this propagated throughout the rest of the system, while the chief engineer considers how to compensate for this negative effect. Missing is the tightly coupled nature of chief engineer and systems engineer in tandem with project management - this is for clarity's sake. At the end of the feedback analysis, the project manager has the final say over the course of action, since they also have control of the budget.

While illustrating only general controllability analysis of project management, it serves to show that a project can only be managed when most variables are known, and *force majeure* can be compensated adequately. Therefore, a PBS must be defined. Figure 2.8 shows from a top level, what this would look like for Wavefront. A full PBS

[1] This is just like a controls problem, if the system cannot be fully controlled and/or observed, additional states or governing equations must be added until it is full-rank. Failing that, a state estimator may be used to compensate, but at a reduced control system effectiveness. In system and program management, this is equivalent to making assumptions. If the program manager or chief engineer assumes incorrectly based on faulty or limited information, the program will "crash".

for Wavefront would be enormous, and would only be representable in a spreadsheet program. An example of how deep the tree would go is represented in Figure 2.9.

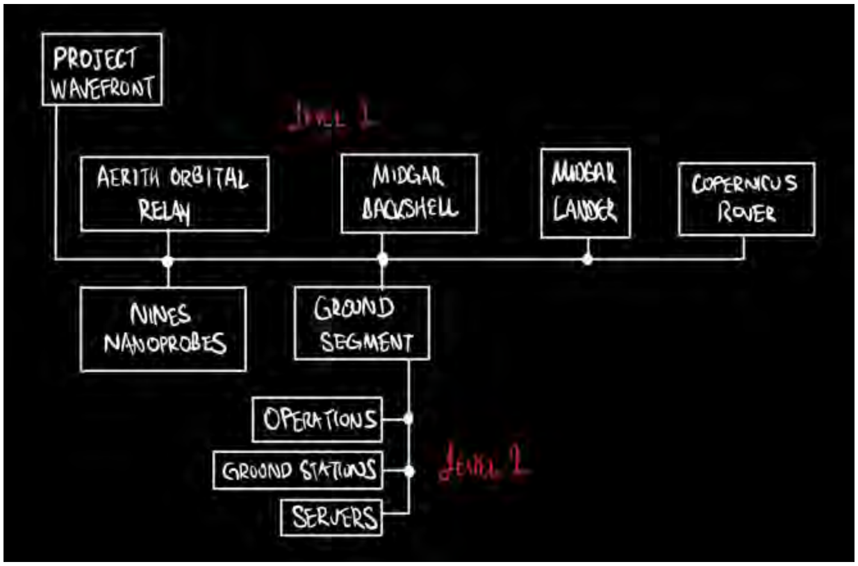


Figure 2.8: Level-1 with a single member node Level-2 product breakdown tree. Each box represents the system, subsystem, and so forth required to formulate the product. When these are fully defined, work can very easily be delegated because the chief engineer and project manager will be able to see the whole picture in terms of the project, and create an appropriately scoped WBS.

This has a visible effect on the finer details of WBS construction. Because of the very fine details required for the project, it is now known that more engineering work and collaboration with scientists is required to meet requirements and deadlines. Because of the higher cost of directly hiring engineers and scientists, work can be leveraged through academic institutions. In some cases, instrument and scientific principal investigators (PIs) are supported through academia, and can then further delegate program-wide Level-3 and higher-level instrument particulars to students. [2]

Not only does the WBS include work breakdown for individual components, it

[2] This is not to say that students are "free labor", students get work experience and further their own educational outcomes and degrees as collateral. There must always be something gained in return; few if any individuals would do something for completely free to expect nothing in return.

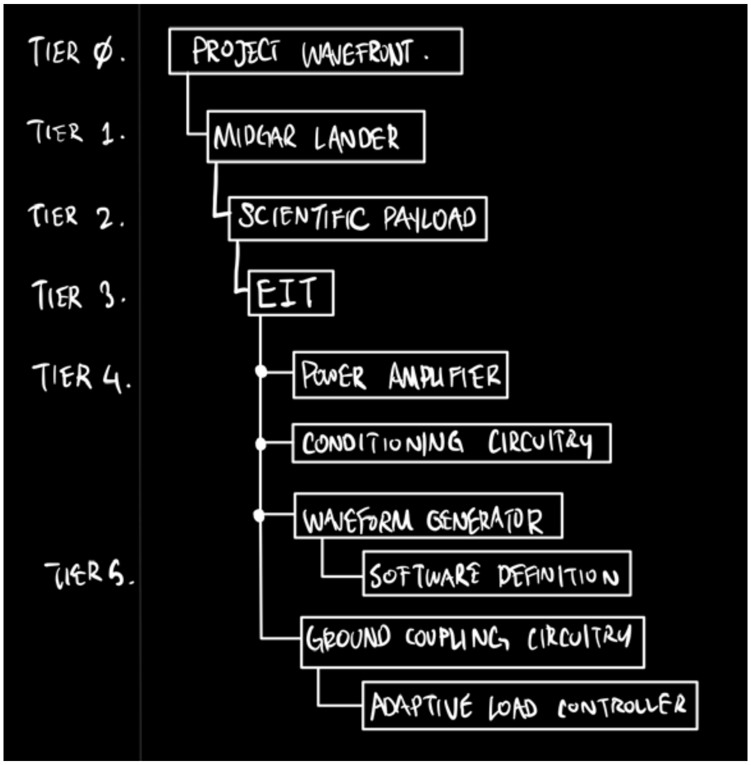


Figure 2.9: Representative Level-5 product breakdown for impedance transmitter on electrical impedance tomography instrument of individual Midgar lander. Because of the distributed nature of this instrument, this is not a complete PBS of the EIT instrument. Aside that, the very high tier numbering shows how much engineering would have to be considered as part of the system, which significantly increases the number of engineers required for this stage of the project.

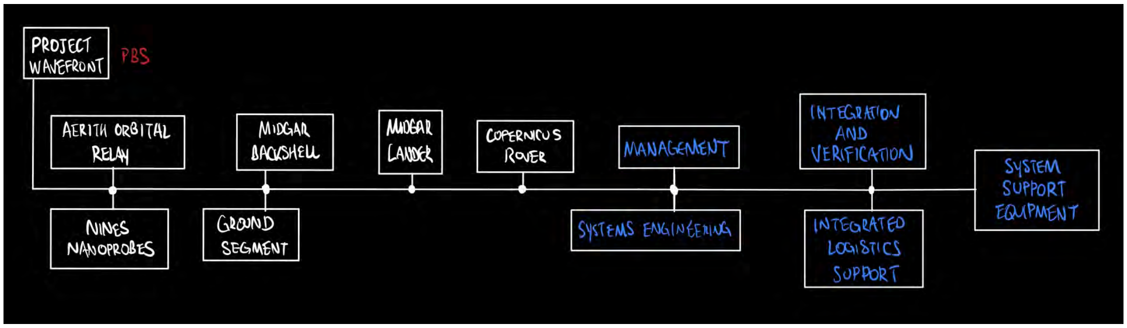


Figure 2.10: Representative overall WBS of Wavefront. Note that at the highest level, it looks similar to a PBS, but now includes management and work systems to tie everything together.

would include the subsystems and tasks needed to "glue" the system together. This is why a WBS can be described as a *gestalt*: the sum of an organized whole is perceived as greater than its individual parts. An example of this is building a complete car from parts in a junkyard - there are a sea of parts to choose from that are in various conditions. An individual or individuals that get together to choose the right parts to create a working car requires background knowledge of how a car works, what parts are needed, the regulations required to pass a smog test and to be in a safe condition to drive, etc. Effectively, this describes a systems engineer - someone that has the knowledge of how to organize the *people*, *knowledge*, and *requirements* together to complete a system, such as a car built from scrap in a junkyard. The WBS in Figure 2.10 accounts for the *gestalt*.

According to the NASA Systems Engineering Handbook, 2007 edition, the WBS is responsible for:

1 - Project and technical planning and scheduling,

2 - Cost estimation and budget formulation (in particular, costs collected in a product-based WBS can be compared to historical data. This is identified as a primary objective by DOD standards for WBSs),

3 - Defining the scope of statements of work and specifications for contract efforts,

4 - project status reporting, including schedule, cost, workforce, technical performance, and integrated cost/schedule data (such as earned valud and estimated cost at completion),

5 - plans, such as the Systems Engineering Management Plan, and other documentation products, such as specifications and drawings [5].

Clearly, without a WBS, a program of this scope will not be able to function properly.

In the interest of leveraging organizations that have significant experience in building spacecraft with rigorous systems engineering, V&V, and I&T practices, both a well-defined WBS and knowledge of contractor experience and competency with appropriately scoped systems is required. For example, JPL/Caltech is viewed by the space and defense industry as one of the most experienced interplanetary probe integrators, with solid management and design, V&V, and I&T processes. Their experience includes learning from challenges and failures, of which they have many lessons learned. This being said, JPL does not have a record for serial production of large amounts of small mission hardware - the Wavefront mission would consist of twelve landers, twelve rovers, and 120 spike-mounted nanoprobes. Despite this, JPL cannot be discounted from IV&V and AI&T and would ultimately play a significant role in R&D (Phase B/C) of the project. The serial production phase would then likely fall with, as of writing, a division of Lockheed Martin or Maxar. Further subcontracts might be necessary, especially those experienced with small robotic systems.

One of the last major points to cover in this section is point 5 in the WBS purpose: formulating a Systems Engineering Management Plan (SEMP). The SEMP is a comprehensive document that conveys technical and engineering activities conducted during the project [5]. It serves as an agreement for how the work within the scope of the program will be accomplished, as communicated to all personnel. The SEMP includes details as specific as what types of tools and facilities are to be used in integration and testing, how strictly the environment must be controlled (such as cleanroom classifications), materials compatibility, responsibility and authority, reviews to ensure compliance with recognized standards and internal program requirements, among many others. It includes the following general sections:

1 - Technical program planning and control, which describes the processes for

planning and control of the engineering efforts for the design, development, test, and evaluation of the system,

2 - Systems engineering processes, which includes specific tailoring of the systems engineering process as described in the NPR, implmentation procedures, trade study methodologies, tools, and modesl to be used,

3 - Engineering specialty integration describes the integration of the technical disciplines' efforts into the systems engineering process and summarizes each technical discipline effort and cross references each of the specific and relevant plans.

The SEMP cannot be completed until the scope and purpose of every single instrument and system is defined, which is dependent on everything covered in this chapter - the program must be defined rigidly enough so there is no ambiguity in downstream documentation like the SEMP.

2.5.3 Cost Breakdown

As discussed in the prior subsection, a WBS is invaluable for cost estimation. A US$3 to 5 billion was initially proposed, but with a basic breakdown, a more rigorous cost estimation can be made.

All things considered thus far, the following tables and subsections document the cost breakdown. These are estimates, and are based on literature reviews and prior experience. Table 2.6 summarizes the cost breakdown.

2.5.3.1 Aerith Orbital Relay

As previously stated, despite being considered a "secondary" payload that carries the primary payloads to the surface, Aerith is the most important component of the actual mission hardware. Based on the literature review and lifespan analyses of similarly classified relay spacecraft, an cost estimate over the full life of Aerith

Table 2.6: Wavefront Basic Cost Breakdown

Line Item	**Cost**
Aerith	**$1400M**
- Hardware Cost	$640M
- Total Launch Cost (aggressive-conservative est.)	$400M
- General Development Margin	$200M
- 25-year Operating Budget	$160M
– Base Operating Budget, 3-shift exempt model	$105M
– Reserve	$55M
Midgar Lander	**$750M**
- 25x Lander and Backshells	$30M each
Nines Nanoprobes	**$120M**
- 173x Nanoprobe Production Budget	$40M
- Development Budget	$60M
- Development Margin	$20M
Pascal Rover	**$340M**
- 25x Rover Production Budget	$175M
- Development and Test Budget	$130M
- Robotic Arm Development Margin	$15M
- Cold-Tolerance Development Margin	$20M
Ground Operations	**$800M**
- Operating Budget	$300M
- Facilities/Federal Real Property	$150M
- Margin for Extended Operations	$350M
Data Research - Direct Funding	**$120M**
- Mars Analogs and Laboratory Follow-ups, 1-year	$75M
- PI Funding, 3-year	$45M

would be $1.4 billion. Hardware development would constitute approximately $1040 million, with the rest ($360 million) allocated to operations. While the spacecraft would return large amounts of data, a small team is required to run it. This is in line with operations of other mainline communications relays at Earth and Mars.

Past the prime mission, Aerith would continue to serve as an orbital relay for other assets, including supporting human spaceflight. A mission length is scoped for 25 years, with 30 operators running instruments, relay functions, and other systems aboard the spacecraft. A three-shift model of 10 operators at exempt pay status would form the team monitoring operations. With no margin, this adds up to $105M. The remaining $255M serves as a contingency budget, shared amongst other assets.

2.5.3.2 Midgar Backshell and Lander

The largest hardware cost will lie in the serial production of the Midgar lander and backshell system. In the manufacture of production units, the general rule of thumb is to produce three copies of hardware for the actual flight article: a "training" unit, an engineering demonstration unit (EDU) that is physically and functionally the same as the flight unit, a backup flight unit, and of course, the flight unit. For serial production of more than one flight unit, this rule can be more relaxed. However, as soon as manufacturing revisions are incurred that significantly change any operational or assembly procedures, the same three units then apply to the serially produced units of that new revision. It is expected that up to three total manufactured revisions are made during the production run of Midgar, which then requires 21 Midgar articles to be produced. In case of any manufacturing or quality assurance issues, a further four units will be produced to use as spares. In all, 25 units would be produced.

The only similarly sized hardware to land on Mars to draw conclusions from is Mars Pathfinder, which cost $150M for the lander, and $25M for the rover. Midgar is a considerably more advanced lander, with many more survival features and scientific instruments, but given the pace of technological advancement and the lower costs of producing hardware [3] , Midgar would likely have the same pricetag. Additionally, assuming economies of scale port directly to spacecraft, the production cost per unit would end up being lower. For Mars Pathfinder, assuming the 3-spare model, each Pathfinder lander and EDL unit cost $37.5M. Assuming a cost of $30M per unit after

[3] This even considers the extreme pressure of inflation as of writing.

the economies of scale assumption, all Midgar landers would cost $750M. Note that this is only the lander, and not the rover or nanoprobe.

Operations is an entire Pandora's box, one unexplored in deep-space exploration thus far. Up to twelve units will operate on the surface of Mars, which does not include the 120 Nines nanoprobes and 12 rovers designed to deploy the nanoprobes. This aspect is further explored in a later section as a separate budget line item.

2.5.3.3 Nines Nanoprobes

Nines is a relatively simple probe to design and build - much of the upfront R&D of an individual unit would come from development of a cold-tolerant, low-friction slip ring to allow solar panel cleaning, and high-sensitivity receivers for ground impedance.

To ensure the EIT system functions to specified requirements, a lander simulator and the entire complement of nanoprobes are required. Holding with the three-unit rule of thumb plus 3 spares, 33 would be produced for V&V and mission operations scenarios. The mission set requires 120 probes, and with a two backup sets, a total of 173 probes would be produced. Development and production would amount to $100M, based off of IPC Class 3 printed circuit board fabrication houses[3] , custom battery manufacturers, extended-range environmental testing, and cold-tolerant electronics. In case of development cost overruns, an additional $20M is allocated.

The pricing of the fixtures for mounting and securing the nanoprobes are counted as part of the lander.

[3] Most board houses with manufacturing to these standards provide quotes classified as proprietary information.

2.5.3.4 Pascal Rover

Developing the Pascal rover is fortunately aided by many advances in mechatronics and a similarly sized precedent: the Pathfinder rover. The Pathfinder rover was developed under the Faster, Better, Cheaper era of NASA; it helped that the rover was only a technology demonstration. Pathfinder's development budget was only U.S.$120 million, but had to deal with a slew of unknowns and how to address them. With decades of roving experience and data, the development of a chassis, drive system, vision system, and radiation-tolerant circuitry is expected to be a fraction of Pathfinder's development costs. Another example working towards Pascal's low cost is the highly succesful Ingenuity helicopter. The entire budget for the helicopter came in at U.S.$27 million, with an expected lifespan of 5 flights, or about 5 minutes in the air. More than one Martian year on, the helicopter has made over 50 flights, and is not showing any significant signs of wear. Revised versions of the Mars Sample Return program envision replacing rovers with helicopters of the same size, each sporting very small robotic arms designed to pick up sample tubes deposited by Perseverance.

These two data points help inform a conservative cost of U.S.$130M to develop and test the platform before mass production. The rover production cost of U.S.$175 million accounts for custom tooling and political agendas, which are coupled factors. In the current political environment, a U.S. mission would likely involve restrictions on outsourcing and domestic production limitations, including components as small as capacitors and resistors. The single, most expensive components are the FPGA's and processors - the U.S. has very limited domestic production capacity, especially for larger feature sizes and special production treatments such as chemical vapor deposition (CVD) plating of iridium for radiation resistance. Therefore, the tooling set up for rover and lander mass production would need to be leveraged in the future for other rovers or similar systems.

The robotic arm and cold-tolerance budget line items are separate from the development and test budget because of their importance to the mission. Ingenuity had recently demonstrated survival to as cold as -85°C for multiple months. A similar electrical system topology is expected to be leveraged, especially control and power electronics, as well as batteries.

2.5.3.5 Ground Operations

The operations budget is expected to comprise a large portion of the program budget. This includes facilities procurement and development, as well as margin for extended operations. A detailed breakdown was skipped because there was little familiarity with this subject. As a result, the budget required for ground and facilities management might have too large of a margin.

2.5.3.6 Direct-funded Data Research

To support future missions and mission planning, direct funding is provided for analysis of Wavefront data. The primary benefactors of this analysis would be scientists and the overarching steering committees, such as MEPAG. The first line item, Mars Analogs and Laboratory follow-ups, is heavily funded to support rapid research and conclusions into findings made by the program. This funding would not be activated until a sufficient amount of data are collected. The second line item is for more detailed investgations, which fund up to 10 PI's at U.S.$1.5 million for three years. This funding includes equipment purchases and intern funding.

2.6 Conclusion

This extensive chapter set the framework and scope of Wavefront starting at the political level. This is necessary due to the large scope of the project - at the total cost of U.S.$3530 million, it is termed a program. In NASA nomenclature, it is considered a Flagship program. Program development and cost scheduling and rationales are developed.

3. Engineering Constraints

This chapter details the highest level engineering requirements in Wavefront - ones that are fundamental to the mission. Given a U.S.$3.53 billion budget, there is an acute need to follow through on all requirements, procedural and engineering. These high-level engineering requirements constrain the problem to reduce the risk of a runaway budget.

Aside from project management requirements and frameworks, the highest level engineering requirement specification to follow is the AFSPCMAN 91-710 document on launch vehicle, range safety, and payload requirements [110]. This document states basic requirements for launch vehicles, safing of hazardous payloads such as explosives and fuels, and redundant system requirements. Wavefront would be launched from a US Space Force (USSF) base, and to ensure minimal risk to technicians, range safety officers, and to ensure minimal downtime on the base or range, compliance with this document is mandatory. Covering the entire document and validation of compliance would require many more chapters and sections than practical. This section summarizes the most important, higher-level requirements necessary for Wavefront. A more intuitive method of describing why all requirements and design efforts for such a large project must flow down from AFSPCMAN 91-710 is the fact that most spacecraft endure the most severe loads *only* during launch. This is the shortest phase of the mission, but obviously the most critical since if the launch vehicle fails or a spacecraft system fails due to launch loads (high g-loading or vibrations), the mission is immediately deemed a failure.

3.1 MSPSP

The Missile System Prelaunch Safety Package (MSPSP), per AFSPCMAN 91-710 page 212, is a required document from all rocket providers, ground support operators, and payloads that describe hazardous hardware and safety-critical equipment. Assuming that the launch provider decouples all hazardous rocket requirements from the payload (which is reasonable), the payload provider shall specify all hazardous systems and materials present. Per MIL-STD-882, *Department of Defense Standard Practice for System Safety*, hazardous systems are those that would present a significant hazard to life and safety if the system were to fail in an inadvertent manner. In the purview of Wavefront, these are systems like batteries, separation devices, propulsion systems, and mortars.

Table 3.1: NFPA 704 Standard "Fire Diamond" - Health[6]

Rating	**HEALTH (blue)**
0	Poses no health hazard, no precautions necessary and would offer no hazard beyond that of ordinary combustible materials.
1	Exposure would cause irritation with only minor residual injury.
2	Intense or continued but not chronic exposure could cause temporary incapacitation or possible residual injury.
3	Short exposure could cause serious temporary or moderate residual injury.
4	Very short exposure could cause death or major residual injury.

Of these four systems, propulsion systems have the highest potential to be a hazard. To maintain heritage with other Mars and lunar landers, hydrazine propellant will be used. It is an extremely hazardous liquid: on the NFPA 704 scale in Table 3.2, 3.1, and 3.3, it rates as a 4 on health hazard, 2 on flammability, and 3 on reactivity. Containment design is therefore extremely important, which does not include the necessary procedures for loading and handling propellant during preparation for flight. Additionally, as shown later, custom tank designs in potentially custom shapes will be

Table 3.2: NFPA 704 Standard "Fire Diamond" - Flammability[6]

Rating	**FLAMMABILITY (red)**
0	Materials that will not burn under typical fire conditions, including intrinsically noncombustible materials such as concrete, stone, and sand. Materials that will not burn in air unless exposed to a temperature of 820°C (1,500°F) for more than 5 minutes.
1	Materials that require considerable preheating, under all ambient temperature conditions, before ignition and combustion can occur. Includes some finely divided suspended solids that do not require heating before ignition can occur. Flash point at or above 93.3°C (200°F).
2	Must be moderately heated or exposed to relatively high ambient temperature before ignition can occur. Flash point between 37.8 and 93.3°C (100 and 200°F).
3	Liquids and solids that can be ignited under almost all ambient temperature conditions. Liquids having a flash point between 22.8°C (73°F) and having a boiling point at or above 37.8°C (100°F) or having a flash point betwen 22.8 and 37.8°C (73 and 100°F).
4	Will rapidly or completely vaporize at normal atmospheric pressure and temperatur, or is readily dispersed in air and will burn readily. Includes pyrophoric substances. Flash point below room temperature at 22.8°C (73°F).

Table 3.3: NFPA 704 Standard "Fire Diamond" - Reactivity[6]

Rating	**INSTABILITY-REACTIVITY (yellow)**
0	Normally stable, even under fire exposure conditions, and is not reactive with water.
1	Normally stable, but can become unstable at elevated temperatures and pressures.
2	Undergoes violent chemical change at elecated temperatures and pressures, reacts violently with water, or may form explosive mixtures with water.
3	Capable of detonation or explosive decomposition but requires a strong initiating source, must be heated under confinement before initiation, reacts explosively with water, or will detonate if severely shocked.
4	Readily capable of detonation or explosive decomposition at normal temperatures and pressures.

required. 91-710 Chapter 12 details the design criteria and verification and validation approach for pressurized tanks, which themselves follow U.S. Department of

Transportation (DOT) standards for tank and pressure vessel design.

The next hazardous system is the parachute mortar. Usually, a black powder charge is used when blowing a parachute assembly from its mortar shell. Due to the risk of explosion during spacecraft integration, launch vehicle handling, or other instances where the spacecraft is interacted with, stringent requirements are levied upon ordinance systems. 91-710 Chapter 13 details these requirements. While the parachute mortar would hold heritage to other Martian systems, they would still need to go through a classification system as they would be brand new designed systems. Per Requirements 13.1.1.1, 13.1.1.2, and 13.1.1.3, the mortars shall have United Nations (UN) explosive hazard classification, shall be tested against U.S. NAVSEAINST directive 8020.3 Explosive Hazard Classification Procedures, and shall obtain a DOT classification.

In the consumer sphere, batteries are generally deemed safe due to their widespread adoption. However, this is because they are handled under "ordinary" circumstances - room temperature, rarely if ever exposed to mechanical and thermal shock, and have usually been tested with a known, maxmimum load. In the space and defense sphere, batteries are treated as a hazardous system due to chemical storage of energy. In the presence of other hazardous systems and human operators, inadvertent catastrophic failure of a battery system could result in catastrophic failure of other systems and injury or death of technicians and/or operators. To comply with range safety and the MSPSP requirement, Chapter 14, Section 1 spells out all required design features, specified transportation methods, and safety devices for batteries and other high-power electrical systems. One particular design feature requires physical connections or connectors that contain a "positive locking mechanism". For example, if in the case mechanical vibration from launch loads induce a resonant mode that (dis)connects a connector for a safety-critical system such as a mortar detonation circuit or activate a high-power load on the battery, inadvertent actuation might occur, which could destroy the spacecraft and/or launch vehicle.

Finally, separation devices would be examined. These may contain either explosive or non-explosive actuation devices. In Wavefront's case, all separation devices aside from the LV separation ring, would be non-explosive electromechanical actuators. Therefore, the separation device would not be considered hazardous, but only safety critical. Design of the system and V&V activities for such systems would thus fall upon software and electrical design, covered in Chapter 15 and 16.

3.2 Inhibit Scheme

To render spacecraft systems inert during the entirety of the vehicle launch, appropriate inhibit schemes shall be used. Even though the hazardous systems will not be used until arrival at Mars, after the cruise stage insertion burn, all systems must under all known circumstances remain inert. Because of the importance of this, all inhibit requirements from 91-710 are listed in Table 3.4.

A catastrophic hazard would be considered one where the launch vehicle completely fails due to inadvertent activation of a hazardous system. The most catastrophic example would be one of the lander parachute mortars blowing, resulting in overpressure of the launch vehicle fairing, blowing it off during ascent. Regardless of launch provider, such a scenario would result in automatic activation of the flight termination system (FTS), since the vehicle would very likely no longer be safely controllable. Thus, the parachute mortar inhibit will require a triple inhibit system. Figure 3.1 shows an electrical example of a solution to this requirement. Mechanically, this implementation would occur through the use of high-reliability, low-resistance plunger switches, such as shown in Figure 3.2, which might contact the cruise stage

Table 3.4: AFSPCMAN 91-710 Inhibit Requirements

Requirement
3.2.1 - If a system failure may lead to a catastrophic hazard, the system shall have three inhibits (dual fault tolerant).
3.2.2 - If a system failure may lead to a critical hazard, the systems shall have two inhibits (single fault tolerant).
3.2.3 - If a system failure may lead to a marginal hazard, the system shall have a single inhibit (no fault tolerant).
3.2.4 - Probabilities of hazard ocurrence shall be taken into consideration when determining the number of required inhibits (See AFSPCMAN 91-710 Volume 1, Chapter 3, Table 3.1).
3.2.5 - Systems shall be able to be brought to a safe state with the loss of an inhibit.
3.2.6 - All inhibits shall be independent and verifiable. Common case failures shall be considered.
3.2.7 - Design inhibits shall consist of electrical and/or mechanical hardware.
3.2.8 - Operator controls shall not be considered a design inhibit. Operator controls are considered a **control** of an inhibit.

wall.

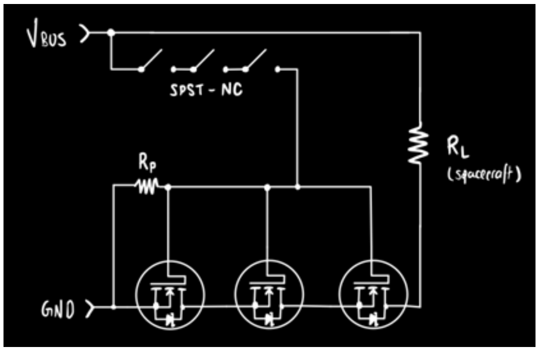


Figure 3.1: Electrical schematic for a dual-fault tolerant inhibit scheme.

Inhibiting the rest of the spacecraft during launch is accomplished using the same mechanical switch scheme, except they drive silicon carbide (SiC) MOSFETs.

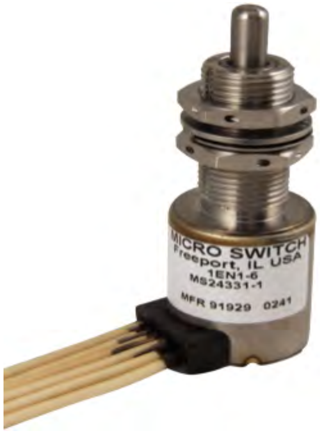


Figure 3.2: Military-grade, hermetically sealed plunger switch. Credit: Honeywell Aerospace

Figure 3.3 shows this modified inhibit scheme. SiC MOSFETs have been used for decades in switching and regulation electronics and have significant spaceflight heritage in military electronics. Due to the safety- and mission-critical nature of these inhibits and the very high current carrying capabilities of SiC MOSFETs, they are used in lieu of purely mechanical switch designs or newer gallium nitride (GaN) FETs.

3.3 Launch Vehicle Identification

The next level down is to identify a launch vehicle (LV). As of writing, a sizable number of companies with different sizes of launch vehicles exist. Almost all companies advertise their capability to low Earth orbit (LEO) from specific launch sites. A big figure of merit in determining total payload energy is launch site inclination: the closer one is to the equator, the more one gets a "boost" for free. The Earth's rotational velocity at the equator is 447 m/s - this is 5.73% of total orbital velocity in an earth-centered, earth fixed frame (ECEF). However, orbital dynamics can still allow more efficient or opportunistic launches from higher inclinations or even retrograde launches. For example, the Double Asteroid Redirection Test

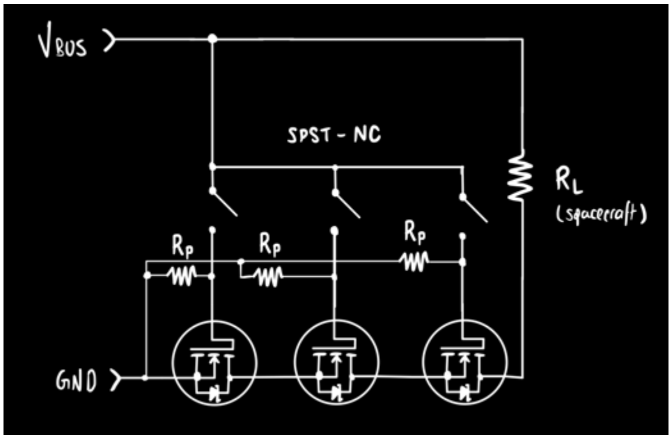


Figure 3.3: Electrical schematic for a dual-fault tolerant, electromechanical inhibit scheme. Resistor R_P is used to "pull-down" the voltage to zero, since MOSFETs behave like parallel plates - capacitors and must be discharged to properly signal. Resistor R_L is the entire spacecraft load. The switches are ahead of the spacecraft load in order to allow power to flow to the inhibit MOSFETs.

(DART) mission launched into a 200x300, 64.7^o parking orbit to then accomplish a Hohmann transfer to Didymos.

3.4 Launch, Cruise, and Arrival

The highest-level requirement is of how the lander hardware will be ferried to Mars. Depending on the phase departure from the lowest energy Hohmann transfer acheiveable, Wavefront will have a wide range of possible approach velocities in a Mars centered Mars frame of reference (MCMF). The Mars Pathfinder and MESUR missions designed for as high of an approach velocity as 7.6 km/s. This strongly affects the thermal protection sizing and mass on entry into Mars's atmosphere, which in turn negatively affects the available landable scientific mass.

However, because an orbital relay will fly with the landers, a hyperbolic entry, descent, and landing is not desirable. The tradeoff is that a higher fuel mass fraction will be required to perform a costly orbital insertion burn. This will narrow the launch window to a smaller number of days; in other words, there will be a reduction in the permissable phase error relative to a Hohmann transfer resulting in the lowest C^3, holding the launch vehicle the same. This concept is illustrated in relative terms in Figure 3.4. A larger rocket will allow larger phase error, however, this also carries it's own constraint: some rocket companies specify a *minimum* launch mass to reduce the acceleration and vibration imparted onto the spacecraft. In Figure 3.4, this is represented by the lower line. Intuitively, the upper bound gives the maximum mass allowable in a given Hohmann transfer. For the final Wavefront mission studied here, there exists a plethora of launch vehicles that could accomplish the job.

The largest constraints are launch service provider availability and reliability. At the time of writing, the only LSP capable of launching to Mars reliably, on schedule, and on budget for the mission size is SpaceX. Their Falcon 9 is capable of transferring up to 4 metric tons (Mt, 4,000 kg) to a Martian transfer orbit. Further details regarding this maximum transfer mass was not available, but an intuitive assumption can be made where this represents the top-right of the porkchop plot. Figure 3.5 shows a Falcon 9 with the final mission configuration, to scale, seated in the fairing.

The next family of tradeoffs are a direct consequence, and benefit, of carrying an orbital relay. The benefit of carrying a relay is such that the actual Wavefront landers do not have to be free-flyers, they can obtain power and thermal control from its' "mothership". Individual free-flight has the following consequences:

- navigation for multiple objects has to be performed, increasing mission support cost,

- complete systems have to be carried with the Wavefront landers to keep them power-positive and thermally regulated, which includes an appropriate guidance, navigation, and control (GNC) system for thermal regulation, increasing individual

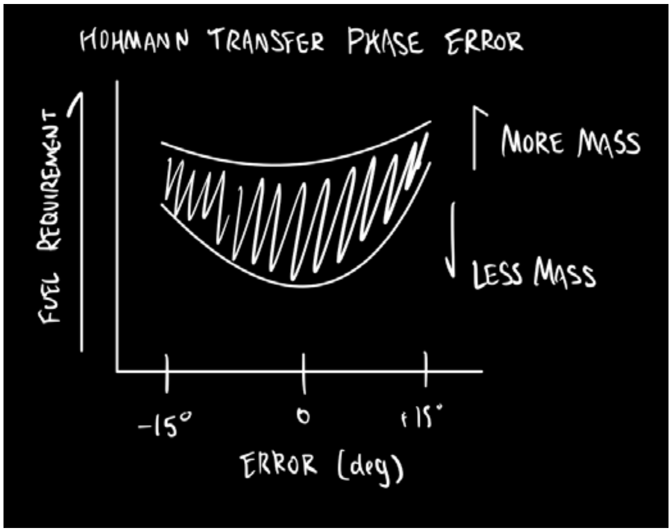


Figure 3.4: A basic tradeoff of the phase error between an ideal Hohmann transfer and the fuel requirement. The space in between the upper and lower lines represent an acceptable mission. In orbital dynamics, derivatives of this plot are called "porkchop plots", due to the shape of the area within the curves forming a porkchop.

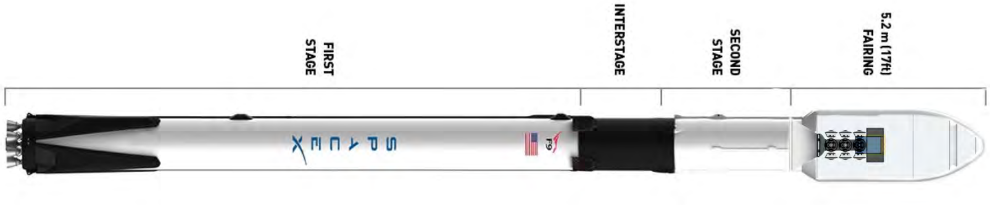


Figure 3.5: An illustration of SpaceX's Falcon 9 with a to-scale representation of Wavefront seated in the payload fairing. Background image credit: SpaceX.

system mass and complexity,

- no possibility for delaying or repositioning landings to a very specific site, and

- TPS mass is increased due to the direct, hyperbolic entry required.

Free-flight was considered early on in the mission design process; Figure 3.6 shows a detailed cruise stage design for such a configuration. Bifacial, fold-out solar panels built using commercial, off-the-shelf modules would provide 30 watts electrical power per wing.

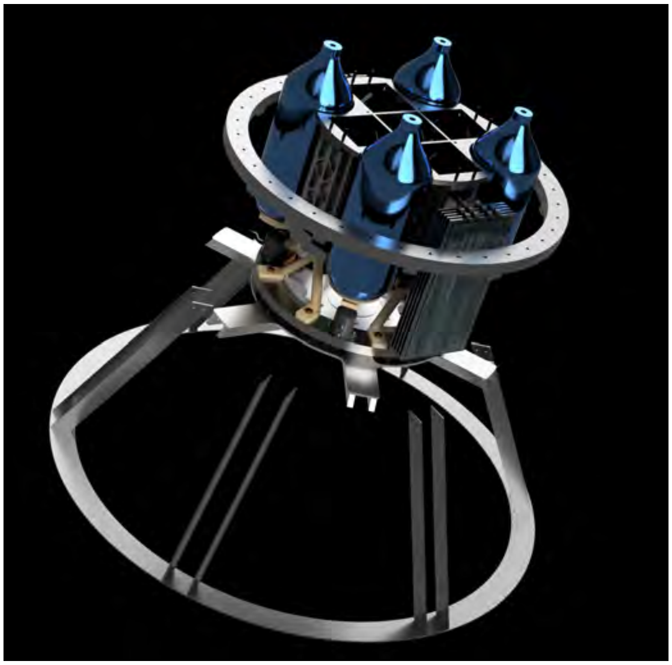


Figure 3.6: Detailed design of a cruise stage with stand-in EDL capsule skeleton. The cruise stage was designed to fit inside of an ESPA ring's portal (shown floating at the midpoint of the anodized fuel tanks), in order to comply with volumetric requirements for a ULA Atlas V 421 launch vehicle. All hardware needed to join all components is modeled; STEP models are taken from hardware distributors. This was necessary to validate the ability to be assembled and whether realistic fasteners were available for the design. To facilitate modularity of this cruise stage, the central cavities were compliant with a 3x1U CubeSat specification. Each cavity would host cruise stage systems, such as the flight computer, batteries, communication systems, and potentially a separate CubeSat.

For the "mothership" or orbital relay concept, the most visible consequence is the mass required to perform an orbital insertion burn, while carrying the extra mass of the landers. Some missions operated off of this principle: the most recent example of this paradigm successfully working is China's *Zhurong* lander. The *Tianwen* 1 orbiter carried *Zhurong* during the orbital insertion burn. *Aerith*, the orbital relay and mothership, is a would carry twelve landers. To reduce the fuel carried to perform a needed circularization (to be as close as possible to landed assets, maximizing bit rate), *Aerith* is designed to perform aerobraking. *Mars Global Surveyor*, *2001 Mars Odyssey*, *Mars Reconnaissance Orbiter*, and *MAVEN* are known to have successfully performed an aerobraking maneuver to perform circularization. According to JPL, this maneuver reduces fuel needs by up to 50%. Such a maneuver carries other major tradeoffs, which will be explored in the detailed design of *Aerith*.

In summary, landing hardware *will* be ferried to Mars by a combination mothership and orbital relay that utilizes an insertion burn and multiple aerobraking maneuvers to circularize before dropping off the landing hardware.

3.5 Communications

MESUR and the Viking landers provided very important insight into the need for an orbital relay, which directly affects the selection of an arrival type. During Martian conjunction, when Mars and Earth are the furthest apart, MESUR was expected to communicate at rates as low as two bits per second direct-to-Earth (DTE), or when forward error correcting is introduced, fewer than 1/4 symbols per second per MESUR probe. For the Viking missions, DTE bit rates from 250 to 1,000 bps were acheived through all phases of the mission. Considering an IoT approach to global in-situ science, even the higher bit rate achieved by Viking is simply untenable and unsustainable for multi-year missions. Viking designers were aware of this issue: the orbiter functioned as a relay to Viking: bit rates of up to 16 kbps were achieved [42], a substantial improvement over a DTE link.

During the extensive field campaigns of the *Nines* nanoprobes, high resolution data from multiple sensor suites simulated Mars IoT deployments. Depending on content and sensors, each probe returned 60 kB to 1 MB of data per day, in the form of comma-separated values (CSV). In a four lander mission, a total of 36 assets would be able to return scientific data. Assuming the worst-case engineering design envelope of 1 MB/day/asset, up to 36 MB per day would be collected on environmental conditions. Since similar IoT strategies, topologies, and hardware would be used, this is a very acheivable goal.

3.6 Power

While there are plenty of solar-powered Mars missions, solar power incurs a certain amount of operational risk for long-term missions. This is due to the unpredictability of dust storms. A great example is the Opportunity rover: it was able to operate for 15 years and survive multiple dust storms, but the June 2018 global dust storm ended it due to an extended period of time without enough sunlight.

3.6.1 Nuclear

Nuclear power has been the choice of energy for many high-power rovers or spacecraft that require long mission durations, such as *Curiosity* and *Perseverance*. However, to date, zero practical missions in the smallsat category have been designed or scoped for nuclear energy [2] . This is due to a wide range of factors,

To avoid a complete and very costly redevelopment of an RTG, the most recent fuel containment mechanism is used. The U.S. Department of Energy's *General Purpose Heat Source* (GPHS) is a highly reinforced, insulated, and ablatively coated enclosure for four standard plutonium dioxide heat sources. Despite this usage, in order to comply with containment requirements in the event of catastrophic failure of the launch vehicle for instance, certification and engineering with high design margins

[2] A notable exception is Breakthrough Starshot. These would use a ground-based laser firing at solar sails for propulsion, and very small RTGs for power generation. However, this mission is still not practical due to the need for construction of a 100-gigawatt laser and light sail material needing almost perfect reflectivity [111].

are required. Such a redesign has the unfortunate side effect of costing tens of millions of dollars. This design process is detailed in later chapters, and resulted in an untenable and unwieldy design for small probes.

3.6.2 Solar

The only other feasible choice for medium- to long-term power generation is the use of solar power. Two types come to mind: rigid and flexible. Up until 2017, rigid, foldable solar arrays have been used in spaceflight. The flight test of the USAF's Roll-out Solar Array (ROSA) on the International Space Station (ISS), shown in Figure 3.7, is a paradigm shift. Higher density packing and lower masses can be achieved, leading to higher power generation and/or larger margins. In terms of lifespan on Mars, the biggest design variable is dust accumulation. Over time, dust eventually accumulates to the point the mission comes to an end. This dust accumulation rate depends on geography and a number of other factors; Spirit and Opportunity were fortunate enough to have dust devils pass over the rovers frequently enough that the missions were extended multiple times. On the other hand, the more recent InSight lander was not fortunate enough to see even one cleaning event - the mission was ended after about two Martian years.

One design note is that all solar powered Mars missions used horizontally deployed panels. By deploying the solar panels at a relatively high angle, dust could roll off the panel over time. Such a design is possible by implementing a ROSA on a gimbaled mount.

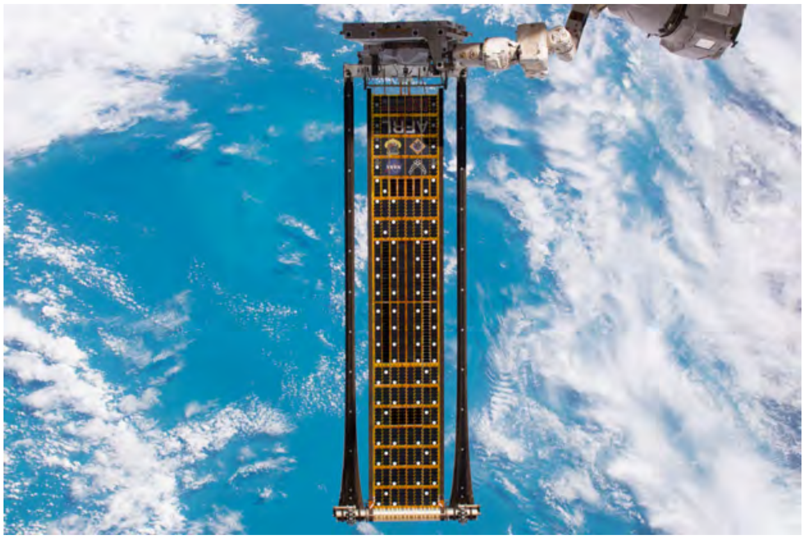


Figure 3.7: U.S. Air Force's Roll-out Solar Array (ROSA) mounted on Canadarm at the ISS for system validation and testing. Credit: NASA/Air Force Research Laboratory.

4. Architectural Design Process and Concept of Operations

With all of the guardrails for design in place, this chapter describes the concepts of operation and development of notional designs into the final product.

4.1 Architecture, Version 1

The original proposal called for a U.S.$25 to 40 million payload of opportunity that would be deployed from a NASA Flagship mission. The specific details of this work are published in [81]. The maximum diameter was specified at 13 inches, which is designed to fit inside an ESPA 6-15-24, shown in Figure 4.1.

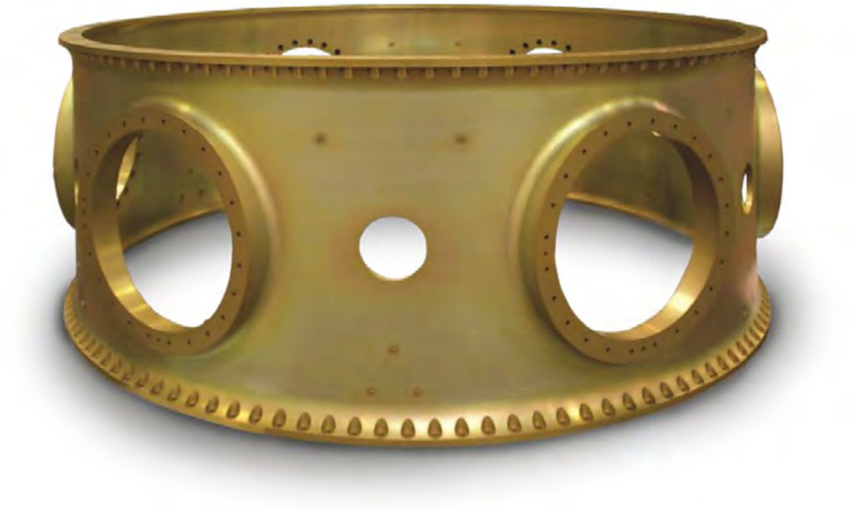


Figure 4.1: The EELV Secondary Payload Adapter, manufactured by MOOG Space and Defense Group. Credit: MOOG Space and Defense Group.

In summary, the nanoprobe would have been a completely independent mission of similar size and mass to the Deep Space 2 microprobes. The point of departure, however, was the complete EDL system. Utilizing COTS and GOTS equipment and advanced additive manufacturing capabilities, the Pathfinder mission was effectively replicated on a 1U CubeSat scale. Figure 4.3 shows this in detail. There are six Estes ammonium perchlorate composite propellant (APCP) retrorockets to slow the entire stack down to zero velocity near the ground. Sandwiched in between each motor are three Crosman carbon dioxide gas cartridges, which provide controllability during the terminal atmospheric entry phase. These details are shown in Figure 4.2.

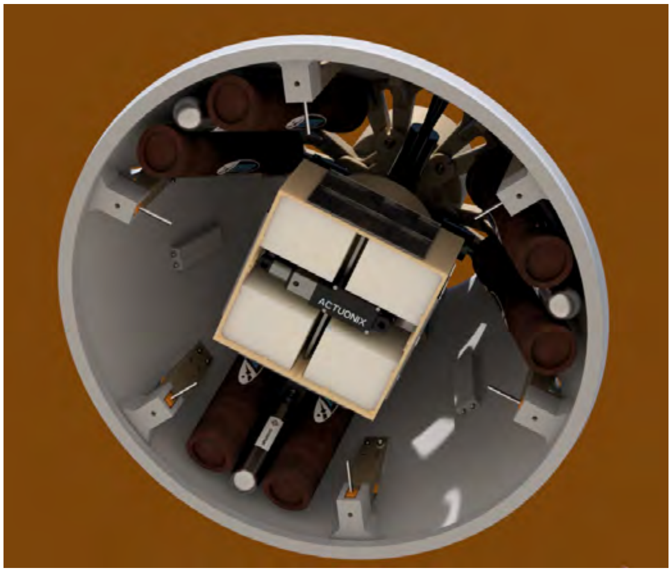


Figure 4.2: Internal render of Nines as an independent, 1U CubeSat mission of opportunity. Promenently shown is the 1U CubeSat, six solid rocket retropropulsion motors, and three gas thruster cartridges.

Since the EDL concept of operations is exactly the same as Pathfinder, Figure 4.4 is accurately representative of this first version of Nines. A hyperbolic entry is conducted, followed by parachute deployment at approximately Mach 1.6 - just before the vehicle becomes unstable. Upon slowing to approximately Mach 0.5, the heatshield is let go, the tetrahedral CubeSat deployer rappels down a 10-meter Zylon braided cable, and the radar begins to seek a solution for retrorocket firing time.

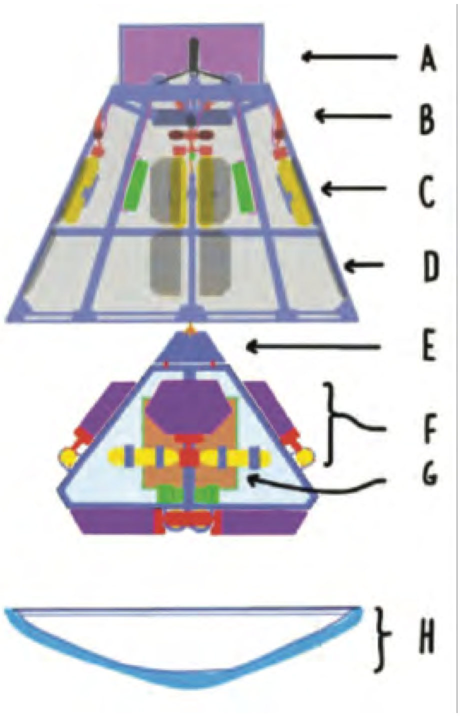


Figure 4.3: 2D drawing of the full Nines concept. The backshell (A-D) contains propellant (C), batteries (D), the parachute (A), and high speed deceleration devices (B). The tetrahedron (E-G) contains the actual lander hardware (G) and self-righting lander petals (E), as well as airbags and airbag inflators (F). The heatshield (H) is at the bottom, which would be 3D printed using ablator and PAEK-based thermoplastics.

Seconds before engine ignition, the CO_2 airbags inflate in about 1 second. At about 50-meters above the ground, the radar commands the rockets to fire, bringing the entire package to a stop 15 to 20-meters above the ground. At this point, the Zylon bridle is cut, the backshell flies into the parachute, and the CubeSat deployer bounces off the ground until it comes to rest. In a compressed timeline compared to Pathfinder, the tetrahedron deploys the CubeSat within 2-3 minutes of commanding the airbags to deflate. This is because of very low thermal inertia: warming up the high-torque axial petal deployer motors would eat into Sol 0/1 power margins. The lander would take advantage of preheating operation in the hours before entry. Therefore, the petals and conformal solar panels would be deployed quickly after landing.

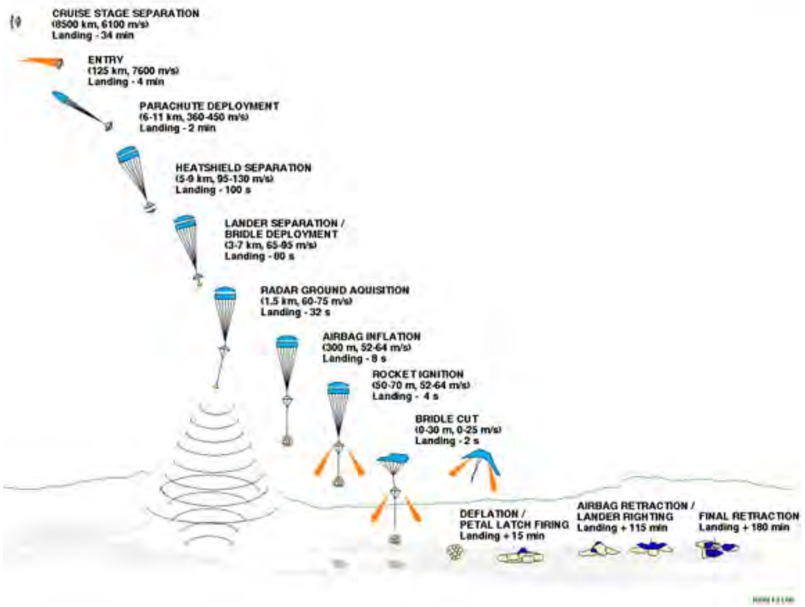


Figure 4.4: Concept of operations of Pathfinder rover mission. It is an accurate representation of a free-flyer version of Nines due to experiencing direct entry and due to the architecture being almost exactly the same. Credit: NASA JPL/Caltech/Lockheed Space Systems

Upon reaching the deployed configuration, the lander goes into a minimal power state to charge the battery. The major constraint for landings are that they need to occur in the local early morning - survival with such small batteries and low thermal inertia would be difficult if not impossible.

Once the state of charge reaches acceptable levels, data transmission from EDL can begin. As customary with all Mars missions, the highest priority data is from heat shield performance, internal thermals, and trajectory reconstruction. Without this data, landing models cannot be refined and margins cannot be reduced confidently.

After about 2-3 sols and 4-6 passes from an MRO-class relay spacecraft, science data collection can begin. This period is continued for a minimum 90 sols, but was expected to last 1 Martian year. The largest survivability constraints were night survival and global dust storms of magnitudes as great as those in 1970 and 2018. Data collection and instrumentation selection was expected to be limited, and was one of the contributing factors to completely revising the design.

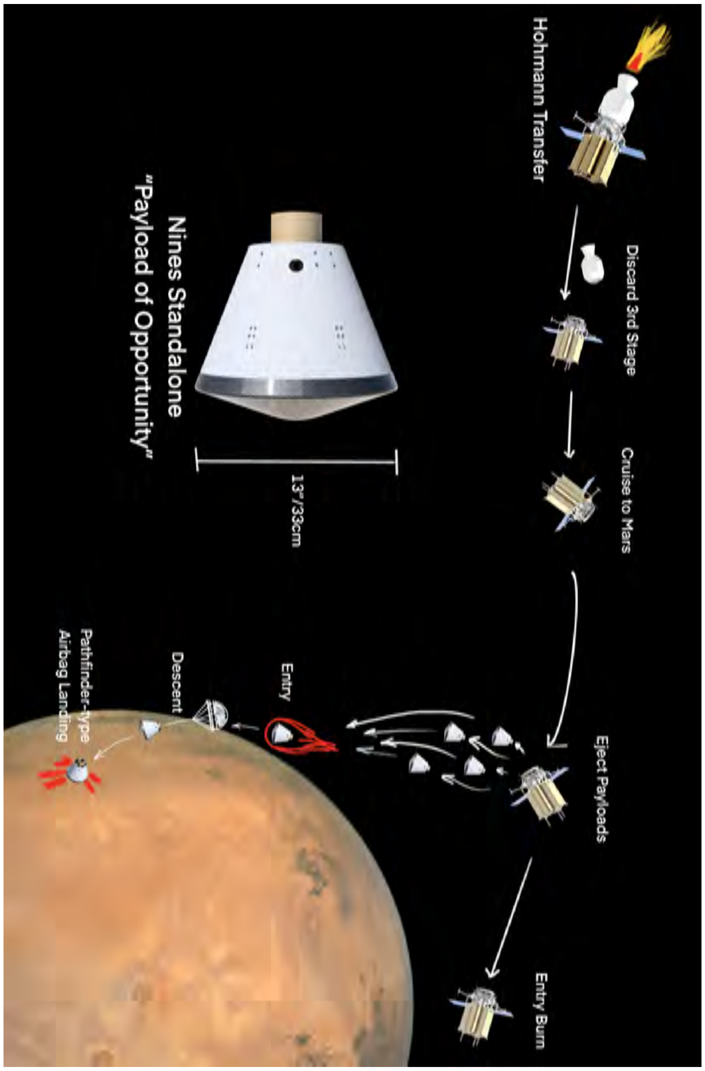


Figure 4.5: Concept of operations of the independent Nines mission, also known as a "payload of opportunity". Many of these would independently deploy from a mothership of an unrelated mission and land on the surface of Mars.

A pictoral representation of mission CONOPs is shown in Figure 4.5.

4.1.1 Architectural Revision Rationale

While very much a feasible and viable concept, the science returns would be extremely limited. Additionally, a top-level power budget analysis shows that the margins would be very limited.

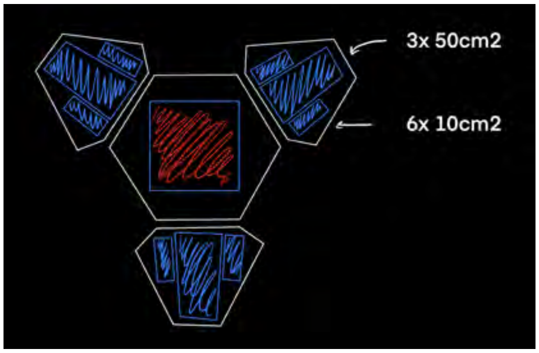


Figure 4.6: Total area in solar cell assumption for lander.

Figure 4.6 sets the stage for the analysis - there is $210cm^2$ of surface area to work with. SolAero IMM4J gallium arsenide (GaAs) solar cells used on the Ingenuity helicopter are 33% efficient, and are tuned specifically for maximum absorption on Mars.

The next step is to determine insolation per unit area per sol. According to data from multiple missions, the solar energy per unit area received is determined based on several variables - this is tabulated in 4.1. τ represents the optical depth ratio, where 0 is no atmospheric attenuation by dust; H_H is total energy received per square meter per sol, and GLI is average insolation received per sol, in watts/meter^2.

Assuming a conservative, but nominal scenario at a mid latitude, a τ of 0.65 is assumed. This results in a 290 W/m^2 peak at midday. Integrating the following formula approximating instantaneous insolation constrained by the average insolation per sol,

Table 4.1: Derived Insolation at Mars

τ	H_H **(W)**	**GLI** (W/m^2)
0.65	3340	290
0.4	3882	308
1.4	1900	178
3.25	1024	95

$$P_{TOTAL} = \int \mathbf{GLI} * sind(t + \frac{360}{24.6})dt \tag{4.1}$$

where P_{TOTAL} is total solar power received in 1 Martian day, **GLI** is average insolation received per sol, and t is time. This yields $\boxed{2.27kWh/m^2}$ per sol. No solar energy conversion system is perfect - multiplying by the efficiency ratio of GaAs solar cells of 33%, $\boxed{749.1Wh/m^2}$ represents a real-world, perfectly manufactured solar panel with no dust on the surface of the panels. With the total panel area in Figure 4.6 considered, the total power output per day, assuming no obstructions or shade on the panels, is $\boxed{15.7311Wh}$.

This result is the primary reason a design change had to be made. While by chance, NASA/JPL learned through the Ingenuity helicopter that almost all electronics are capable of surviving Martian winters with no electrical heating, the same would almost certainly not be said about scientific instruments. If the spacecraft were to use a nuclear heat source, such as a 1-watt thermal radioisotope heater unit (RHU), assuming that a fantastically thermally isolated design was feasible, the probe could possibly survive. This being said, nuclear power sources are not friendly with budgets or politics - replacing this 1-watt heat source with electrical heating would mean 76% of total energy per sol at landing would be consumed just to keep the probe, instrumentation, and power systems warm. Additionally, given that many RF communnications packages on the surface of Mars require power on the order of tens of watts during transmission, this further excludes this design from the trade space.

4.2 Architecture, Version 2

The next version of the system diverged from dedicated assets to much smaller "payloads of opportunity". It leaned on the Ingenuity-type model, where a primary payload like a rover or helicopter would drop off each nanoprobe somewhere on the surface of Mars. Figure 4.7 shows, with some changes needed for ruggedization, what each probe would have looked like at one to one scale.

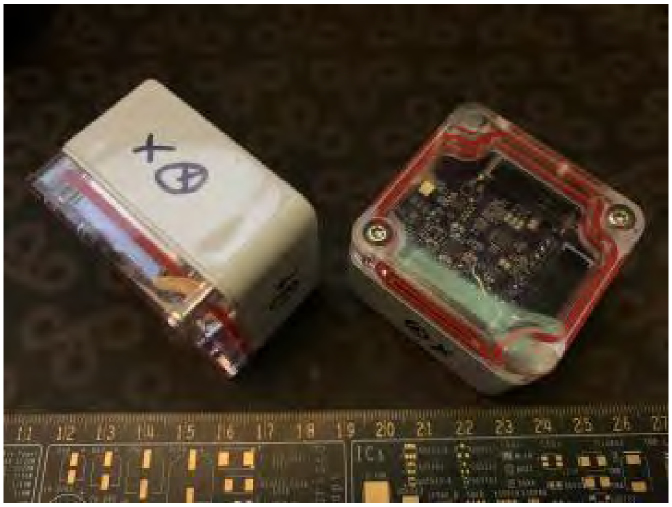


Figure 4.7: Nines probes, version 2, with ruler for scale. The polycarbonate boxes held a complete Internet of Things node inside, fabricated on two stacked PCBs.

After dropping off each node at various places in close proximity to an operating rover or helicopter, the nodes would gather and store data whenever there was enough power to do so. This was under the assumption that all electronics, batteries, and instruments would tolerate very low night time temperatures. Assuming that the rover or helicopter was within range of its' radio, each node would transmit the cached data, which would occur whenever the opportunity presented itself. An illustration of this conops is shown in Figure 4.8.

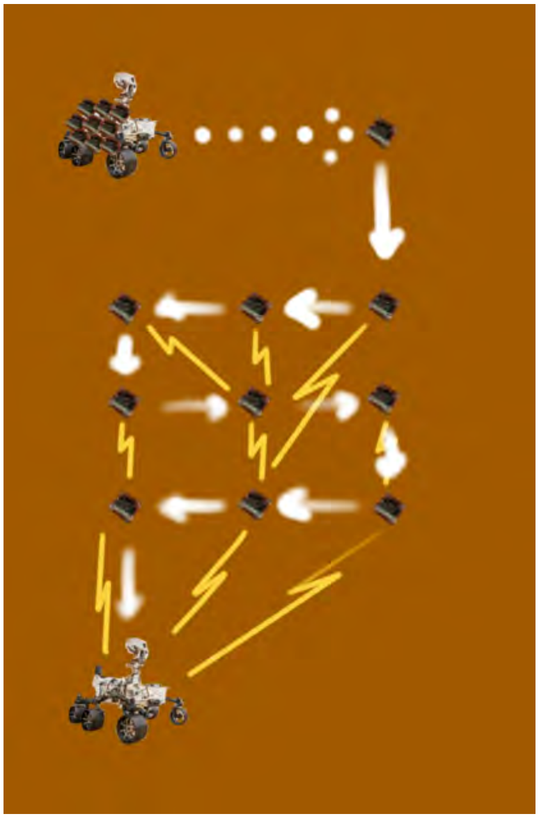


Figure 4.8: Concept of operations of Nines nodes being dropped off by a rover, one at a time. After all nanoprobes are deployed and activated, each communicate in a mesh network to eventually move the data to the rover when in range.

While a good exercise in IoT design, Nines as-is in this form factor has severe limitations. Aside from an opportunistic deployment scenario, data collection intervals would be inconsistent. Additionally, the solar power system was designed with minimal margin for collection on Earth - a complete recharge would require two to three days of sun exposure. From 4.1, with 25% efficient solar cells as used in the design, the probe's 5.7 cm^2 of cells would only generate 1.30 watt-hours per day. While sufficient for Earth testing, it would be completely insufficient for Mars especially when survival heaters are considered and survival every night would amount to gambling.

4.3 Architecture, Version 3

The next design iteration widened the scope to allow for more volume for better electrical systems. For this version, a new geometric constraints were given: the aeroshell diameter was to be no greater than 940 mm, a 70-degree half angle heat shield would be used, and the backshell would be no taller than 1-meter. By its' nature, the spacecraft system departed from the possibility for Wavefront to be a "rideshare" or "payload of opportunity" mission. This revision firmed up the lander-rover concept, and opted for a stackable cruise stage ring as shown in 4.9. The cruise stage was not considered; however, would have looked similar to the final iterations of Wavefront. The design would have allowed for larger lander systems and potentially more probes; but in the interest of maintaining flexibility for other launch vehicles and configurations, the outer mole line of the aeroshell was maintained at 940mm. It would have also endured the lightest loads, since the final revision would encounter significant cantilevered scenarios. By having a stacking semi-independent cruise stage with propellant, all probes would have allowed release days to weeks before entry, permitting high landing flexibility at the expense of hyperbolic entry speeds.

4.4 Version 4 and Shinra RTG

One of the primary reasons nuclear spacecraft were landed on Mars was to have guaranteed power in all situations, especially during dust storms. While rovers like Spirit and Opportunity were not designed to last through dust storms, it showed it was possible to survive dust storms with very high τ. After accepting that the mission cost could no longer even be a NASA Discovery-class mission, a miniaturized RTG was studied. The Shinra RTG is shown in Figures 4.10 and 4.11. The main

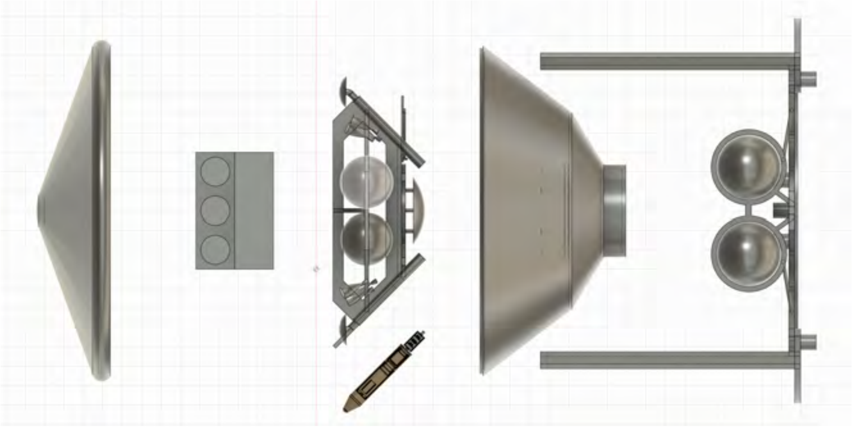


Figure 4.9: Notional, initial design for the Wavefront mission system. From left to right: 70 deg half-angle instrumented heatshield, 18U CubeSat form factor rover, ultra-light lander with base station and unrollable solar panels, Nines nanoprobes, instrumented backshell, cruise stage with stacking struts.

requirement was to design around the GPHS, as shown in Figure 1.25. This was critical to keeping development costs of a new RTG down, since requialification of nuclear fuel though required destructive testing due to ablation and impact could amount to U.S.$100 million.

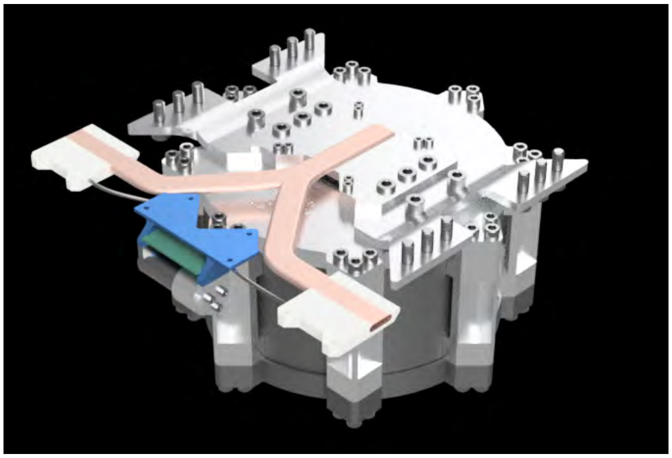


Figure 4.10: Shinra Radioisotope Thermoelectric Generator for nuclear lander option, which was ultimately not selected.

The minutiae of design choices and requirements were bypassed by creating

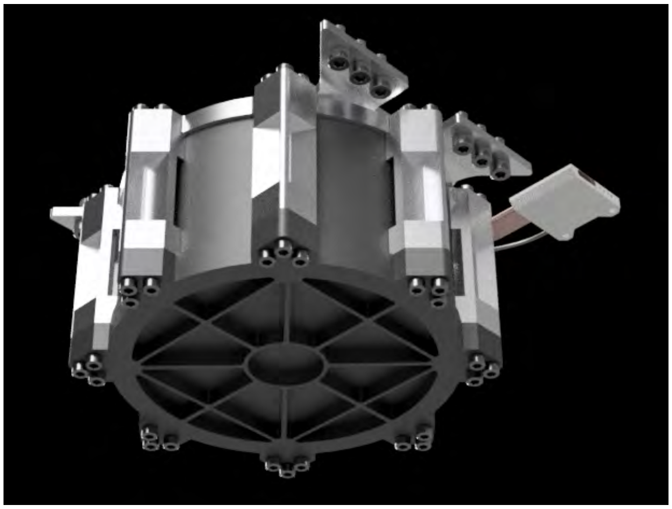


Figure 4.11: Shinra Radioisotope Thermoelectric Generator for nuclear lander option, which was ultimately not selected.

very large mechanical design margins. This, in turn, created a likely overdesigned and excessively heavy RTG. After fully modeling the correct material densities for all fasteners, insulators, and shock-absorbing material, the entire RTG was 9.4 kg and took up an excessive amount of volume in the backshell. The overdesign had fatal ramifications in the systems engineering design space, since the physical size would leave no more room for any vehicle electronics and would potentially very negatively affect stability due to the high mass at a high vehicle CG. When comparing Figure 4.12 with the final render of the Midgar lander, it is painfully apparent that there would simply not be enough room. If the Wavefront program were to go ahead with an RTG, several revisions of RTGs would be required. This was necessary to reduce the design margins to an acceptable balance of safety, power output, and structural rigidity.

Had the design been viable, Shinra would have produced 250W thermal and 12.5W electrical BOL (beginning of life) with the most efficient thermocouple junctions. Despite the low electrical output power, no electrical load would be needed

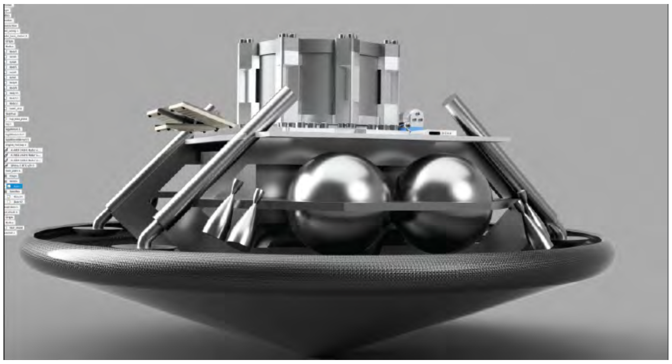


Figure 4.12: Shinra RTG mounted on lander, showing the extensive amount of volume it occupies.

for heat management or survival on the Martian surface, since waste heat would radiate throughout the spacecraft. To supply peak load demands past Shinra's capabilities, the system would have charged a battery whenever scientific measurements or transmissions were not being conducted.

4.5 Version 5 and 6 - New Space LSP Version

To keep the possibility open for "New Space" launch service providers (LSPs), a descoped version was considered, with a smaller version of Aerith (version numbering taken from the first version of Nines) as a free-flyer cruise stage. The nominal concept is shown in Figure 4.13. As of writing, the only viable New Space LSP aside from SpaceX is Firefly Aerospace's Alpha LV. A vehicle that can fit snugle inside the Alpha payload inner mold line (IML) is shown in Figure 14, which comprises the smaller Aerith cruise stage and relay, and a single Wavefront lander package. Unfortunately, according to the Alpha User's Guide, the Star 37D third stage plus Wavefront mission was 960 kg, barely under the maximum mass Firefly is able to launch to low-Earth orbit. Additionally, assuming a total impulse of 1.8 MN*s, only 784 m/s ΔV would be achieved - only 23.8% of needed Earth escape velocity.

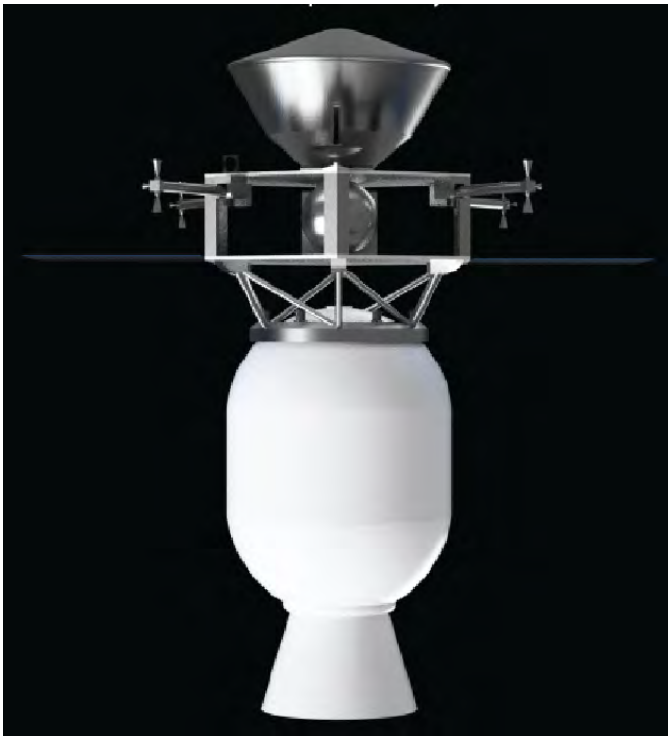


Figure 4.13: Initial revision of a free-flyer version of Wavefront. The white engine is a Star 37D solid rocket kick motor.

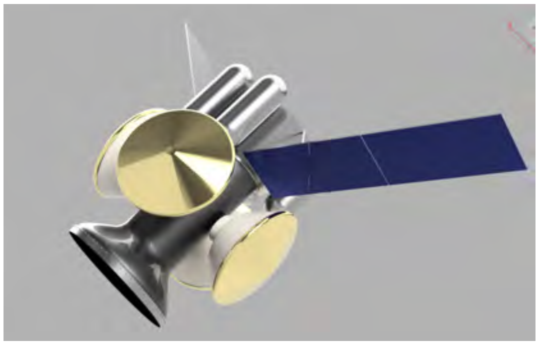


Figure 4.14: Notional, complete system design with orbital relay. Four nanoprobe landers surround the center tube, which serves as both a mounting point for all nanoprobes and relay, and extra internal space for communication equipment, including a high-gain antenna. Four external tanks surround a hexagonal prism-shaped main spacecraft body of the relay. All tanks will have a hypervelocity Whipple shield to mitigate catastrophic damage to the spacecraft during the orbital phase.

Another version was investigated with the EDL aeroshell fixed in design, which sported a composite-wound, single-piece tube, shown in Figure 4.14. The flared portion, which would provide a means for mechanical fastening to the rocket 2nd stage, doubled as a Ka-band antenna dish and volume for a laser communications system. This design was specific to a Firefly Alpha LV, but given the need for a third stage to perform a transfer burn to Mars, and the lack of volume (this design maximized the LV IML), the design was seen as a non-starter. Design elements, however, such as the single-piece composite tube, survived to the final iteration.

4.6 Version 7 - Final Version

After many top-level configuration revisions over 1 1/2 years, the design space finally converged with a 12-lander mission. The complete mission is shown in Figure 4.15. This design first set a budget for US$3 to 5 billion over the life of a NASA Flagship program, and took into consideration such activities as management and real-time data analysis and processing. These details were covered in the previous chapters. The next step was to select a viable launch vehicle that would be generously flexible in terms of payload volume and mass to a Martian transfer orbit.

4.7 Complete Concept of Operations

One of the only vehicles that could support these requirements with on-time delivery and reliative flexibility is the SpaceX Falcon 9 Block 5 (F9). The complete Wavefront mission as mounted in an F9 is shown in Figure 4.16. In the F9 User's Guide, a Mars transfer mass of 4 metric tons (Mt) is specified. To allow for vehicle and spacecraft margins, 3 Mt is given as the upper limit. With a 95 kg lander mass, all landers added up to 1.14 Mt, leaving relatively small margins for Aerith. To

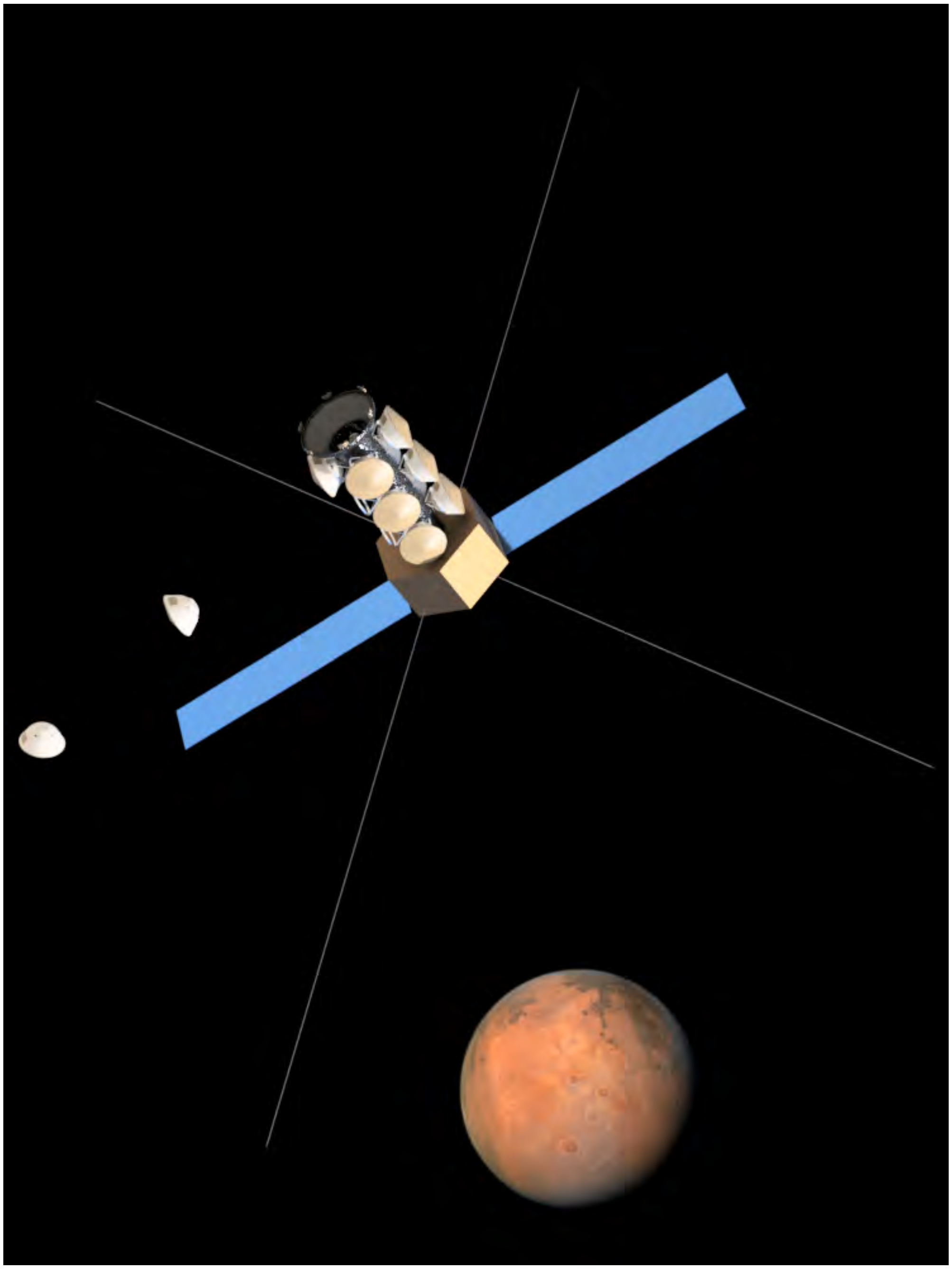


Figure 4.15: Aerith releasing two Midgar landers at apoapsis. By releasing the landers at apoapsis, fine-tuning the landing site by the lander's limited fuel supply is maximized.

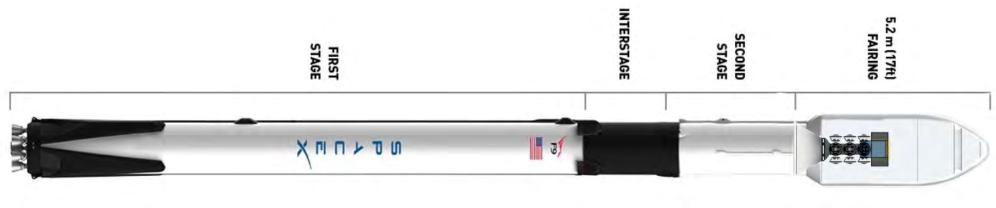


Figure 4.16: An illustration of SpaceX's Falcon 9 with a to-scale representation of Wavefront seated in the payload fairing. Background image credit: SpaceX.

increase these margins, very stiff deployable solar wings were incorporated, in order to allow for aerodynamic maneuvers.

A hybrid aerocapture and retropropulsion maneuver would be used to reduce fuel needed to bring the mission to an orbit with $0.9 < e < 0.98$. In this state, landers are released at the apoapsis - they use their own fuel and power to conduct a retrograde burn to lower the periapsis to come in for a landing. Once all probes are released, Aerith conducts an aggressive aerobraking campaign to bring the orbit down to a 500x500 km sun-synchronous. This portion of the operations are shown in Figure 4.17. SSO allows for minimum mission downtime when functioning as an orbital relay with a high power radio. In between relay campaigns, Aerith would conduct Mars observations and site surveys of potential human landing sites, and use its HF radar to produce a map of subsurface ice or brine.

On Mars, all 168 assets including backshell and heatshield would transmit scientific data. Each group lands as a complete unit on the Midgar lander. After a safe landing in a particular orientation, Midgar's solar panel is unrolled and locked at an angle that allows dust to roll off the solder wing. The Pascal rover is unlocked and untethered, and drives out from under the lander to unroll its own solar wing. Once Pascal is charged adequately and the robotic arm is checked out, the Midgar lander swivels the tray holding the Nines nanoprobes 45 deg so Pascal can grapple them one

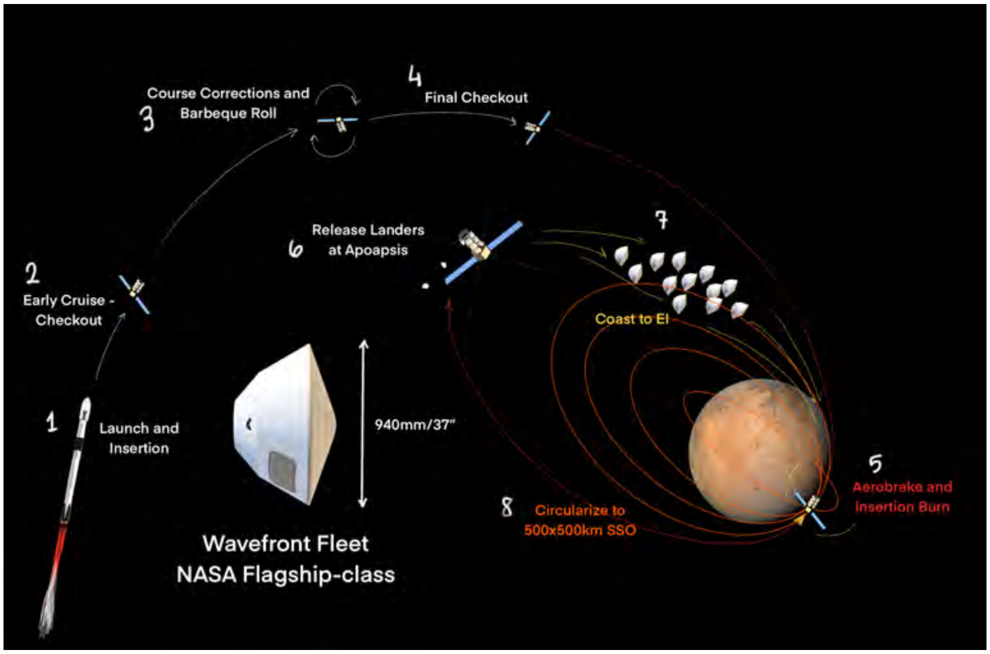


Figure 4.17: Concept of Operations for Wavefront, part 1. (1) Wavefront is launched and inserted into a Hohmann transfer by a SpaceX Falcon 9 Block 5. (2) Cruise operations begin after a complete system checkout on all hardware pieces. Lost assets are accounted for and when possible, a root-cause analysis is conducted. (3) Most of the cruise time will be spent in a "barbeque roll" spin-stabilized mode, evenly distributing heat throughout all components of the spacecraft. As-needed course corrections are performed. (4) Before the insertion burn, functionality is checked once again on all hardware. (5) A deep aeropass maneuver in tandem with an insertion burn is performed. The solar panels are retracted to move the center of pressure behind the center of mass to reduce RCS firing. (6) At multiple apoapses, two to three Midgar landers are released from Aerith in order to position all assets. (7) All probes coast to the entry interface and enter the atmosphere. The CONOPS for this portion continues in the next frame. (8) To support the missions with as minimal downtime as possible, an aggressive aerobraking campaign is carried out. The campaign ends when the orbital parameters reach a 500x500 km sun-synchronous orbit with an LTAN of 6 PM.

at a time. At predetermined locations, Pascal takes each Nines probe to the site and uses a percussion end effector to hammer the probe into the ground. At the final depth, a pin is pressed inside the stainless steel rod on Nines by the robotic arm, unlocking the solar panels and powering on the probe. Data is received by Midgar, relayed to Aerith, and received on Earth for confirmation that the nanoprobe is functioning. This process is repeated 12 more times, until all nanoprobes are in the ground and the Midgar VLF transmitter is hammered into the ground. Because time is of the essence, all 12 lander groups would conduct this simultaneously, necessitating a high degree of autonomy to reduce mission control demands during this period. Figures 4.18 and 4.19 provide an illustration of this process.

In orbit, Aerith collects data from all Midgar landers during a twice-daily overflight window of 15 minutes. At an average effective bit rate of 4 Mbps, 900 MB may be collected per pass. This is a conservative estimate; adaptive bit rates can substantially increase data throughput.

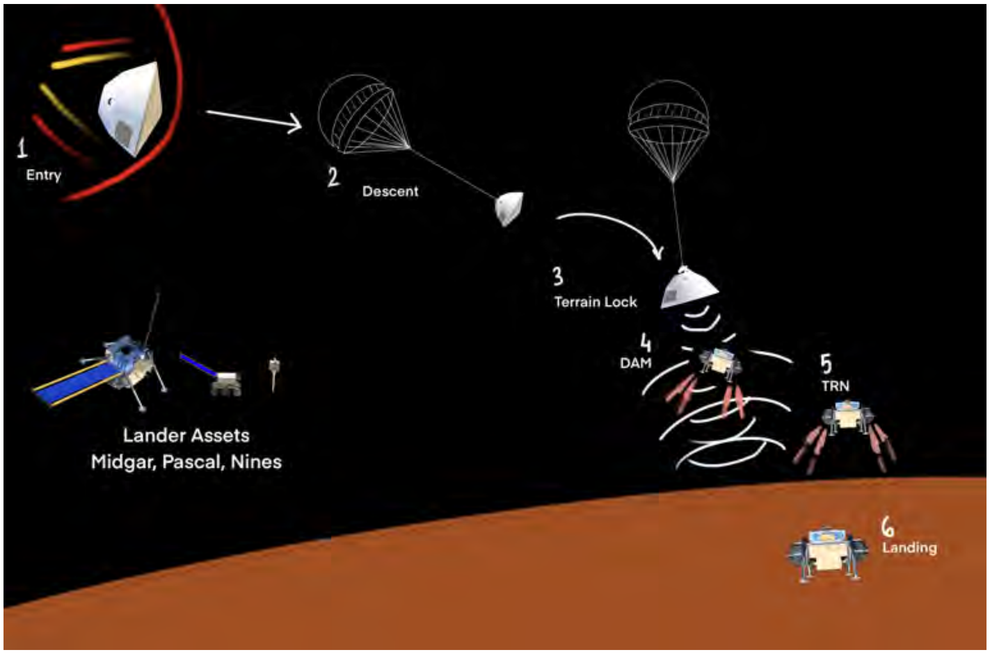


Figure 4.18: (1) A much more mundane 4.1 km/s entry is performed, due to having no hyperbolic excess velocity with respect to the Mars frame. (2) Upon reaching about Mach 1.6, the supersonic disc-gap-band parachute is deployed. It stays on for the majority of the descent. Once the system reaches terminal velocity, the heatshield is released and (3) a terrain solution is generated. This solution is used to time the release and firing of the landing engines. (4) At about 1 kilometer above the surface, the lander is released from the backshell, the engines fire, and a debris avoidance maneuver (DAM) is performed to steer clear of the parachute and backshell. (5) Terrain relative navigation (TRN) is engaged as soon as the lander vision dynamics system (LVDS) locks onto the ground, and quickly identifies as safe of a landing site as possible given its fuel margins. Midgar has very little fuel reserves for higher altitude destinations, so LVDS must use a multitude of remote sensing techniques to identify even the smallest zone of safety. (6) Within 5 to 7 minutes, Midgar comes in for a landing. For the fleet of landers, this is repeated 11 more times.

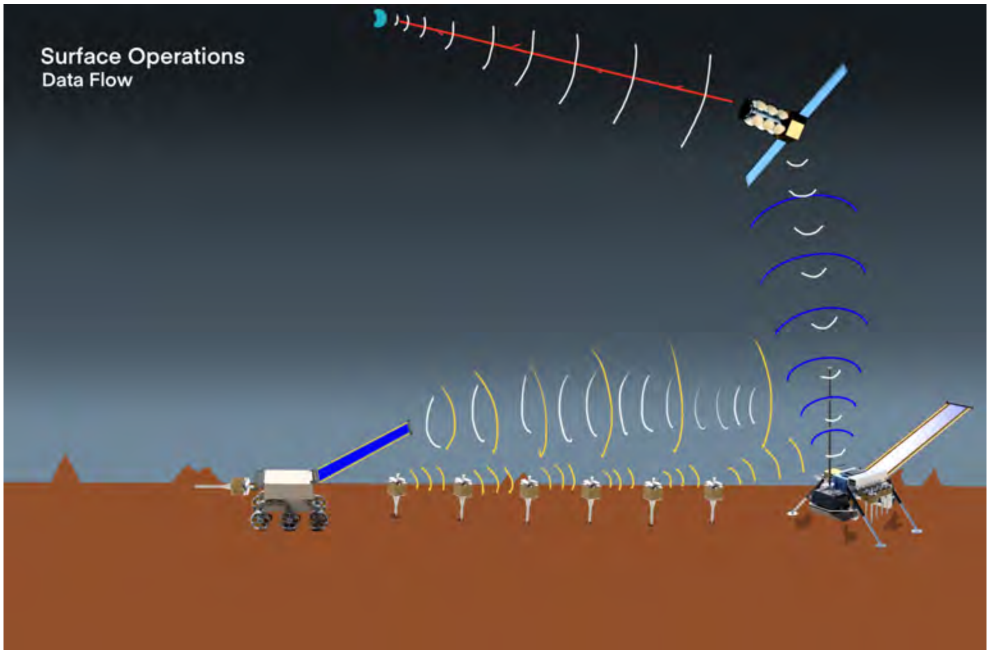


Figure 4.19: From bottom left, counterclockwise: Pascal is shown carrying a Nines nanoprobe with its robotic arm. The robotic arm has an impact driver attachment to drive the nanoprobe into the ground. To reduce vibration associated with impact driving, the electronics and instruments are mechanically stood off from the 316L stainless steel spike. Moving counterclockwise, each Nines probe is illustrated as driven in the ground at regular intervals. To analyze regolith impedence and develop a tomographic map of the subsurface, the probes are driven in these intervals. In the lower right corner, the Midgar lander acts as a relay between Nines probes, Pascal, and Aerith, whenever it is in view. While relay operations occur, the tomography transmitter emits RF into the ground via two spikes, and the weather station takes high resolution data. Twice a day, Midgar transmits up to 900 MB per pass to Aerith, which then forwards this data to Earth via optical communications.

5. Nanoprobe Design, Development, and Testing

This chapter details the development of compact, field-tested remote sensing units, considered precursors to the spike-mounted nanoprobes deployed by the Pascal rover. Three versions of nodes are built, spanning three years of development.

5.1 Version 0 - SEEDS-A

SEEDS-A, or Space Extreme Environment Detection System - Atacama, was a project formed in June 2019 as a collaboration between NASA, SETI, and San Jose State University. The objective of SEEDS-A was to deploy an array of sensors to the top of Cerro Simba, a 20,500-foot (6.24-km) tall stratovolcano, in order to perform follow-up observations to anomalously high UV indices [112]. The top of the volcano has a high altitude lake, which was also a science focus as an analog to the late Hesperian period of Mars. Monitoring extreme UV at the top of this volcano would have allowed future astrobiologists to test theories of how, if any, extant life on Mars would have begun to adapt to Mars drying out [113]. The full design, test, integration, validation, and deployment sequence was compressed into 5 months. Ultimately, the system was sent but never deployed due to geopolitical factors beyond the team's control [114].

5.1.1 Work Breakdown

Due to the small team size, a WBS for the team was very small. The SEEDS team was given only 90 days to design, develop, demonstrate, and deliver the product, so development work needed to commence immediately. The team was broken down into three main groups: the undergraduate senior design team, the graduate design team, and the administrative team. The undergraduate team focused on development of the sensor units and instrumentation - a chiefly software and electrical discipline; the graduate team worked on mechanical design, custom PCB design, concept of operations, and deployment procedures of the sensors at the top of the volcano; and the administration team coordinated requirements, purchases, and deliverable timelines. The deployment procedures, as written in 2019, is listed in Appendix 8.

5.1.2 Product Breakdown

Figure 5.1 shows the high complexity of the SEEDS system. Although the PBS was for the follow-on project, it was focused on higher data throughputs and a smaller, more optimized design, and still served to demonstrate the work required to turn the project around.

5.2 Revision 1

The earliest test of an encapsulated system occurred in mid-2020, and involved soldering several, off-the-shelf sensors to another off-the-shelf Cortex M0 processor evaluation board. 13 test campaigns were carried out, the final one ending when rainwater seeped in through a hole in the seal.

Figure 5.9 shows the first revision of the nanoprobe in the test environment before being left outside until the battery was depleted. Several more revisions of this system were made until the firmware and concept were firmed up.

5.3 Revision 2

Figure 5.10 shows two identical nodes, which were designed around the constraint of a PocketQube-like form factor container. It is the product of several design iterations mostly involving power optimization strategies.

Figures 5.11, 5.12, and 5.13 are images of the internal electronic hardware. All these images are telling of how little room there was to work with.

5.4 Gateway Electronics

The second component of IoT-type sensing and networking is developing a gateway, which relays data from one network to another. A substantial effort was made, for at least one year, to develop the hardware and custom scheduler to cache and forward data to the Internet. Figure 5.14 shows the complete, prototyped gateway.

On the left of the image, there are six antennae - three rubber duckys are for the 915 MHz ISM band, the long, skinny vertical antenna is for 2.4 GHz WiFi, an aluminum substrate-mounted patch antenna is for GPS, and the isolated black, substrate patch antenna is for L-band Iridium. This section forms the RF subsystem. Note that the GPS antenna and Iridium antenna are as far away as possible - this is due to their very close adjacent frequency bands. The Iridium modem can transmit in bursts with greater than 1W RF, which could easily overwhelm the extremely sensitive GPS frontend if not accounted for.

The next system is to the right - a small red board in the center of the breadboard. This is a precision, low bandwidth inclinometer from Murata. With a running average filter and other DSP, it can be repurposed as a strong-motion seismometer. The data output from this is a raw stream of floats in 3 axes at 1.6 kHz, which is fed to an intermediate processor characterizing the data output. If an event is detected as determined experimentally by listening for months for any local seismic activity, the peak ground acceleration (PGA) is converted into the Modified Mercalli

Index (MMI), used by the U.S. Geological Survey to provide a qualitative scale for ground motion. However, this complete subsystem was not tested due to a lack of time and other hardware priorities.

Directly below the inclinometer is an environmental sensor suite. Most prominent are the four green boards, which are screen-printed electrodes utilizing electrochemical sensing techniques. From left to right, these sensors pick up carbon monoxide, hydrogen sulfide, sulfur dioxide, and ozone at the parts per billion level. While not suitable for Mars, these sensors would deliver suitable science for volcanology - an extreme environment analog for space missions. In fact, the intention was to validate this entire system concept in proximity of a local volcano if time permitted. These sensors represented science-grade instrumentation that is a hard requirement of any interplanetary space mission. To the immediate right of the SPECs are consumer-grade and some military-grade weather and temperature sensors. These served to provide an engineering and scientific reference for the rest of the system. Data included several high-accuracy temperature sensors, atmospheric pressure, carbon dioxide concentration, and a small, 16-pixel spectrometer. This subsystem produced the most data, and was formatted as comma-separated values for readability and ease of debugging.

Above the weather and temperature sensors and to the right of the inclinometer is the debug interface. This included the parallel character display. There are a large number of firmware libraries available - development of this interface was fairly straightforward. However, debug statements used a lot of flash memory and greatly slowed execution of routines, so by the end of the development period, this interface was not used except during boot.

To the right of the parallel character display is a very complex rat's nest of wiring. This is the command and data handling system. It is centered around an

Arduino Due, which itself runs on an 84 MHz, 32-bit ARM(R) Cortex-M3. This board was chosen due to the large number of I/O lines broken out. This fact allowed usage of parallel static RAM from Panasonic, using a 500 nanometer (nm) feature size. While extremely outdated - the SRAM was from 1994 - the very large feature size has intrinsically high radiation tolerance. It allowed for experience to be gained in interfacing with potentially space-rated hardware. Four memory chips were used, totaling 16 Mbit of external RAM. Underneath the rat's nest is a precision real-time clock (RTC) that also doubles as a temperature-compensated, accurate frequency reference.

Finally, all the way on the right of the board is the optical and radar evaluation subsystem. A 32x24 thermal camera is evaluated for sun tracking and attitude knowledge, the 5 megapixel rolling shutter camera takes pictures for situational awareness and context, and the radar is to determine if there is motion in the immediate vicinity.

Given enough time and work-hours to complete, Figure 5.15 shows the flight-like production system for Midgar. It was to use several Raspberry Pi RP2040s with a fully custom PCB in a PC-104 form factor for maximum system compatibility. While the RP2040 is an extremely cheap processor, the large feature size (45 nm) was a driving factor in its usage, as larger feature sizes are more tolerant to radiation and single event upsets to a degree.

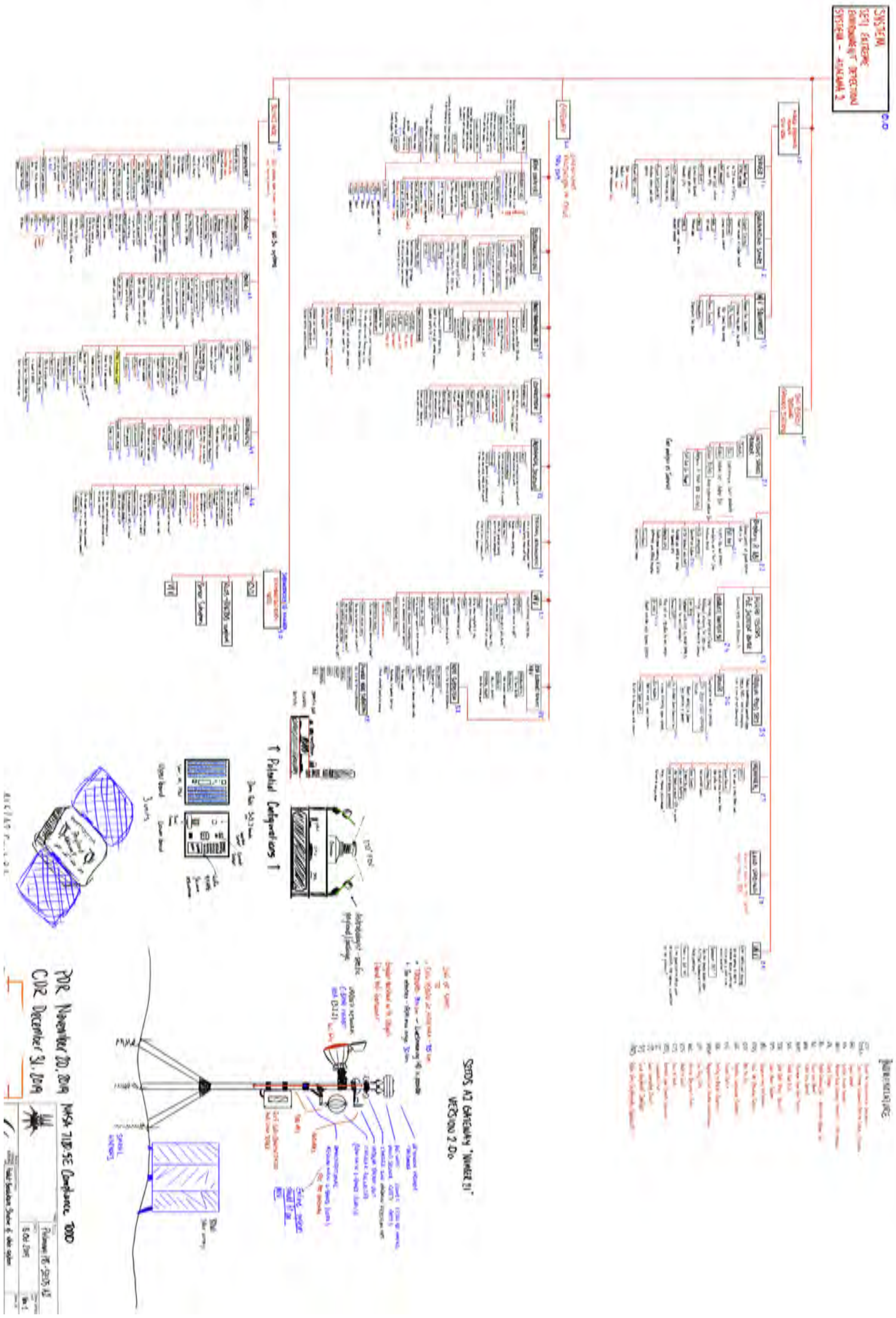


Figure 5.1: SEEDS-A version 2 master product breakdown structure of the gateway unit. This overview illustrates the very complex nature of systems engineering for even an Earth-analog mission for Mars. Drawings in the lower-right were the earliest version of the small environmental monitoring nodes. Credit: Stanley Krześniak, Kayla Parcero.

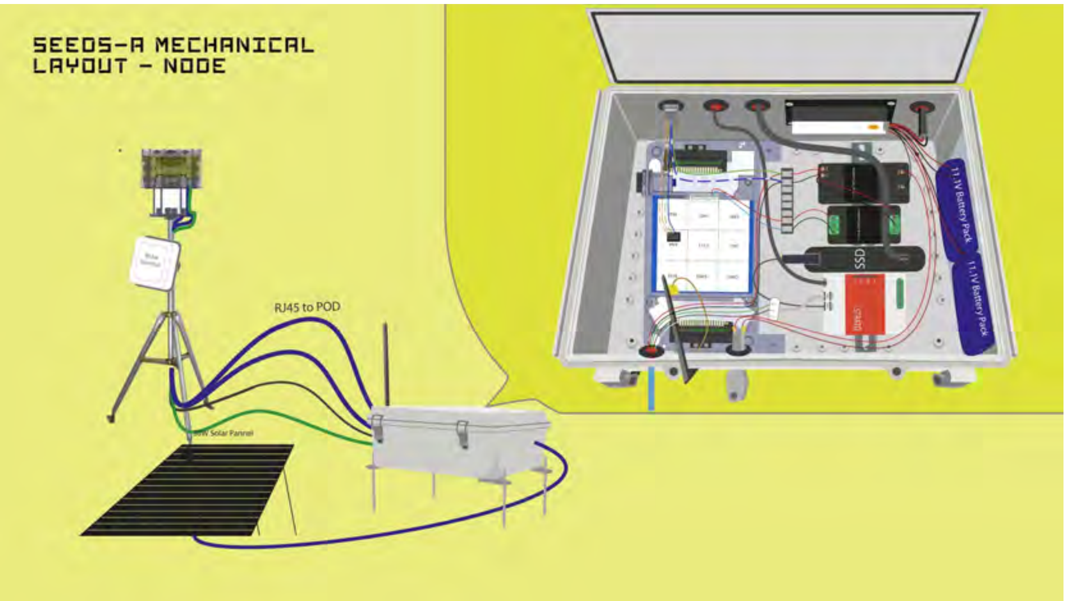


Figure 5.2: SEEDS-A mechanical configuration of *gateway*. The project used a different nomenclature that is incompatible with IoT terminology. A ground-mountable mast held the optical sensor head and Inmarsat(R) communications unit. A ground-staked polycarbonate enclosure held the electronics and mass storage. On the lower-left, a 30-watt monocrystalline solar panel powered the entire gateway. Credit: Christian Espinoza.

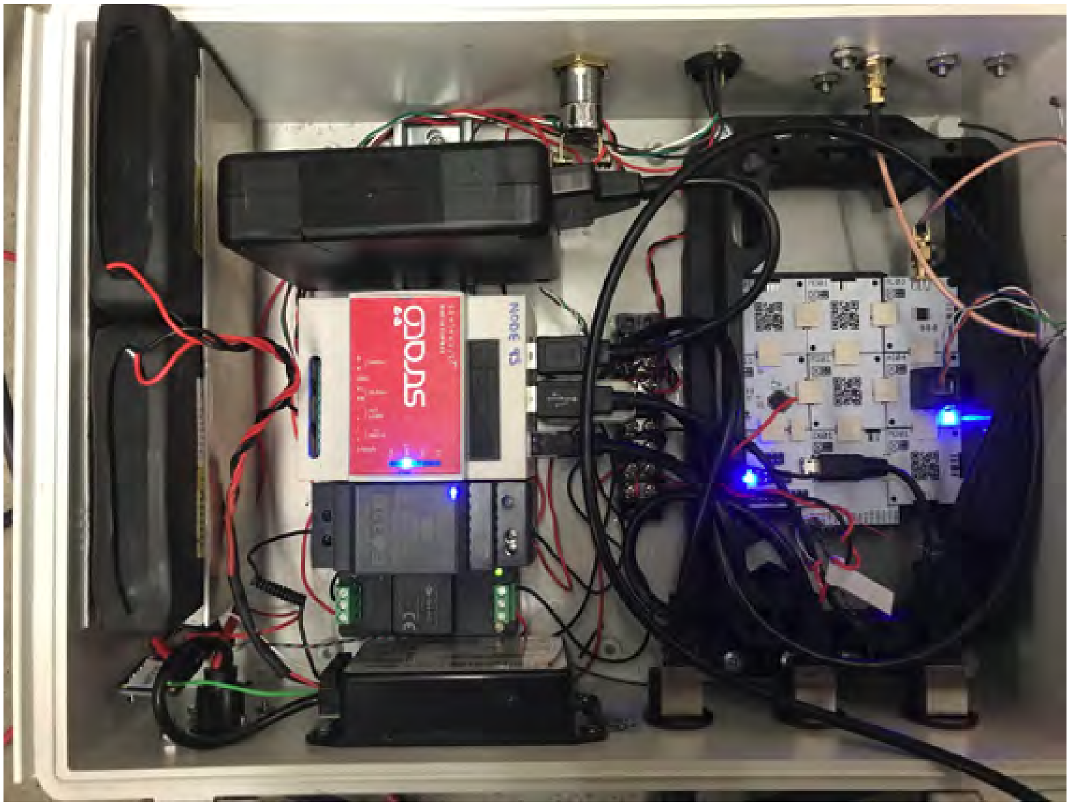


Figure 5.3: SEEDS-A interal mechanical configuration of *gateway*. From left to right: 3S2P, lithium-ion, 200 watt-hour; 128 GB SLC SSD; Raspberry Pi 3B+; 12V buck regulator; 3.3V buck regulator; 27V MPPT solar to battery charger; terminal block; modular central computer. Credit: Stanley Krześniak

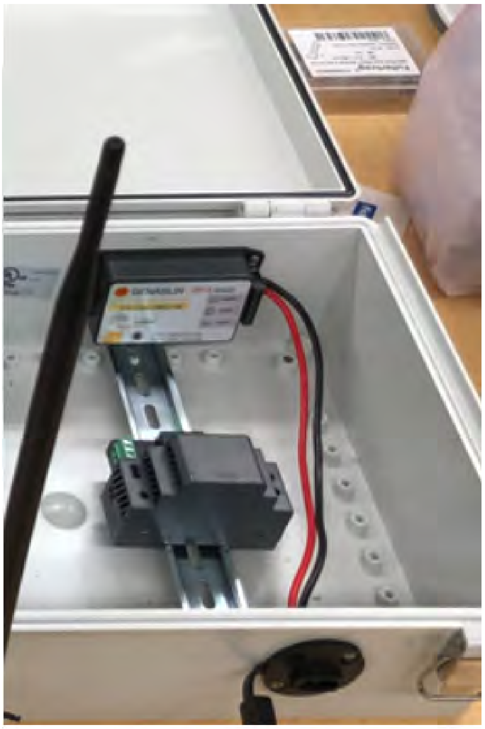


Figure 5.4: DIN rail mounting mechanism for power supplies and industrial Raspberry Pi. Credit: Stanley Krześniak

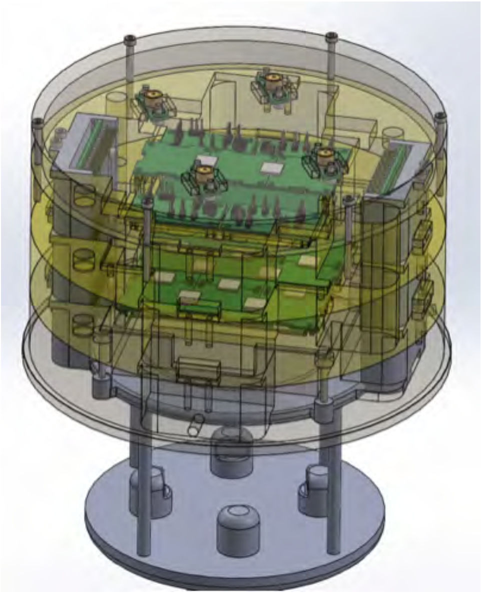


Figure 5.5: Sensor head CAD with internals shown. The optical sensor head was designed to be mounted on top of a Windsonic(R) ultrasonic wind sensor. Credit: Stanley Krześniak

Figure 5.6: Initial sensor head camera picture. Left: unfocused, right: focused. Credit: Afrah Siddiqi.

Figure 5.7: Completing a test of the Windsonic ultrasonic wind sensor with a standard room fan. Credit: Nataliya Grigoryan.

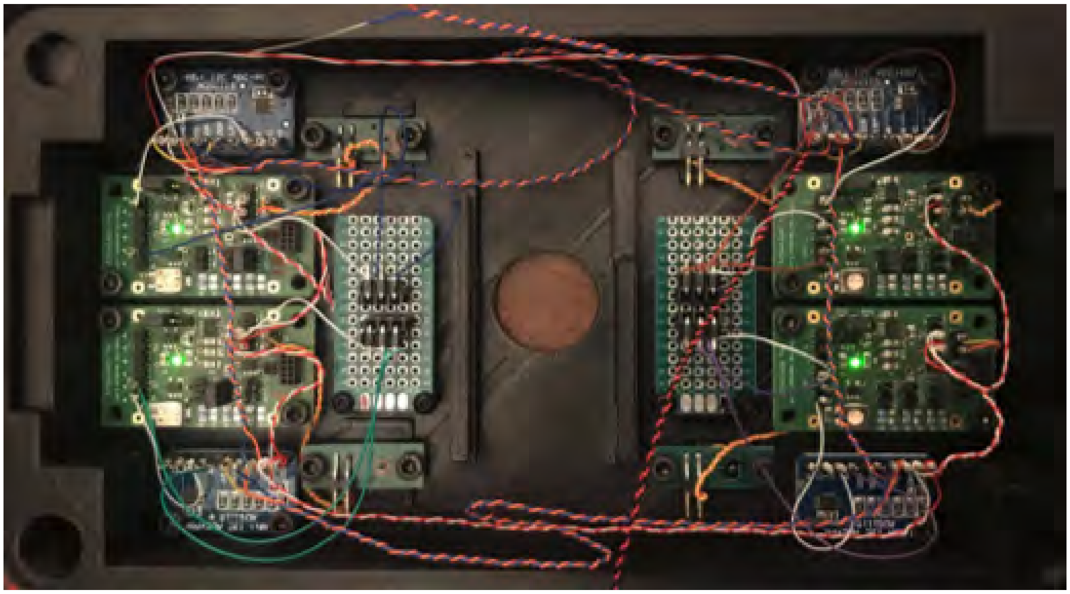


Figure 5.8: Completed assembly of SEEDS-A node sensor head with IFW(R) diodes, the type flight-certified and flown on both the Curiosity and Perseverance rovers in the Spanish Astrobiology Center's Rover Environmental Monitoring Station payload. Wires were in twisted pairs to reduce RF interference due to extremely sensitive analog readings from the photodiodes. To ensure universal compatibility with other systems, the entire sensor head was controlled and telemetered by I^{2}C. Credit: Stanley Krześniak.

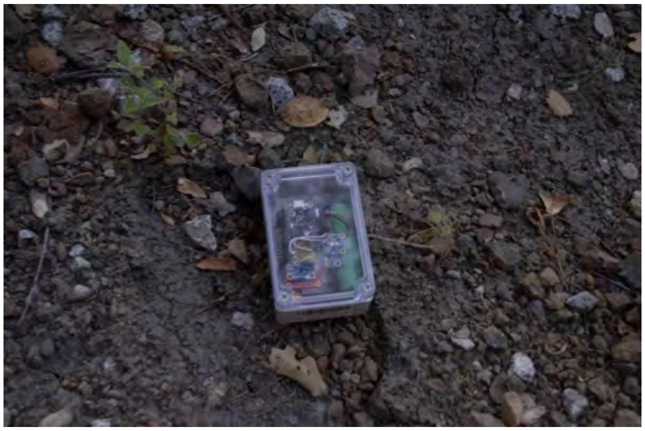


Figure 5.9: R2 node in a clear polycarbonate junction box in the test environment.

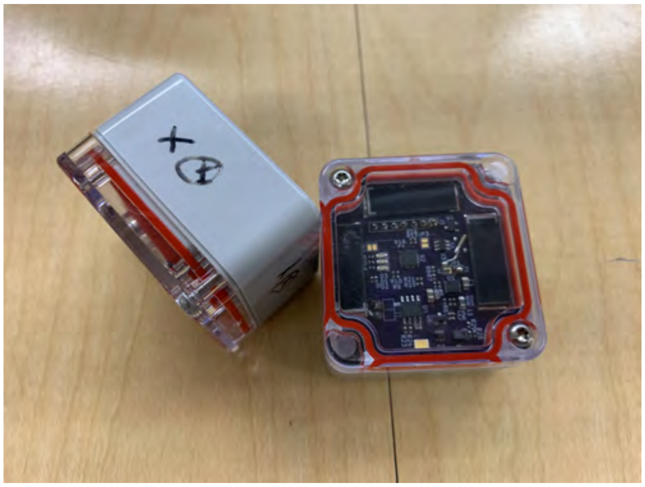


Figure 5.10: Two R3 Nines nodes. The containers are off-the-shelf and contain two custom PCBAs stacked internally.

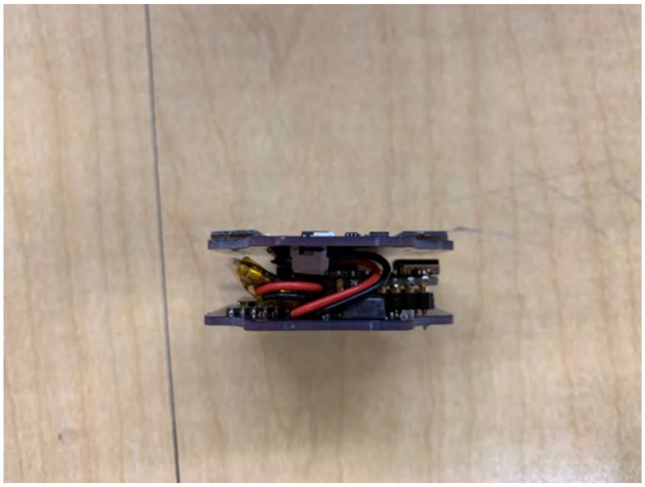


Figure 5.11: Stack removed from container. This image clearly illustrates the very limited volume to work with.

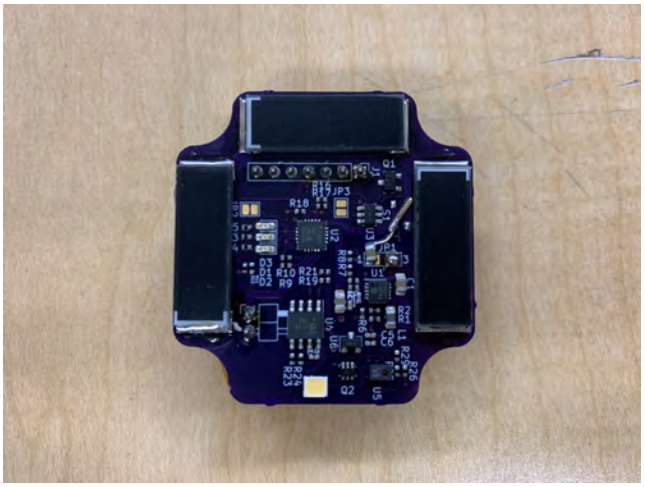


Figure 5.12: The upper stack contains the solar panels, power conditioning, and a light sensor.

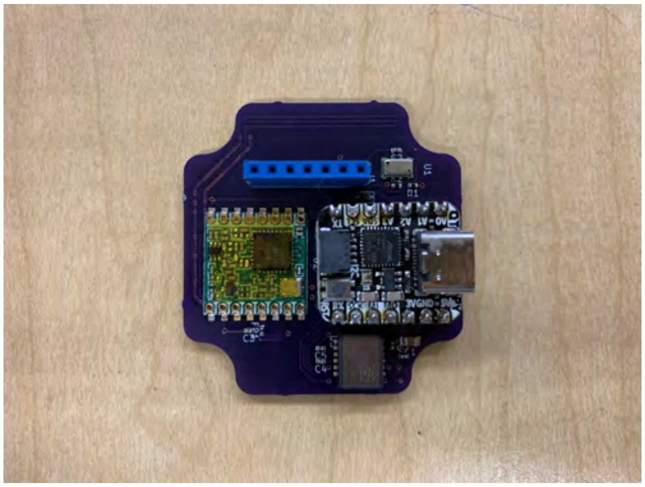


Figure 5.13: Lower stack. This contains a LoRa modem under Kapton tape, an off-the-shelf microcontroller, precision inclinometer, and pressure/temperature/humidity sensors. The blue header is a board-to-board connector, which contains 3.3V, I^{2}C data, and three GPIO control lines.

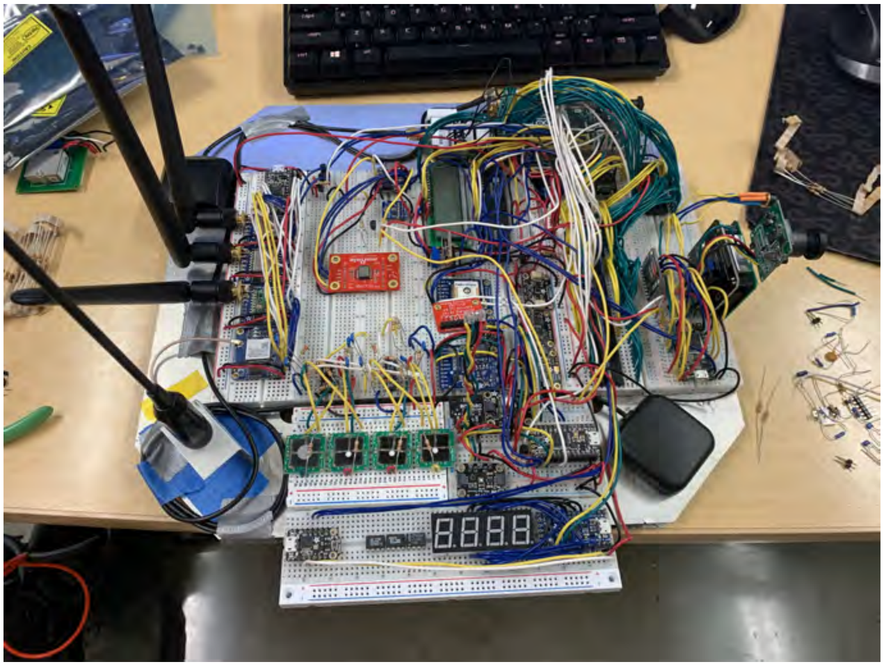


Figure 5.14: The complete gateway electronics mounted on an aluminum-2000 series honeycomb panel. The highly complex system is broken up into the following subsystems: command and data handling, mass storage, radar evaluation and machine vision, environmental and gas sensing, debug, seismometry, and communications.

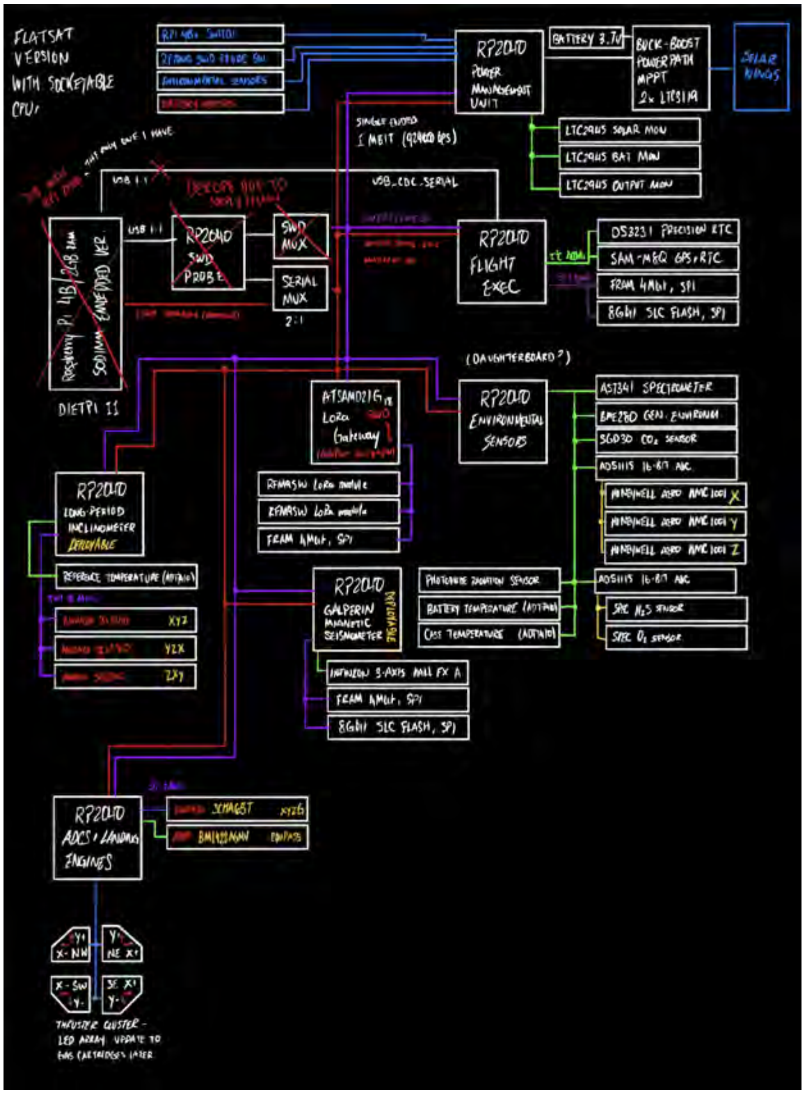


Figure 5.15: Wavefront R1 lander electronics. The prototype was meant to be breadboardable across several boards for prototyping. No propulsion system was to be a part of it; an LED quadrant simulates reaction control system thrusters.

6. Entry, Descent, and Landing System

The design of the entry, descent, and landing system (EDLS) is documented here. Supporting CFD and assumptions are given in this chapter for all stages of flight.

6.1 Assumptions and Initial Requirements

In the design of Wavefront, the most important design requirement made early in the process was a 1-meter aeroshell maximum diameter. This was to ensure a universal fit in a variety of launch vehicles regardless of configuration or launch provider. Late in the design process, the Falcon 9 was chosen as the launch vehicle for the configuration, allowing for a generously sized mothership and communications relay. By retaining the same 1-meter aeroshell requirement, up to 12 landers could be flown. As covered later, this still heavily constrained landed volume, fuel budgets for the Midgar lander after release from the backshell, and reduced the landable sites to areas below zero-meters MOLA.

An assumption of an aeroshell shape was the first specific design consideration. According to the literature, nearly all larger Martian lander probes, especially those from NASA, utilized a 70-degree half-angle heat shield profile. This maximized the internal volume and lowered the center of mass, critical for stability at hypersonic speeds.

Regardless, blunt-body stability is Mach-dependent. As the capsule slows, the center of pressure shifts forward, closer to the center of mass. At a critical Mach number, the center of pressure moves ahead of the center of mass, rendering the capsule an effective inverted pendulum. Figures 6.2 and 6.3 clearly show this from

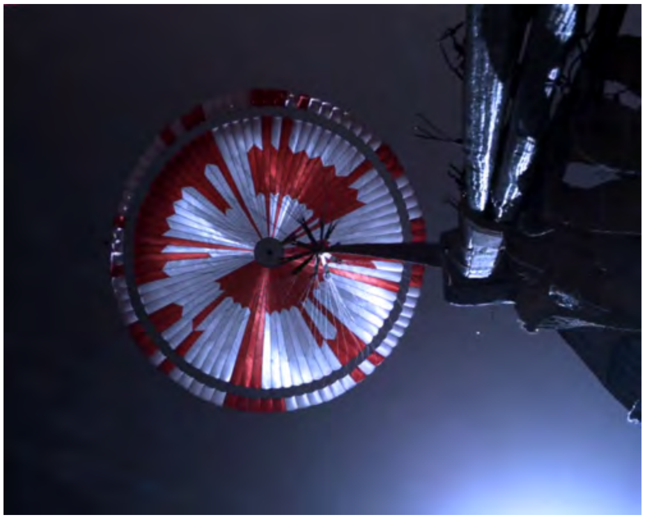


Figure 6.1: Disk-band-gap parachute of Mars 2020, fully inflated. The gap is clearly visible. While sacrificing on maximum drag, it is the only proven parachute design that deploys at supersonic speeds. Credit: NASA JPL/Caltech.

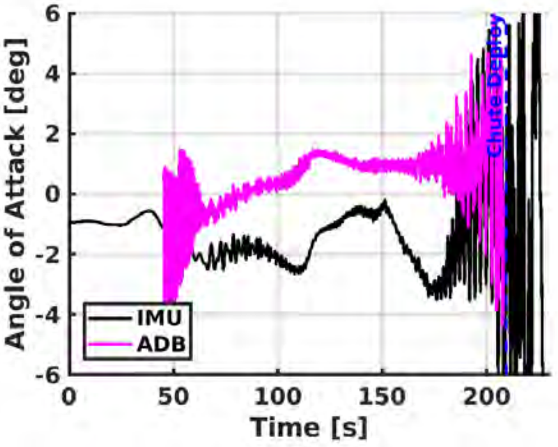


Figure 6.2: InSight reconstructed angle of attack, based on data from IMU and other factors. The black line is the direct data from the inertial measurement unit, and the pink line is from a ratio of normal and side acclerations for determination or angle of attack and sideslip, respectively. Credit: NASA JPL/Caltech.

JPL InSight flight reconstructed landing data [115]. As time past peak deceleration increases, the angle of attack oscillations increase. Past a certain point, this becomes unstable. It is highly impractical from a mass- and volume-budget perspective to

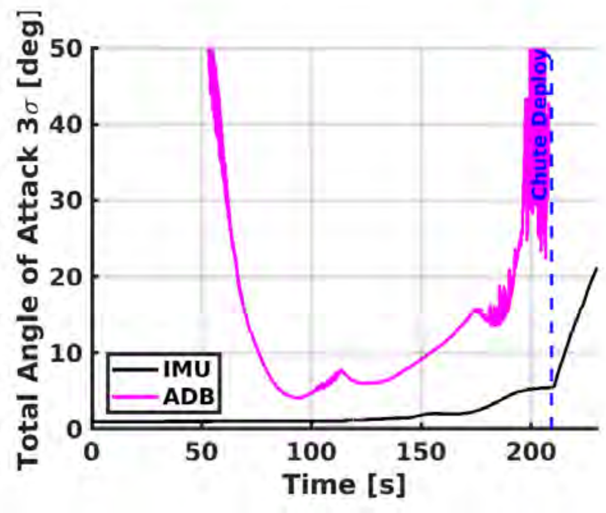


Figure 6.3: Reconstructed angle of attack, with 3-sigma error bounds. Credit: NASA JPL/Caltech.

control the pitch angle of the capsule, as most designs have used gas thrusters to control angle of attack[1] . This is the main rationale behind the supersonic disk-band-gap (DGB) parachute in Figure 6.1: the parachute would deploy before the capsule reached the critical Mach number. These considerations lead to the following assumptions:

> A6.1 - a 70-degree half-angle heat shield is assumed to be the best design in terms of flight heritage and volumetric efficiency.
> A6.2 - a supersonic, DGB parachute is assumed due to extensive flight heritage on Mars and the impracticality of devoting more fuel margins to stabilization of the vehicle below the critical Mach of the vehicle.

Following the heatshield is the backshell shape. The initial assumption was to start with a Mars 2020 and MSL-type backshell, with a "biconic" design conforming to the vehicle internals. The backshell shape, also a contributing factor to stability,

[1] Tianwen 1 is an exception - it used novel, popout fins that further guided the capsule below Mach 2.8.

was chosen (i.e. assumed) as a shape similar to that of Mars 2020 and MSL. An additional study on the optimal shape would have been beyond the scope of this paper. Due to time constraints, only one CFD run was made at Mach 2.0, which would inform very rough stability margins of the final iteration. Future work would more rigidly evaluate the stability based on more optimized CFD codes. The fidelity and uncertainty of roll rates at a given Mach is further constrained by the center of mass, which was not rigidly defined at the time of the CFD analysis. In summary:

> A6.3 - a backshell of Mars 2020 shape is assumed. Due to time constraints, changes to the backshell profile are assumed to minimally alter the aerodynamics of the aeroshell all the way down to low Mach numbers.
> A6.4 - the assumed deployment Mach number of the parachute is Mach 1.8.

With the aeroshell assumptions squared away, the assumptions for aerodynamic analysis are made. Table 6.1 lists a summary of these assumptions.

The Ansys Fluent multiphysics package was available through the Aerospace Engineering department; the tight integration to other analysis types especially for future research and development was the driving consideration for its usage. New to the 2022 R1 version was the inclusion of mixtures and reaction rates for both Earth and Mars atmospheres; this greatly simplified the modeling and setup of nonequilibrium chemically reacting flows. Conservation of energy included the radiative terms, but with non-optimal settings. Per [75], this was not the best mode of operation, as the radiative term considerably contributes to the flow energy on entry into a CO_2-dominant atmosphere, as shown in Figure 6.4. However, correctly modeling and validating results against NEQAIR, LAURA, and HARA codes as a complete system was deemed far beyond the scope of this paper.

Table 6.1: CFD Analysis Assuptions

Assumption
A6.5: ANSYS Fluent 2022 R1 is used for all computational fluid dynamics analyses.
A6.6: The Park 8-species Mars model is used for chemically reacting flows.
A6.7: The built-in *Stiff Chemistry Solver* is used to solve for nonequilibrium chemically reacting flow, and a minimum reaction temperature of 1,800K is considered to reduce computational overhead.
A6.8: The Ansys proprietary Transition-SST 4-equation viscosity model is used.
A6.9: Entry conditions following the ESA/Roscosmos Schiaparelli lander are considered at the following Mach numbers: 18, 9.79, 7, 5, and 1.8.
A6.10: A similar trajectory to NASA's Viking landers are considered due to time constraints.
A6.11: To analyze moment coefficients of the vehicle, a 5-degree flow angle of attack is used for all Mach numbers.
A6.12: A simplified OML with rounded corners is used.
A6.13: The Spalart-Allmaras and k-omega SST viscosity models are used.

6.2 Individual Cases

6.2.1 Hypersonic, Mach 18 and 9.79

To begin the initial analysis, the geometry was traced by screenshot in Ansys SpaceClaim, shown in Figure 6.5. Per assumption **A6.13**, geometric simplification, especially for 3D simulations, is necessary. Modeling extra surfaces, internal volumes, and other geometry is impractical, out of scope, and highly time- and resource-consuming. In addition, memory requirements increase exponentially especially with fully reacting, nonequilibrium flows.

Definition of the control volume lies in a number of smaller assumptions, but primarily tie back to **A6.12**. If radiative terms were included, according to [116], the

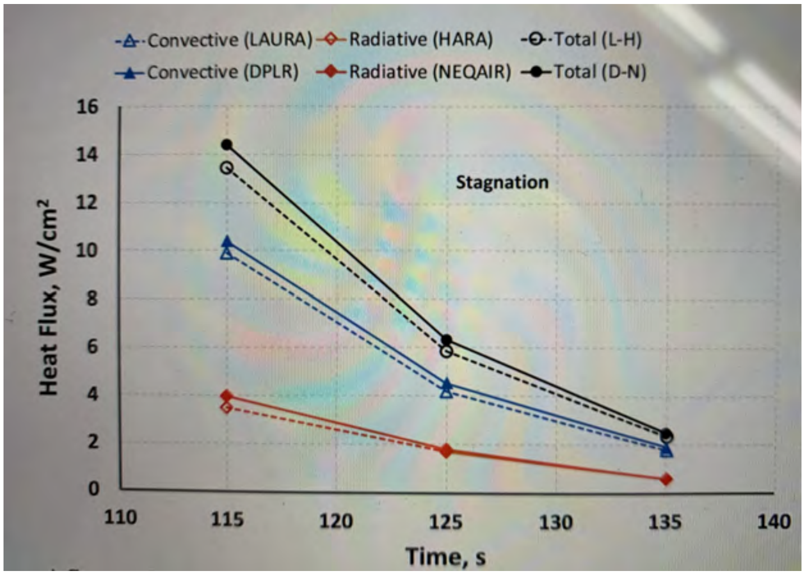


Figure 6.4: A chart of Schiaparelli's estimated total heat flux contributions just after peak heating. Due to the chemical kinetics of CO_2, the radiative component cannot be neglected for high-fidelity, high-precision landing simulations.

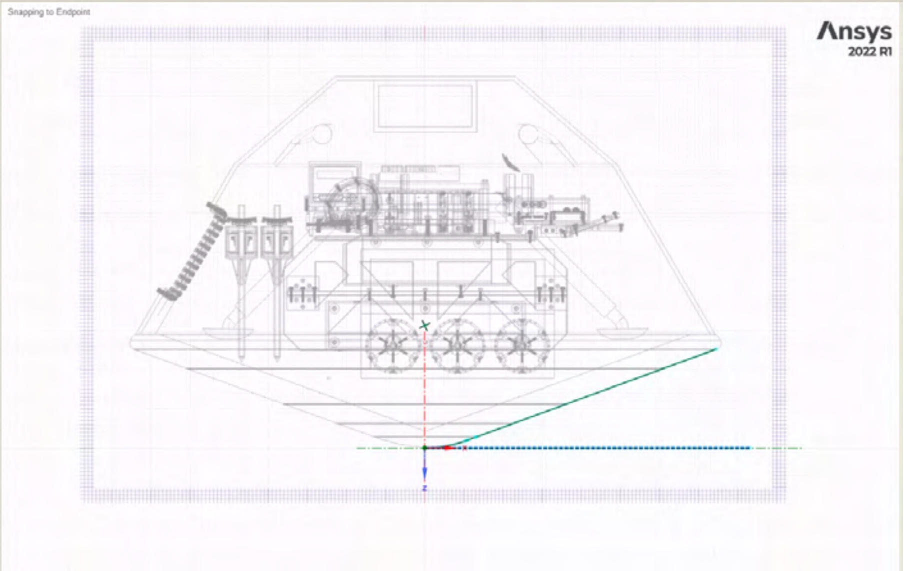
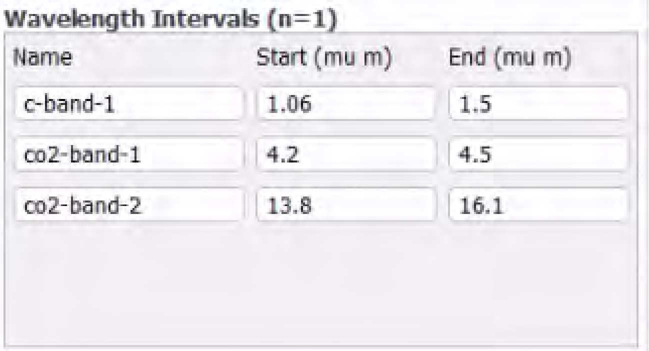


Figure 6.5: A screenshot of the SpaceClaim CAD program in sketch mode. The screenshot was scaled to the correct dimensions in order to prepare it for CFD analysis.



Wavelength Intervals (n=1)

Name	Start (mu m)	End (mu m)
c-band-1	1.06	1.5
co2-band-1	4.2	4.5
co2-band-2	13.8	16.1

Figure 6.6: n=1 wavelength radiative bands inputted into the Fluent non-gray radiation model.

wake length should be at least 12 body lengths to converge radiative emissions from reacting flows in nonequilibrium states. Given the sheer number of cells needed for such an analysis, even with NEQAIR and HARA, radiative terms were only considered to demonstrate that a full, multiphysics simuation can be achieved with Fluent. These parameters were included with the non-gray model, using the NIST Atomic Spectra Database (ASD), inputted as shown in Figure 6.6. This allowed the control volume to be substantially reduced to just over one body length to observe recirculation at the backshell, an indication of a correct CFD implementation. Upstream flow is assumed to be steady; therefore, the volume ahead of the shock is kept as small as practically possible. This distance is informed by

$$\frac{\delta}{R} = \frac{\rho_1/\rho_2}{1+\sqrt{2(\rho_1/\rho_2)}} \tag{6.1}$$

However, as freestream goes to infinity, Equation 6.1 simplifies to

$$\frac{\delta}{R} \approx \frac{\rho_1}{\rho_2} = \frac{1}{(\rho_2\rho_1)} \tag{6.2}$$

per Anderson [117]. Since flight data from Mars is available, the ρ_1/ρ_2 ratio can be found from the Schiaparelli flight data. Using a rearrangement of the momentum equation,

$$p_2 = p_1 + \rho_1 u_1^2 (1 - \frac{\rho_1}{\rho_2}) \tag{6.3}$$

the density ratio can be solved for from aerodynamic data at point S_3 in [118], yielding a shock standoff distance of $\boxed{6.63cm}$ at the stagnation point at Mach 9.79. To account for solution instability during the first few hundred iterations, the upwind portion of the control volume was offset by about 2.5x from the surface of the heatshield.

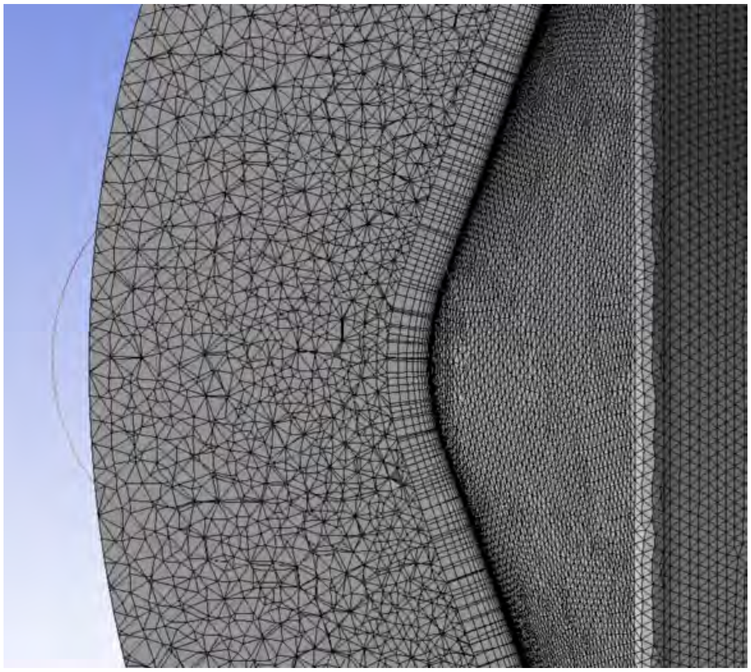


Figure 6.7: ANSYS DesignModeler (R) meshing software showing a cross-section of the 3D control volume for the Wavefront lander. This particular mesh has 2.9 million cells, with an O-type structured mesh and unstructured tetrahedral filler mesh. Note the semi-unstructured form of the O-type mesh - each layer is a hexahedral/triangular prism.

Meshing of the body was performed with the DesignModeler (DM) meshing system. To ensure accurate boundary layer physics, a structured O-type hexahedral

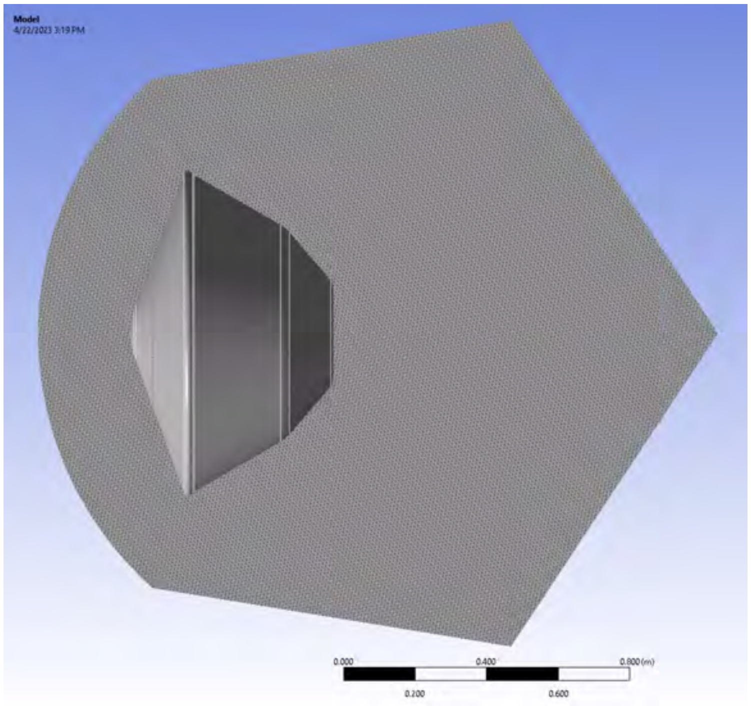


Figure 6.8: ANSYS DesignModeler (R) meshing software showing a cross-section of the blank 3D control volume for the Wavefront lander. After simulating flow, it was determined a loss of accuracy might have occurred due to part of the boundary condition intersecting the shoulders of the bow shock flow.

mesh was created. In DM, this option is called "Inflation Layers". It is called such due to the algorithm generating the O-type mesh - every subsequent layer from the adjacent wall layer grows, or inflates, by a specific ratio. The selected ratio was 1.35 - 35% larger every layer for 40 layers. Initial wall layer thickness started at 100 micron. O-type meshing is only possible when the entire, selected wall surface is free of sharp edges - to create a structured mesh, hexahedral cells (8 edges) are required. A low resolution example is included in Figure 6.7. Filling the rest of the control volume out to the boundary was accomplished by unstructured, tetrahedral meshing. While not ideal, this was chosen due to time limitations - creating a structured mesh considering shock capturing is a highly time-consuming process in the ANSYS ecosystem. It was especially difficult to accomplish with 3D simulations. In addition to being time-consuming for the operator to set up flow feature refinement, on a single body, meshing can only be performed serially: with a single core. This vastly increases the mesh generation time when converting from a tetrahedral to hexahedral mesh - several hours with over 1 million cells.

3 to up to 7 million cells were targeted to balance RAM usage and CPU time for initial solution exploration. A single, 2U rack-mount server with 8 physical cores and 144GB DDR3 ECC RAM was available - the bottleneck was the relatively low core count. Years of trial and error with FLUENT yielded optimal estimates of RAM usage and CPU time for a given set of hardware - with double precision, chemically reacting, nonequilibrium flow in 3D, iteration time was expected to be approximately 3 minutes. Later in the analysis, the cell count was increased to a maximum of 6.3 million cells for the problem, limited by total RAM in the server. Finer meshes, with an initially proposed target of 40 million cells, were not possible due to double-precision required for numerical stability in hypersonic flows and modeling of fully-reacting nonequilibrium flow. At minimum, 1.14TB RAM would be required for such a mesh, and a small supercomputer cluster with cores in the hundreds would be needed to solve the flow in reasonable amounts of time.

Table 6.2: Major Fluent Parameters, Initialization Run

Parameter	**Value**
Gauge Pressure	195.4 Pa
Mach Number	9.79
Flow Direction	5°
Turbulent Viscosity Ratio	2:1
Freestream Temperature	193.5K

After creating the mesh, it is automatically converted to a Fluent mesh. The major Fluent parameters for the first run attempting to establish an initial flowfield are tabulated in Table 6.2. To increase the solution speed as much as possible while still retaining acceptable accuracy, a single-variable synthetic viscosity model by Spalart and Allmaras is used [119]. According to the ANSYS (R) Fluent (R) theory guide, Spalart-Allmaras (S-A) has gained in popularity in wall-bounded flows, including turbomachinery simulations, and is shown to have reasonably accurate results against experiment in aerospace appliations. The original S-A turblulence model requires a well-defined boundary layer, with a Y_+ starting at ≈ 1.0 for adequate simulation of heat and stress transfer. The ANSYS Fluent treatment includes a less sensitive treatment of heat and stress transfer - but still recommends at least 15 to 30 layers within the boundary layer to model the viscous regions correctly. In 3D flows, this is a considerable constraint when considering the O-type boundary layer meshing. The shortfall of S-A is it cannot be used to predict the decay of isotropic turbulence. In the case of hypersonic aerodynamics, this could potentially have an effect on chemical mixing and radiative emissions depending on the amount of CO2 dissociation. Further investigation, which surveys the differences in turbulence models in hypersonic applications, is necessary.

Running hypersonic simulations with any reasonable acuracy requires chemical species to be considered in the energy term of the Navier-Stokes equations. Starting with ANSYS Fluent 2022 R1, a few gas mixtures were incorporated with 2, 5, and 11-species Earth air, as well as 5- and 8-species Mars air. Table 6.3 shows the chemical kinetics for the selected 8-species Mars air case.

Mach 25 to 18 cases were not considered due to a high Knudsen number. At 10 Pa, about 40 km deep from the edge of the Mars atmosphere, the Knudsen number is computed from the relation

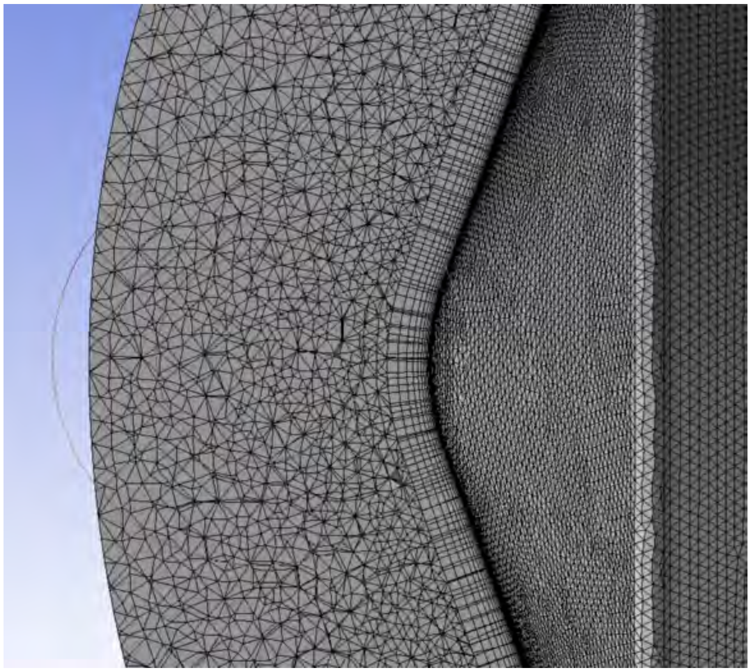


Figure 6.9: ANSYS DesignModeler (R) meshing software showing a cross-section of the blank 3D control volume for the Wavefront lander. After simulating flow, it was determined a loss of accuracy might have occurred due to part of the boundary condition intersecting the shoulders of the bow shock flow.

Table 6.3: Park 8-species Mars atmospheric model - reactions [7].

No.	**Reactions**	**Rate Expression**	**Remark**
1	$CO_2 \rightleftarrows CO + O$	$6.9E18T_a^{-1.5}exp(5.260976E8/T_a)$	—
2	$CO \rightleftarrows C + O$	$2.3E17T_a^{-1.}exp(1.072566E9/T_a)$	—
3	$O_2 \rightleftarrows O + O$	$2.0E18T_a^{-1.5}exp(4.967891E8/T_a)$	—
4	$CO_2 + O \rightleftarrows O_2 + O$	$2.1E10T_a^{0}exp(5.260976E8/T_a)$	—
5	$CO_2 \rightleftarrows CO + O$	$5.260978E21T_a^{-1.5}exp(5.260976E8/T_a)$	—
6	$CO_2 \rightleftarrows CO + O$	$5.260978E21T_a^{-1.5}exp(5.260976E8/T_a)$	—
7	$CO_2 \rightleftarrows CO + O$	$5.260978E21T_a^{-1.5}exp(5.260976E8/T_a)$	—
8	$CO_2 \rightleftarrows CO + O$	$5.260978E21T_a^{-1.5}exp(5.260976E8/T_a)$	—
9	$CO_2 \rightleftarrows CO + O$	$5.260978E21T_a^{-1.5}exp(5.260976E8/T_a)$	—

$$Kn = \frac{Ma}{Re}\sqrt{\frac{\gamma\pi}{2}} \tag{6.4}$$

where Kn is the Knudsen number, Ma is the Mach number, Re is the cell

Reynolds number, and γ is the heat capacity ratio of the gas in the cell. Assuming a length scale of the smallest freestream cell of $0.01m^2$,

$$Re = \frac{\rho \nu L}{\mu} \tag{6.5}$$

Re is 33.3. Mach at 10 Pa is 18, and γ is 1.28. This results in a Kn of $\boxed{0.77}$. According to Anderson, this is considered *transition* flow, meaning Navier-Stokes is unusable for any reasonable amount of accuracy. Kn should be under 0.01 to use traditional CFD methods. While there was access to Direct Simulation Monte Carlo (DSMC) codes, such as SPARTA, the time to set up the problem was out of scope of this paper and simulations were kept below the equivalent altitude of Mach 10. Additionally, dissociative radiative heat transfer from the bow shock must be accounted for as demonstrated in several publications and data from several Mars landers, so a tightly coupled simlulation involving SPARTA, LAURA, and HARA would also be necessary. This is far out of the scope of this paper.

6.2.1.1 Results

Due to the usage of unstructured mesh, hotspots were observed on the surface of the heat shield. The bow shock uncertainty from tetrahedral cells resulted in non-uniform energy balance, propagating to the rest of the coupled equations. Figure 6.10 shows the state of convergence in this case. Convergence to an accurate solution in a density-based solver is 0.1% change (residuals), which was still more than an order of magnitude away after 5 days of simulation with 2.997 million cells. However, comparing Figure 6.12 and 6.11, the pressure is well within the same order of magnitude.

Figure 6.14 shows the radiative response of the aeroshell. Using the non-gray

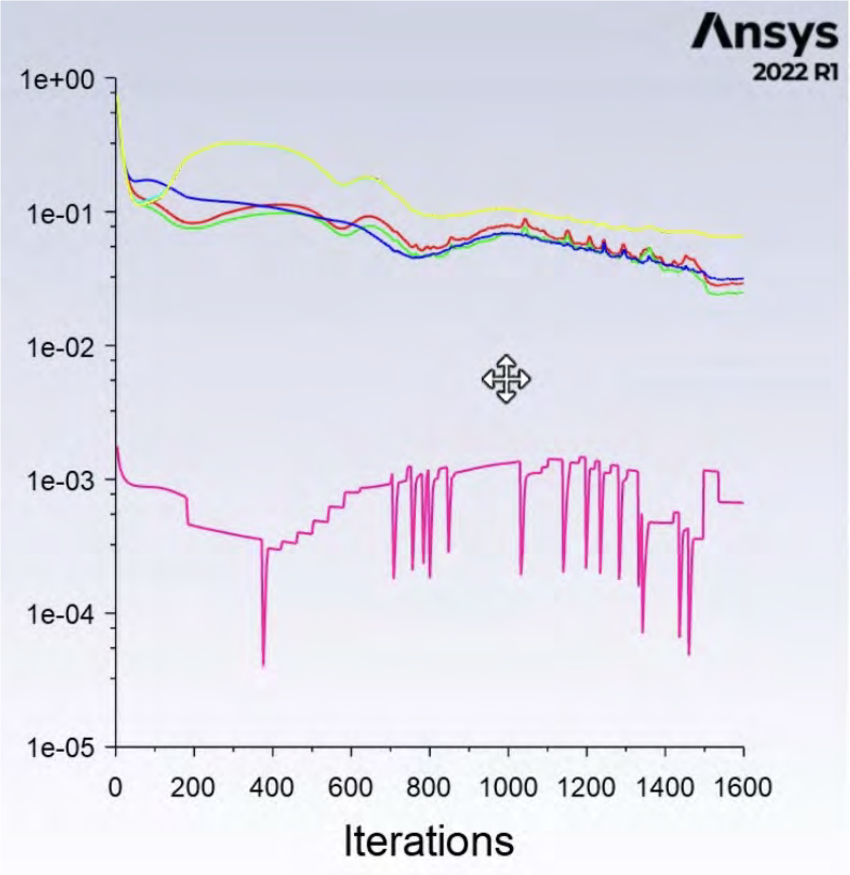


Figure 6.10: Convergence results after running the simulation with 2.997 million cells for 5 days. The stairstepping occurring in the ν_t variable shows constant divergence protection, which is a symptom of very high gradients, poor meshing in shoulder flow, and a generally unsolved flow condition.

and chemical kinetics model, the results are in general agreeance with experimental data from MSL and Mars 2020 MEDLI instrumentation as shown in Figure 6.13, as well as the Schiaparelli lander COMARS2 instrumentation.

The shoulder flow was tricky to mesh with sufficient resolution whilst beign bounded by a 3 million cell limit for hypersonic simulation, and is potentially the reason for convergence stalling. Figure 6.15 clearly shows rays emanating from the general direction of the shoulders, which is not realistic. Due to low dissociation at Mach 10, the radiative flux should gradually increase starting from the shoulders all

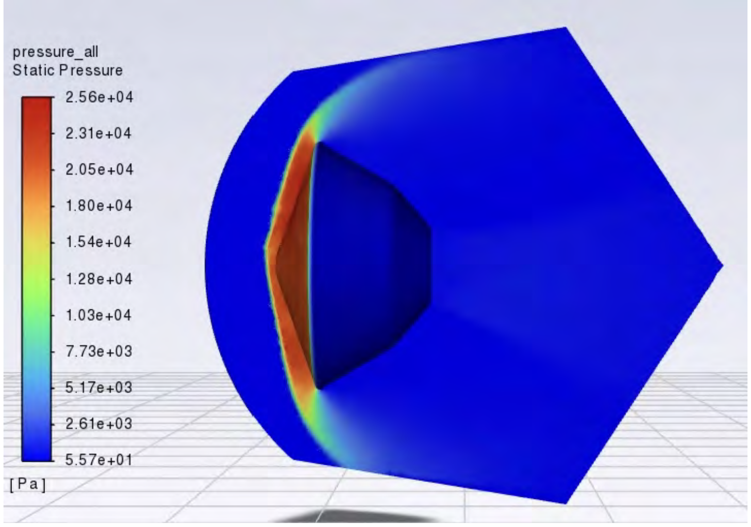
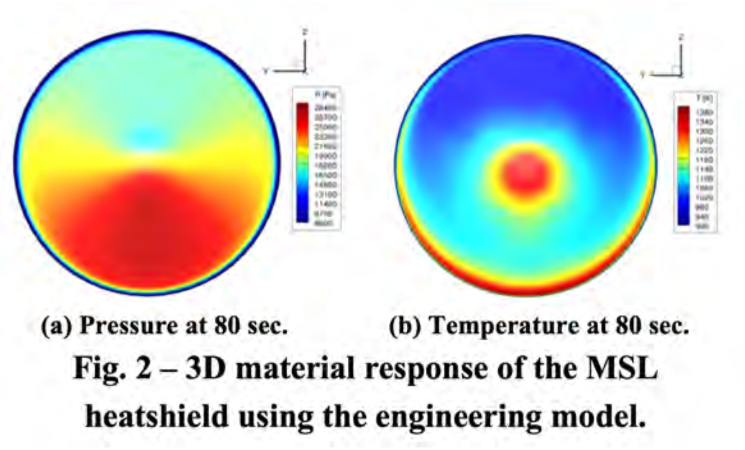


Figure 6.11: ANSYS DesignModeler (R) meshing software showing a cross-section of the blank 3D control volume for the Wavefront lander. After simulating flow, it was determined a loss of accuracy might have occurred due to part of the boundary condition intersecting the shoulders of the bow shock flow.


(a) Pressure at 80 sec.

(b) Temperature at 80 sec.

Fig. 2 – 3D material response of the MSL heatshield using the engineering model.

Figure 6.12: This figure shows the material response on the MSL heatshield. When compared to Figure 6.11, the results are well within the order of magnitude, and show that the approach is more or less correct.

the way to some distance back from the aeroshell as CO_2 recombines in the flow, which is indeed observed to some extent. It is obscured by the higher radiation temperature from the shoulder cells. Additional evidence of shoulder flow mesh issues

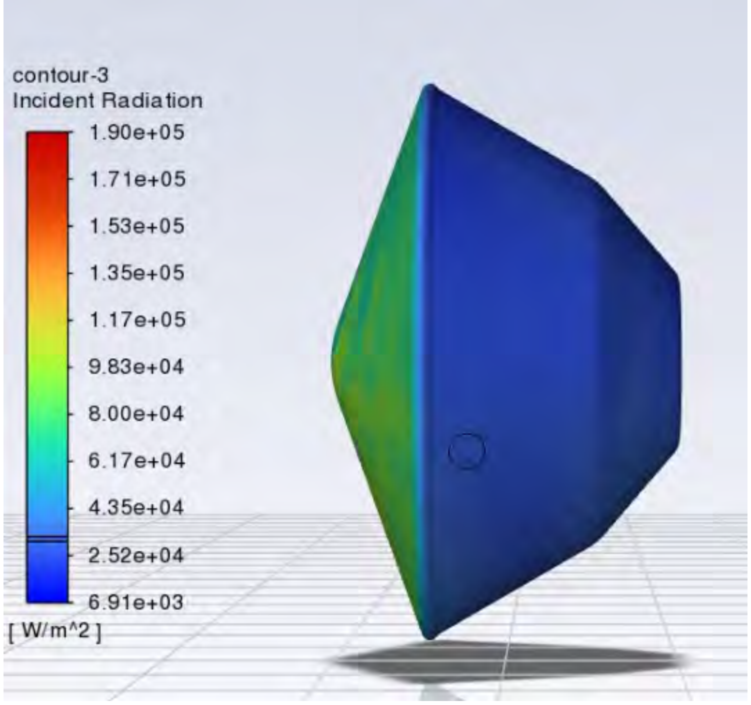


Figure 6.13: Radiative response of the surface of the aeroshell. The circle (reticle) on the backshell corresponds to Figure 6.13's MTB09 radiometer response.

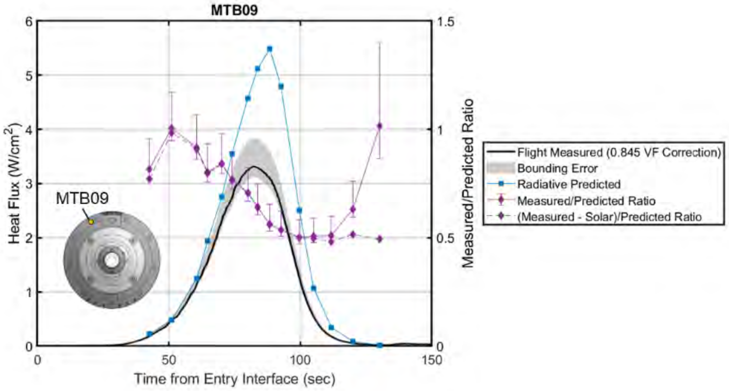


Figure 6.14: Radiative response of the surface of the MSL aeroshell.

is shown in Figure 6.16. Shoulder flow should not be generating heat greater than the stagnation point - the most severe part of a hypersonic flow.

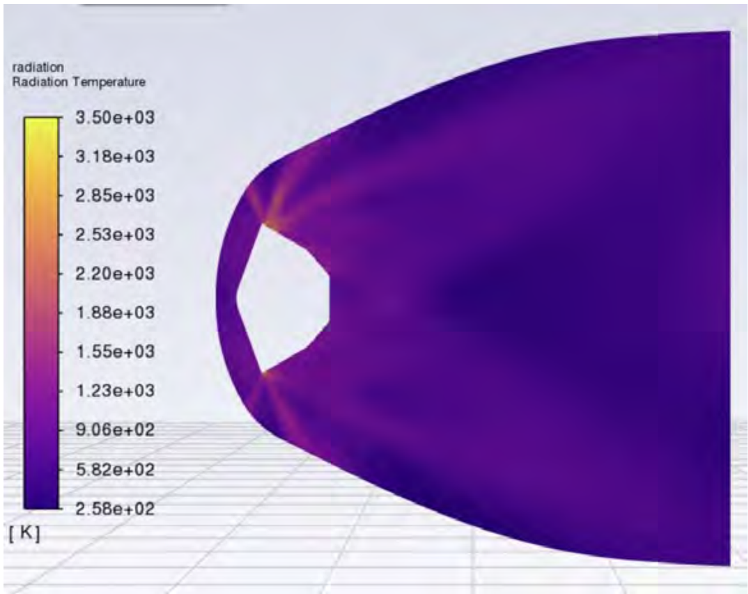


Figure 6.15: Radiation temperature response from the surface of the aeroshell.

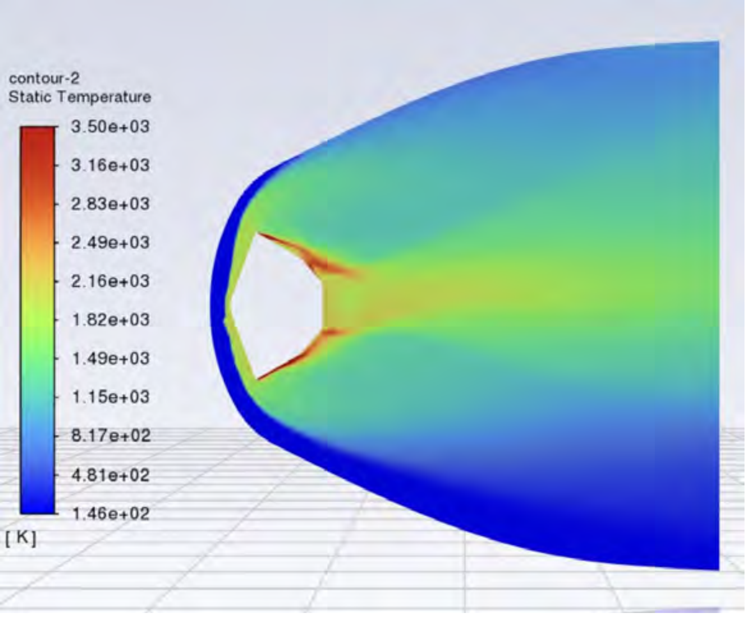


Figure 6.16: Radiative response of the surface of the MSL aeroshell.

6.2.2 Mach 2.1

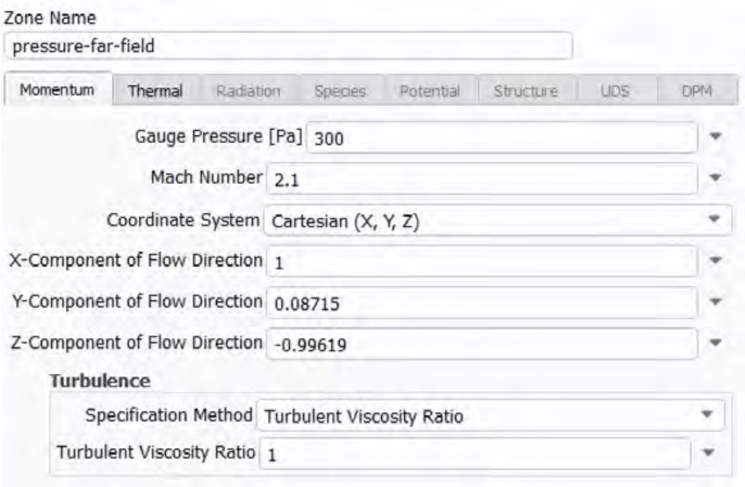
CFD was performed at a much lower Mach number as Wavefront would descend down to Hellas Basin. Supersonic conditions afforded much easier simulation conditions: chemical kinetics and radiative heat transfer mechanisms could be completely discounted. This also had the added benefit of being able to run higher fidelity simulations. From Mach 2.1 to 0.16, 30 million cells were targeted, since additional RAM would not be occupied by the extremely computationally expensive chemistry and radiation view factor tables.

The simulation was run at 300 Pa, Mach 2.1, and 207 Kelvin, which are conditions taken from trajectory reconstruction of InSight's landing. The setup dialog is shown in Figure 6.17. Erroneous results were discovered days into the simulation, which can be traced back to the X-component setting - it was not set to zero. The Y and Z components are set such that the flow gives a 5-degree angle of attack, allowing for derivation of stability derivatives. These are run at all Mach numbers, because stability of a blunt-body is a highly Mach-dependent phenomenon. This is clearly shown in the trajectory reconstructed data of InSight shown in Figure 6.18, top-left subpanel.

Due to the relaxed requirements, the k-omega shear stress transport equations (SST) are used from this point forward.

A section plane shows the internal cell structure after meshing in Figure 6.19. No mesh alignments were performed, as high accuracy was not desired. However, a boundary layer mesh of 16 cells thickness was used, in order to accurately simulate shear stress transition in the boundary layer.

Unfortunately, the error in the boundary conditions resulted in a potentially compromising scenario being simulated - the total angle of attack was approximately



Zone Name

pressure-far-field

Gauge Pressure [Pa] 300

Mach Number 2.1

Coordinate System Cartesian (X, Y, Z)

X-Component of Flow Direction 1

Y-Component of Flow Direction 0.08715

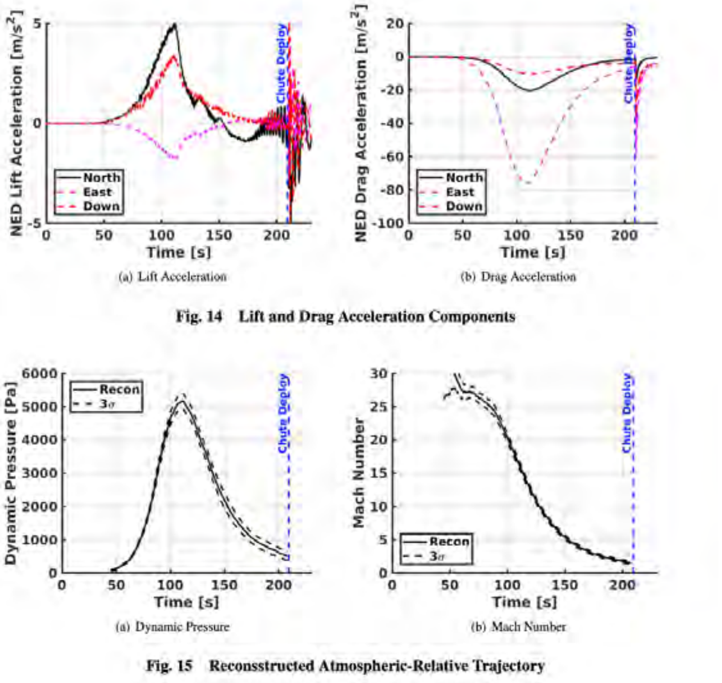
Z-Component of Flow Direction -0.99619

Turbulence

Specification Method Turbulent Viscosity Ratio

Turbulent Viscosity Ratio 1

Figure 6.17: Dialog showing boundary condition settings for Mach 2.1. Note the X-component error.



(a) Lift Acceleration

(b) Drag Acceleration

Fig. 14 Lift and Drag Acceleration Components

(a) Dynamic Pressure

(b) Mach Number

Fig. 15 Reconsstructed Atmospheric-Relative Trajectory

Figure 6.18: InSight reconstructed trajectory data. In the upper-left subpanel, it is very apparent that there is a Mach dependency on stability, as diverging oscillations occur with small perturbations from the Martian atmosphere.

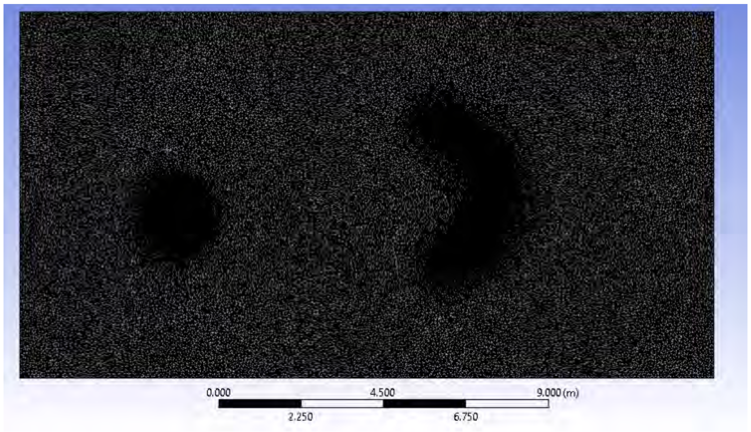


Figure 6.19: Unstructured grid showing backshell and parachute meshing.

40 deg. However, this was kept as a worst-case scenario.

The convergence plot is shown in Figure 6.20. Monitoring convergence was not sufficient; aerodynamic parameters needed to be converged as well. The force readout of the capsule is shown in Figure 6.21.

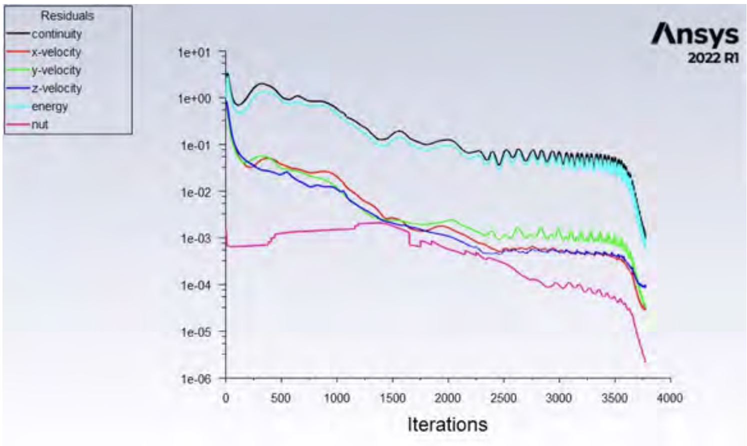


Figure 6.20: Convergence plot for Mach 2.1.

The results of this off-nominal scenario are shown in Figures 6.22, 6.23, and 6.24.

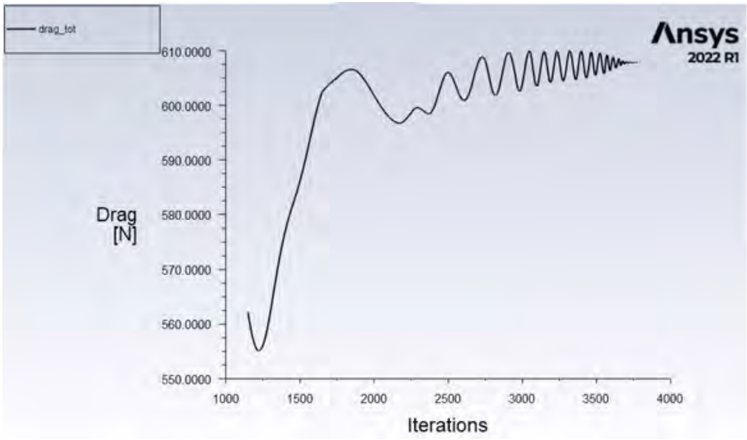


Figure 6.21: Aerodynamic convergence plot for Mach 2.1.

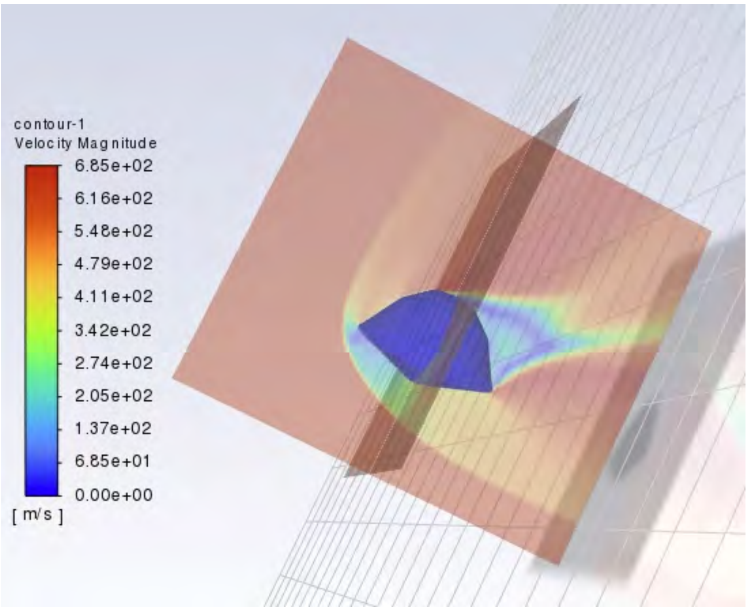


Figure 6.22: Oblique view of XZ and YZ planes of 3D flow around the aeroshell at over 40 deg AOA.

6.2.3 Mach 1.4 - Parachute Deployed

The same mesh and flow settings were retained for Mach 1.4, except the pressure and temperature were propagated downstream in accordance with InSight's reconstructed trajectory and atmospheric data, in addition to setting the airflow to

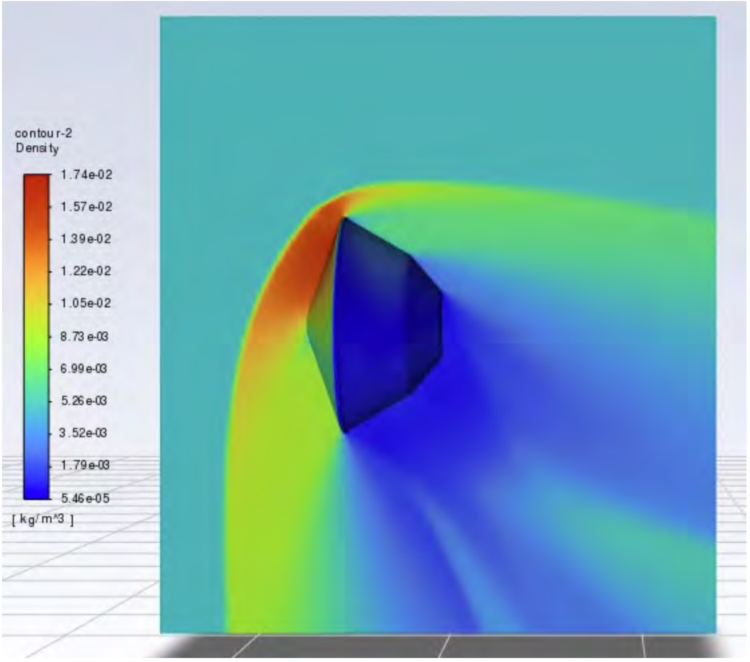


Figure 6.23: Density of flow on the XZ plane of flow around the aeroshell at over 40 deg AOA.

zero angle of attack. Figure 6.25 shows the boundary conditions.

An angle of attack is simulated with the parachute - it is modeled being 5 deg off-axis in the pitch and yaw axes. This is to simulate parachute sway, and to calculate for restoring force and the effect on vehicle dynamics after deployment. The CAD is shown in 6.26 shows the parachute off-axis.

To accelerate convergence, especially when using nearly 30 million cells, FAS FMG initialization is used. Care must be taken to use a small enough CFL number when running FMG because a misleading solution can be initialized and take much more computational effort than a simple Bernoulli-style hybrid initialization. After a 10 iteration initialization routine of approximately 30 minutes, the flowfield in the YZ axis is shown in Figure 6.27, which is a relatively accurate result for the algorithm it is.

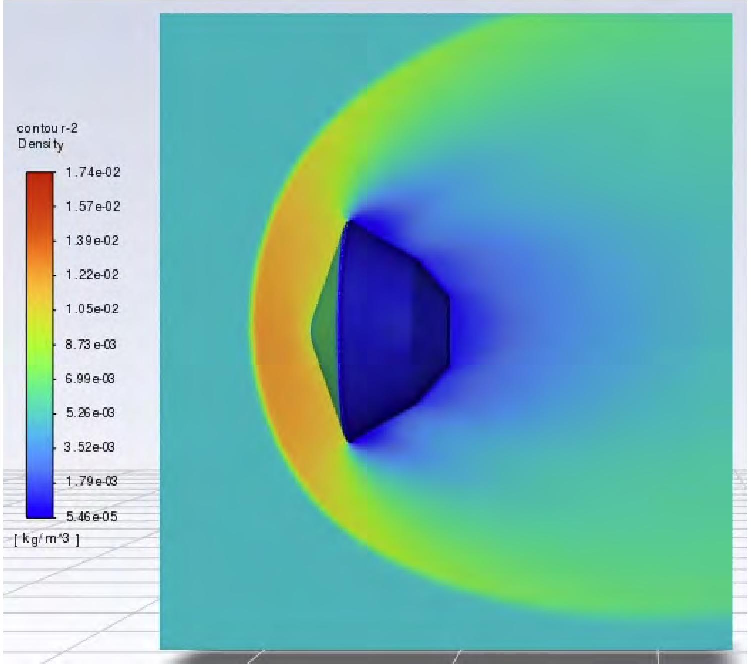
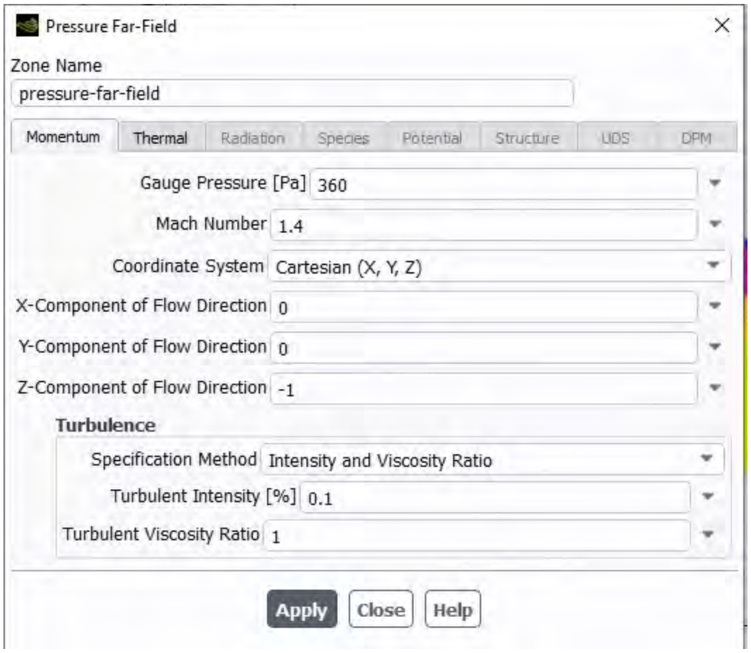


Figure 6.24: Density of flow on the YZ plane of flow around the aeroshell at over 40 deg AOA.



Zone Name: pressure-far-field

Gauge Pressure [Pa]: 360

Mach Number: 1.4

Coordinate System: Cartesian (X, Y, Z)

X-Component of Flow Direction: 0

Y-Component of Flow Direction: 0

Z-Component of Flow Direction: -1

Turbulence

Specification Method: Intensity and Viscosity Ratio

Turbulent Intensity [%]: 0.1

Turbulent Viscosity Ratio: 1

Figure 6.25: Boundary conditions dialog. Note that there is no angle of attack.

Figure 6.26: Off-axis parachute. No bridles are modeled to allow for computational simplicity.

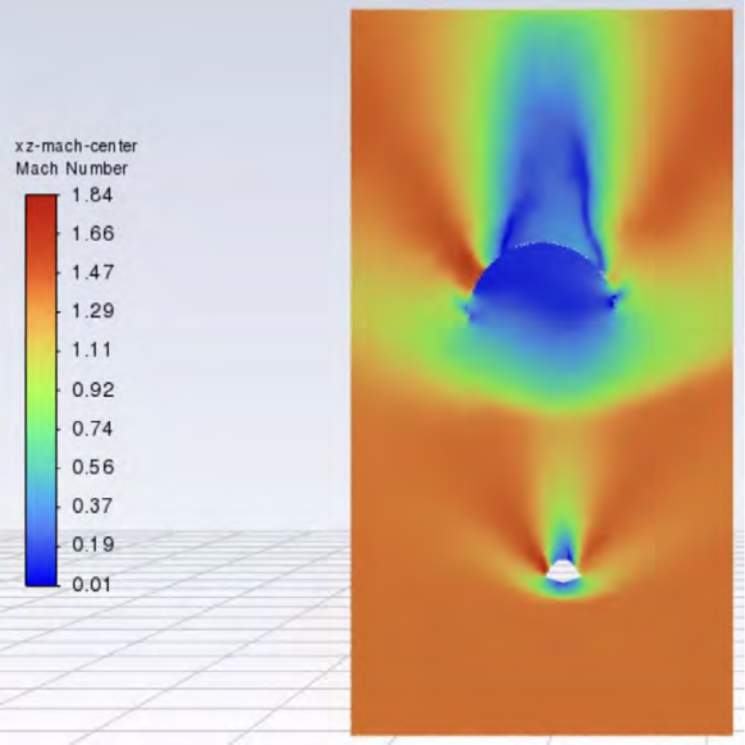


Figure 6.27: Initial solution obtained using FAS FMG initialization.

To analyze the effect of turbulence on aerodynamics, the initial solution was used to perform a transient time-stepped simulation. After 150 timesteps at 160 microseconds for approximately 6 days of simulation time, the simulation was advanced to 1 milisecond per timestep and convergence criteria was relaxed to 0.5%.

By this point, timesteps were solved in only one to two iterations, indicating a very steady flow. The residual plot shows this exponentially convergent behavior in Figure

reffig:140mach-pp4.

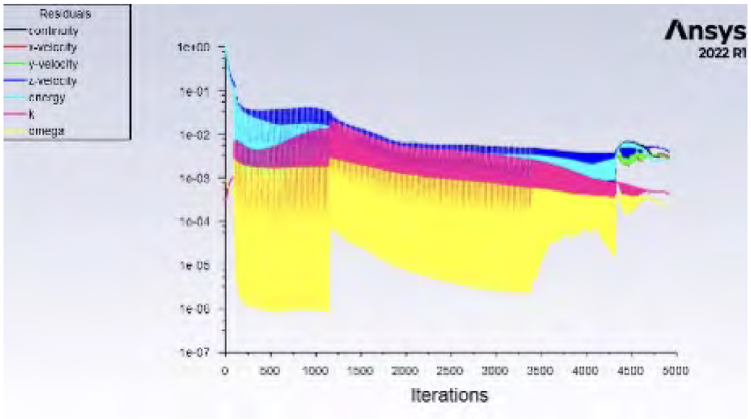


Figure 6.28: Residual plot of transient simulation. The first 100 iterations were performed at steady state to antialias the FMG solution, 150 timesteps were performed at 160 microseconds to converge the solution, and 251 timesteps were then performed at 1 millisecond once sufficiently converged.

To show convergence of the solution and minimal effect of turbulence on drag, Figures 6.29, 6.29 and 6.29 show the convergent behavior and the overall drag of the entire system, the capsule, and the parachute.

The final result after 331.3 ms of timestepping is shown in Figure 6.32. It is a composite scene, with the Mach number showing on the YZ plane, density on the parachute and aeroshell, and an iso-surface of 14% turbulent viscosity ratio with flow velocity plotted on the iso-surface.

6.2.4 Mach 0.16 - Pre-Deployment

One more multi-day simulation was performed with the same mesh. This was to validate a hand calculation of terminal velocity and drag on the system seconds before release into the lowest points of Hellas Basin.

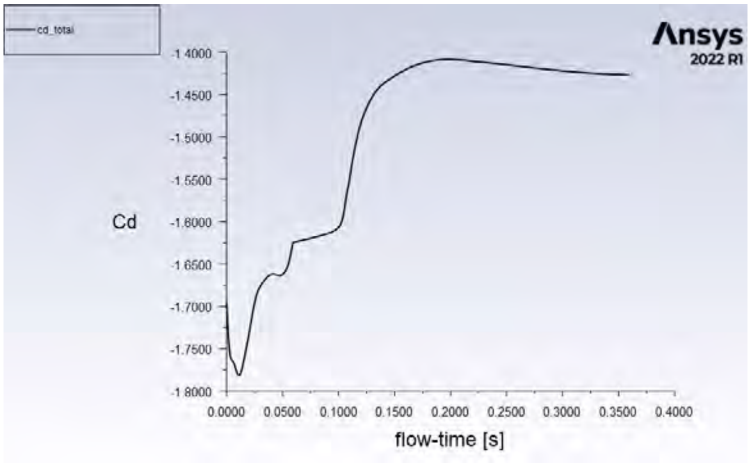


Figure 6.29: Total coefficient of drag on the capsule-parachute system.

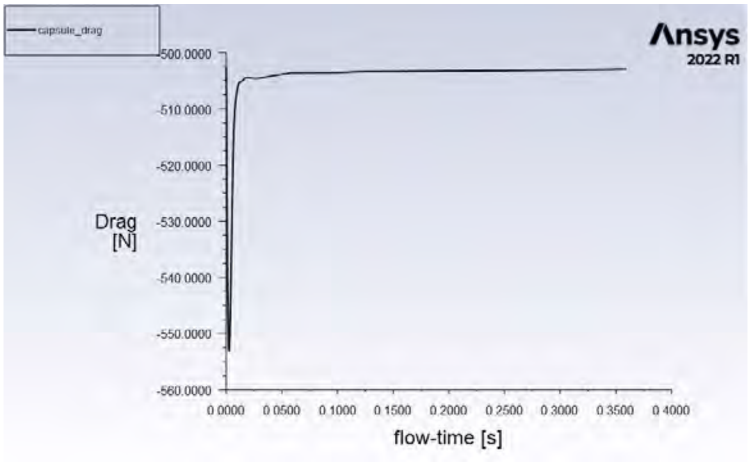


Figure 6.30: Drag in Newtons on the aeroshell.

To compute this, the coefficient of drag during Mach 1.4 flight was taken, assuming that the C_D holds relatively constant through the transonic and subsonic flight regime. This C_D was taken as 1.428.

One form of the ideal gas equation is given as

$$\rho = \frac{P}{RT} \tag{6.6}$$

At 1 km above the lowest point in Hellas Basin, the pressure is approximately

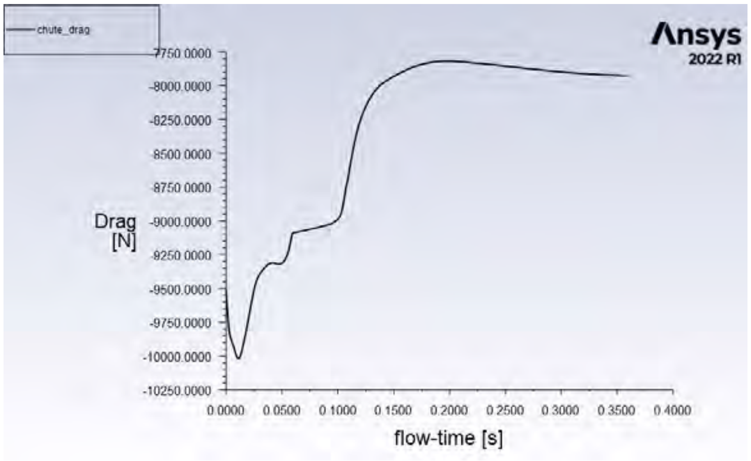


Figure 6.31: Drag in Newtons on the parachute.

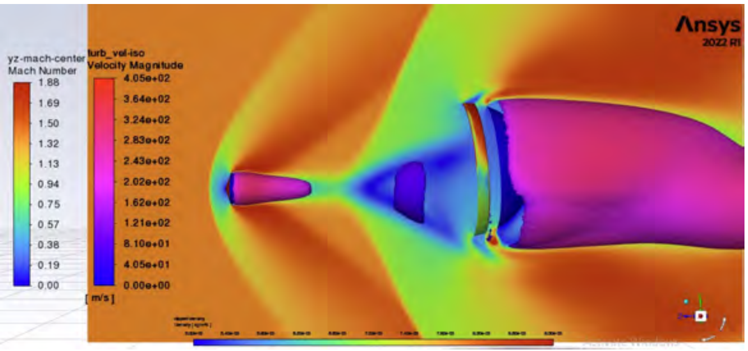


Figure 6.32: Mach 1.4 scene at 331.4ms flow time.

1150 Pa. Using 6.6, a density of $2.623E-2\ kg/m^3$ is determined at 232 Kelvin.

Terminal velocity is then given as

$$V_T = \sqrt{\frac{2mg}{\rho A C_D}} \tag{6.7}$$

where m is the mass of the system (91 kg - the heat shield was ejected by this point), g is gravitational acceleration, which on Mars, is 3.728 m/s^2; ρ is density, which was just solved for; and A is the total projected area of the system. Assuming the flow is slow enough, the reference area used is both the aeroshell and parachute, which yielded 12.56637 m^2. Therefore, the terminal velocity at this altitude is $\boxed{37.96 m/s}$. Converting this to Mach on pure CO2 at 232K results in Mach 0.16.

The CFD run to validate these numbers is the same, except for changing temperature and pressure to match the conditions near deployment.

The ending residuals are shown in Figure 6.33. The ending residuals are just above 0.1%, which is likely due to an undersized control volume and very close proximity of the turbulent core of the parachute to the boundary condition. Given the limited time, CPU cores, and only 144 GB RAM, enlarging the control volume was a non-starter, as the problem size peaked at 120 GB RAM for 29.9 million cells. This convergence condition is an acceptable tradeoff for the hardware and time limitations, however.

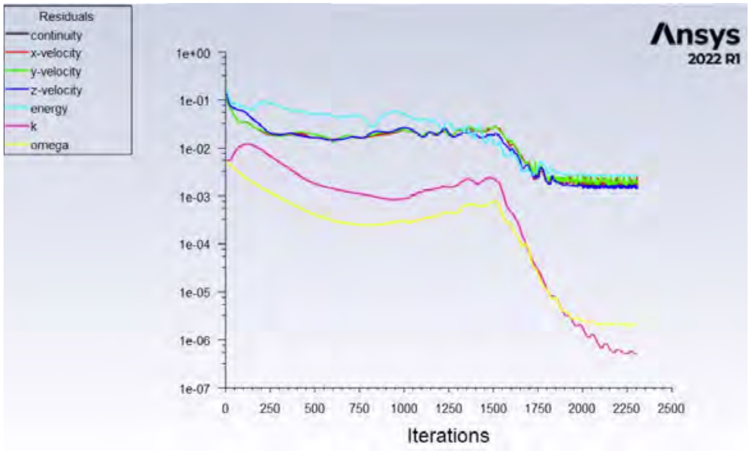


Figure 6.33: Residuals at Mach 0.16. The solution converges just above 0.1%, which is likely due to the boundary conditions being too close to the turbulent wake core of the parachute.

Figures 6.34, 6.35, and 6.36 are additional aerodynamic plots that demonstrate numerical convergence. The final drag value computed at steady state is 419.8 Newton, significantly above the 338.5N system weight at this point of the flight. Observing the total C_D assuming all other aerodynamic parameters are correct (which is a reasonable assumption given the very low speed flow) gives the reason why - the computational value is 1.77. Recomputing the terminal velocity with this coefficient of drag yields $\boxed{34.01m/s}$.

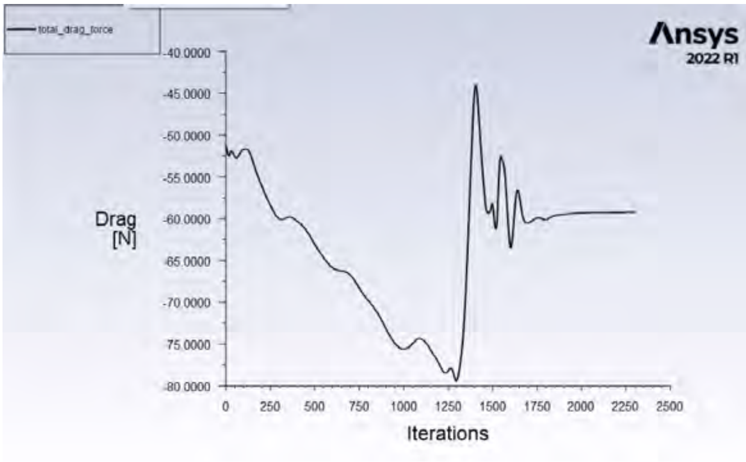


Figure 6.34: Plot of *lift* force in the XY plane. The plot is erroneously labeled as drag.

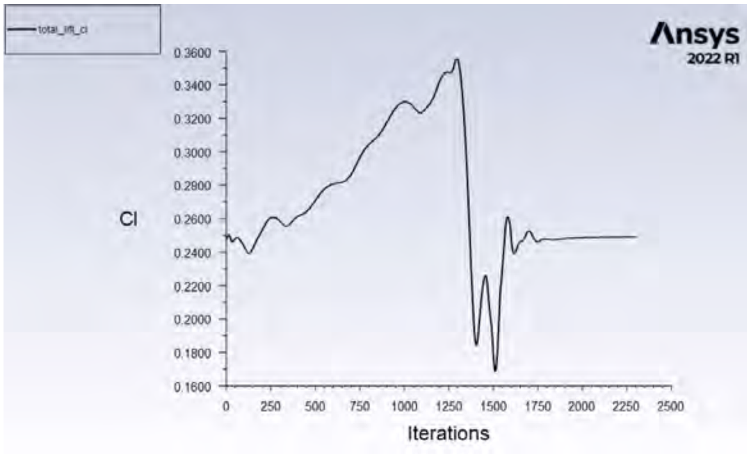


Figure 6.35: Plot of coefficient of lift in the XY plane. The initial estimate generated by FAS FMG was nearly spot-on with the final result.

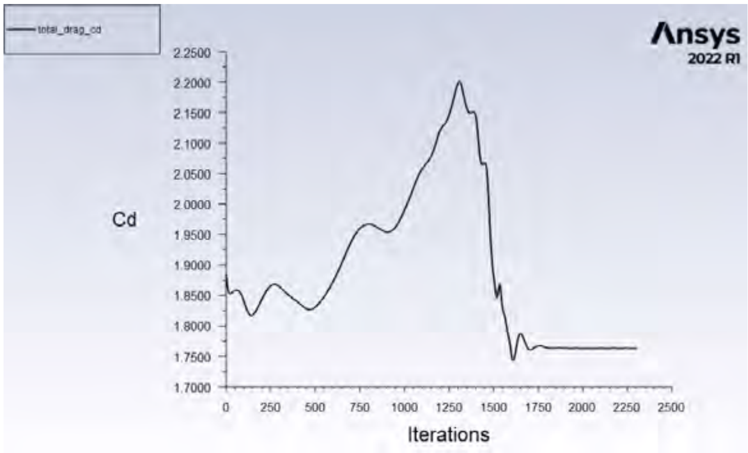


Figure 6.36: Total CD for the entire package. Convergence was observed at 1800 iterations, or roughly 3 days of wall-clock time.

6.2.5 Fuel Margin Validation

At this point, it is now possible to validate the fuel margin available in the Midgar lander. As designed, there is 9.21 kg of hydrazine fuel loaded in the four conformal tanks. Using the Tsiolkovsky rocket equation,

$$m_0 = m_f e^{\Delta v / v_e} \tag{6.8}$$

the initial mass is assumed to be 84 kg without the aeroshell, heatshield, and parachute, the final mass is 74.79 kg, and the specific impulse of hydrazine is 220 seconds. This translates to 2.157 km/s. Rearranged to solve for ΔV, the final solution is $\boxed{250.50 m/s}$. This is a very favorable result - the resulting factor of safety is $\boxed{6.756}$. This factor of safety can then be used by machine vision to search for a safe landing spot.

Table 6.4: Park 11-species Earth atmospheric model [7].

No.	Reactions	Rate Expression	Remark
1	$O_2 + O_2 \rightarrow O + O + O_2$	$2E21T_a^{-1.5}exp(-5.95E4/T_a)$	—
2	$O_2 + NO \rightarrow O + O + NO$	$2E21T_a^{-1.5}exp(-5.95E4/T_a)$	Est.
3	$O_2 + N_2 \rightarrow O + O + N_2$	$2E21T_a^{-1.5}exp(-5.95E4/T_a)$	—
4	$O_2 + O \rightarrow O + O + O$	$1E22T_a^{-1.5}exp(-5.95E4/T_a)$	—
5	$O_2 + N \rightarrow O + O + N$	$1E22T_a^{-1.5}exp(-5.95E4/T_a)$	Est.
6	$NO + O_2 \rightarrow N + O + NO$	$5E15T_a^{0}exp(-7.55E4/T_a)$	Est.
7	$NO + NO \rightarrow N + O + NO$	$1.1E17T_a^{0}exp(-7.55E4/T_a)$	—
8	$NO + N_2 \rightarrow N + O + N_2$	$5E15T_a^{0}exp(-7.55E4/T_a)$	—
9	$NO + O \rightarrow N + O + O$	$1.1E17T_a^{0}exp(-7.55E4/T_a)$	Est.
10	$NO + N \rightarrow N + O + N$	$1.1E17T_a^{0}exp(-7.55E4/T_a)$	Est.
11	$N_2 + O_2 \rightarrow N + N + O_2$	$7E21T_a^{-1.6}exp(-1.132E5/T_a)$	Est.
12	$N_2 + NO \rightarrow N + N + NO$	$7E21T_a^{-1.6}exp(-1.132E5/T_a)$	Est.
13	$N_2 + N_2 \rightarrow N + N + N_2$	$7E21T_a^{-1.6}exp(-1.132E5/T_a)$	—
14	$N_2 + O \rightarrow N + N + O$	$3E22T_a^{-1.6}exp(-1.132E5/T_a)$	Est.
15	$N_2 + NO \rightarrow N + N + NO$	$3E22T_a^{-1.6}exp(-1.132E5/T_a)$	—
16	$N_2 + e \rightarrow N + N + e$	$3E24T_a^{-1.6}exp(-1.132E5/T_e)$	Est.
17	$N_2 + O \rightarrow NO + N$	$6.4E17T_a^{-1}exp(-3.82E4/T_a)$	—
18	$NO + O \rightarrow O_2 + N$	$8.4E12T_a^{0}exp(-1.94E4/T_a)$	—
19	$N + O \rightarrow NO^+ + e$	$5.3E12T_a^{0}exp(-3.19E4/T_a)$	—
20	$N + N \rightarrow N_2^+ + e$	$2E13T_a^{0}exp(-6.75E4/T_a)$	—
21	$N_2 + O \rightarrow NO + N$	$6.4E17T_a^{-1}exp(-3.82E4/T_a)$	—
22	$O + e \rightarrow O^+ + e + e$	$3.9E33T_a^{-3.78}exp(-1.585E5/T_e)$	Est.
23	$N + e \rightarrow N^+ + e + e$	$2.5E33T_a^{-3.82}exp(-1.682E5/T_a)$	—
24	$NO^+ + O \rightarrow N^+ + O_2$	$1E12T_a^{0.5}exp(-7.72E4/T)$	—
25	$O_2^+ + N \rightarrow N^+ + O_2$	$8.7E13T_a^{0.14}exp(-2.86E4/T)$	—
26	$NO + O^+ \rightarrow N^+ + O_2$	$1.4E5T_a^{1.9}exp(-1.53E4/T)$	—
27	$O_2^+ + N_2 \rightarrow N_2^+ + O_2$	$9.9E12T_a^{-1.08}exp(-4.07E4/T)$	—
28	$O_2^+ + O_2 \rightarrow Ok + O_2$	$6.4E17T_a^{-1}exp(-3.82E4/T_a)$	—
29	$NO^+ + N \rightarrow O^+ + N_2$	$3.4E13T_a^{-1.08}exp(-1.28E4/T)$	—
30	$NO^+ + O_2 \rightarrow O_2^+ + NO$	$2.4E13T_a^{0.41}exp(-3.26E4/T)$	—
31	$NO^+ + O \rightarrow O_2^+ + N$	$7.2E12T_a^{0.29}exp(-4.86E4/T)$	—
32	$O^+ + O \rightarrow NO + N$	$6.4E17T_a^{-1}exp(-3.82E4/T_a)$	—

7. Universal Electrical Power System Design

7.1

Although the design of this system was incomplete, the only major items that were needed to complete the electrical design were backplane (spacecraft bus) configuration and tuning of power system control parameters, as well as the establishment of a safe operating envelope for all adjustable systems. This adjustment and characterization is what would take the most amount of R&D time: the electrical systems need to be validated in a program such as MATLAB(R) Simulink(R), or Spice, an electronic circuit simulator. Further, porting the simulated and validated envelopes of control would take a large amount of time, especially to validate program execution under a variety of conditions and coupling.

Figure 7.1 shows the overall schematic. However complicated this PCBA might be, it is necessary due to the need for redundant circuitry and enough self-sensing.

Figure 7.2 shows the GPSDO subsystem. There are three further subsystems in this circuit: a dual-redundant, precision in-flight adjustable voltage regulator, a high-frequency clock prescaler, and the rubidium atomic clock unit itself along with supporting passive and monitoring electronics.

Figure 7.3 is the digital potentiometer power supply and the redundant watchdog timeout circuit. Due to how critical the potentiometers are in precisely trimming the output voltages to the entire spacecraft during the life of the mission, the potentiometer power supply is quadruple redundant, featuring Schottky barrier diodes to block lower voltage from a faulty or burned out regulator circuit. The watchdog timer, however, is the most critical circuit on the spacecraft. Two watchdog ASICs send a signal to either of the active redundant CPUs (through an EX-NOR

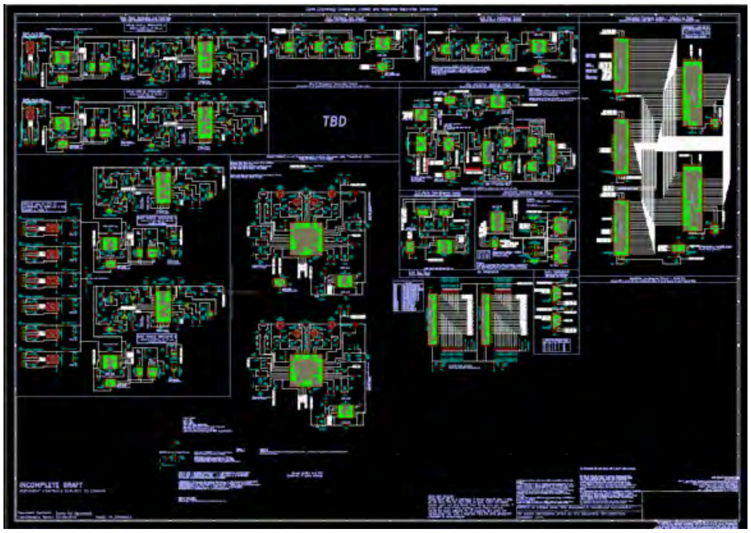


Figure 7.1: Overview of Claire II. This screenshot illustrates the very high degree of complexity of an autonomous power system.

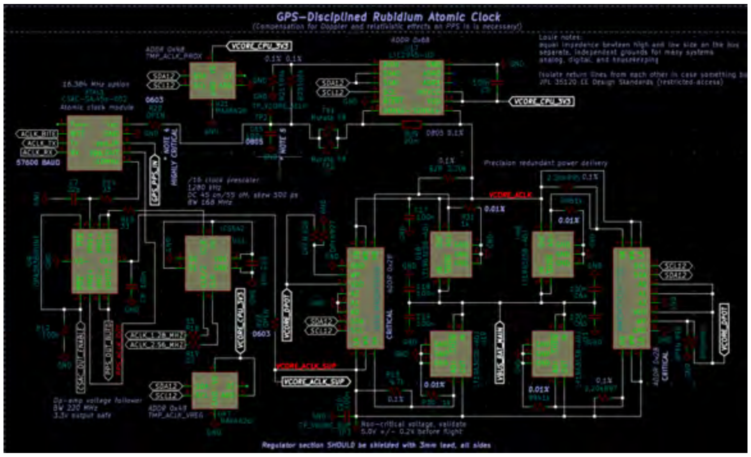

GPS-Disciplined Rubidium Atomic Clock
(Compensation for Doppler and relativistic effects on PPS is is necessary!)
VCORE_CPU_3V3
VCORE_ACLK
VCORE_ACLK_SUP
CRITICAL
Precision redundant power delivery
Regulator section SHOULD be shielded with 3mm lead, all sides

Figure 7.2: Power delivery and control schematic for GPSDO. There are three sub-subsystems: a dual-redundant, precision in-flight adjustable voltage regulator, a high-frequency clock prescaler, and the rubidium atomic clock unit itself.

gate) if one of them does not receive a signal by their hardwired timeout period. This signal turns off the frozen CPU and powers on the other. To protect against failure of one of the watchdog timers, an EX-NOR gate is used.

Table 7.1 shows the logic truth table for this gate: if the state if either watchdog

Table 7.1: 2-bit Exclusive-NOR gate truth table.

Input A	**Input B**	**Output**
False	*False*	*False*
False	*True*	*True*
True	*False*	*True*
True	*True*	*False*

fails, regardless of a watchdog failure or not, the reset signal will be triggered.

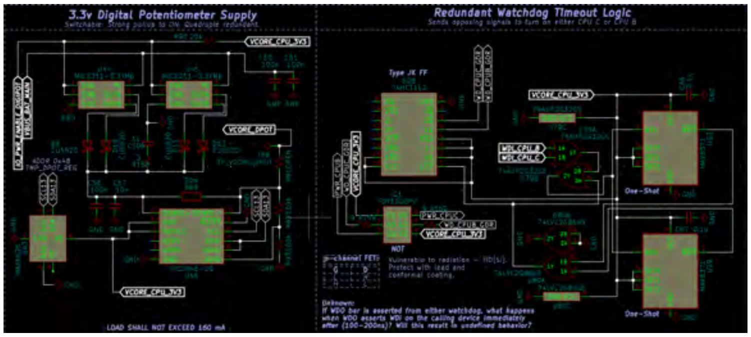

3.3v Digital Potentiometer Supply
Switchable: Strong pullup to ON. Quadruple redundant.
VCORE_CPU_3V3
VCORE_DPOT
ADDR 0x48
TMP_DPOT_REG
VCORE_CPU_3V3
LOAD SHALL NOT EXCEED 160 mA
Redundant Watchdog Timeout Logic
Sends opposing signals to turn on either CPU C or CPU B
Type JK FF
VCORE_CPU_3V3
WDI_CPU_B
WDI_CPU_C
One-Shot
PWR_CPUC
WD_CPUB_GOR
VCORE_CPU_3V3
NOT
p-channel FET:
Vulnerable to radiation — HD(s).
Protect with lead and
conformal coating.
Unknown:
If WDO bar is asserted from either watchdog, what happens
when WDO asserts WDI on the calling device immediately
after (100-200ns)? Will this result in undefined behavior?
One-Shot

Figure 7.3: Potentiometer power supply and CPU watchdog schematics.

Figure 7.4 shows the solar wing power regulator. The core of the regulator circuit, an LTC3119MPFE, is a highly monolithic, military temperature-rated integrated circuit that is able to track a variable input voltage, while maintaining control over the maximum power point. The drawback of this highly integrated chip is that it is not radiation tolerant. To counteract potential early failure or latchup due to radiation, multiple regulators are used in parallel. [Elaborate more on this later.]

Figure 7.5 shows the GaNFET-supported battery charger circuit. To support two parallel cells, two circuits are included. Simplification of this highly complex subsystem, centered around a monolithic controller IC, is possible by using multi-cell batteries with internal regulator circuitry. These are available from various manufacturers; some are custom-manufactured.

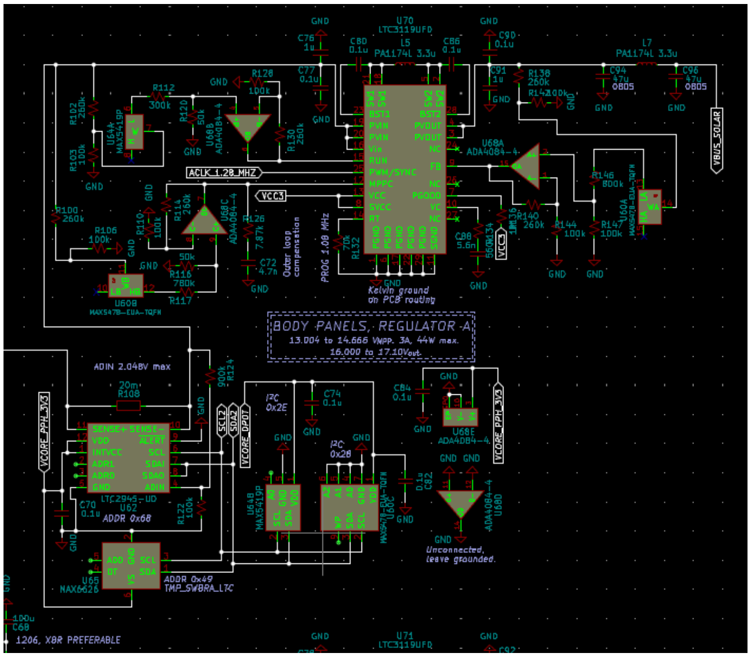


Figure 7.4: Solar wing power regulator subsystem schematic.

7.1.1 Dual-Redundant CPU Topology

The implications for switching over CPUs from one active set to another is one of RAM. Because the power system does not require fast response times, the CPUs do not need to execute in lockstep mode. This additionally preserves the inactive CPU from some radiation-induced effects. However, when power is lost to the CPU, the contents of RAM are lost. To offset this, a shadow copy of all runtime variables and states is saved into external ferroelectric RAM (FRAM). FRAM is a relatively new type of memory that repackages and miniaturizes the strengths of core memory, discussed in Section 1.1. According to [14], it is intrinsically radiation resistant, since magnetic domains, not capacitive domains, store memory. Additionally, the feature size and insulation oxides are relatively speaking, thick compared to many other wafer processes as of writing. By storing a shadow copy, as long as the executing CPU's microcode is not corrupted, the other CPU can pick up back where the failed

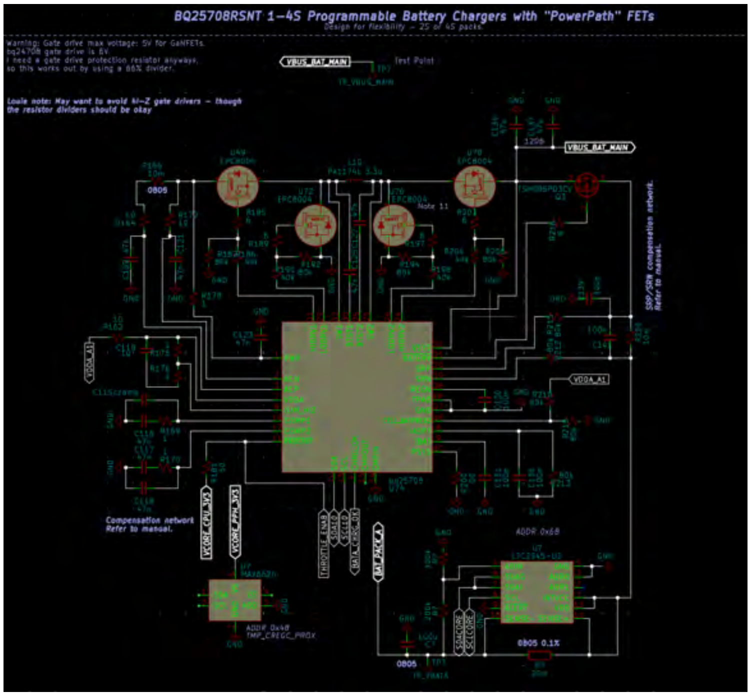

BQ25708RSNT 1–4S Programmable Battery Chargers with "PowerPath" FETs
Design for flexibility – 2S or 4S packs.
Warning: Gate drive max voltage: 5V for GaNFETs.
bq24708 gate drive is 6V.
I need a gate drive protection resistor anyways,
so this works out by using a 86% divider.
Louie note: May want to avoid hi-Z gate drivers – though
the resistor dividers should be okay
VBUS_BAT_MAIN
Test Point
TP_VBUS_MAIN
EPC8004
Note 11
0805
VDDA_A1
Compensation network
Refer to manual.
SRP/SRN compensation network.
Refer to manual.
VCORE_CPU_3V3
VCORE_PPH_3V3
THROTTLE_ENAB
SDA1
SCL1
BATA_CHRG_OK
BAT_PACK_A
bq25708
U74
ADDR 0x68
ADDR 0x48
TMP_CREGC_PROX
SDACORE
SCLCORE
0805 0.1%
GND

Figure 7.5: Schematic for battery charger. There are two battery circuits for parallel battery packs. The circuit does not include per-cell balancing; this is contained in the individual batteries.

or timed out CPU left off.

7.2 System-level I&T Plan

As previously mentioned, the large amount of programming and validation work still needed did not allow for physical layout and routing of a PCBA.

8. Next Steps

The most time-consuming aspect of the project as documented is the research and development work. Despite the four years of work on the project, there are substantial amounts of work and data omitted from the report due to time limitations. During final compilation of this report, seven chapters were omitted, which would have showcased most of the design work, including on physical deliverables. These include detailed subsystem design at the electrical and firmware level, data analysis from operational validation efforts, and in-the-field testing campaigns, of which there were over 20 conducted. This being said, some of the design work is included in the following appendices, which includes code, screenshots and drafts of subsystems, among others.

Beyond this, Wavefront can be a real mission, given 2- to 3-years of intensive, full-time effort to clearly define and vet requirements and procedures to 7120.5F program management requirements.

8.1 Report Work

Much remains to be documented and potentially published to journals. Table 8.1 gives an overview of all subsystems remaining to be documented, and Table **??** shows the component system theoretical designs remaining.

8.2 Research and Development

To generate a very strong mission proposal, work must continue to advance each system's TRL. One of the most important characterizations required is an extended test in a Martian environment exposed to equivalent temperatures,

Table 8.1: Summary of individual subsystems remaining to be documented

System	**Description**
Claire	Controlled, Limited, And Integrated Regulation Electronics - universal electrical power system. A nearly complete schematic was developed. Some of the most basic design summaries are included in a prior chapter.
Serah	Spacecraft Engineering Reference, Attitude, Health - spacecraft engineering instrumentation and scientific reference electronics. A complete system was built for the TechEdSat program as part of a Space Act Agreement, and was subsequently taken up by TechEdSat to be further developed internally.
Etro	Extended Temperature Range Omnibus - temperature monitoring extension of Serah. Etro was also taken up by TechEdSat to be further developed as a multimission temperature monitor.
Shinra	Spacecraft Heating In Nuclear Regulated Apparatus - radioisotope thermoelectric generator and power delivery controller. A very thorough chapter detailing the design of Shinra was outlined, but near the end, it was descoped due to limited time. The Shinra RTG was technically a 4-month diversion near the end of the project, due to the late perception that one would be required for reliable operation. The RTG was extremely conservatively oversized to ensure full containment of the ceramic plutonium fuel, as well as to represent the worst possible case in terms of landable mass, which unfortunately turned out to be a bad design choice. After further iterations of lander hardware and further research into budgetary and political constraints, work short of a report was discontinued.
Cloud	Closed-Loop Ordered Unified Driver - hardware controller for GNC system. The conceptual baseline for the controller was developed, but no hardware was developed by the conclusion of the project.
Tifa	Tracked Intrinsic Fine Attitude - GNC sun and star tracker. Initial evaluation of in-family hardware was performed, and a breadboard version was built. Some of the evaluation code is included in the Appendix.
Snow	Spectrometry of Nuclear and Orbital Whistlers - electromagnetic FFT and nuclear spallation detection and classification. Parts for this system, including avalanche photodiodes (APDs) for classification of nuclear radiation, were purchased. An initial aspect of Snow was fabricated in the PCB as part of Serah to evaluate electromagnetic FFT.

pressures, and insolation. This is required to validate the Nines concept with respect to survivability and functionality of electrical systems without any night time heating.

Another important aspect is to complete the design of Pascal. This includes the robotic arm and hammer mechanism, since deployment of Nines depends on the success of the functionality of this system.

Beyond these two items, an end-to-end test of lower-TRL versions of the complete mission can only be completed with additional funding. Given a thorough effort in completing Nines and Pascal, Midgar and the Wavefront EDL aeroshells can be easily developed, given JPL and Lockheed Martin's extensive experience with Martian landers.

While technically a dead-end, work should continue with development of the Shinra RTG. For missions beyond Mars, RTGs will continue to be a viable - if not the only - option for power generation. Of keen interest is potential prebiotic conditions on Titan. Given a less than 1% insolation level compared to Earth on Titan, and the cryogenic temperatures present, RTGs are a necessity. Smaller RTGs would benefit multiprobe missions, since volume, mass, and consequently budget constraints to any destination beyond Mars are extremely tight.

8.3 Mission Proposal

Once R&D activities and relevant tech demos have been completed to the best of a college's ability, a complete mission proposal will be needed. This process is expected to take 2-3 years with dedicated funding and time. Industry guidance will be needed, and specific SMEs must guide the science to justify the narrative for such a large mission.

9. Conclusion

This report attempted to cover a tremendous amount of ground to develop a complete, NASA Flagship mission to Mars. The initial concept of geophysical and atmospheric science monitoring stations mounted near the summit of an active volcano in Chile grew into a concept for a Mars mission. This concept was shrunk down to a CubeSat as a dedicated mission - effectively a 1U version of Mars Pathfinder. Due to several engineering constraints in the design space, notably power generation to keep warm during Martian nights, a years-long journey began to iterate the design space to an acceptable mission. This continued to grow the scope and budget of this theoretical mission, which necessitated larger literature reviews and new forays into the politics of budgetary allocation in science and engineering. At the end of this project, a Flagship-class mission of $3.5 billion was developed, utilizing one orbital relay/mothership and twelve aeroshells, each containing twelve landers, twelve rovers, and ten nanoprobes. All in all, 193 assets including heatshields and backshells will be deployed to Mars.

Meanwhile, work on hardware continued in the background. Starting with the SEEDS effort, multiple iterations of hardware were conducted in order to miniaturize remote sensing and Internet of Things (IoT) concepts and to evaluate the feasibility of execution on Mars. The final hardware iteration ended with a 52x48x35mm hermetically sealed enclosure, with two PCBs containing instruments and power interfaces. This effort is partly covered in Chapter 5.

In the end, there was so much work that occurred that much of the design effort and data was not included into the r eport. Future efforts would round out the documentation of this project and its deliverables.

References

[1] “Mars Pathfinder Fact Sheet,” Website. Jet Propulsion Laboratory, Caltech„ 1997.

[2] Hubbard, S., Haberle, R., Wercinski, P., Sarver, G., Tauber, M., Lemke, L., and DeVincenzi, D., “Mars Environmental Survey (MESUR) Science Objectives and Mission Description,” Unpublished Report Manuscript. NASA Ames Research Center, Moffett Field, CA 94035. July 19, 1991.

[3] “Scientific instrument diagram of Perseverance rover,” 2015, `http://photojournal.jpl.nasa.gov/jpeg/PIA19672.jpg`.

[4] Zou, Y., Zhu, Y., Bai, Y., Wang, L., Jia, Y., Shen, W., Fan, Y., Liu, Y., Wang, C., Zhang, A., Yu, G., Dong, J., Shu, R., He, Z., Zhang, T., Du, A., Fan, M., Yang, J., Zhou, B., Wang, Y., and Peng, Y., “Scientific objectives and payloads of Tianwen-1, China’s first Mars exploration mission,” *Advances in Space Research*, Vol. 67, No. 2, 2021, pp. 812–823.

[5] Kapurch, S. J. and Rainwater, N. E., *NASA Systems Engineering Handbook*, Office of the Chief Engineer, National Aeronautics and Space Administration, Washington, DC 20001, 2007, NASA SP-2007-6105.

[6] A., N., “NFPA 704: Standard System for the Identification of the Hazard of Materials for Emergency Response,” Tech. rep., National Fire Protection Association, 2017,

https://www.nfpa.org/codes-and-standards/all-codes-and-standards/list-of-codes-and-standards/detail?code=704.

[7] Chul, P., *Nonequilibrium Hypersonic Aerothermodynamics*, Wiley International, Wiley Interscience, 1990.

[8] Hoag, D. G., *The History of Apollo On-Board Guidance, Navigation, and Control*, The Charles Stark Draper Laboratory, Inc., 1976.

[9] Hall, E., "Hugh Blair-Smith's Introduction," *Apollo Guidance Computer History Project*, California Institute of Technology, http://authors.library.caltech.edu/5456/1/hrst.mit.edu/hrs/apollo/public/conference3/blairsmith.htm.

[10] "Mach Scheduling and Thread Interfaces," Tech. rep., Apple Inc., 2013, https://developer.apple.com/library/archive/documentation/Darwin/Conceptual/KernelProgramming/scheduler/scheduler.html.

[11] Fox, O. D., Kutyrev, A. S., Rapchun, D. A., Klein, C. R., and et. al., "Performance and Calibration of H2RG Detectors and SIDECAR ASICs for the RATIR Camera," Tech. rep., NASA Goddard Space Flight Center, 2012, https://ntrs.nasa.gov/api/citations/20120009967/downloads/20120009967.pdf.

[12] Hall, E., *Journey to the Moon: The History of the Apollo Guidance Computer*, Reston, VA, USA. AIAA, 1996, ISBN 1-56347-185-X.

[13] *"nRF52840 Product Specification"*, https://infocenter.nordicsemi.com/pdf/nRF52840_PS_v1.5.pdf.

[14] *"MSP430FR203x Mixed-Signal Microcontrollers"*, https://www.ti.com/lit/ds/symlink/msp430fr2033.pdf.

[15] Hall, E., “Apollo Guidance, Navigation, and Control,” Tech. rep., MIT Charles Stark Draper Laboratory, 1972, http://ibiblio.org/apollo/hrst/archive/1029.pdf.

[16] Wegener, H. A. R., Doig, M. B., Marraffino, P., and Robinson, B., “Radiation Resistant MNOS Memories,” *IEEE Transactions on Nuclear Science*, Vol. 19, No. 6, 1972, pp. 291–298, https://dx.doi.org/10.1109/TNS.1972.4326847.

[17] Logek, B., *History of Semiconductor Engineering*, Springer Science and Business Media, 2007, ISBN 9783640342588.

[18] Spohn, T., “’Undercover’ Mole,” *DLR Blogs*, Deutsches Zentrum fur Luft- und Raumfahrt, 2020, https://www.dlr.de/blogs/en/desktopdefault.aspx/tabid-5893/9577_read-1144/.

[19] Drake, B. G. and Watts, K. D., “Human Exploration of Mars Design Reference Architecture 5.0,” Tech. rep., NASA Johnson Space Center, Houston, TX, 2009, https://ntrs.nasa.gov/api/citations/20160003093/downloads/20160003093.pdf.

[20] Restrepo, C. I., Petro, N. F., Barker, M. K., and Mazarico, E., “Building Lunar Maps for Terrain Relative Navigation and Hazard Detection Applications,” *AIAA SciTech*, 2021, https://ntrs.nasa.gov/citations/20210024816.

[21] Cramer, N., Cellucci, D., Adams, C., Sweet, A., and Hejase, M., “Design and Testing of Autonomous Distributed Space Systems,” *35th Annual Small*

Satellite Conference, 2021, https://ntrs.nasa.gov/api/citations/20210016930/downloads/SmallSat2021.pdf.

[22] Hanson, J., Chartres, J., Sanchez, H., and Oyadomari, K., "The EDSN Intersatellite Communications Architecture," *28th Annual Small Satellite Conference*, 2014, https://ntrs.nasa.gov/api/citations/20160006437/downloads/20160006437.pdf.

[23] *Time History of Events and Macroscale Interactions during Substorms (Explorer 85)*, NASA. Web. http://www.nasa.gov/mission_pages/themis/main/index.html.

[24] Dudukovich, R., LaFuente, B., Hylton, A., and Tomko, B., "A Distributed Approach to High-Rate Delay Tolerant Networking Within A Virualized Environment," *IEEE Cognitive Communications for Aerospace Applications*, edited by IEEE, 2021, https://ntrs.nasa.gov/api/citations/20210014035/downloads/HDTN_CCAA_21_final.pdf.

[25] Ely, T. A., Koch, T., Kuang, D., Lee, K., and Murphy, D., "The Deep Space Atomic Clock Mission," Tech. rep., Jet Propulsion Laboratory, 2012, https://hdl.handle.net/2014/43016.

[26] "Luna 13," Tech. rep., NASA Solar System Exploration - Science Directorate, 2018, https://solarsystem.nasa.gov/missions/luna-13/in-depth/.

[27] Mahmood, A., Hossain, M. M. A., Cavdar, C., and Gidlund, M., "Energy-Reliability Aware Link Optimization for Battery-Powered IoT Devices With Nonideal Power Amplifiers," *IEEE Internet of Things Journal*, Vol. 6, No. 3, 2019, pp. 5058–5067, https://dx.doi.org/10.1109/JIOT.2019.2895228.

[28] Siddiqi, A., *Beyond Earth: A Chronicle of Deep Space Exploration, 1958-2016*, NASA Headquarters, Washington, DC, 2018.

[29] Williams, D. R., "Ranger 7," Tech. rep., NASA Space Science Data Coordinated Archive, 1964, `https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1964-041A`.

[30] Williams, D. R., "Luna 9," Tech. rep., NASA Space Science Data Coordinated Archive, 1966, `https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1966-006A`.

[31] "Luna 9 Radiation Sensor," Tech. rep., NASA Space Science Data Coordinated Archive, 1966, `https://nssdc.gsfc.nasa.gov/nmc/experiment/display.action?id=1966-006A-02`.

[32] Pyle, R., "Fifty Years of Moon Dust: Surveyor 1 was a Pathfinder for Apollo," *NASA JPL*, Jet Propulsion Laboratory, 2017, `https://www.nasa.gov/feature/jpl/fifty-years-of-moon-dust-surveyor-1-was-a-pathfinder-for-apollo`.

[33] "Surveyor 1," Tech. rep., NASA Space Science Data Coordinated Archive, 1966, `https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1966-045A`.

[34] "Surveyor 3," Tech. rep., NASA Space Science Data Coordinated Archive, 1968, `https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1967-035A`.

[35] "Luna 16," Tech. rep., NASA Space Science Data Coordinated Archive, 1970, https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1970-072A.

[36] *Apollo 15 Press Kit*, NASA Headquarters, 1971, https://history.nasa.gov/alsj/a15/A15_PressKit.pdf.

[37] Williams, D. R., "Mariner 4," Tech. rep., NASA Space Science Data Coordinated Archive, 1965, https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1964-077A.

[38] Huntress, W. T. J. and Marov, M. Y., *Soviet Robots in the Solar System: Mission Technologies and Discoveries*, Springer-Praxis, 2011, ISBN 9781441978974.

[39] "Venera 7," Tech. rep., NASA Space Science Data Coordinated Archive, 1970, https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1970-060A.

[40] "Mars 3," Tech. rep., NASA Space Science Data Coordinated Archive, 1971, https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1971-049F.

[41] "Viking 1," Tech. rep., NASA Space Science Data Coordinated Archive, 1975, https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1975-075C.

[42] "Telecommunications and Data Acquisition Systems Support for the Viking 1975 Mission to Mars," Tech. rep., NASA Jet Propulsion Laboratory, 1982, JPL Publication 82-107.

[43] Levin, G. V. and Straat, P. A., “The Case for Extant Life on Mars and Its Possible Detection by the Viking Labeled Release Experiment,” *Astrobiology*, Vol. 16, No. 10, 2016, pp. 798–810, PMID: 27626510.

[44] Klein, H. P., “The Viking biological experiments on Mars,” *Icarus*, Vol. 34, No. 3, 1978, pp. 666–674, `https://www.sciencedirect.com/science/article/pii/0019103578900532`.

[45] “MARS OBSERVER INVESTIGATION REPORT RELEASED,” Press release, Malin Space Science Systems, 1994, `https://www.msss.com/mars/observer/project/mo_loss/nasa_mo_loss.txt`.

[46] Albee, A. L., Arvidson, R. E., Palluconi, F., and Thorpe, T., “Overview of the Mars Global Surveyor mission,” *Journal of Geophysical Research: Planets*, Vol. 106, No. E10, 2001, pp. 23291–23316, `https://doi.org/10.1029/2000JE001306`.

[47] Lyons, D. T., Beerer, J. G., Esposito, P., Johnston, M. D., and Willcockson, W. H., “Mars Global Surveyor: Aerobraking Mission Overview,” *Journal of Spacecraft and Rockets*, Vol. 36, No. 3, 1999, pp. 307–313, `https://doi.org/10.2514/2.3472`.

[48] “Mars Global Surveyor,” Tech. rep., NASA Space Science Data Coordinated Archive, 1996, `https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1996-062A`.

[49] Nieberding, J. and Ross, L., “Mission Success First: Lessons Learned: Lecture #94,” Lecture slides. Aerospace Engineering Associates, LLC, Bay Village, OH, 2006.

[50] McCuiston, D., "Mars Exploration Group: Mars - the search for life," Conference slides. Mars Exploration Program, NASA, 2009, `http://mepag.jpl.nasa.gov/meeting/mar-09/02_MEPAG_McCuistion_Mar_09.pdf`.

[51] "Mars Pathfinder," Tech. rep., NASA Space Science Data Coordinated Archive, 1997, `https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=1996-068A`.

[52] "Sojourner Rover, Mars Pathfinder Rover," Tech. rep., NASA Space Science Data Coordinated Archive, 1996, `https://nssdc.gsfc.nasa.gov/nmc/spacecraft/display.action?id=MESURPR`.

[53] "NASAfacts: Radioisotope Power Systems for NASA," Web. Jet Propulsion Laboratory, Caltech, 2009, `https://web.archive.org/web/20120311002411/http://www.jpl.nasa.gov/news/fact_sheets/radioisotope-power-systems.pdf` - original has been taken down.

[54] Bechtel, R., "Radioisotope Missions," U.S. Department of Energy, 2011, `https://web.archive.org/web/20120201232852/http://www.jpl.nasa.gov/msl/pdf/MMRTG_RyanBechtel_DOE.pdf`.

[55] Ritz, F. and Peterson, C. E., "Multi-Mission Radioisotope Thermoelectric Generator (MMRTG) Program Overview," *Institute for Electrical and Electronic Engineers*, Vol. 4, No. 1595, 2004, `https://web.archive.org/web/20111216101915/http://trs-new.jpl.nasa.gov/dspace/bitstream/2014/38246/1/04-0191.pdf`.

[56] Shure, L. I. and Schwartz, H. J., "Survey of Electric Power Plants for Space Applications," Tech. rep., NASA Lewis Research Center, Cleveland, Ohio, USA,

1965, https://web.archive.org/web/20100525084704/https://ntrs.nasa.gov/archive/nasa/casi.ntrs.nasa.gov/19660005486_1966005486.pdf.

[57] "InSight Seismometer," 2018, https://mars.nasa.gov/insight/mission/instruments/seis/.

[58] Banerdt, W. B., Smrekar, S. E., Banfield, D., Giardini, D., Golombek, M., Johnson, C. L., Lognonné, P., Spiga, A., and Spohn, T., "Initial results from the InSight mission on Mars," *Nature Geoscience*, Vol. 13, 2020, https://doi.org/10.1038/s41561-020-0544-y.

[59] Banfield, D., Spiga, A., Newman, C., and et al., "The atmosphere of Mars as observed by InSight," *Nature Geoscience*, Vol. 13, 2020, https://doi.org/10.1038/s41561-020-0534-0.

[60] Albee, A., Leising, C., Battel, S., MacPherson, D., Casani, J., and Whetsel, C., "Report on the Loss of the Mars Polar Lander and Deep Space 2 Missions," Tech. rep., NASA Jet Propulsion Laboratory, Pasadena, CA, 2000, https://web.archive.org/web/20151213144413/ftp://ftp.hq.nasa.gov/pub/pao/reports/2000/2000_mpl_report_1.pdf.

[61] Young, T., "Mars Program Independent Assessment Team Summary Report," Tech. rep., United States House Science and Technology Committee, 2000, https://spaceref.com/press-release/testimony-of-thomas-young-chairman-of-the-mars-program-independent-assessment

[62] Russell, P., Carmen, D., Marsh, C., Reddy, T., Bugga, R., Deligiannis, F., and Frank, H., "Development of a lithium/thionyl chloride battery for the Mars Microprobe Program," *Thirteenth Annual Battery Conference on Applications and Advances. Proceedings of the Conference*, 1998, pp. 341–346.

[63] Holt, J. W., Safaeinili, A., Plaut, J. J., Head, J. W., Phillips, R. J., Seu, R., Kempf, S. D., and Choudhary, P., “Radar Sounding Evidence for Buried Glaciers in the Southern Mid-Latitudes of Mars,” *Science*, Vol. 322, No. 5905, 2008, https://doi.org/10.1126/science.1164246.

[64] Orosei, R. and et al., “Radar evidence of subglacial liquid water on Mars,” *Science*, Vol. 361, No. 6401, 2018, https://doi.org/10.1126/science.aar7268.

[65] Laboratory, N. J. P., “Ingenuity Mars Helicopter Press Kit,” Web, 2021, https://www.jpl.nasa.gov/news/press_kits/mars_2020/download/ingenuity_landing_press_kit.pdf.

[66] Balaram, B., Canham, T., Duncan, C., Golombek, M., Grip, H. F., Johnson, W., Maki, J., Quon, A., Stern, R., and Zhu, D., “Mars Helicopter Technology Demonstrator,” *AIAA SciTech Forum*, Vol. 6, No. 23, 2018, DOI 10.2514/6.2018-0023.

[67] Agle, D., “NASA’s Ingenuity in Contact With Perseverance Rover After Communications Dropout,” JPL Ingenuity Status Updates, 2022, https://mars.nasa.gov/technology/helicopter/status/379/nasas-ingenuity-in-contact-with-perseverance-rover-after-communications-dropo

[68] Brown, T., “Perseverance’s Four Legged Companion is Ready,” JPL Ingenuity Status Updates, 2023, https://mars.nasa.gov/technology/helicopter/status/441/perseverances-four-legged-companion-is-ready/.

[69] Brown, T., “The Race Is On,” JPL Ingenuity Status Updates, 2023, https://mars.nasa.gov/technology/helicopter/status/450/the-race-is-on/.

[70] Bapst, J., Tzanetos, T., and Withrow-Maser, S., "Helicopters on Mars: Technology Demonstration to Future Science Missions," Tech. rep., NASA Jet Propulsion Laboratory, 2021, http://fiso.spiritastro.net/telecon19-21/Bapst-Tzanetos-WithrowMaser_9-29-21/Bapst-Tzanetos-WithrowMaser_9-29-21.pdf.

[71] Pipenberg, B. T., Langberg, S. A., Tyler, J. D., and Keennon, M. T., "Conceptual Design of a Mars Rotorcraft for Future Sample Fetch Missions," *2022 IEEE Aerospace Conference (AERO)*, 2022, pp. 01–14.

[72] Wu, N., "Next Stop - Mars: China aims to send rover to Red Planet within six years," South China Morning Post, 2014, http://www.scmp.com/news/china/article/1539568/next-stop-mars-china-aims-send-rover-red-planet-within-six-years.

[73] Jones, A., "Here's What You Need to Know About China's Mars Rover," IEEE Spectrum, 2021, hhttps://spectrum.ieee.org/what-you-need-to-know-about-china-mars-rover-tianwen-1.

[74] Cheatwood, F. M., Bose, D., Karlgaard, C. D., Kuhl, C. A., Santos, J. A., and Wright, M. J., "Mars Science Laboratory (MSL) Entry, Descent, and Landing Instrumentation (MEDLI): Complete Flight Data Set," Tech. rep., NASA Ames Research Center, 2014, https://ntrs.nasa.gov/citations/20140016393/.

[75] Thornton, J. M., Meurisse, J. B. E., Prabhu, D. K., Borner, A. P., Monk, J. D., and Cruden, B. A., "ANALYSIS OF THE MSL/MEDLI ENTRY DATA WITH COUPLED CFD AND MATERIAL RESPONSE," Tech. rep., NASA Ames Research Center, 2021, https://ntrs.nasa.gov/api/citations/20210014103/.

[76] Way, D., Dutta, S., Zumwalt, C., and De León, S. S., "EDL Simulation Results for the Mars 2020 Landing Site Safety Assessment," *2020 IEEE Aerospace Conference*, 2020, pp. 1–5, `https://ieeexplore.ieee.org/document/9172525`.

[77] Wu, B., Dong, J., Wang, Y., Rao, W., Sun, Z., Li, Z., Tan, Z., Chen, Z., Wang, C., Liu, W. C., Chen, L., Zhu, J., and Li, H., "Landing Site Selection and Characterization of Tianwen-1 (Zhurong Rover) on Mars," *Journal of Geophysical Research: Planets*, Vol. 127, No. 4, 2022, pp. e2021JE007137, `https://agupubs.onlinelibrary.wiley.com/doi/abs/10.1029/2021JE007137`.

[78] Huang, X., Li, M., Wang, X., Jinchang, H., Guo, M., Liu, W., Wang, Y., and Xu, L., "The Tianwen-1 Guidance, Navigation, and Control for Mars Entry, Descent, and Landing," *Space Science and Technology*, Vol. 2021, No. 9846185, 2021, `https://spj.science.org/doi/10.34133/2021/9846185?permanently=true`.

[79] Mallapaty, S., "What's happened to China's first Mars rover?" *Nature Communications*, , No. 610, 2023, `https://www.nature.com/articles/d41586-023-00111-3`.

[80] Mazhari, A. A., Ticknor, R., Swei, S., Krześniak, S., and Teodorescu, M., "Automated Characterization and Testing of Additive Manufacturing (ATCAM)," *Journal of Materials Engineering and Performance*, Vol. 30, 2021, pp. 6862–6873, `https://dx.doi.org/10.1007/s11665-021-06042-2`.

[81] Krześniak, S. and Papadopoulos, P., "Martian Microprobe Entry, Descent, and Landing System," *Interplanetary Probe Workshop*, 2021.

[82] Quinlan, G. D. and Tremaine, S., "Symmetric Multistep Methods for the Numerical Integration of Planetary Orbits," *Astronomical Journal*, Vol. 100, No. 5, 1990.

[83] Leroy, R. and Leroy, P., "Principia," GitHub, 2023, `https://github.com/mockingbirdnest/Principia`.

[84] "HORIZONS Service," NASA Jet Propulsion Laboratory, 2023, `https://ssd.jpl.nasa.gov/horizons/`.

[85] "JOINT STRIKE FIGHTER - DOD Actions Needed to Further Enhance Restructuring and Address Affordability Risks," Tech. rep., U.S. Government Accountabiilty Office, 2012, GAO Report 12-437.

[86] "Galileo Final Report," Tech. rep., NASA Jet Propulsion Laboratory, Pasadena, CA, USA, 1991, NASA JPL Report D-28516, Volumes 1-3.

[87] "NASA PROCEDURAL REQUIREMENTS: NASA RESEARCH AND TECHNOLOGY PROGRAM AND PROJECT MANAGEMENT REQUIREMENTS," NASA Headquarters, 2023, `https://nodis3.gsfc.nasa.gov/displayDir.cfm?t=NPR&c=7120&s=8A`.

[88] Serrano, J. E., "Commerce, Justice, Science, and Related Agencies Appropriations for 2020," U.S. House of Representatives Subcommittee on Appropriations, 2019, Section DEXP-2.

[89] "U.S. Department of Defense Budget Overview," Comptroller, United States Department of Defense, 2023, `https://comptroller.defense.gov/Portals/45/Documents/defbudget/FY2024/FY2024_Budget_Request_Overview_Book.pdf`.

[90] Biden, J. R., “Budget of the U.S. Government for Fiscal Year 2024,” U.S. White House, 2023, https://www.whitehouse.gov/wp-content/uploads/2023/03/budget_fy2024.pdf.

[91] “Ingenuity Spots Rover Tracks During Ninth Flight,” NASA, 2021, https://mars.nasa.gov/resources/26046.

[92] “NASA FY2024 Budget Request,” NASA Headquarters, 2023, https://www.nasa.gov/sites/default/files/atoms/files/fiscal_year_2024_nasa_budget_summary.pdf.

[93] Smith, D. H., *Review and Assessment of Planetary Protection Policy Development Processes*, National Acadamies Press, 500 Fifth Street, NW, Washington, D.C. 20001, 2018.

[94] Nilekani, N., Dixon, H., Chaibong, H., and Sherman, W., *World War Web*, Foreign Affairs, 58 E. 68th Street, New York, NY 10065, 2018.

[95] Aspaturian, V. V., Hill, C., Joffe, J., Macridis, R. C., Odom, D., Roett, R., Safran, N., Scalapino, R. A., Sundelius, B., White, B., Whiting, A. S., and Wright, S., *Foreign Policy in World Politics, Eighth Edition*, Prentice-Hall, 1991.

[96] Smith, N., “7000-7999 Program Formulation,” NASA Online Directives Information System, 2023, https://nodis3.gsfc.nasa.gov/lib_docs.cfm?range=7.

[97] Smith, N., “8000-8999 Program Management,” NASA Online Directives Information System, 2023, https://nodis3.gsfc.nasa.gov/lib_docs.cfm?range=8.

[98] Hang, W. and Erickson, S., “About Us,” Mars Exploration Program Analysis Group, Jet Propulsion Laboratory, Pasadena, CA, 2021, `https://mepag.jpl.nasa.gov/about.cfm`.

[99] Blair, P. D., “The evolving role of the US National Academies of Sciences, Engineering, and Medicine in providing science and technology policy advice to the US government,” *Palgrave Communications*, Vol. 1, No. 2, 2016, `https://doi.org/10.1057%2Fpalcomms.2016.30`.

[100] Banfield, D., “Mars Science Goals, Objectives, Investigations, and Priorities: 2020 Version,” Tech. rep., Mars Exploration Program Analysis Group, Jet Propulsion Laboratory, Pasadena, CA, 2020, `https://mepag.jpl.nasa.gov/reports/MEPAGGoals_2020_MainText_Final.pdf`.

[101] Korablev, O., Vanadele, A. C., Montmessin, F., and Federova, A. A., “No detection of methane on Mars from early ExoMars Trace Gas Orbiter observations,” *Nature Communications*, Vol. 2019, No. 568, 2019, `https://www.nature.com/articles/s41586-019-1096-4#citeas`.

[102] Yung, Y. L., Chen, P., Nealson, K., Atreya, S., Beckett, P., and Blank, J. G., “Methane on Mars and Habitability: Challenges and Responses,” *Astrobiology*, Vol. 10, No. 18, 2018, `https://www.ncbi.nlm.nih.gov/pmc/articles/PMC6205098/`.

[103] Etiope, G. and Oehler, D. Z., “Methane spikes, background seasonality and non-detections on Mars: A geological perspective,” *Elsevier Planetary and Space Science*, Vol. 168, No. 2, 2019, `https://www.sciencedirect.com/science/article/abs/pii/S0032063318303404?via%3Dihub`.

[104] Budhu, M., *Soil Mechanics and Foundation, 3rd Edition*, John Wiley and Sons, Inc., Hoboken, New Jersey, USA, 2011.

[105] Tikhonov, A. N., "O edinstvennosti resheniya zadachi elektrorazevedki (in Russian)," *Doklady Akademii Nauk SSSR*, Vol. 69, No. 6, 1949.

[106] Brown, B. H., "Electrical impedance tomography (EIT): A review," *Journal of Medical Engineering and Technology*, Vol. 27, No. 3, 2003, `https://pubmed.ncbi.nlm.nih.gov/12775455`.

[107] Hassler, D. M., Zeitlin, C., Wimmer-Schwingruber, R. F., Ehrsmann, B., Rafin, S., Eigenbrode, J. L., Brinza, D. E., Weigle, G., Böttcher, S., and Böhm, E., "Mars' Surface Radiation Environment Measured with the Mars Science Laboratory's Curiosity Rover," *Science*, Vol. 343, No. 6169, 2013, `https://pubmed.ncbi.nlm.nih.gov/24324275`.

[108] Parker, C. and Smith, N., "NODIS Library," NASA Online Directives Information System, NASA Goddard Spaceflight Center, Cleveland, OH, 2023, `https://nodis3.gsfc.nasa.gov/main_lib.cfm`.

[109] "1300-Series Spacecraft Platform," Space Systems Loral/Maxar, 2010, Web. `https://web.archive.org/web/20100210192715/http://www.ssloral.com/html/products/1300.html`.

[110] Humphries, J. and Colwell, B., Tech. rep., United States Air Force Space Command.

[111] "Breakthrough Starshot: Concept," Breakthrough Initiatives, 2016, `http://breakthroughinitiatives.org/Concept/3`.

[112] Cabrol, N. A., Feister, U., Häder, D.-P., Piazena, H., Grin, E. A., and Klein, A., "Record solar UV irradiance in the tropical Andes," *Frontiers in Environmental Science*, Vol. 2, No. 19, 2014, DOI: 10.3389/fenvs.2014.00019.

[113] Cabrol, N. A., Grin, E. A., Chong, G., Minkley, E., Hock, A. N., and Yu, Y., "The High-Lakes Project," *Journal of Geophysical Research*, Vol. 114, No. 6, 2009, `https://dx.doi.org/10.1029/2008JG000818`.

[114] "Bloomberg: Santiago despierta en la devestacion," El Mostrador, 2019, `https://web.archive.org/web/20191020042309/https://www.elmostrador.cl/dia/2019/10/19/bloomberg-santiago-despierta-en-la-devastacion/`.

[115] Karlgaard, C. D., Korzun, A. M., Schoenenberger, M., Bonfiglio, E. P., Kass, D. M., and Grover, M. R., "Mars InSight Entry, Descent, and Landing Trajectory and Atmosphere Reconstruction," Tech. rep., NASA Langley Research Center, 2020.

[116] Brandis, A. M., Saunders, D. A., Johnston, C. O., Cruden, B. A., and White, T. R., "Radiative Heating on the After-Body of Martian Vehicles," *Journal of Thermophysics and Heat Transfer*, Vol. 34, No. 1, 2019, `https://arc.aiaa.org/doi/10.2514/1.T5613`.

[117] Anderson, J. D., *Hypersonic and High-temperature Gas Dynamics*, American Institute of Aeronautics and Astronautics, Reston, VA, 2006.

[118] Brandis, A. M., White, T. R., Saunders, D. A., Hill, J. P., and Johnston, C. O., "Simulation of the Schiaparelli Entry and Comparison to Aerosciences Flight Data," *Journal of Spacecraft and Rockets*, Vol. 59, No. 1, 2022, `https://dx.doi.org/10.2514/1.A35049`.

[119] Spalart, P. and Allmaras, S., "A one-equation turbulence model for aerodynamic flows," Tech. rep., American Institute of Aeronautics and Astronautics, 1992, Report AIAA-92-0439.

Appendices

1. MATLAB (R) 2D hypersonic propagator code

```
clearvars;
clc;
% Written ca. 2019.
% Stanley Krzesniak
% Data based on:
% NASA research papers (see References for this)
% NASA Chemical Equilibrium with Applications - MATLAB
% NASA Mars-GRAM 2010
% Modified Newtonian Theory
% Local Slope Inclination, valid from Mach 8 and above

%% static and inital definitions
decelDragForce = zeros(27,1);
decelDragForce(27) = 0;
vehicleMass = 80000; %kg
marsGrav = 3.72076; %m/s^2
flightPathAngle = zeros(28,1);
flightPathAngle(28) = -1; %degrees
vehicleRadius = 23; %meters, yes, this is a bicc boi
surfAngle = 20; %degrees, average surface inclination
deltaVerticalDist = 5000; %meters
u1 = zeros(28,1);
u1(28) = 5600; %inital entry interface total velocity
u1_x = zeros(28,1);
u1_x(28) = u1(28).*cosd(flightPathAngle(28)); %EI horiz
u1_y = zeros(28,1);
u1_y(28) = u1(28).*sind(flightPathAngle(28)); %EI vert
dragAcc = zeros(27,1);
dragAcc(27) = 0;
dragAcc_x = zeros(27,1);
dragAcc_y = zeros(27,1);

% For kinetic theory of gases
boltzmann = 1.3806e-23;
diaCO = 376e-12;
diaCO2 = 330e-12;
diaO2 = 346e-12;
diaAr = 340e-12;
diaN2 = 364e-12;
p2 = zeros(27,1);
p2(27) = 1e-2; % initial condition, Pa (just for a guess to get it started)
```

```
h2 = zeros(27,1);
h2(27) = 4000; % initial condition, kJ/kg
controlVolumeDepth = 1; %meters
L = ((pi*vehicleRadius*(vehicleRadius*cosd(surfAngle)+vehicleRadius)-(pi*vehicleRadius.^2))*controlVolumeDepth).^(1/3); % Characteristic length

% CEA convergence solver parameters
t2Stepping = 20; % Kelvin, smaller means more accurate convergence but longer computation time
rho2Stepping = [12000E-8;16000E-8;35000E-8;540000E-8;800000E-8;600000E-8;400000E-8;120000E-8;72000E-8;28000E-8;14400E-8;9600E-8;4800E-8;2400E-8;1200E-8;720E-8;360E-8;180E-8;164E-8;128E-8;64E-8;32E-8;16E-8;8E-8;4E-8;2E-8;1E-8];
t2 = zeros(27,1);
rho2 = [1.5E-01;8.7E-02;6.5E-03;4.5E-03;3.3E-3;3.2E-3;3.1E-3;3.09E-3;...
  3.08E-03;2.33E-03;1.37E-03;7.62E-04;4.12E-04;2.43E-04;1.66E-04;...
  1.04E-04;5.87E-05;2.99E-05;1.45E-05;6.72E-06;3.08E-06;1.42E-06;...
  6.63E-07;3.15E-07;1.51E-07;7.25E-08;3.72E-08]; %initial guesses to converge faster
rhoConvgError = zeros(27,1);
rhoConvgError_old = zeros(27,1);
h2ConvgError = zeros(27,1);
h2ConvgError_old = zeros(27,1);
rhoConvgError(27) = 100; % arb
rhoConvgError_old(27) = 100; %arb
h2ConvgError(27) = 100; % arb
iterationTimeout = 25;
t2GuessNew = 0; % for self-learning guess - refines based on the previous converged temperature

% OUTPUT MATRICES
u2 = zeros(27,1); %preallocate for output
m2 = zeros(27,1); %preallocate for output

% Atmospheric Properties from Mars-GRAM 2010
Ar =  [1.5860;1.5890;1.5900;1.5890;1.5890;1.5880;1.5880;1.5870;1.5870;1.5860;1.5850;1.5850;1.5840;1.5830;1.5830;1.5820;1.5810;1.5810;1.6300;1.6200;1.6070;1.5860;1.5750;1.5570;1.5460;1.5440;1.5510];
CO =  [0.0670000000000000;0.0680000000000000;0.0680000000000000;0.0680000000000000;0.0670000000000000;0.0670000000000000;0.0670000000000000;0.0670000000000000;0.0670000000000000;0
```

```
.0670000000000000;0.0670000000000000;0.0670000000000000;0
.0670000000000000;0.0670000000000000;0.0670000000000000;0
.0670000000000000;0.0670000000000000;0.0670000000000000;0
.163000000000000;0.256000000000000;0.348000000000000;0
.438000000000000;0.530000000000000;0.619000000000000;0
.710000000000000;0.805000000000000;0.919000000000000];
CO2 = [96.0840000000000;96.2900000000000;96.3660000000000;96
.3800000000000;96.3850000000000;96.3880000000000;96.3890000000000
;96.3910000000000;96.3920000000000;96.3930000000000;96
.3950000000000;96.3960000000000;96.3980000000000;96.3990000000000
;96.4010000000000;96.4020000000000;96.4040000000000;96
.4060000000000;96.4070000000000;96.4090000000000;96.4140000000000
;96.4210000000000;96.4470000000000;96.4500000000000;96
.4500000000000;96.4510000000000;96.4510000000000];
kmMOLA = [
-5;0;5;10;15;20;25;30;35;40;45;50;55;60;65;70;75;80;85;90;95;100;105;110;115;12
];
N2 = [1.85400000000000;1.85700000000000;1.85800000000000;1
.85700000000000;1.85700000000000;1.85600000000000;1.85500000000000
;1.85500000000000;1.85400000000000;1.85300000000000;1
.85300000000000;1.85200000000000;1.85100000000000;1.85000000000000
;1.84900000000000;1.84900000000000;1.84800000000000;1
.84700000000000;1.90800000000000;1.89800000000000;1.88600000000000
;1.86400000000000;1.85300000000000;1.83500000000000;1
.82500000000000;1.82500000000000;1.90900000000000];
O2 = [0.100000000000000;0.100000000000000;0.100000000000000;0
.100000000000000;0.100000000000000;0.100000000000000;0
.100000000000000;0.100000000000000;0.100000000000000;0
.100000000000000;0.100000000000000;0.100000000000000;0
.100000000000000;0.100000000000000;0.100000000000000;0
.100000000000000;0.100000000000000;0.100000000000000;0
.109000000000000;0.115000000000000;0.121000000000000;0
.126000000000000;0.132000000000000;0.137000000000000;0
.143000000000000;0.149000000000000;0.159000000000000];
P1 = [895;567;358;216;128;73.7000000000000;41.6000000000000;22
.9000000000000;12.3000000000000;6.52000000000000;3.39000000000000;1
.73000000000000;0.920000000000000;0.537000000000000;0
.361000000000000;0.217000000000000;0.116000000000000;0
.0563000000000000;0.0262000000000000;0.0116000000000000;0
.00517000000000000;0.00232000000000000;0.00107000000000000;0
.000506000000000000;0.000243000000000000;0.000121000000000000;6
.24000000000000e-05];
rho1_avg = [0.0196000000000000;0.0132000000000000;0
.00897000000000000;0.00575000000000000;0.00359000000000000;0
.00219000000000000;0.00131000000000000;0.000772000000000000;0
.000437000000000000;0.000248000000000000;0.000135000000000000;6
```

```
.90000000000000e-05;3.48000000000000e-05;1.94000000000000e-05;1
.30000000000000e-05;7.92000000000000e-06;4.33000000000000e-06;2
.16000000000000e-06;1.02000000000000e-06;4.61000000000000e-07;2
.06000000000000e-07;9.24000000000000e-08;4.24000000000000e-08;1
.97000000000000e-08;9.16000000000000e-09;4.31000000000000e-09;2
.09000000000000e-09];
T1 = [241.800000000000;227.800000000000;211;198.700000000000;188
.300000000000;178.100000000000;168;157.100000000000;149
.400000000000;139.300000000000;133.200000000000;132.400000000000
;140.400000000000;146.600000000000;147;144.600000000000;141
.200000000000;133.800000000000;128.600000000000;126.300000000000
;125.700000000000;126.100000000000;127.200000000000;129
.400000000000;133.400000000000;140.800000000000;149];

Cp_max_aero = zeros(27,1);
Minf = zeros(27,1);
p02_p1 = zeros(27,1);
a1 = zeros(27,1);
coeffPressureLower = zeros(27,1);
coeffPressureUpper = zeros(27,1);
Kn = zeros(27,1);

% define capsule in 2D (heat shield only because hypersonic
assumes zero backshell pressure)
capsule_2d = [0.57566 0.57914 0.58168 0.58334 0.58417 0.58423 0
.58358...
  0.58227 0.58037 0.57792 0.57499 0.57163 0.56791 0.56387 0
.55957...
  0.55508 0.55044 0.54572 0.54097 0.53626 0.53162 0.52710 0
.52270...
  0.51840 0.51420 0.51012 0.50613 0.50224 0.49844 0.49474 0
.49113...
  0.48761 0.48418 0.48082 0.47755 0.47435 0.47123 0.46818 0
.46520...
  0.46229 0.45944 0.45665 0.45391 0.45121 0.44855 0.44591 0
.44330...
  0.44070 0.43810 0.43550 0.43290 0.43028 0.42763 0.42496 0
.42224...
  0.41949 0.41668 0.41381 0.41087 0.40786 0.40478 0.40165 0
.39850...
  0.39534 0.39220 0.38909 0.38605 0.38307 0.38020 0.37744 0
.37483...
  0.37237 0.37009 0.36802 0.36617 0.36456 0.36322 0.36216 0
.36140...
  0.36216 0.36322 0.36456 0.36617 0.36802 0.37009 0.37237 0
.37483...
```

```
  0.37744 0.38020 0.38307 0.38605 0.38909 0.39220 0.39534 0
.39850...
  0.40165 0.40478 0.40786 0.41087 0.41381 0.41668 0.41949 0
.42224...
  0.42496 0.42763 0.43028 0.43290 0.43550 0.43810 0.44070 0
.44330...
  0.44591 0.44855 0.45121 0.45391 0.45665 0.45944 0.46229 0
.46520...
  0.46818 0.47123 0.47435 0.47755 0.48082 0.48418 0.48761 0
.49113...
  0.49474 0.49844 0.50224 0.50613 0.51012 0.51420 0.51840 0
.52270...
  0.52710 0.53162 0.53626 0.54097 0.54572 0.55044 0.55508 0
.55957...
  0.56387 0.56791 0.57163 0.57499 0.57792 0.58037 0.58227 0
.58358...
  0.58423 0.58417 0.58334 0.58168 0.57914 0.57566; 0.00000  0
.10693...
  0.20322  0.28704  0.35914  0.42026  0.47115  0.51255  0.54521...
  0.56988  0.58729  0.59821  0.60337  0.60351  0.59939  0.59175...
  0.58134  0.56890  0.55518  0.54092  0.52685  0.51333  0.50036...
  0.48790  0.47593  0.46442  0.45334  0.44267  0.43238  0.42244...
  0.41282  0.40351  0.39446  0.38566  0.37707  0.36868  0.36044...
  0.35234  0.34435  0.33645  0.32860  0.32080  0.31305  0.30535...
  0.29769  0.29008  0.28251  0.27498  0.26748  0.26002  0.25259...
  0.24519  0.23781  0.23046  0.22313  0.21583  0.20854  0.20127...
  0.19401  0.18676  0.17950  0.17219  0.16478  0.15724  0.14953...
  0.14160  0.13341  0.12492  0.11609  0.10688  0.09725  0.08715...
  0.07655  0.06540  0.05366  0.04129  0.02825  0.01450  0.00000...
  -0.01450 -0.02825 -0.04129 -0.05366 -0.06540 -0.07655 -0
.08715...
  -0.09725 -0.10688 -0.11609 -0.12492 -0.13341 -0.14160 -0
.14953...
  -0.15724 -0.16478 -0.17219 -0.17950 -0.18676 -0.19401 -0
.20127...
  -0.20854 -0.21583 -0.22313 -0.23046 -0.23781 -0.24519 -0
.25259...
  -0.26002 -0.26748 -0.27498 -0.28251 -0.29008 -0.29769 -0
.30535...
  -0.31305 -0.32080 -0.32860 -0.33645 -0.34435 -0.35234 -0
.36044...
  -0.36868 -0.37707 -0.38566 -0.39446 -0.40351 -0.41282 -0
.42244...
  -0.43238 -0.44267 -0.45334 -0.46442 -0.47593 -0.48790 -0
.50036...
  -0.51333 -0.52685 -0.54092 -0.55518 -0.56890 -0.58134 -0
```

```
.59175...
  -0.59939 -0.60351 -0.60337 -0.59821 -0.58729 -0.56988 -0.54521...
  -0.51255 -0.47115 -0.42026 -0.35914 -0.28704 -0.20322 -0.10693...
  0.00000];

%% CEA + Mars-GRAM > newtonian theory and rarified gas dynamics

for ai = length(kmMOLA):-1:1
  %cearun1 = CEA('prob','TP','p,bar',P1(ai)/10000,'t,K',T1(ai),'reac',...
  %    'fuel','na','CO2','wt%',CO2(ai),'na','N2','wt%',N2(ai),'na',...
  %    'Ar','wt%',Ar(ai),'na','O2','wt%',O2(ai),'na','CO','wt%',...
  %    CO(ai),'end'); % CEArun for accurate gas Cp at every altitude
  gammaMarsStatic = 1.3319;
  h1 = (0.7677).*T1(ai); % assuming still air
  p2(ai) = P1(ai)+rho1_avg(ai)*u1(ai+1).^2*(1-(rho1_avg(ai)./rho2(ai))); % behind shock, guess
  h2(ai) = h1+(u1(ai+1).^2/2)*(1-(rho1_avg(ai)./rho2(ai)).^2)/1000; % behind shock, guess
  if rho1_avg(ai)/rho2(ai) >= 1
    error(message("Error:␣rho2␣is␣smaller␣than␣rho1.␣Halting."));
  end
  % convergence parameters BEGIN
  t2(ai) = 8000; %init guess
  rhoConvgError(ai) = 100; % arb
  rhoConvgError_old(ai) = 100; % arb
  rhoPrecision = 1e-2;
  % convergence parameters END
  rho_run = 0; % number of times CEA has run for density convergence
  iterationTotal = 0; %total number of times CEA has run
  iterationTotalAi = 0; %total number of times CEA has run for the current altitude
  while rhoConvgError(ai) >= rhoPrecision || rhoConvgError(ai) <= -rhoPrecision %medium precision solver
    if ai < 8 %quick and dirty condition to optimize calculations
      t2Stepping = 5;
      t2(ai) = 3000;
    end
    h2ConvgError(ai) = 60; % arb
    h2ConvgError_old(ai) = 60; % arb
    cearun2 = CEA('prob','TP','p,bar',p2(ai)/10000,'t,K',t2(ai),'reac',...
      'fuel','na','CO2','wt%',CO2(ai),'na','N2','wt%',N2(ai),'na',...
      'Ar','wt%',Ar(ai),'na','O2','wt%',O2(ai),'na','CO','wt%',...
      CO(ai),'end');
      iterationTotal = iterationTotal+1;
      iterationTotalAi = iterationTotalAi+1;
      h2_run = 0;
      while h2ConvgError(ai) >= 1e-2 || h2ConvgError(ai) <= -1e-2 %requesting very high precision
        if h2_run >= iterationTimeout && sign(h2ConvgError(ai)) > 0
           fprintf('Unable␣to␣converge␣a␣solution␣in␣%i␣iterations,␣increasing␣temperature...\n',iterationTimeout);
           t2GuessNew =
           t2(ai) = t2(ai)-(t2Stepping*iterationTimeout-1)+20;
           h2_run = 0;
           h2convgError(ai) = 60;
           h2ConvgError_old(ai) = 60;
        elseif h2_run >= iterationTimeout && sign(h2ConvgError(ai)) > 0
           fprintf('Unable␣to␣converge␣a␣solution␣in␣%i␣iterations,␣reducing␣temperature...\n',iterationTimeout);
           t2(ai) = t2(ai)-(t2Stepping*iterationTimeout-1)-20;
           h2_run = 0;
           h2convgError(ai) = 60;
           h2ConvgError_old(ai) = 60;
        end
        cearun2 = CEA('prob','TP','p,bar',p2(ai)/10000,'t,K',t2(ai),'reac',...
          'fuel','na','CO2','wt%',CO2(ai),'na','N2','wt%',N2(ai),'na',...
          'Ar','wt%',Ar(ai),'na','O2','wt%',O2(ai),'na','CO','wt%',...
          CO(ai),'end');
        iterationTotal = iterationTotal+1;
        iterationTotalAi = iterationTotalAi+1;
        h2ConvgError(ai) = 1-(cearun2.output.enthalpy/(h2(ai)));
        fprintf("h2␣coverr:␣%f,␣h2␣iteration:␣%i\n",h2ConvgError(ai),h2_run);
        h2ConvgErrorSign = sign(h2ConvgError_old(ai)/h2ConvgError(ai));
        % Check last convergence case:
        if h2ConvgError_old(ai) > h2ConvgError(ai) && h2ConvgErrorSign >= 0
```

```
          %guess higher:
          t2(ai) = abs(t2(ai) + t2Stepping);
        elseif h2ConvgError_old(ai) < h2ConvgError(ai) && h2ConvgErrorSign >= 0
          %guess lower:
          t2(ai) = abs(t2(ai) - t2Stepping);
        elseif h2ConvgError_old(ai) > h2ConvgError(ai) && h2ConvgErrorSign == -1
          % if negative number do this
          t2(ai) = abs(t2(ai) - t2Stepping);
        elseif h2ConvgError_old(ai) < h2ConvgError(ai) && h2ConvgErrorSign == -1
          % also if negative number do this
          t2(ai) = abs(t2(ai) + t2Stepping);
        end
        h2ConvgError_old(ai) = h2ConvgError(ai)-1e-17; %Deadlock if equal to each other for any reason
        h2_run=h2_run+1; % add 1
      end
    rhoConvgError(ai) = 1-(rho2(ai)./cearun2.output.density);
    fprintf("rho␣coverr:␣%f,␣rho2␣iteration:␣%i\n", rhoConvgError(ai),rho_run);
    rhoConvgErrorSign = sign(rhoConvgError_old(ai)/rhoConvgError(ai));
    if rhoConvgError_old(ai) > rhoConvgError(ai) && rhoConvgErrorSign > 0
      %guess higher:
      rho2(ai) = rho2(ai) + rho2Stepping(ai);
    elseif rhoConvgError_old(ai) < rhoConvgError(ai) && rhoConvgErrorSign > 0
      %guess lower:
      rho2(ai) = rho2(ai) - rho2Stepping(ai);
    elseif rhoConvgError_old(ai) > rhoConvgError(ai) && rhoConvgErrorSign == -1
      %negative guess higher
      rho2(ai) = rho2(ai) - rho2Stepping(ai);
    elseif rhoConvgError_old(ai) < rhoConvgError(ai) && rhoConvgErrorSign == -1
      %negative guess lower
      rho2(ai) = rho2(ai) + rho2Stepping(ai);
    %else
    %    continue; %continue beacuse rhoConvgError = 0;
    end

    % if "ai" is the same as the last run,
    % then increase the precision
```

```
      % check for race condition or deadlock due to reduced precision ***VERY IMPORTANT***
      % Add more conditions
      if rho_run >= iterationTimeout && rhoConvgError(ai) <= rhoPrecision*3 && rhoConvgError(ai) >= rhoPrecision
        rho2Stepping(ai) = rho2Stepping(ai) - 0.5e-8; % should prevent race condition
        rho_run = 0; % should bypass the final case if it's␣only␣rho_run␣that␣meets␣the␣conditions
␣␣␣␣␣␣elseif␣rho_run␣>=␣iterationTimeout␣&&␣rhoConvgError(ai)␣>=␣-rhoPrecision*3␣&&␣rhoConvgError(ai)␣<=␣rhoPrecision
␣␣␣␣␣␣␣␣rho2Stepping(ai)␣=␣rho2Stepping(ai)␣-␣0.5e-8;␣%␣should␣prevent␣race␣condition
␣␣␣␣␣␣␣␣rho_run␣=␣0;
␣␣␣␣␣␣elseif␣rho_run␣>=␣iterationTimeout␣&&␣sign(rhoConvgError(ai))␣>␣0␣%positive
␣␣␣␣␣␣␣␣rho2(ai)␣=␣rho2(ai)*0.5;␣%guess␣2x␣higher␣than␣initial␣guess
␣␣␣␣␣␣␣␣rho_run␣=␣0;
␣␣␣␣␣␣elseif␣rho_run␣>=␣iterationTimeout␣&&␣sign(rhoConvgError(ai))␣==␣-1
␣␣␣␣␣␣␣␣rho2(ai)␣=␣rho2(ai)*2;␣%guess␣2x␣lower␣than␣init␣guess
␣␣␣␣␣␣␣␣rho_run␣=␣0;
␣␣␣␣␣␣end
␣␣
␣␣␣␣␣␣rho_run=rho_run+1;
␣␣␣␣␣␣rhoConvgError_old(ai)␣=␣rhoConvgError(ai);
␣␣␣␣␣␣p2(ai)␣=␣P1(ai)+rho1_avg(ai).*u1(ai+1).^2.*(1-(rho1_avg(ai)./rho2(ai)));␣%␣recalc␣if␣not␣good
␣␣␣␣␣␣h2(ai)␣=␣h1+(u1(ai+1).^2/2).*(1-(rho1_avg(ai)./rho2(ai)).^2)/1000;␣%␣recalc␣if␣not␣good
␣␣␣␣end
␣␣␣␣fprintf('%i times to convergence for %i km.',iterationTotalAi, kmMOLA(ai));
␣␣␣␣atomicDiaAvg␣=␣((CO2(ai).*diaCO2)/100+(N2(ai).*diaN2)/100+(Ar(ai).*diaAr)/100+(O2(ai).*diaO2)/100+(CO(ai).*diaCO)/100);
␣␣␣␣Kn(ai)␣=␣(boltzmann*T1(ai))/(sqrt(2).*pi.*atomicDiaAvg.^2.*p2(ai).*L);
␣␣␣␣u2(ai)␣=␣(rho1_avg(ai).*u1(ai))/cearun2.output.density;
␣␣␣␣m2(ai)␣=␣u2(ai)./cearun2.output.sonvel;
␣␣␣␣a1(ai)␣=␣sqrt(gammaMarsStatic.*277.*T1(ai));
␣␣␣␣Minf(ai)␣=␣u1(ai+1)./a1(ai);
␣␣␣␣gammaMars␣=␣cearun2.output.gamma;
␣␣␣␣p02_p1(ai)␣=␣((1+(gammaMars-1)/2.*m2(ai).^2).^(gammaMars/(gammaMars-1))).*((1-gammaMars+2.*gammaMars.*Minf(ai).^2)/(gammaMars+1));
␣␣␣␣Cp_max_aero(ai)␣=␣(2./(gammaMarsStatic.*Minf(ai).^2)).*
```

```
((p02_p1(ai))-1);
␣␣␣␣if␣Kn␣>=␣1␣%␣Knudsen␣number␣greater␣than␣1␣indicates␣you␣can't use
      % aerodynamic equations anymore
      % rarified gas dynamics in here
      % not necessary because Kn << 1
    else
      % again, be mindful of order of operations. There are initial
      % conditions specified in the header of the code
      cp_lsi = 0;
      for p=1:size(capsule_2d,2)-1 % for every panel of the capsule,
        LSI_y = capsule_2d(2,p+1)-capsule_2d(2,p); %slope
        LSI_x = capsule_2d(1,p+1)-capsule_2d(1,p); %slope
        if LSI_y <= 0 && LSI_x <= 0
          %fprintf("Ignored␣shadowed␣upper␣surface␣above␣Mach␣8...\n");
        elseif LSI_y <= 0 && LSI_x >= 0
          %fprintf("Ignored␣shadowed␣lower␣surface␣above␣Mach␣8...\n");
        else
          cp_lsi_cat = Cp_max_aero(ai).*sind(atand((LSI_y)/(LSI_x))).^2;
          if cp_lsi_cat ~= 0 %#ok<BDSCI>
            cp_lsi = [cp_lsi;cp_lsi_cat]; %#ok<AGROW>
          end
        end
      end
      coeffPressureUpper(ai) = sum(cp_lsi(1:size(cp_lsi,1)/2))/(size(cp_lsi,1)/2);
      coeffPressureLower(ai) = sum(cp_lsi((size(cp_lsi,1)/2)+1:size(cp_lsi,1)))/(size(cp_lsi,1)/2); %+1 because it is even
      decelDragForce(ai) = 0.5*rho1_avg(ai)*u1(ai+1).^2*coeffPressureUpper(ai)*((pi*23^2)/2);
      dragAcc(ai) = decelDragForce(ai)/vehicleMass;
      dragAcc_x(ai) = dragAcc(ai)*cosd(flightPathAngle(ai+1));
      dragAcc_y(ai) = dragAcc(ai)*sind(flightPathAngle(ai+1));
      u1_x(ai) = sqrt(u1_x(ai+1).^2+2.*((dragAcc_x(ai)).*cosd(flightPathAngle(ai+1))).*((deltaVerticalDist.*sind(90-flightPathAngle(ai+1)))./sind(flightPathAngle(ai+1))));
      u1_y(ai) = -sqrt(u1_y(ai+1).^2+2.*((marsGrav+dragAcc_y(ai)).*deltaVerticalDist));
      u1(ai) = sqrt(u1_x(ai).^2+u1_y(ai).^2);
      flightPathAngle(ai) = atand(u1_y(ai)/u1_x(ai));
    end
    % save all data into a matrix for graphing:
    % (insert everything here)
```

```

  if Minf(ai) <= 8
    fprintf("Execution␣complete");
    break;
  end
end

```

2. RF Gateway I&T Unit

```
#include <Arduino.h>
#include <ArduinoJson.h>
#include <WiFiUdp.h>
#include <RH_RF95.h>
#include <RH_RF69.h>
#include <WiFi101.h>
#include <Adafruit_SleepyDog.h>
#include <RTClib.h>
#include <PriUint64.h>
#include <Adafruit_DotStar.h>

//RFM95
#define RFM95_CS  A2
#define RFM95_INT 9u
#define RFM95_RST 7u
#define RFM69_CS  A4
#define RFM69_INT 11u
#define RFM69_RST 13u
#define DEFAULT_FREQ 922.0 //MHZ
RH_RF95 r(RFM95_CS, RFM95_INT);
//RH_RF69 cmd(RFM69_CS, RFM69_INT);

RTC_Millis rtc;
uint32_t timeUnix = 0;
String snd= "";

//WiFi credentials for known networks:
//(reside in flash memory only):
#define NUM_SSID_ENTRIES 6
const char *ssidlist[NUM_SSID_ENTRIES] =  {
  "REDACTED", "REDACTED", "REDACTED", "REDACTED", "REDACTED", "
  REDACTED"
};

const char *passlist[NUM_SSID_ENTRIES] = {
  "REDACTED","REDACTED","REDACTED","REDACTED","NONE","NONE"
};

//two SSIDs in case the first one doesn't work for latency's sake
const char *ssid3 = "REDACTED";
const char *ssid2 = "REDACTED";
const char *pass2 = "REDACTED";
const char *ssid1 = "REDACTED";
const char *pass1 = "REDACTED";

int netsRange = 0;

#define NUMPIXELS     1
#define DATAPIN       8u
#define CLOCKPIN      6u
Adafruit_DotStar strip(NUMPIXELS, DATAPIN, CLOCKPIN, DOTSTAR_BGR);

////////////////////////////////////////////////////////////
//wifi:
#define ATWIFI_SS     A5
#define ATWIFI_ACK    12u
#define ATWIFI_RST    5u
#define ATWIFI_EN     2u
int status = WL_IDLE_STATUS;
WiFiSSLClient client;     //internet
WiFiUDP    udp; //HTTP-only

//FOR HIDDEN SSIDs: BSSID REQUIRED
uint8_t bssid_1[6] = {REDACTED, REDACTED, REDACTED, REDACTED, REDACTED, REDACTED}; //Asus router

#define NTP_PACKET_SIZE 48
uint8_t netNTPpacketBuffer[48];

uint16_t localPort = 2390; //NTP server
#define UTCMINUS7 28800//+3600 //seconds
//THE FOLLOWING KEYS/ADDRESSES ARE PRIVATE - DO NOT LET ANYONE ELSE USE THEM.
/*
 * Discord private keys are redacted for publication.
 */

//end wifi:
////////////////////////////////////////////////////////////


/*
 * 64-bit microsecond timer. Number will roll over after 213 million days.
 * Best if run on a CPU that can handle sub-microsecond precision, such as
 * a Cortex M4, M7, or A-series.
 * MUST CALL TWICE EVERY 71.6 MINUTES TO WORK.
 */
uint64_t micros64(void) {
  static uint32_t low32, high32;
  uint32_t new_low32 = micros();
  if(new_low32 < low32)high32++;
  low32 = new_low32;
  return (uint64_t)high32 << 32 | low32;
}

//Convenience function: microseconds after boot time:
//Dependancy: Serial must be enabled before using.
//Takes a few uS to execute.
void SPTime(String msg) {
  Serial.print("[");
  Serial.print(PriUint64<DEC>(micros64()));
  Serial.print("]␣");
  Serial.print(msg);
}

//Same function as above but prints newline:
void SPLTime(String msg) {
  Serial.print("[");
  Serial.print(PriUint64<DEC>(micros64()));
  Serial.print("]␣");
  Serial.println(msg);
}

//
  //////////////////////////////////////////////////////////////////
//                        serial forwarder - RFM 69W
           //
//                               PID: XX
           //
//
  //////////////////////////////////////////////////////////////////
//void sfr69(String str) {
//  cmd.send((uint8_t *)str.c_str(),strlen(str.c_str()));
//}

//
  //////////////////////////////////////////////////////////////////
//                        iso8610 format - discord
           //
```

```
//                                  PID: 22
          //
//
  /////////////////////////////////////////////////////////////////////////////

//Take an RTC and output the current time as an ISO 8610 String:
String rtcd_iso8610(DateTime &nowf, bool trailingComma = true,\
  bool startup = false) {
  //if(startup == false) m.lastProcessIDActive = 0x22;
  String iso8610 = "";
  iso8610 += nowf.year()       + (String)"-";
  if(nowf.month() < 10) iso8610 += (String)"0";
  iso8610 += nowf.month()      + (String)"-";
  if(nowf.day() < 10)   iso8610 += (String)"0";
  iso8610 += nowf.day()        + (String)"T";
  if(nowf.hour() < 10)  iso8610 += (String)"0";
  iso8610 += nowf.hour()       + (String)".";
  if(nowf.minute() < 10)   iso8610 += (String)"0";
  iso8610 += nowf.minute()     + (String)".";
  if(nowf.second() < 10)   iso8610 += (String)"0";
  iso8610 += nowf.second();
  if(trailingComma) iso8610 += ",";
  return iso8610;
}

//
  /////////////////////////////////////////////////////////////////////////////

//                                restart CPU
          //
//                                  PID: FF
          //
//
  /////////////////////////////////////////////////////////////////////////////

//Writes a 1 to the SYSRESETREQ register to restart the CPU.
void restart(void)  {
  //m.lastProcessIDActive = 0xFF;
  //SYNCFRAM;
  __asm volatile ("cpsid␣i" ::: "memory"); //disable interrupt
  reporting
  __asm volatile ("dsb␣0xF":::"memory"); //commit
  SCB->AIRCR = ((0x5FAUL << SCB_AIRCR_VECTKEY_Pos)\
    | SCB_AIRCR_SYSRESETREQ_Msk); //write to system control block to
   reset
  __asm volatile ("dsb␣0xF":::"memory");
  for(;;) __asm volatile("nop");
}

//Stratum 1 NIST.GOV. I hope the USG doesn't mind that I'm spamming their server.
IPAddress timeServer(129, 6, 15, 28);
//ntp packet
//Sends an NTP packet to request the time. Use this provision when there is internet.
//Attempt to connect to internet for 1 minute before running this
//if unable to make connection, revert to GPS-derived time.
unsigned long sendNTPpacket(IPAddress &address) {
  //fbuf.print("1"); FFPUSH;
  // set all bytes in the buffer to 0
  memset(netNTPpacketBuffer, 0, NTP_PACKET_SIZE);
  // Initialize values needed to form NTP request
  netNTPpacketBuffer[0] = 0b11100011;   // LI, Version, Mode
  netNTPpacketBuffer[1] = 0;     // Stratum, or type of clock
  netNTPpacketBuffer[2] = 6;     // Polling Interval
  netNTPpacketBuffer[3] = 0xEC;  // Peer Clock Precision
  // 8 bytes of zero for Root Delay & Root Dispersion
  netNTPpacketBuffer[12]  = 49;
  netNTPpacketBuffer[13]  = 0x4E;
  netNTPpacketBuffer[14]  = 49;
  netNTPpacketBuffer[15]  = 52;
  Serial.println(F("writing␣pKT"));
  // all NTP fields have been given values, now
  // you can send a packet requesting a timestamp:
  udp.beginPacket(address, 123); //NTP requests are to port 123
  udp.write(netNTPpacketBuffer, NTP_PACKET_SIZE);
  int tst = udp.endPacket();
  Serial.println(tst);
  Serial.println(F("sent␣packet␣to␣time␣server"));
  return 0;
}

uint32_t inputTimeNIST(void) {
  udp.begin(localPort);
  int count = 0;
  while(true) {
    Watchdog.reset();
    Serial.println(F("loopp"));
    sendNTPpacket(timeServer);
    //wait for packet
    delay(1000);
    Serial.println(F("Waiting␣for␣pkt"));
    if(udp.parsePacket()) {
      Serial.println(F("Success"));
      udp.read(netNTPpacketBuffer, NTP_PACKET_SIZE);
      uint32_t sSince1900 = netNTPpacketBuffer[40] << 24 | netNTPpacketBuffer[41] << 16 \
                          | netNTPpacketBuffer[42] << 8  | netNTPpacketBuffer[43];
      //uint32_t sSince1900 = highWord << 16 | lowWord;
      Serial.println(sSince1900,BIN);
      //convert to unix time:
      const uint32_t unixepoch = 2208988800UL;
      uint32_t unixtime = sSince1900 - unixepoch - UTCMINUS7;
      return unixtime;
    }
    Watchdog.reset();
    Serial.println(F("reached␣this␣p[oint"));
    delay(9000);
    count++;
    if(count > 3) restart();
  }
}


// /////////////////////////////////////////////////////////////////////////////
//                              iso8610 format //
//                                  PID: 22 //
// /////////////////////////////////////////////////////////////////////////////
//Take an RTC and output the current time as an ISO 8610 String:
String rtc_iso8610(RTC_Millis &rtcf, bool trailingComma = true,\
  bool startup = false) {
  //if(startup == true) m.lastProcessIDActive = 0x22;
  //SYNCFRAM;
  DateTime nowf = rtcf.now();
  String iso8610 = "";
  iso8610 += nowf.year()        + (String)"-";
  if(nowf.month() < 10) iso8610 += (String)"0";
  iso8610 += nowf.month()       + (String)"-";
  if(nowf.day() < 10)   iso8610 += (String)"0";
  iso8610 += nowf.day()         + (String)"T";
  if(nowf.hour() < 10)  iso8610 += (String)"0";
  iso8610 += nowf.hour()        + (String)":";
  if(nowf.minute() < 10)   iso8610 += (String)"0";
  iso8610 += nowf.minute()      + (String)":";
  if(nowf.second() < 10)   iso8610 += (String)"0";
  iso8610 += nowf.second();
  if(trailingComma) iso8610 += ",";
  return iso8610;
}

//send csv function, general for gateway.
/*!
 * @brief Send CSV string to Discord. General function for
 * both gateway and node. Retry not implemented in this function to allow developer
 * flexibility for retry strategies.
 * @param token
 * Set URL for webhook token, HTTP 1.3 format
 * @param content
 * Content string - the text. Already formatted for monospaced text.
 * Pointer to ADDRESS of existing content string to save RAM.
 * @param nodeName
 * Name of node for username. Optional.
 * @param imageUrl
 * URL to profile picture of current message of bot. Optional.
 * @param msp
 * Whether to enclose in monospaced form or not.
 * @return Returns true on success, false on failure to send.
 */
bool sendDiscordCSV(String token, String content, String nodeName, String imageUrl, bool msp) {
  Watchdog.reset();
  //sfr69("HEARTBEAT LOSS"); //this message will be shown if it fails.
  StaticJsonDocument<4096> NODE; //allocate to stack. ONLY ON CORTEX M4, M7, OR R5 OR BETTER.
  //delay(1500); //delay so Cloudflare doesn't reject our messages
  SPLTime("[DBG]␣Allocated␣JSON␣doc");
  NODE["username"] = nodeName;
  NODE["avatar_url"] = imageUrl;
  if(msp) { NODE["content"] = "‘‘" + content + "‘‘"; }
  else { NODE["content"] = content; }
  //JsonArray embeds1 = NODE.createNestedArray("embeds");
  //JsonObject embed_def1 = embeds1.createNestedObject();
  //embed_def1["description"] = "Courtyard readings - " + a4;
  RETR:
  Watchdog.reset();
  if(client.connect("discord.com", 443)) {
    //WiFi.setLEDs(64,255,255);
    //Send HTTP header to Discord:
    //HTTP header is extremely particular about order of flags, and Discord is very particular about using HTTP 1.3
    client.println("POST␣" + token + "␣HTTP/1.1");
    client.println("Host:␣discord.com");
    client.println("Accept:␣*/*");
    client.println("Content-Type:␣application/json");
    client.print("Content-Length:␣");
    //compute length of minified JSON:
    client.println(measureJson(NODE));
    client.print("\r\n"); //terminate HEADER by sending a newline
    //Now send JSON header to Discord:
    serializeJson(NODE, client); //This sends without any additional RAM
    //free((void *)NODE);
    //timeout implementation - 1 second timeout.
    uint32_t timeout = millis();
    uint32_t t_timeout = 1500;
    //WiFi.setLEDs(64,0,255);
    //Edit 17 October 2022:
    //Dynamic retry needed. Cloudflare is extremely strict with ANY missing/malformed packets.
    //Analyze the output of this function by outputting the entire CF response:
    //wait for response from server:
    Watchdog.reset();
    while(!client.available() && (millis() < (timeout + t_timeout)));
    /*
     * There were two reasons a TX failed frequently:
     * 1. Low power mode called and a call to shut off LPM never existed after initialization.
     *    This resulted in the radio misbehaving frequently and/or needing to be really REALLY
     *    close to an AP.
     * 2. On busy networks, the radio might transmit on an overlapping frame (radio is very
     *    slow over SPI) with another client and the AP will throw that frame away and tell
     *    the legitimate client to retransmit. My radio is dumb and won't recognize that
     *    command. So the easiest solution will simply be to retry til I get a frame from
     *    Cloudflare (specifically an HTTP 1.1/204 No Content).
     */
    //if we time out retry TX:

    if(client.available()) {
      timeout = millis();
      while(client.available() && (millis() < (timeout + t_timeout)) ) {

        char c = client.read();
        Serial.print(c);
      }
    }
    else goto RETR;
    client.stop();
    //sfr69("Discord TX'd"); //this message will be shown if it fails.
    return true;
      //if(millis() > timeout + t_timeout) return false;
  } return false;
}

//Agent to control WiFi coprocessor, including CONNECTION AT BOOT.
//Major update May 4: loss of STA reassociation and no known networks found
//handler.
uint64_t wifi_delegate(int w_stat) {
  #define CONN_TIMEOUT 30000 //msec

  Watchdog.reset();
  WiFi.noLowPowerMode();
  bool poll_requested = false;
  uint8_t netsReturned = 0;
  //Basic state check: are we connected to a STA?
  if(w_stat != WL_CONNECTED) {
    //Figure out what state we're in and act upon it to return to WL_CONNECTED
    //Worst case:
    if(w_stat == WL_NO_SHIELD) {
      SPTime(F("[wifi_delegate]␣CRITICAL:␣TOTAL␣LOSS␣OF␣WIFI␣COPROCESSOR."));
      Serial.println(F("␣UNABLE␣TO␣TRANSFER␣ANY␣DATA,␣PERIOD."));
      return micros64();
    }
    //we've probably just restarted, so proceed with SSID search
    else {
      connect: //GOTO used as alias only within this function as a
```

```
lazy way to
    //transfer control.
    Watchdog.disable();
    //Reenable for a longer time to allow for scanning:
    Watchdog.enable(32000);
    SPLTime(F("[wifi_delegate]␣Starting␣WiFi␣SSID␣search..."));
    netsReturned = WiFi.scanNetworks();
    netsRange = netsReturned;
    Watchdog.enable(16000);
    SPLTime(F("[Watchdog]␣NOTICE:␣Watchdog␣enabled,␣16s␣timeout"))
;
    SPTime(F("[wifi_delegate]␣Returned␣"));
    Serial.print(netsReturned);
    Serial.println(F("␣networks␣in␣range."));
    if(WiFi.status() == WL_NO_SSID_AVAIL) {
      SPTime(F("[wifi_delegate]␣Warning:␣No␣SSIDs␣found.␣Reducing␣
sensor"));
      Serial.println(F("␣polling␣rate␣until␣SSIDs␣are␣in␣range."))
;
      //actually follow up on this later
    return micros64();
    }
    for(int thisNet = 0; thisNet < netsReturned; thisNet++) {
      const char * tmp_ssid = WiFi.SSID(thisNet);
      uint8_t tmp_bssid[6];
      //look up if this entry is in our table:
      for(int i=0;i<NUM_SSID_ENTRIES;i++) {
        //INCLUDE BSSID LOOKUP, this will be really long if
statement:
        uint8_t *tmp_bssid_actual = {WiFi.BSSID(thisNet, tmp_bssid
)};
        //printMacAddress(tmp_bssid_actual);
        if(strcmp((const char *)ssidlist[i],(const char *)tmp_ssid
) == 0 || (\
        tmp_bssid_actual[0] == bssid_1[0] && \
        tmp_bssid_actual[1] == bssid_1[1] && \
        tmp_bssid_actual[2] == bssid_1[2] && \
        tmp_bssid_actual[3] == bssid_1[3] && \
        tmp_bssid_actual[4] == bssid_1[4] && \
        tmp_bssid_actual[5] == bssid_1[5])) {
          //we're done searching - attempt connection and return.
          SPTime(F("[wifi_delegate]␣Attempting␣to␣connect␣to␣"));
          Serial.println(ssidlist[i]);
          //decide if we're open or WPA/WPA2:
          Watchdog.disable();
          SPLTime(F("[watchdog]␣NOTICE:␣Watchdog␣disabled"));
          uint32_t tot = millis();
          if(strcmp(passlist[i],"NONE") == 0) {
            while(WiFi.status() != WL_CONNECTED && (millis() < tot + CONN_TIMEOUT)) {
              WiFi.begin(ssidlist[i]);
              delay(6000);
            }
            Watchdog.enable(16000);
            SPLTime(F("[Watchdog]␣NOTICE:␣Watchdog␣enabled,␣16s␣timeout"));
            if(WiFi.status() == WL_CONNECTED) {
              SPLTime(F("[wifi_delegate]␣Connected"));
              //WiFi.setLEDs(64,0,255);
              return micros64();
            } else goto failw;
          }
          else {
            while(WiFi.status() != WL_CONNECTED && (millis() < tot + CONN_TIMEOUT)) {
              WiFi.begin(ssidlist[i],passlist[i]);
              delay(6000);
            }
            Watchdog.enable(16000);
            SPLTime(F("[Watchdog]␣NOTICE:␣Watchdog␣enabled,␣16s␣timeout"));
            if(WiFi.status() == WL_CONNECTED) {
              SPLTime(F("[wifi_delegate]␣Connected"));
              //WiFi.setLEDs(64,0,255);
              //w.RES_CONNECTED_TO_WIFI = true;
              //SYNCFRAM;
              return micros64();
            }
          }
          failw:
          if(WiFi.status() == WL_CONNECT_FAILED || WiFi.status() == WL_CONNECTION_LOST) {
            SPLTime(F("[wifi_delegate]␣Failed␣to␣connect.␣Retrying␣in␣1␣minute."));
            //w.RES_CONNECTED_TO_WIFI = false;
            //WiFi.setLEDs(0,0,0);
          }
          return micros64();
        }
      }
      //if we get this far we have a hidden network:
      if(netsReturned > 0) {
          SPLTime(F("[wifi_delegate]␣Failed␣to␣find␣network␣in␣list␣but␣networks␣available.␣Trying␣hidden␣network␣DB:"));
          Watchdog.disable();
          SPLTime(F("[watchdog]␣NOTICE:␣Watchdog␣disabled"));
          uint32_t tot = millis();
          while(WiFi.status() != WL_CONNECTED && (millis() < tot + CONN_TIMEOUT)) {
            WiFi.begin(ssid1,pass1);
            delay(6000);
          }
          Watchdog.enable(16000);
          SPLTime(F("[watchdog]␣NOTICE:␣Watchdog␣enabled,␣16s␣timeout"));
          if(WiFi.status() == WL_CONNECTED) {
            SPLTime(F("[wifi_delegate]␣Connected␣to␣Orion."));
            //WiFi.setLEDs(64,0,255);
            //w.RES_CONNECTED_TO_WIFI = true;
            return micros64();
          }
        }
        else {
          SPLTime(F("No␣networks␣in␣range.␣Sleeping␣processes␣for␣5␣minutes."));
          WiFi.lowPowerMode();
          delay(300000);
        }
      }
    }
  }
  //IN CASE MY PROGRAMMING IS A PIECE OF SHIT:
  //w.RES_CONNECTED_TO_WIFI = true;
  return micros64();
}


void setup() {
  Serial.begin(11000000); //max speed USB
  digitalWrite(LED_BUILTIN, LOW);
  //while(!Serial);
  strip.begin();
  strip.show();

  //Serial1 slave+master mode. C&DH can command the board to go into master mode
  Serial1.begin(115200);

  //NEW AND PROVISIONAL: WAIT FOR "AT" COMMAND. DO NOT PROCEED UNTIL THIS HANDSHAKE
  //HAS BEEN ACCOMPLISHED.
  //MUST HAVE 4-WAY HANDSHAKE.
  while(true) {
    if(Serial1.available()) {
      String str = Serial1.readStringUntil('*');
      if(str == "AT") {
        Serial1.print("OK*");
        break;
      }
    }
  }

  Watchdog.enable(60000);
  SPLTime(F("Watchdog␣started"));

  //set to handshake or long-range mode settings:
  //Handshake: 10bps
  //Main data throughput: 200bps
  //RH_RF95::Bw125Cr48Sf4096


  //turn on wifi radio
  SPLTime(F("Starting␣WiFi..."));
  digitalWrite(ATWIFI_EN, LOW);
  delay(2000); //FULL RESET
  digitalWrite(ATWIFI_EN, HIGH);
  delay(100);
  digitalWrite(ATWIFI_RST, HIGH);
  //delay(100);
  //digitalWrite(ATWIFI_RST, LOW);
  delay(100);
  WiFi.setPins(ATWIFI_SS,ATWIFI_ACK,ATWIFI_RST);

  //RTC start.
  //rtc.begin();
  wifi_delegate(WiFi.status());

  Serial.println(F("Asking␣for␣time..."));
 // if(!rtc.initialized() || rtc.lostPower()) rtc.adjust(DateTime(inputTimeNIST()));
  rtc.begin(DateTime(inputTimeNIST()));
  status = WiFi.status();
  if(status == WL_CONNECTED) {
    SPTime(F("Sending␣beacons␣to␣Discord..."));
    DateTime compileTime(F(__DATE__), F(__TIME__));
    String se = "Boot␣OK.␣Firmware␣compiled␣";
    se += rtcd_iso8610(compileTime,false,true);
    se += ".␣Connected␣to␣";
    se += String(WiFi.SSID());
    se += ",␣RSSI␣";
    se += String(WiFi.RSSI());
    se += "␣dBm.␣";
    se += String(netsRange);
    se += "␣SSIDs␣in␣range.\n␣Firmware␣specifically␣for␣AE␣110␣labs.";
    sendDiscordCSV(CSV_RDA,se,"Aerith␣7.0.1",AERITH_pfp,false);
    sendDiscordCSV(AE_RDA,se,"Aerith␣7.0.1",AERITH_pfp,false);
  }

  Serial.println(F("success"));
  //sfr69(String(timeUnix));

  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);
  delay(100);
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(100);


  if(!r.init()) {
    Serial.println(F("Device␣not␣responding.␣HALT."));
    String snd = "**Fatal**:␣LoRa␣radio␣hardware␣error.␣Disabling␣watchdog␣to␣permanently␣halt␣until␣POR.";
    sendDiscordCSV(CSV_RDA,snd,"Aerith␣7.0.1",AERITH_pfp,false);
    Watchdog.disable();
    sendDiscordCSV(CSV_RDA,"Watchdog␣disabled","Aerith␣7.0.1",AERITH_pfp,false);
    sendDiscordCSV(CSV_RDA,"Halt.","Aerith␣7.0.1",AERITH_pfp,false);
    while(1);
  }
  if(!r.setFrequency(DEFAULT_FREQ)) {
  }
  snd = "LoRa␣started.␣Center␣frequency:␣";
  snd += (String)DEFAULT_FREQ;
  snd += "␣MHz.";
  //sendDiscordCSV(CSV_RDA,snd,"Aerith 7.0.1",AERITH_pfp,false);
  r.setModemConfig(RH_RF95::Bw31_25Cr48Sf4096);
  r.setTxPower(20,false);
  //r.setLowDatarate();


  sendDiscordCSV(CSV_RDA,String("RTC␣started␣from␣TIME.NIST.GOV.␣Time:␣" + rtc_iso8610(rtc,true,false)),"Aerith␣7.0.1",AERITH_pfp,false);
  sendDiscordCSV(CSV_RDA,"Init␣complete.","Aerith␣7.0.1",AERITH_pfp,false);

  //need to tell C&DH we're ready:
  Serial1.print(F("READY"));

  delay(2000);
  //initialize Aerith C&DH:
  strip.setPixelColor(0,0x00FF4000); //visual hold
  strip.show();
    while(true) {
    if(Serial1.available()) {
      String str = Serial1.readStringUntil('*');
      if(str == "AT+REQTM") {
        DateTime now = rtc.now();
        Serial1.write((uint8_t *)&now, sizeof(now));
        delay(3); //arbitrary delay
        break;
      }
    }
  }
  strip.setPixelColor(0,0x0000FF00);
  strip.show();
  sendDiscordCSV(CSV_RDA,"Aerith␣C&DH␣RTC␣initialized␣from␣RF␣board.","Aerith␣7.0.1",INGENUITY_pfp,false);
  Watchdog.enable(160000);
  sendDiscordCSV(CSV_RDA,"Watchdog␣enabled.","Aerith␣7.0.1",AERITH_pfp,false);
  delay(2000);
  //execute only once!!!!!!!!!!!!!
  //sendDiscordCSV(TEAM1_WH,"test","Aerith 7.0.1",AERITH_pfp,false);
  //sendDiscordCSV(TEAM2_WH,"test","Aerith 7.0.1",AERITH_pfp,false);
  //sendDiscordCSV(TEAM3_WH,"test","Aerith 7.0.1",AERITH_pfp,false);
  //sendDiscordCSV(TEAM4_WH,"test","Aerith 7.0.1",AERITH_pfp,false);
  //sendDiscordCSV(TEAM5_WH,"test","Aerith 7.0.1",AERITH_pfp,false);
}


//Transmits data on RF board's terms. Can be dangerous if the master board is not designed to handle such situations
```

```
bool altMasterEnabled = false; //AT+ALTMI - on, AT+ALTMO - off

int pkrxd[5] = {0,0,0,0,0};
String pkrxd_a = "";


void loop() {
  Watchdog.reset();
  //no handshake for commands, only for time
  if(Serial1.available()) {
    String str = Serial1.readStringUntil('*');
    //crude switch statement for multicharacter commands:
    if(str == "AT+ALTMI") {
      altMasterEnabled = true;
      sendDiscordCSV(CSV_RDA,"RFB␣Alt␣master␣mode␣enabled.","Aerith␣7.0.1",AERITH_pfp,false);
    }
    else if(str == "AT+ALTMO") {
      altMasterEnabled = false;
      sendDiscordCSV(CSV_RDA,"RFB␣Alt␣master␣mode␣disabled.","Aerith␣7.0.1",AERITH_pfp,false);
    }
    else if(str == "AT+DISUL") {
      Serial1.print("OK*");
      while(true) {if(Serial1.available()) break;}
      String tmpp = Serial1.readStringUntil('*');
      sendDiscordCSV(E236_RDA,tmpp,"Aerith␣II",AERITH_pfp,false);
    }
    else {
      while(Serial1.available()); //flush, could be dangerous without a watchdog
    }
  }
  if(r.available()) {
    strip.setBrightness(255);
    strip.setPixelColor(0,0x00FFFFFF);
    strip.show();
    digitalWrite(LED_BUILTIN, HIGH);
    uint8_t buf[RH_RF95_MAX_MESSAGE_LEN];
    //dynamically allocated data, the string (not String) must be null-terminated.
    memset(buf,'\000',sizeof(buf));
    uint8_t len = 64;
    if(r.recv((uint8_t *)&buf,&len)) {

      uint8_t hdr = r.headerFrom();
      //memset((void *)buf,'\000',sizeof(buf)); //<-- this line screwed it all up, not supposed to be here
      int rssi,snr,from,ferr; rssi = 0; snr = 0; from = 0; ferr = 0;
      rssi = r.lastRssi(); snr = r.lastSNR(); ferr = r.frequencyError();
      String ds = rtc_iso8610(rtc, true,false);
      ds += String(from, HEX); ds += ",";
      ds += String(rssi, DEC); ds += ",";
      ds += String(snr, DEC);  ds += ",";
      ds += String(ferr, DEC); ds += ",";
      ds += String(len, DEC); ds += ",";
      //peek the data in buf in byte form:
      for(int i=0;i<sizeof(buf);i++) {
        if(buf[i] < 0x100) Serial1.print("0");
        Serial1.print(buf[i],HEX);
        Serial1.print("␣");
      }
      ds += String((char *)buf);
      if(hdr == 0x90) {
        pkrxd[0] += 1;
        sendDiscordCSV(TEAM1_WH,ds,"Aerith␣7.0.1",AERITH_pfp,false);
      }
      else if(hdr == 0xA6) {
        pkrxd[1] += 1;
        sendDiscordCSV(TEAM2_WH,ds,"Aerith␣7.0.1",AERITH_pfp,false);
      }
      else if(hdr == 0xB1) {
        pkrxd[2] += 1;
        sendDiscordCSV(TEAM3_WH,ds,"Aerith␣7.0.1",AERITH_pfp,false);
      }
      else if(hdr == 0xC4) {
        pkrxd[3] += 1;
        sendDiscordCSV(TEAM4_WH,ds,"Aerith␣7.0.1",AERITH_pfp,false);
      }
      else if(hdr == 0xFC) {
        pkrxd[4] += 1;
        sendDiscordCSV(TEAM5_WH,ds,"Aerith␣7.0.1",AERITH_pfp,false);
      }
      else if(hdr == 0xAE) {
        //in order to get proper null termination, set the buf to NULL
        memset((void *)buf,'\000',sizeof(buf));
        RH_RF95::printBuffer("Received:␣",buf,len);
        //Serial.println((char*)buf);

        //AE110 specific code:
```

```
      //Parsing the stupid buffer.

      //also send over serial to aerith if in master mode
      if(altMasterEnabled) {
        Serial1.print(ds);
      }

      //sendDiscordCSV(AE_RDA,ds,"Aerith VII",AERITH_pfp,true);
      //cmd.send((uint8_t *)buf,RH_RF69_MAX_MESSAGE_LEN);
      ds = "";
      //send handshake:
      r.send((uint8_t *)"OK",3);
      r.waitPacketSent(); //important for super slow rates
      digitalWrite(LED_BUILTIN, LOW);
    } else {
      digitalWrite(LED_BUILTIN, HIGH);
      RH_RF95::printBuffer("Received:␣",buf,len);
      Serial.println((char*)buf);
      int rssi,snr,from,ferr; rssi = 0; snr = 0; from = 0; ferr = 0;
      rssi = r.lastRssi();
      snr = r.lastSNR();
      ferr = r.frequencyError();
      Serial.print(F("RSSI:␣"));
      Serial.println(rssi,DEC);
      Serial.print(F("SNR:␣"));
      Serial.println(snr, DEC);
      from = r.headerFrom();
      Serial.print(F("From:␣"));
      Serial.println(from, HEX);
      String ds = rtc_iso8610(rtc, true,false);
      ds += String(from, HEX); ds += ",";
      ds += String(rssi, DEC); ds += ",";
      ds += String(snr, DEC); ds += ",";
      ds += String(ferr, DEC); ds += ",";
      ds += String((char *)buf);

      sendDiscordCSV(TILT_RDA,ds,"Aerith␣7.0.1",AERITH_pfp,true);
      //cmd.send((uint8_t *)buf,RH_RF69_MAX_MESSAGE_LEN);
      ds = "";
      //send handshake:
      r.send((uint8_t *)"OK",3);
      r.waitPacketSent(); //important for super slow rates
      digitalWrite(LED_BUILTIN, LOW);
    }
    pkrxd_a = "0x90:␣" + (String)pkrxd[0] + (String)"," + "0xA6:␣"
```

```
 + (String)pkrxd[1] + (String)"," +"0xB1:␣" + (String)pkrxd[2] +
(String)"," + "0xC4:␣" + (String)pkrxd[3] + (String)"," + "0xFC:␣"
+ (String)pkrxd[4];
      sendDiscordCSV(AE_RDA,pkrxd_a,"Aerith␣7.0.1",AERITH_pfp,false)
;
    }
    digitalWrite(LED_BUILTIN, LOW);
    strip.setBrightness(128);
    strip.setPixelColor(0,0x0000FF00);
    strip.show();
  }
}
```

3. Etro Thermocouple Test Code

```
/* ****************************************************************************80
* Etro Eval Board Microcode
* Extended Temperature Range Omnibus
* Universal multiplexed thermocouple amplifier
* Stanley Krzesniak
* Discord: @Nines#4444. Other methods of contact are unreliable, including
* by phone or (non-government) email. Please do not attempt to call or email
* me unless for official business.
*
* version 1.0-08_2022
* 16 August 2022
*
* This work is mostly to support this as a USBSerial thermocouple readout
* device, usable on any computer with a **user interface**.
*
* No warranty is implied:
* this code is provided "as-is". I do not guarantee this microcode  to be
* bug-free, that it is portable across multiple architectures, or that it will
* work for your application, wheter that be an IoT (Internet of Things)
* project or a mainline spacecraft. The end user agrees that in the case this
* microcode is used for mission-, safety-, or life support-critical systems,
* all damages as a result of any bugs, misuse, misimplementation, or gross
* negligence fall upon the party utilizing this code.
*/// ///////////////////////////////////////////////////////////////////////////

 #include <Arduino.h>
 #include <Adafruit_I2CDevice.h>
 #include <Adafruit_I2CRegister.h>
 #include <Adafruit_MCP9601.h>

 //convenience functions
 #define FSP(x) Serial.print(F(x))
 #define FSPL(x) Serial.println(F(x))

 //BREN and BREN2 shall be switched at the same time
 #define BREN 26
 #define BREN2 27
 #define A0SW 28 //addr0
 #define A1SW 29 //addr1
 #define MCP0ADDR 0x67
 #define MCP1ADDR 0x60

 Adafruit_MCP9601 mcp0;
 Adafruit_MCP9601 mcp1;

 void setup() {
 Serial.begin(1000000);
 while(!Serial);

 //starting UI:
 FSPL("
********************************************************************************
");
 FSPL("**␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Etro␣-␣Extended␣Temperature␣Range␣
Omnibus␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣**");
 FSPL("**␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Firmware␣1.0␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣**");
 FSPL("**␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣Stanley␣M.␣Krzesniak␣␣␣␣␣␣␣␣␣␣␣
␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣␣**");
 FSPL("**␣␣␣␣␣␣␣␣␣␣␣San␣Jose␣State␣University␣/␣NASA␣Ames␣Research
␣Center␣␣␣␣␣␣␣␣␣␣␣␣**");
 FSPL("
********************************************************************************
");
 //flush serial buffer before beginning:
 while(Serial.available()) Serial.read();
 FSP("Press␣any␣key␣to␣start...");
 while(true) {
   if(Serial.available()) break;
 }

 //explicitly set pin modes
 pinMode(BREN,OUTPUT);
 pinMode(BREN2, OUTPUT);
 pinMode(A0SW, OUTPUT);
```

```
  pinMode(A1SW, OUTPUT);
  digitalWrite(BREN, LOW);
  digitalWrite(BREN2, LOW);
  digitalWrite(A0SW, LOW); //addr 0
  digitalWrite(A1SW, LOW); //addr 1

  //start chip0
  FSP("Starting␣channel␣1-4␣thermocouple␣reader...");
  if(!mcp1.begin(MCP1ADDR)) {
    FSPL("ERROR");
    FSPL("U5␣failed␣to␣initialize,␣chip␣is␣damaged.");
    digitalWrite(LED_BUILTIN, HIGH);
    while(1);
  } else {
    FSPL("ok");
    FSPL("Set␣thermocouple␣to␣type␣K");
    mcp1.setThermocoupleType(MCP9600_TYPE_K);
    FSPL("Set␣ADC␣resolution␣to␣14-bit");
    mcp1.setADCresolution(MCP9600_ADCRESOLUTION_18);
    FSPL("Set␣LPF␣decimation␣to␣3");
    mcp1.setFilterCoefficient(3);
    FSPL("Read␣circuit␣powered␣on");
    mcp1.enable(true);
  }
  //check chip0 and multiplexers
  int32_t tmpReading = 0;
  FSPL("Checking␣ch␣1-4␣readings␣for␣faults...");
  //channel 1
  digitalWrite(BREN, HIGH); digitalWrite(A0SW, LOW);
 digitalWrite(A1SW, LOW);
  FSP("␣␣channel␣1.");
  tmpReading = mcp1.readADC();
  if(tmpReading < -5000 || tmpReading > 5000)
    FSPL("Error:␣mux␣channel␣1␣or␣TCA␣0␣out␣of␣range.");
  else FSPL("..ok");
  //channel 2
  digitalWrite(BREN, HIGH); digitalWrite(A0SW, LOW);
 digitalWrite(A1SW, HIGH);
  FSP("␣␣channel␣2.");
  tmpReading = mcp1.readADC();
  if(tmpReading < -5000 || tmpReading > 5000)
    FSPL("Error:␣mux␣channel␣2␣or␣TCA␣1␣out␣of␣range.");
  else FSPL("..ok");
  //channel 3
  digitalWrite(BREN, HIGH); digitalWrite(A0SW, HIGH);
 digitalWrite(A1SW, HIGH);
```

```
  FSP("␣␣channel␣3.");
  tmpReading = mcp1.readADC();
  if(tmpReading < -5000 || tmpReading > 5000)
    FSPL("Error:␣mux␣channel␣3␣or␣TCA␣2␣out␣of␣range.");
  else FSPL("..ok");
  //channel 4
  digitalWrite(BREN, HIGH); digitalWrite(A0SW, HIGH); digitalWrite(A1SW, LOW);
  FSP("␣␣channel␣4.");
  tmpReading = mcp1.readADC();
  if(tmpReading < -5000 || tmpReading > 5000)
    FSPL("Error:␣mux␣channel␣4␣or␣TCA␣3␣out␣of␣range.");
  else FSPL("..ok");
  digitalWrite(BREN, LOW); //turn reader1 off


  //start chip1
  FSP("Starting␣channel␣5-8␣thermocouple␣reader...");
  if(!mcp0.begin(MCP1ADDR)) {
    FSPL("ERROR");
    FSPL("U6␣failed␣to␣initialize,␣chip␣is␣damaged.");
    digitalWrite(LED_BUILTIN, HIGH);
    while(1);
  } else {
    FSPL("ok");
    FSPL("Set␣thermocouple␣to␣type␣K");
    mcp0.setThermocoupleType(MCP9600_TYPE_K);
    FSPL("Set␣ADC␣resolution␣to␣14-bit");
    mcp0.setADCresolution(MCP9600_ADCRESOLUTION_18);
    FSPL("Set␣LPF␣decimation␣to␣3");
    mcp0.setFilterCoefficient(3);
    FSPL("Read␣circuit␣powered␣on");
    mcp0.enable(true);
  }

  FSPL("Checking␣ch␣5-8␣readings␣for␣faults...");
  //channel 5
  digitalWrite(BREN2, HIGH); digitalWrite(A0SW, LOW); digitalWrite(A1SW, LOW);
  FSP("␣␣channel␣5.");
  tmpReading = mcp0.readADC();
  if(tmpReading < -5000 || tmpReading > 5000)
    FSPL("Error:␣mux␣channel␣1␣or␣TCA␣4␣out␣of␣range.");
  else FSPL("..ok");
  //channel 6
  digitalWrite(BREN2, HIGH); digitalWrite(A0SW, LOW);
```

```
digitalWrite(A1SW, HIGH);
 FSP("␣␣channel␣6.");
 tmpReading = mcp1.readADC();
 if(tmpReading < -5000 || tmpReading > 5000)
   FSPL("Error:␣mux␣channel␣2␣or␣TCA␣5␣out␣of␣range.");
 else FSPL("..ok");
 //channel 7
 digitalWrite(BREN2, HIGH); digitalWrite(A0SW, HIGH);
digitalWrite(A1SW, HIGH);
 FSP("␣␣channel␣7.");
 tmpReading = mcp1.readADC();
 if(tmpReading < -5000 || tmpReading > 5000)
   FSPL("Error:␣mux␣channel␣3␣or␣TCA␣6␣out␣of␣range.");
 else FSPL("..ok");
 //channel 8
 digitalWrite(BREN2, HIGH); digitalWrite(A0SW, HIGH);
digitalWrite(A1SW, LOW);
 FSP("␣␣channel␣8.");
 tmpReading = mcp0.readADC();
 if(tmpReading < -5000 || tmpReading > 5000)
   FSPL("Error:␣mux␣channel␣4␣or␣TCA␣7␣out␣of␣range.");
 else FSPL("..ok");

 FSPL("All␣channels␣ok.");
 delay(1000);
 if(Serial.available()) Serial.read(); //empty buffer

 digitalWrite(BREN, HIGH);
 digitalWrite(BREN2, HIGH);
// if(!mcp0.begin(MCP0ADDR)) {Serial.println("MCP0 dead.");
while(1);}
// if(!mcp1.begin(MCP1ADDR)) {Serial.println("MCP1 dead.");
while(1);}

 }

 bool bit0 = false;
 bool bit1 = false;
 bool mux60 = false;
 bool mux67 = false;
 bool constRead = false;

 float atr(Adafruit_MCP9601 inp,uint32_t samps) {
 float accum = 0.0;
 for(uint32_t sp = 0; sp <= samps; sp++) {
   accum += inp.readThermocouple();
```

```
  delay(5);
}
accum /= samps;
return accum;
}

void loop() {
//task1: check for keypresses:
if(Serial.available()) {
  char c = Serial.read();
  switch(c) {
  case ’1’:
    if(!bit0) {
    bit0 = true;
    digitalWrite(A0SW, HIGH);
    Serial.println(F("Address␣0␣bit␣on"));
    } else {
    bit0 = false;
    digitalWrite(A0SW, LOW);
    Serial.println(F("Address␣0␣bit␣off"));
    }
    break;
  case ’2’:
    if(!bit1) {
    bit1 = true;
    digitalWrite(A0SW, HIGH);
    Serial.println(F("Address␣1␣bit␣on"));
    } else {
    bit1 = false;
    digitalWrite(A0SW, LOW);
    Serial.println(F("Address␣1␣bit␣off"));
    }
    break;
  case ’3’:
    if(!mux60) {
    mux60 = true;
    digitalWrite(BREN, HIGH);
    Serial.println(F("U5␣TC␣mux␣on"));
    } else {
    mux60 = false;
    digitalWrite(BREN, LOW);
    Serial.println(F("U5␣TC␣mux␣off"));
    }
    break;
  case ’4’:
    if(!mux67) {
```

```
    mux67 = true;
    digitalWrite(BREN2, HIGH);
    Serial.println(F("U6␣TC␣mux␣on"));
    } else {
    mux67 = false;
    digitalWrite(BREN2, LOW);
    Serial.println(F("U6␣TC␣mux␣off"));
    }
    break;
  case ’a’:
    Serial.print(F("MCP␣U6:␣"));
    Serial.print(mcp0.readADC());
    Serial.print(F("␣ADC,␣"));
    Serial.print(mcp0.readAmbient());
    Serial.print(F("*C␣internal,␣"));
    Serial.print(mcp0.readThermocouple());
    Serial.print(F("*C␣thermocouple.\r\n"));
    break;
  case ’s’:
    Serial.print(F("MCP␣U5:␣"));
    Serial.print(mcp1.readADC());
    Serial.print(F("␣ADC,␣"));
    Serial.print(mcp1.readAmbient());
    Serial.print(F("*C␣internal,␣"));
    Serial.print(mcp1.readThermocouple());
    Serial.print(F("*C␣thermocouple.\r\n"));
    break;
  case ’z’:
    if(!constRead) constRead = true;
    else constRead = false;
  default:
    break;
  }
}
/*
delay(250);
digitalWrite(A0SW, LOW); digitalWrite(A1SW, LOW); delay(10);
Serial.print(mcp0.readADC()); Serial.print(F(","));
Serial.print(mcp1.readADC()); Serial.print(F(","));
digitalWrite(A0SW, HIGH); digitalWrite(A1SW, LOW); delay(10);
Serial.print(mcp0.readADC()); Serial.print(F(","));
Serial.print(mcp1.readADC()); Serial.print(F(","));
digitalWrite(A0SW, LOW); digitalWrite(A1SW, HIGH); delay(10);
Serial.print(mcp0.readADC()); Serial.print(F(","));
Serial.print(mcp1.readADC()); Serial.print(F(","));
digitalWrite(A0SW, HIGH); digitalWrite(A1SW, HIGH); delay(10);
```

```
 Serial.print(mcp0.readADC()); Serial.print(F(","));
 Serial.print(mcp1.readADC()); Serial.println(F(""));
 */

//conditional constant read
 if(constRead) {
   digitalWrite(BREN , HIGH); digitalWrite(BREN2 , LOW);
   digitalWrite(A0SW , LOW); digitalWrite(A1SW , LOW); delay(5000);
   Serial.print(atr(mcp1 ,512)); FSP(",");
   digitalWrite(A0SW , LOW); digitalWrite(A1SW , HIGH); delay(5000);
   Serial.print(atr(mcp1 ,512)); FSP(",");
   digitalWrite(A0SW , HIGH); digitalWrite(A1SW , HIGH); delay(5000)
;
   Serial.print(atr(mcp1 ,512)); FSP(",");
   digitalWrite(A0SW , HIGH); digitalWrite(A1SW , LOW); delay(5000);
   Serial.print(atr(mcp1 ,512)); FSP(",");

   digitalWrite(BREN , LOW); digitalWrite(BREN2 , HIGH);
   digitalWrite(A0SW , LOW); digitalWrite(A1SW , LOW); delay(5000);
   Serial.print(atr(mcp0 ,512)); FSP(",");
   digitalWrite(A0SW , LOW); digitalWrite(A1SW , HIGH); delay(5000);
   Serial.print(atr(mcp0 ,512)); FSP(",");
   digitalWrite(A0SW , HIGH); digitalWrite(A1SW , HIGH); delay(5000)
;
   Serial.print(atr(mcp0 ,512)); FSP(",");
   digitalWrite(A0SW , HIGH); digitalWrite(A1SW , LOW); delay(5000);
   Serial.print(atr(mcp0 ,512)); FSP(",");

   Serial.print(mcp1.readAmbient()); FSP(","); delay(100);
   Serial.println(mcp0.readAmbient());
   delay(1000);
 }
 }
```

4. Gateway Code - Main Processor

```
/*
* Wavefront Gateway
* Integrated Version with Environmental Sensing
* Dual Processor - second processor for dedicated seismometry output
* Final Version
*
* Stanley Krzesniak - v1.0 April 15, 2022
*
* Goal: write a basic Discord bot in C, which is a matter of implementing the
* API. Adjustable cadence, but default 10 minutes. Writeout all stats from all
* sensors in one, giant line as a "packet".
*
* This is the second-to-last iteration. I will only design a PTH PCB for
* Wavefront due to the supply chain and associated need to recode the system.
*
* Immediately writeout wireless node data to Discord.
* This file, in full, will not be released to the public domain. Segments of
* it will be showcased in the masters report.
*
* I'm inspired by the COBOL style of writing programs: one gigantic file.
* What's the big deal? It's like a novel. As long as it's well documented,
* there's really no problem.
*
* 9 April, 2023 - I changed my mind for all code. The full code is released with
* sensitive information redacted.
*/

//required core libraries
#include <Arduino.h>
#include <Adafruit_Sensor.h>
#include <Wire.h>
#include <SPI.h>
#include <Adafruit_I2CDevice.h>
```

```
    #include <Adafruit_SleepyDog.h> //watchdog config
    #include <atomic> //utility
    #include <inttypes.h> //to print uint64_t
    #include <PriUint64.h>
    #include <CRC32.h>

    //sensor and peripheral libraries
    #include <Adafruit_BME680.h>    //i2c Address 0x77
    #include <Adafruit_SCD30.h>     //i2c Address 0x61
    #include <Adafruit_ADT7410.h>   //i2c Address 0x48, conflict w ADS1115
    #include <Adafruit_AS7341.h>    //i2c Address 0x39
    #include <Adafruit_LIS3MDL.h>   //i2c Address 0x1C
    #include <Adafruit_CCS811.h>    //i2c Address 0x5A
    #include <Adafruit_ADS1X15.h>   //i2c Address 0x4A, jumpered to avoid conflict
    #include <Adafruit_INA219.h>    //i2c Address 0x40, 0x41

    //timing and others
    #include <RTClib.h>             //i2c address 0x68
    #include <Sparkfun_Ublox_Arduino_Library.h> //i2c address 0x42
    #include <Adafruit_FRAM_SPI.h>
    #include <Adafruit_FRAM_I2C.h>  //i2c address 0x50
    #include <RH_RF95.h>
    #include <RH_RF69.h>
    #include <Adafruit_DotStar.h>

    //internet access
    //#include <WiFiNINA.h>
    #include <WiFi101.h> //new WiFi module, Atmel ATWINC1500
    #include <ArduinoJson.h>

    // >:)
    //macros for extremely frequent one to two line instructions
    #define EXEC      void loop(){tifa();}
    //Due to the archaic way I used SYNCFRAM, I will insert the whole function
    //here:
    #define SYNCFRAM  write_fram()
    #define DLFRAM    read_fram()
    #define WROBJ 0x000002, m
    #define FRAMA_ADDR_MASTER 0x0002
    #define FRAMA_ADDR_WFLGS  0x2000
    #define FRAMA_ADDR_CRCS   0x2B95

    //for seismometer, use the already existing code to simplify,
 and treat as
    //a single device. Simplify the access schema for commands -
 single-drop bus.

    //instances of each peripheral and their respective #defines
    //GPS
    #define GPS_INIT_TIMEOUT 60000
    #define MTU_INTERNAL 128

    //wifi:
    #define ATWIFI_SS     10
    #define ATWIFI_ACK     9
    #define ATWIFI_RST     7
    //wifi:
    #define SPIWIFI_SS   10
    #define SPIWIFI_ACK   9
    #define ESP32_RESETN  7
    #define ESP32_GPIO0  A4
    int status = WL_IDLE_STATUS;
    WiFiSSLClient client;     //internet
    WiFiClient    clienthttp; //HTTP-only
    //two SSIDs in case the first one doesn't work for latency's
 sake
    const char *ssid3 = "REDACTED";
    const char *ssid2 = "REDACTED";
    const char *pass2 = "REDACTED";
    const char *ssid1 = "REDACTED";
    const char *pass1 = "REDACTED";

    //FOR HIDDEN SSIDs: BSSID REQUIRED
    uint8_t bssid_1[6] = {REDACTED}; //Asus router

    //Smarter way of SEEING what is around and auto connecting to a
  known list of SSIDs
    //Way faster and way WAY more reliable.
    #define NUM_SSID_ENTRIES 6
    char *ssidlist[NUM_SSID_ENTRIES] =  {
   "REDACTED", "REDACTED", "REDACTED", "REDACTED", "REDACTED", "
 REDACTED"
    };

    char *passlist[NUM_SSID_ENTRIES] = {
   "REDACTED","REDACTED","REDACTED","REDACTED","REDACTED","REDACTED
 "
    };

```

```
    //lora node radio:
    #define RFM95_CS     12
    #define RFM95_RST     2
    #define RFM95_INT    13
    RH_RF95 rf95(RFM95_CS, RFM95_INT);

    //command radio:
    #define RFM69_CS     A2
    #define RFM69_RST     2
    #define RFM69_INT    A3
    RH_RF69 rf69(RFM69_CS, RFM69_INT);

    //sensors:
    Adafruit_ADT7410 pts = Adafruit_ADT7410(); //temp sensor, precision
    Adafruit_AS7341   as; // light sensor
    Adafruit_BME680  bme; // PTHG
    Adafruit_CCS811  ccs; // VOCs
    Adafruit_LIS3MDL mag; // Magnetometer
    Adafruit_SCD30   co2; // CO2 and TH
    Adafruit_ADS1115 ads; // SO2 and H2S
    Adafruit_INA219  ina; // Battery stats

    //other (timing and memory):
    #define RAD2DEG   57.295779513

    //READBACK REQUIRED ON ALL NONDESTRUCTIVE FRAM WRITE OPERATIONS
    #define FRAM_CS    11
    #define SM_SIG     0x2895
    Adafruit_FRAM_SPI RAM  = Adafruit_FRAM_SPI(FRAM_CS);
    Adafruit_FRAM_I2C RAM1;
    RTC_DS3231        rtc;
    SFE_UBLOX_GPS     gps; //Time sync, PPS sync (sub-usec sync), space weather
    #define NUMPIXELS     1
    #define DATAPIN      41
    #define CLOCKPIN     40
    Adafruit_DotStar strip(NUMPIXELS, DATAPIN, CLOCKPIN, DOTSTAR_BGR);

    const char e[2] = "";

    //complete this section once I have everything else
    /*
  * Discord Bot section
  * minimal bot implemented, only enough to accomplish what I
```

```
need.
  */

    //THE FOLLOWING KEYS/ADDRESSES ARE PRIVATE - DO NOT LET ANYONE ELSE USE THEM.
    #define DISCORD_BOT_TOKEN     "REDACTED"
    #define DISCORD_WEBHOOK_URL   "REDACTED"
    #define DISCORD_PASTEBIN_URL  "REDACTED"
    #define DISCORD_RAWDATA_URL   "REDACTED"
    #define DISCORD_ALERTS_URL    "REDACTED"
    #define INGENUITY_pfp         "https://upload.wikimedia.org/wikipedia/␣commons/5/58/Mars_helicopter_on_sol_46.png"

    //Referencing the Discord API v10 manual, and @Rapptz's discord.py library.
    //As of writing, no Discord API exists for embedded systems written in C.

    //Scheduler and state machine:
    //Fixed list of tasks:
    #define NUMTASKS 7
    #define TASK_LOOP_MACH           0
    #define TASK_LOOP_SCANLORA       1
    #define TASK_LOOP_MAINSENSE      2
    #define TASK_LOOP_MACHSTATS      3
    #define TASK_LOOP_WIFIDELEG      4
    #define TASK_LOOP_OPSUMMARY      5
    #define TASK_LOOP_FRAMINTEG      6
    #define TASK_LOOP_SCANGFSK       7
    //loop_mach() - prio 0. state transition mechanism
    //  -- also evaluate RAM
    //loop_scanLora() - prio 1
    //loop_MainSense() - prio 2
    //f_discordSend() - prio R.
    //f_serialWriteOut() - prio R. For debugging.
    //f_deserializeBuf() - prio R. FRAM deserializer for all science datastreams
    //delays in micros64()
    //if delay == 1, not a routinely called task.
    static uint64_t sched_blks[NUMTASKS] = {
    10000ULL,     //loop_mach() 10mS - execute almost as often as possible
    1000000ULL,   //loop_scanLora() 1s
    120000000ULL,  //loop_MainSense() 120s
    300000000ULL,  //loop_machstats() 5m
    30000000ULL,   //wifi_delegate() 30s
    7250000000ULL,  //bihourly_summ() 2h 50s
    3600000000ULL,  //hourly FRAM integrity check
    //14400000000ULL      //GPS
    };

    //"last process run time"
    //set to initializer that ensures every process runs ONCE after ooot.
    static uint64_t lprc[NUMTASKS] {
    0ULL,
    0ULL,
    0ULL,
    0ULL,
    0ULL,
    0ULL,
    0ULL,
    //0ULL
    };

    //Sensor reads, straight to Serial
    //Include algorithms internally
    //Save external variables and configuration to struct, which is saved
    //to 0x000000 to 0x000FFF reserved block of FRAM
    //All sensor reads are blocking and gather statistical information.
    //To overcome this limitation and make it nonblocking would require an
    //FPGA and dedicated I2C lines or multiple dedicated CPUs, which, in the
    //interest of time, will not be done. Remember to talk about the architecture
    //in the project paper.

    //All functions are designed to access the global struct, with ALL raw values
    //available for extra automated preprocessing/cross-correlation.

    struct master{
    //be sure to sync struct with fram after every sensor read, doesn't matter
    //small it is

    //Ensure aligned memory access - 4 byte alignment

    //Mach State Assessment
    uint32_t    ram0_sz; //I2C FRAM
    uint32_t    ram1_sz; //SPI FRAM
    uint32_t    data_counter_total;
    uint32_t    data_counter_lora;
    uint32_t    ram0_mem_free;
    uint32_t    ram1_mem_free; //VERY IMPORTANT
    uint64_t    lastRecordedMicros;
    uint8_t     lastProcessIDActive;
    uint8_t     lastRecordedExecState;
    uint32_t    powerState;
    uint16_t    ramfree;

    //Last Active Time +/- 2 Mins
    uint16_t    lastYear;
    uint8_t     lastMonth;
    uint8_t     lastDay;
    uint8_t     lastHour;
    uint8_t     lastMinute;
    uint8_t     lastSecond;
    uint8_t     align000;

    //Sensor Configuration (can be dynamically changed)
    int         scd30_num_samps;
    int         adt7410_num_samps;
    float       adt7410_samp_rate;
    int         bme680_num_samps;
    float       bme680_samp_rate;
    int         ccs811_num_samps;
    float       ccs811_samp_rate;
    int         mag_num_samps;
    float       mag_samp_rate;

    //Data SCD30 CO2 sensor
    float       scd30_temp_max;
    float       scd30_temp_min;
    float       scd30_temp_avg;
    float       scd30_temp_stddev;
    float       scd30_rh_max;
    float       scd30_rh_min;
    float       scd30_rh_avg;
    float       scd30_rh_stddev;
    float       scd30_co2_max;
    float       scd30_co2_min;
    float       scd30_co2_avg;
    float       scd30_co2_stddev;
    uint32_t    scd30_micros_op;

    //Data ADT7410 prec temperature sensor
    float       adt7410_temp_max;
    float       adt7410_temp_min;
    float       adt7410_temp_avg;
    float       adt7410_temp_stddev;
    uint32_t    adt7410_micros_op;

    //Data BME680 "weather" sensor
    float       bme680_temp_max;
    float       bme680_temp_min;
    float       bme680_temp_avg;
    float       bme680_temp_stddev;
    float       bme680_rh_max;
    float       bme680_rh_min;
    float       bme680_rh_avg;
    float       bme680_rh_stddev;
    float       bme680_prs_max;
    float       bme680_prs_min;
    float       bme680_prs_avg;
    float       bme680_prs_stddev;
    float       bme680_gas_res_max;
    float       bme680_gas_res_min;
    float       bme680_gas_res_avg;
    float       bme680_gas_res_stddev;
    uint32_t    bme680_micros_op;

    //Data CCS811 VOC sensor
    float       ccs811_tvoc_max;
    float       ccs811_tvoc_min;
    float       ccs811_tvoc_avg;
    float       ccs811_tvoc_stddev;
    float       ccs811_eco2_max;
    float       ccs811_eco2_min;
    float       ccs811_eco2_avg;
    float       ccs811_eco2_stddev;
    uint32_t    ccs811_micros_op;

    //Data LIS3MDL Magnetometer
    float       mag_x_max;
    float       mag_x_min;
    float       mag_x_avg;
    float       mag_x_stddev;
    float       mag_y_max;
    float       mag_y_min;
    float       mag_y_avg;
    float       mag_y_stddev;
    float       mag_z_max;
    float       mag_z_min;
    float       mag_z_avg;
    float       mag_z_stddev;
    uint32_t    mag_micros_op;

    //Data AS7341 Light sensor
    uint32_t    lss_415nm_abs;
    uint32_t    lss_445nm_abs;
    uint32_t    lss_480nm_abs;
    uint32_t    lss_515nm_abs;
    uint32_t    lss_555nm_abs;
    uint32_t    lss_590nm_abs;
    uint32_t    lss_630nm_abs;
    uint32_t    lss_680nm_abs;
    uint32_t    lss_wband_abs;
    uint32_t    lss_890nm_abs;
    int32_t     lss_415nm_diff;
    int32_t     lss_445nm_diff;
    int32_t     lss_480nm_diff;
    int32_t     lss_515nm_diff;
    int32_t     lss_555nm_diff;
    int32_t     lss_590nm_diff;
    int32_t     lss_630nm_diff;
    int32_t     lss_680nm_diff;
    int32_t     lss_wband_diff;
    int32_t     lss_890nm_diff;
    uint32_t    lss_micros_op;

    //Data GPS
    float       gps_HDOP_avg    ;
    float       gps_VDOP_avg    ;
    float       gps_PDOP_avg    ;
    float       gps_Lat_avg     ;
    float       gps_Long_avg    ;
    float       gps_HDOP_stddev ;
    float       gps_VDOP_stddev ;
    float       gps_PDOP_stddev ;
    float       gps_Lat_stddev  ;
    float       gps_Long_stddev ;
    int32_t     gps_HDOP_max    ;
    int32_t     gps_VDOP_max    ;
    int32_t     gps_PDOP_max    ;
```

```
int32_t     gps_Lat_max     ;
int32_t     gps_Long_max    ;
int32_t     gps_HDOP_min    ;
int32_t     gps_VDOP_min    ;
int32_t     gps_PDOP_min    ;
int32_t     gps_Lat_min     ;
int32_t     gps_Long_min    ;
uint16_t    gps_year        ;
uint8_t     gps_month       ;
uint8_t     gps_day         ;
uint8_t     gps_hour        ;
uint8_t     gps_min         ;
uint8_t     gps_sec         ;
//pad byte:
uint8_t     nop_pad1        ;
//msec time:
uint16_t    gps_msec        ;
//pad bytes:
uint16_t    machine_flags   ; //bitmath to access this
uint32_t    gps_micros_op   ;

//Data INA219 Power:
float       ina_volt_max    ;
float       ina_volt_min    ;
float       ina_volt_avg    ;
float       ina_volt_sdv    ;
float       ina_curr_max    ;
float       ina_curr_min    ;
float       ina_curr_avg    ;
float       ina_curr_sdv    ;
float       ina_powr_max    ;
float       ina_powr_min    ;
float       ina_powr_avg    ;
float       ina_powr_sdv    ;
uint32_t    ina_micros_op   ;

//Data Inclinometer:
uint32_t    scl_ms_reported ;
int         scl_sps         ;
double      scl_temp_raw    ;
double      scl_xt_decim    ;
double      scl_yt_decim    ;
double      scl_zt_decim    ;
uint32_t    scl_micros_op   ;
float       scl_tilt16_max_x;
float       scl_tilt16_min_x;
```

```
    float       scl_tilt16_avg_x;
    float       scl_tilt16_sdv_x;
    float       scl_tilt16_max_y;
    float       scl_tilt16_min_y;
    float       scl_tilt16_avg_y;
    float       scl_tilt16_sdv_y;
    float       scl_tilt16_max_z;
    float       scl_tilt16_min_z;
    float       scl_tilt16_avg_z;
    float       scl_tilt16_sdv_z;
    //Seismometer data - listen for earthquakes or high accel
    //ideally this will be externally mounted or buried
    uint32_t    scl_sdv_delay_ms;
    float       scl_t16_max_x_mp;
    float       scl_t16_min_x_mp;
    float       scl_t16_sdv_mpdx;
    float       scl_t16_max_y_mp;
    float       scl_t16_min_y_mp;
    float       scl_t16_sdv_mpdy;
    float       scl_t16_max_z_mp;
    float       scl_t16_min_z_mp;
    float       scl_t16_sdv_mpdz;
    //Fredericks ultra-precision tilt sensor
    //Zero temperature dependency!
    float      fts_tilt_y1_max  ;
    float      fts_tilt_y1_min  ;
    float      fts_tilt_y1_avg  ;
    float      fts_tilt_y1_sdv  ;
    float      fts_tilt_y2_max  ;
    float      fts_tilt_y2_min  ;
    float      fts_tilt_y2_avg  ;
    float      fts_tilt_y2_sdv  ;

    //Data Volcanic Gas:
    int         gasv_sps        ;
    float       gasv_so2_max    ;
    float       gasv_so2_min    ;
    float       gasv_so2_avg    ;
    float       gasv_so2_sdv    ;
    float       gasv_h2s_max    ;
    float       gasv_h2s_min    ;
    float       gasv_h2s_avg    ;
    float       gasv_h2s_sdv    ;
    float       gasv_o3_max     ;
    float       gasv_o3_min     ;
    float       gasv_o3_avg     ;
    float       gasv_o3_sdv     ;

    float       gasv_so2_av_v   ;
    float       gasv_h2s_av_v   ;
    float       gasv_o3_av_v    ;

    uint32_t    gasv_micros_op  ;

    };
    //prior versions to April 15, 2022 did NOT allocate this as a static entity.
    //This could be why the processor hung after about 22 hours.
    static master m;

    //struct for flags at 0x1000
    struct working_flags {
    //RAM size: 1 BYTE per _bool.
    bool    RES_FRAM_FORMATTED; //if it was JUST formatted. Should almost never happen.
    bool    RES_NEW_FIRMWARE;
    bool    RES_FRAM_INTEG_CHK_FAILED;
    bool    RES_FRAM_INTEG_MSG_FIRED;
    bool    RES_RESTART_LOW_SRAM;
    bool    SCI_SCL3300_STDDEV_EXCEEDED_CHX;
    bool    SCI_SCL3300_CHX_FIRED;
    bool    SCI_SCL3300_STDDEV_EXCEEDED_CHY;
    bool    SCI_SCL3300_CHY_FIRED;
    bool    SCI_SCL3300_STDDEV_EXCEEDED_CHZ;
    bool    SCI_SCL3300_CHZ_FIRED;
    bool    SCI_ADT7410_HIGH_TEMP_FLAG;
    bool    SCI_ADT7410_HTF_FIRED;
    bool    SCI_ADT7410_LARGE_DT_DT;
    bool    SCI_SCD30_CO2_HIGH_2000_PPM;
    bool    SCI_SCD30_CO2H_FIRED;
    bool    SCI_AS7341_LARGE_DI_DT;
    uint8_t RES_FIRST_LOOP;
    uint8_t pad1;
    bool    RES_FRAM_FULL;
    bool    RES_CONNECTED_TO_WIFI;

    int     ping_result;
    int     fail_times;
    int     rpt_times;
    int     ping_high;
    int     ping_low;
    int     ping_avg;
```

```
//data:
float   sci_scl3300_sdv_x_prev;
float   sci_scl3300_sdv_y_prev;
float   sci_scl3300_sdv_z_prev;
float   sci_adt7410_avgtmp_prev;
float   sci_scd30_co2_prev;
}; static working_flags w;

//CRC32 storage for both working structs at 0x2B95:
//the known values are computed in RAM, read back, and validated that they are
//the same thing.
struct crc32storage {
uint32_t crc_master;
uint32_t crc_workng;
}; static crc32storage c;

//LoRa node v1 struct:
struct LoRa_typeA {
  uint32_t node_id;
  int32_t tilt32_x;
  int32_t tilt32_y;
  int32_t tilt32_z;
  int32_t tilt32_max_x;
  int32_t tilt32_max_y;
  int32_t tilt32_max_z;
  int32_t tilt32_min_x;
  int32_t tilt32_min_y;
  int32_t tilt32_min_z;
  uint32_t spect[13];
  float   meas_pressure;
  float   btmp_lower;
  float   meas_humidity;
  float   adt7410;
  float   scltemp;
  float   sysvolt;
}; static LoRa_typeA LTA;

//LoRa node v2 struct:

//Core and utility functions:

//actual sync fram instructions:
void write_fram(void) {
RAM1.writeObject(FRAMA_ADDR_MASTER,m);
```

```
RAM1.writeObject(FRAMA_ADDR_WFLGS,w);
RAM1.writeObject(FRAMA_ADDR_CRCS,c);
}

void read_fram(void) {
RAM1.readObject(FRAMA_ADDR_MASTER,m);
RAM1.readObject(FRAMA_ADDR_WFLGS,w);
RAM1.readObject(FRAMA_ADDR_CRCS,c);
}

void printEncryptionType(int thisType) {
// read the encryption type and print out the name:
switch (thisType) {
  case ENC_TYPE_WEP:
  Serial.print("WEP");
  break;
  case ENC_TYPE_TKIP:
  Serial.print("WPA");
  break;
  case ENC_TYPE_CCMP:
  Serial.print("WPA2");
  break;
  case ENC_TYPE_NONE:
  Serial.print("None");
  break;
  case ENC_TYPE_AUTO:
  Serial.print("Auto");
  break;
  case ENC_TYPE_UNKNOWN:
  default:
  Serial.print("Unknown");
  break;
}
}

void print2Digits(byte thisByte) {
if (thisByte < 0xF) {
  Serial.print("0");
}
Serial.print(thisByte, HEX);
}

void printMacAddress(byte mac[]) {
for (int i = 5; i >= 0; i--) {
  if (mac[i] < 16) {
  Serial.print("0");
```

```
     }
     Serial.print(mac[i], HEX);
     if (i > 0) {
     Serial.print(":");
     }
   }
   Serial.println();
   }


   ///memory:
   //credit Kevin Townsend
   int32_t readBack(uint32_t addr, int32_t data) {
   int32_t check = !data;
   int32_t wrapCheck, backup;
   RAM.read(addr, (uint8_t*)&backup, sizeof(int32_t));
   RAM.writeEnable(true);
   RAM.write(addr, (uint8_t*)&data, sizeof(int32_t));
   RAM.writeEnable(false);
   RAM.read(addr, (uint8_t*)&check, sizeof(int32_t));
   RAM.read(0, (uint8_t*)&wrapCheck, sizeof(int32_t));
   RAM.writeEnable(true);
   RAM.write(addr, (uint8_t*)&backup, sizeof(int32_t));
   RAM.writeEnable(false);
   // Check for warparound, address 0 will work anyway
   if (wrapCheck==check)
     check = 0;
   return check;
   }

   ///memory:
   //credit Kevin Townsend
   bool testAddrSize(uint8_t addrSize) {
   RAM.setAddressSize(addrSize);
   if (readBack(4, 0xbeefbead) == 0xbeefbead)
     return true;
   return false;
   }


   /*
  * Credit: Edgar Bonnet - Stack Overflow (2015)
  * Set millisecond timer on CPU to a specified, uint32_t value. No particular
  * use envisioned, but an ability to restore mission time might be possible.
```

```
 * @param ms unsigned milliseconds value
 */
  void setMillis(uint32_t ms) {
  extern uint32_t timer0_millis;
  ATOMIC_BLOCK(ATOMIC_RESTORESTATE) {
    timer0_millis = ms;
  }
  }

  /*
 * 64-bit microsecond timer. Number will roll over after 213 million days.
 * Best if run on a CPU that can handle sub-microsecond precision, such as
 * a Cortex M4, M7, or A-series.
 * MUST CALL TWICE EVERY 71.6 MINUTES TO WORK.
 */
  uint64_t micros64(void) {
  static uint32_t low32, high32;
  uint32_t new_low32 = micros();
  if(new_low32 < low32)high32++;
  low32 = new_low32;
  return (uint64_t)high32 << 32 | low32;
  }

  /*
 * decimal to String, adapted for arduino
 * @param val Input value, double prec float
 * @param width Baseline width in characters
 * @param prec Precision of decimal value
 * @return String-formatted decimal number of _prec_ precision.
 */
  String dtosstrf(double val, signed char width, unsigned char prec) {
  asm(".global␣_printf_float");
  char sout[64];
  char fmt[20];
  sprintf(fmt, "%%%d.%df", width, prec);
  sprintf(sout, fmt, val);
  String ssout(sout);
  return ssout;
  }

  //Print a 64-bit unsigned integer (because Arduino can't do that)
  //doesn't work, have to do this manually
```

```
/*
String print64(uint64_t val) {
char num[20];
sprintf(num,"%" PRIu64,val);
String out(num);
return out;
}
*/

//Convenience function: microseconds after boot time:
//Dependancy: Serial must be enabled before using.
//Takes a few uS to execute.
void SPTime(String msg) {
Serial.print("[");
Serial.print(PriUint64<DEC>(micros64()));
Serial.print("]␣");
Serial.print(msg);
}

//Same function as above but prints newline:
void SPLTime(String msg) {
Serial.print("[");
Serial.print(PriUint64<DEC>(micros64()));
Serial.print("]␣");
Serial.println(msg);
}

//
////////////////////////////////////////////////////////////////////////////////

//                                    iso8610 format
                //
//                                          PID: 22
                 //
//
////////////////////////////////////////////////////////////////////////////////

//Take an RTC and output the current time as an ISO 8610 String
:
String rtc_iso8610(RTC_DS3231 &rtcf, bool trailingComma = true
,\
bool startup = false) {
if(startup == true) m.lastProcessIDActive = 0x22;
SYNCFRAM;
DateTime nowf = rtcf.now();
String iso8610 = "";
```

```
    iso8610 += nowf.year()        + (String)"-";
    if(nowf.month() < 10) iso8610 += (String)"0";
    iso8610 += nowf.month()       + (String)"-";
    if(nowf.day() < 10)   iso8610 += (String)"0";
    iso8610 += nowf.day()         + (String)"T";
    if(nowf.hour() < 10)  iso8610 += (String)"0";
    iso8610 += nowf.hour()        + (String)":";
    if(nowf.minute() < 10)   iso8610 += (String)"0";
    iso8610 += nowf.minute()      + (String)":";
    if(nowf.second() < 10)   iso8610 += (String)"0";
    iso8610 += nowf.second();
    if(trailingComma) iso8610 += ",";
    return iso8610;
    }

    //
////////////////////////////////////////////////////////////////////////////////
    //                           iso8610 format - discord
              //
    //                                    PID: 22
              //
    //
////////////////////////////////////////////////////////////////////////////////
    //Take an RTC and output the current time as an ISO 8610 String
:
    String rtcd_iso8610(RTC_DS3231 &rtcf, bool trailingComma = true
,\
    bool startup = false) {
    if(startup == false) m.lastProcessIDActive = 0x22;
    DateTime nowf = rtcf.now();
    String iso8610 = "";
    iso8610 += nowf.year()        + (String)"-";
    if(nowf.month() < 10) iso8610 += (String)"0";
    iso8610 += nowf.month()       + (String)"-";
    if(nowf.day() < 10)   iso8610 += (String)"0";
    iso8610 += nowf.day()         + (String)"T";
    if(nowf.hour() < 10)  iso8610 += (String)"0";
    iso8610 += nowf.hour()        + (String)".";
    if(nowf.minute() < 10)   iso8610 += (String)"0";
    iso8610 += nowf.minute()      + (String)".";
    if(nowf.second() < 10)   iso8610 += (String)"0";
    iso8610 += nowf.second();
    if(trailingComma) iso8610 += ",";
    return iso8610;
    }

    ////////////////////////////////////////////////////////////////////////////////
    //                           iso8610 format - discord                         //
    //                                   PID: 22                                  //
    ////////////////////////////////////////////////////////////////////////////////
    //Take an RTC and output the current time as an ISO 8610 String:
    String rtcd_iso8610(DateTime &nowf, bool trailingComma = true,\
    bool startup = false) {
    if(startup == false) m.lastProcessIDActive = 0x22;
    String iso8610 = "";
    iso8610 += nowf.year()        + (String)"-";
    if(nowf.month() < 10) iso8610 += (String)"0";
    iso8610 += nowf.month()       + (String)"-";
    if(nowf.day() < 10)   iso8610 += (String)"0";
    iso8610 += nowf.day()         + (String)"T";
    if(nowf.hour() < 10)  iso8610 += (String)"0";
    iso8610 += nowf.hour()        + (String)".";
    if(nowf.minute() < 10)   iso8610 += (String)"0";
    iso8610 += nowf.minute()      + (String)".";
    if(nowf.second() < 10)   iso8610 += (String)"0";
    iso8610 += nowf.second();
    if(trailingComma) iso8610 += ",";
    return iso8610;
    }

    //send csv function, general for gateway.
    /*!
   * @brief Send CSV string to Discord. General function for
   * both gateway and node. Retry not implemented in this function to allow developer
   * flexibility for retry strategies.
   * @param token
   * Set URL for webhook token, HTTP 1.3 format
   * @param content
   * Content string - the text. Already formatted for monospaced text.
   * Pointer to ADDRESS of existing content string to save RAM.
   * @param nodeName
```

```
  * Name of node for username. Optional.
  * @param imageUrl
  * URL to profile picture of current message of bot. Optional.
  * @param msp
  * Whether to enclose in monospaced form or not.
  * @return Returns true on success, false on failure to send.
  */
   bool sendDiscordCSV(String token, String content, String
nodeName, String imageUrl, bool msp) {
   Watchdog.reset();
   StaticJsonDocument<4096> NODE; //allocate to stack. ONLY ON
CORTEX M4, M7, OR R5 OR BETTER.
   SPLTime("[DBG]␣Allocated␣JSON␣doc");
   NODE["username"] = nodeName;
   NODE["avatar_url"] = imageUrl;
   if(msp) { NODE["content"] = "‘‘" + content + "‘‘"; }
   else { NODE["content"] = content; }
   //JsonArray embeds1 = NODE.createNestedArray("embeds");
   //JsonObject embed_def1 = embeds1.createNestedObject();
   //embed_def1["description"] = "Courtyard readings - " + a4;
   if(client.connect("discord.com", 443)) {
     WiFi.setLEDs(64,255,255);
     //Send HTTP header to Discord:
     //HTTP header is extremely particular about order of flags,
and Discord is very particular about using HTTP 1.3
     client.println("POST␣" + token + "␣HTTP/1.1");
     client.println("Host:␣discord.com");
     client.println("Accept:␣*/*");
     client.println("Content-Type:␣application/json");
     client.print("Content-Length:␣");
     //compute length of minified JSON:
     client.println(measureJson(NODE));
     client.print("\r\n"); //terminate HEADER by sending a newline
     //Now send JSON header to Discord:
     serializeJson(NODE, client); //This sends without any
additional RAM
     //free((void *)NODE);
     //timeout implementation - 1 second timeout.
     uint32_t timeout = millis();
     uint32_t t_timeout = 1000;
     WiFi.setLEDs(64,0,255);
     while(true) {
     //deal with satcom latency
     if(client.available()) {
       client.stop(); //disconnect!!
       return true;
```

```
      }
      if(millis() > timeout + t_timeout) return false;
      }
    } return false;
    }

    //matching file size is a REQUIREMENT for Discord.
    //Function required to determine
    //Update 29 April: after analysis with a netcat http server, I determined that
    //packet retries or fragmentation is critical. If I drop even ONE packet, or am
    //even ONE byte off, the Cloudflare frontend rejects the transfer with a 400
    //Bad Request.
    //Fragmentation, in all honesty, is a stretch goal. It requires a tremendous
    //amount of work and a major or masters in computer networking.
    bool sendDiscordFileRNG(String token, String content, String nodeName, String imageUrl, bool msp, int cont_len) {
    StaticJsonDocument<4096> NODE; //allocate to stack. ONLY ON CORTEX M4, M7, OR R5 OR BETTER.
    NODE["username"] = nodeName;
    NODE["avatar_url"] = imageUrl;
    if(msp) { NODE["content"] = "‘‘" + content + "‘‘"; }
    else { NODE["content"] = content; }
    //payload_sz = measureJson(client);
    //payload_sz += 312;
    if(client.connectSSL("ptb.discord.com", 443)) {
      WiFi.setLEDs(64,255,255);
      //Send HTTP header to Discord:
      client.println("POST␣" + token + "␣HTTP/1.1");
      client.println("Host:␣ptb.discord.com");
      client.println("User-Agent:␣GNU␣GCC");
      client.println("Accept:␣*/*");
      client.print("Content-Length:␣");
      client.println((String)cont_len);
      client.println("Content-Type:␣multipart/form-data;␣boundary=--000");
      client.println();
      client.println("----000");
      client.println("Content-Disposition:␣form-data;␣name=\"file1\";␣filename=\"aaa.txt\"");
      client.println("Content-Type:␣text/plain\r\n");
      client.println("aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
\r\n");
      client.println("----000--"); //end of file
      uint32_t timeout = millis();
      uint32_t t_timeout = 2000;
      WiFi.setLEDs(64,0,255);
      delay(1000);
      if(client.available()) {
      while(client.available() >0) {
        Serial.write(client.read());
      }
      }
      while(true) {
      //deal with satcom latency
      if(client.available()) {
        client.stop(); //disconnect!!
        return true;
      }
      if(millis() > timeout + t_timeout) return false;
      }
    } return false;
    }

    bool dumpframdiscord(String token, String imageUrl,bool erase,
int fudgeFactor) {
    Watchdog.reset();
    if(client.connect("ptb.discord.com", 443)) {
      //Compute the actual number of bytes transmitted:
      //uint32_t modulo_add = 0;
      //if(((m.ram1_sz-m.ram1_mem_free) % 256) > 0) modulo_add =
(m.ram1_sz-m.ram1_mem_free) % 256;
      uint32_t len = (m.ram1_sz-m.ram1_mem_free)+(uint32_t)121+
fudgeFactor/*+modulo_add*/;
      WiFi.setLEDs(64,255,255);
      //Send HTTP header to Discord:
      client.println("POST␣" + token + "␣HTTP/1.1");
      client.println("Host:␣ptb.discord.com");
      client.println("User-Agent:␣GNU␣GCC");
      client.println("Accept:␣*/*");
      client.print("Content-Length:␣");
      client.println((String)len);
      client.println("Content-Type:␣multipart/form-data;␣boundary=
------000");
      client.println();
      client.println("--------000"); //13 bytes
      client.println("Content-Disposition:␣form-data;␣name=\"file1
\";␣filename=\"dump.txt\""); //67 bytes
```

```
     client.println("Content-Type:␣text/plain\r\n"); //28 bytes
     //Header total (incl. postamble): 121 bytes

     //fill buffer until buffer is full
     //The problem here is that I read in 256 byte sectors. Cloudflare and Discord ALWAYS
     //expect EXACTLY the number of bytes you say you're going to send.
     char block_buf[256] = {'\000'};
     for(uint32_t sz=0;sz<(m.ram1_sz-m.ram1_mem_free);) {
     RAM.read(sz,(uint8_t *)block_buf,sizeof(block_buf)-1);
     client.write((const char *)block_buf);
     sz +=sizeof(block_buf);
     }
     client.print("\r\n--------000--\r\n"); //end of file

     uint32_t timeout = millis();
     uint32_t t_timeout = 2000;
     WiFi.setLEDs(64,0,255);
     Watchdog.reset();
     delay(2000);
     Serial.println("");
     if(client.available()) {
     while(client.available() >0) {
      Serial.write(client.read());
     }
     }
     while(true) {
     Watchdog.reset();
     //deal with satcom latency
     if(client.available()) {
      client.stop(); //disconnect!!
      goto end;
     }
     if(millis() > timeout + t_timeout) {
      SPLTime(F("[discord]␣Failed␣to␣transmit␣file."));
      goto end;
     }
     }
    }
    end:
    //Erase if commanded to do so:
    if(erase == true) {
     RAM.writeEnable(true);
     //wipe:
     for(uint32_t i=0;i<m.ram1_sz;i++) {
```

```
  RAM.write8(i,'\000');
  }
  RAM.writeEnable(false);
  m.ram1_mem_free = m.ram1_sz; //set to start
  SYNCFRAM;
}
strip.setPixelColor(0,0x00FFFFFF);
strip.show();
delay(500);
strip.setPixelColor(0,0);
strip.show();
return false;
}

#define BYTESPACING 16
//Hexdump function:
//robj: I2C FRAM object
//read_struct_ident: start reading at 0x000000 instead of 0x000002
String framhexdump(Adafruit_FRAM_I2C &robj, bool read_struct_ident) {
String dump = "";
dump += "FRAM␣A␣hexdump:\r\n";
uint32_t i = 0;
if(read_struct_ident == false) i = 2;
for(int j=1;i<sizeof(m);i++, j++) {
  char tmp;
  tmp = robj.read(i);
  if(tmp < 0x10) dump += "0";
  //the data:
  dump += String(tmp,HEX);
  //padding:
  if(j == BYTESPACING/4) dump += "␣";
  if(j == BYTESPACING) {
  dump += "\r\n";
  j=0;
  }
}
//on return, dump should be deallocated by default, because I have no
//way of manually calling free() after this.
Serial.print(dump);
return dump;
}

//Hexdump FRAM A to Discord. Leverage previously existing
```

```
function:
    bool dumpstructdiscord(String token, String obj, String
nodeName, String imageUrl, bool msp) {

    Watchdog.reset();
    StaticJsonDocument<6144> NODE; //allocate to stack. ONLY ON
CORTEX M4, M7, OR R5 OR BETTER.
    NODE["username"] = nodeName;
    NODE["avatar_url"] = imageUrl;
    if(msp) { NODE["content"] = "‘‘" + obj + "‘‘"; }
    else { NODE["content"] = obj; }
    //JsonArray embeds1 = NODE.createNestedArray("embeds");
    //JsonObject embed_def1 = embeds1.createNestedObject();
    //embed_def1["description"] = "Courtyard readings - " + a4;
    if(client.connect("discord.com", 443)) {
      WiFi.setLEDs(64,255,255);
      //Send HTTP header to Discord:
      //HTTP header is extremely particular about order of flags,
and Discord is very particular about using HTTP 1.3
      client.println("POST␣" + token + "␣HTTP/1.1");
      client.println("Host:␣discord.com");
      client.println("Accept:␣*/*");
      client.println("Content-Type:␣application/json");
      client.print("Content-Length:␣");
      //compute length of minified JSON:
      client.println(measureJson(NODE));
      client.print("\r\n"); //terminate HEADER by sending a newline
      //Now send JSON header to Discord:
      serializeJson(NODE, client); //This sends without any
additional RAM
      uint32_t timeout = millis();
      //if we don't get a response within 1 sec, something's wrong
      uint32_t t_timeout = 1000;
      WiFi.setLEDs(64,0,255);
      while(true) {
      //deal with satcom latency
      if(client.available()) {
        client.stop(); //disconnect!!
        return true;
      }
      if(millis() > timeout + t_timeout) return false;
      }
    } return false;
    return false;
    }

```

```
//04 May 2022: add contingencies
bool dumpframhttp(bool erase) {
Watchdog.reset();
SPTime(F("Pinging␣Cloudflare...\r\n"));
//Rationale: Builtin network conditions analysis to determine if I should
//send data or not. Expand to dynamic file sizing.
//Analyze ping to Cloudflare (because they're always up)
//Expect no greater than 100ms and no greater than 50ms spread
int ping_result = 0;
int fail_times = 0;
int rpt_times = 8;
int high = 0;
int low = 0;
int avg = 0;
bool send_flag = true;
for(rpt_times = 0;rpt_times < 8;rpt_times++) {
  if(ping_result > 0) {
  ping_result = WiFi.ping(IPAddress(1,1,1,1),64U);
  if(ping_result < low) low = ping_result;
  if(ping_result > high) high = ping_result;
  avg += ping_result;
  SPTime(F("[ping]␣Response␣from␣1.1.1.1:␣"));
  Serial.print(ping_result);
  Serial.println((F("␣ms")));
  }
  else {
  fail_times++;
  SPLTime(F("[ping]␣Destination␣unreachable"));
  }
}
Serial.println();

//fill out working flags:
w.ping_result = ping_result;
w.fail_times  = fail_times;
w.rpt_times   = rpt_times;
w.ping_high   = high;
w.ping_low    = low;
w.ping_avg    = avg;
SPLTime(F("Analysing␣ping..."));
if(high-low <= 50) {
  SPTime(F("␣␣...jitter␣ok␣("));
  Serial.print(ping_result);
  Serial.println(F("␣ms)"));
}
```

```
else {
  SPTime(F("␣␣...jitter␣failed␣("));
  Serial.print(ping_result);
  Serial.println(F("␣ms"));
}
//decimate by 8:
avg /= 8;
if(avg <= 100) {
  SPTime(F("␣␣...latency␣ok␣("));
  Serial.print(avg);
  Serial.println(F("␣ms"));
}
else {
  SPTime(F("␣␣...latency␣failed␣("));
  Serial.print(avg);
  Serial.println(F("␣ms"));
}
if(fail_times > 0) {
  SPTime(F("␣␣...packet␣loss␣failed␣("));
  Serial.print(fail_times);
  Serial.println(F("␣times"));
}
else {
  SPTime(F("␣␣...packet␣loss␣ok␣("));
  Serial.print(fail_times);
  Serial.println(F("␣times"));
}
SPLTime(F("Current␣RSSI␣to␣SSID:␣"));
Serial.print(WiFi.RSSI());
Serial.println(F("␣dBm"));

//now determine if it's okay to send based on available data:
//top-level criteria: packet loss:
if(fail_times == 0) {
  //next criteria: RSSI:
  if(WiFi.RSSI() > -77) {
  //jitter:
  if(high-low <= 50) {
    //latency:
    if(avg <= 100) {
    //now we can send:
    send_flag = true;
    }
  }
  }
}
```

```


    if(send_flag == true) {
    clienthttp.connect("REDACTED", 61126);
      uint32_t len = (m.ram1_sz-m.ram1_mem_free)+(uint32_t)121;
      WiFi.setLEDs(64,255,255);
      //Send HTTP header to Discord:
      clienthttp.println("POST␣/api/wavefront/␣HTTP/1.1");
      clienthttp.println("Host:␣REDACTED");
      clienthttp.println("User-Agent:␣GNU␣GCC");
      clienthttp.println("Accept:␣*/*");
      clienthttp.print("Content-Length:␣");
      clienthttp.println((String)len);
      clienthttp.println("Content-Type:␣multipart/form-data;␣
boundary=------000");
      clienthttp.println();
      clienthttp.println("--------000"); //13 bytes
      clienthttp.println("Content-Disposition:␣form-data;␣name=\"
file1\";␣filename=\"dump.txt\""); //67 bytes
      clienthttp.println("Content-Type:␣text/plain\r\n"); //28
bytes
      //Header total (incl. postamble): 121 bytes



      char block_buf[256] = {'\000'};
      for(uint32_t sz=0;sz<(m.ram1_sz-m.ram1_mem_free);) {
      RAM.read(sz,(uint8_t *)block_buf,sizeof(block_buf)-1);
      clienthttp.print((String)block_buf);
      sz +=sizeof(block_buf);
      delay(1);
      }
      clienthttp.print("\r\n--------000--\r\n"); //end of file
      clienthttp.println(F("Connection:␣close"));

      uint32_t timeout = millis();
      uint32_t t_timeout = 2000;
      WiFi.setLEDs(64,0,255);
      Watchdog.reset();
      delay(2000);
      Serial.println("");
      while(true) {
      Watchdog.reset();
      //deal with satcom latency
      if(clienthttp.available()) {
        while(clienthttp.available() > 0) {
```

```
        Serial.print(clienthttp.read());
        }
        clienthttp.stop(); //disconnect!!
        goto end;
      }
      if(millis() > timeout + t_timeout) {
        SPLTime(F("[http-upload]␣No␣response␣received.␣Retrying␣in␣60␣seconds."));
        return false;
      }
      }
    }
    else {
      SPLTime(F("Criteria␣to␣send␣file␣failed.␣Will␣retry␣until␣conditions␣are␣suitable."));
      return false;
    }

    end:
    //Erase if commanded to do so:
    if(erase == true) {
      RAM.writeEnable(true);
      //wipe:
      for(uint32_t i=0;i<m.ram1_sz;i++) {
      RAM.write8(i,'\000');
      }
      RAM.writeEnable(false);
      m.ram1_mem_free = m.ram1_sz; //set to start
      SYNCFRAM;
      SPLTime(F("[fram]␣FRAM␣erased."));
    }
    strip.setPixelColor(0,0x00FFFFFF);
    strip.show();
    delay(500);
    strip.setPixelColor(0,0);
    strip.show();
    return true;
    }

    // ///////////////////////////////////////////////////////////////////////////
    //                                  WiFi delegate                            //
    //                                     PID: 0F                               //
```

```
  // //////////////////////////////////////////////////////////////////////////////
  //Agent to control WiFi coprocessor, including CONNECTION AT BOOT.
  //Major update May 4: loss of STA reassociation and no known networks found
  //handler.
  uint64_t wifi_delegate(int w_stat) {
  #define CONN_TIMEOUT 30000 //msec
  m.lastProcessIDActive = 0x0F;
  SYNCFRAM;
  Watchdog.reset();
  WiFi.noLowPowerMode();
  bool poll_requested = false;
  uint8_t netsReturned = 0;
  //Basic state check: are we connected to a STA?
  if(w_stat != WL_CONNECTED) {
    WiFi.setLEDs(64 ,0 ,0);
    //Figure out what state we're in and act upon it to return to WL_CONNECTED
    //Worst case:
    if(w_stat == WL_NO_SHIELD) {
    SPTime(F("[wifi_delegate]␣CRITICAL:␣TOTAL␣LOSS␣OF␣WIFI␣COPROCESSOR."));
    Serial.println(F("␣UNABLE␣TO␣TRANSFER␣ANY␣DATA,␣PERIOD."));
    return micros64();
    }
    //we've probably just restarted, so proceed with SSID search
    else {
    connect: //GOTO used as alias only within this function as a lazy way to
    //transfer control.
    Watchdog.disable();
    //Reenable for a longer time to allow for scanning:
    Watchdog.enable(32000);
    SPLTime(F("[wifi_delegate]␣Starting␣WiFi␣SSID␣search..."));
    netsReturned = WiFi.scanNetworks();
    Watchdog.enable(16000);
    SPLTime(F("[Watchdog]␣NOTICE:␣Watchdog␣enabled,␣16s␣timeout"));
    SPTime(F("[wifi_delegate]␣Returned␣"));
    Serial.print(netsReturned);
    Serial.println(F("␣networks␣in␣range."));
    if(WiFi.status() == WL_NO_SSID_AVAIL) {
      SPTime(F("[wifi_delegate]␣Warning:␣No␣SSIDs␣found.␣Reducing
␣sensor"));
       Serial.println(F("␣polling␣rate␣until␣SSIDs␣are␣in␣range.")
);
       //actually follow up on this later
     return micros64();
     }
     for(int thisNet = 0; thisNet < netsReturned; thisNet++) {
       const char * tmp_ssid = WiFi.SSID(thisNet);
       uint8_t tmp_bssid[6];
       //look up if this entry is in our table:
       for(int i=0;i<NUM_SSID_ENTRIES;i++) {
       //INCLUDE BSSID LOOKUP, this will be really long if
statement:
       uint8_t *tmp_bssid_actual = {WiFi.BSSID(thisNet, tmp_bssid)
};
       //printMacAddress(tmp_bssid_actual);
       if(strcmp((const char *)ssidlist[i],(const char *)tmp_ssid)
 == 0 || (\
       tmp_bssid_actual[0] == bssid_1[0] && \
       tmp_bssid_actual[1] == bssid_1[1] && \
       tmp_bssid_actual[2] == bssid_1[2] && \
       tmp_bssid_actual[3] == bssid_1[3] && \
       tmp_bssid_actual[4] == bssid_1[4] && \
       tmp_bssid_actual[5] == bssid_1[5])) {
         //we're done searching - attempt connection and return.
         SPTime(F("[wifi_delegate]␣Attempting␣to␣connect␣to␣"));
         Serial.println(ssidlist[i]);
         //decide if we're open or WPA/WPA2:
         Watchdog.disable();
         SPLTime(F("[watchdog]␣NOTICE:␣Watchdog␣disabled"));
         uint32_t tot = millis();
         if(strcmp(passlist[i],"NONE") == 0) {
         while(WiFi.status() != WL_CONNECTED && (millis() < tot +
CONN_TIMEOUT)) {
           WiFi.begin(ssidlist[i]);
           delay(6000);
         }
         Watchdog.enable(16000);
         SPLTime(F("[Watchdog]␣NOTICE:␣Watchdog␣enabled,␣16s␣
timeout"));
         if(WiFi.status() == WL_CONNECTED) {
           SPLTime(F("[wifi_delegate]␣Connected"));
           WiFi.setLEDs(64,0,255);
           return micros64();
         } else goto failw;
         }
```

```
          else {
          while(WiFi.status() != WL_CONNECTED && (millis() < tot + CONN_TIMEOUT)) {
            WiFi.begin(ssidlist[i],passlist[i]);
            delay(6000);
          }
          Watchdog.enable(16000);
          SPLTime(F("[Watchdog]␣NOTICE:␣Watchdog␣enabled,␣16s␣timeout"));
          if(WiFi.status() == WL_CONNECTED) {
            SPLTime(F("[wifi_delegate]␣Connected"));
            WiFi.setLEDs(64,0,255);
            w.RES_CONNECTED_TO_WIFI = true;
            SYNCFRAM;
            return micros64();
          }
          }
          failw:
          if(WiFi.status() == WL_CONNECT_FAILED || WiFi.status() == WL_CONNECTION_LOST) {
          SPLTime(F("[wifi_delegate]␣Failed␣to␣connect.␣Retrying␣in␣1␣minute."));
          w.RES_CONNECTED_TO_WIFI = false;
          WiFi.setLEDs(0,0,0);
          }
          return micros64();
        }
        else {
          //Additional section added 04 May 2022:
          //Handle zero known networks in range:
          SPLTime(F("[wifi_delegate]␣No␣known␣networks␣in␣range.␣Pushing␣out␣respawn␣to␣5␣minutes."));
          w.RES_CONNECTED_TO_WIFI = false;
          WiFi.setLEDs(0,0,0);
          WiFi.lowPowerMode();
          SYNCFRAM;
        }
        }
        SPLTime(F("[wifi_delegate]␣Failed␣to␣find␣network␣in␣list␣but␣networks␣available.␣Trying␣hidden␣network␣DB:"));
        Watchdog.disable();
        SPLTime(F("[watchdog]␣NOTICE:␣Watchdog␣disabled"));
        uint32_t tot = millis();
        while(WiFi.status() != WL_CONNECTED && (millis() < tot + CONN_TIMEOUT)) {
        WiFi.begin(ssid1,pass1);
```

```
    delay(6000);
    }
    Watchdog.enable(16000);
    SPLTime(F("[watchdog]␣NOTICE:␣Watchdog␣enabled,␣16s␣timeout"));
    if(WiFi.status() == WL_CONNECTED) {
    SPLTime(F("[wifi_delegate]␣Connected␣to␣Orion."));
    WiFi.setLEDs(64,0,255);
    w.RES_CONNECTED_TO_WIFI = true;
    return micros64();
    }
  }
  }
}
//IN CASE MY PROGRAMMING IS A PIECE OF SHIT:
w.RES_CONNECTED_TO_WIFI = true;
return micros64();
}

// //////////////////////////////////////////////////////////////////////////////
//                             debug blink routine                            //
//                                   PID: A0                                  //
// //////////////////////////////////////////////////////////////////////////////
//divert here for errors and info blinks so LCD can be saved for data out
//reduce code mess in setup and main
#define BLINK_GPS_T_SYNC              44
#define BLINK_INIT_SETUP              32
#define BLINK_LINK_ACQUISITION        36
#define BLINK_CORE_ERR_UNRECOVERABLE  255
void dbg_blink(uint8_t blink_code) {
m.lastProcessIDActive = 0xA0;
switch(blink_code) {
  ////////
  //initial start, formatting and FRAM integrity check
  ////////
  case BLINK_GPS_T_SYNC:
  goto dbg_blink_end;
  case BLINK_INIT_SETUP:
  for(int i=0;i<32;i++) {
    //blink to show state is in initial setup
    strip.setPixelColor(0,0x00FF0000);
    strip.show();
    delay(50);
    strip.setPixelColor(0,0);
    strip.show();
    delay(50);
  }
  goto dbg_blink_end;
  case BLINK_LINK_ACQUISITION:
  Watchdog.reset();
  strip.setPixelColor(0,0x00FF0000);
  strip.show();
  delay(750);
  strip.setPixelColor(0,0);
  strip.show();
  delay(500);
  goto dbg_blink_end;
  case 255: //core error, unrecoverable
  strip.setPixelColor(0,0x00FF0000);
  strip.show();
  //do NOT disable watchdog in case of problem! It can resolve
  //the problem on its own!!
  //Watchdog.disable();
  for(;;); //end of CPU execution
}
dbg_blink_end:
asm("nop");
}


// ///////////////////////////////////////////////////////////////////////////////
//                                  set RTC                                    //
//                                  PID: 20                                    //
// ///////////////////////////////////////////////////////////////////////////////
//Set an external battery-backed, temperature-compensated RTC from GPS time.
bool setRTCGPS(RTC_DS3231 &rtc, SFE_UBLOX_GPS &GPS) {
m.lastProcessIDActive = 0x20;
rtc.adjust(DateTime(GPS.getYear(),\
          GPS.getMonth(),\
          GPS.getDay(),\
          GPS.getHour(),\
          GPS.getMinute(),\
          GPS.getSecond()));
return true;
}

//
//////////////////////////////////////////////////////////////////////////////
//                                   free RAM
             //
//                                   PID: N/A
             //
//
//////////////////////////////////////////////////////////////////////////////
//Check how much RAM is free between heap and stack
//make this universal:
#ifdef __arm__
extern "C" char *sbrk(int incr);
#else
extern char *__brkval;
#endif

int freeRam() {
char top;
#ifdef __arm__
return &top - reinterpret_cast<char*>(sbrk(0));
#elif defined(CORE_TEENSY) || (ARDUINO > 103 && ARDUINO != 151)
return &top - __brkval;
#else
return __brkval ? &top - __brkval : &top - __malloc_heap_start;
#endif
}



//
//////////////////////////////////////////////////////////////////////////////
//                                  update  RTC
             //
//                                    PID: 21
             //
```

```
   //
////////////////////////////////////////////////////////////////////////////

   //Update RTC on the condition that the GPS time is valid
   //and that the date is greater than 2021.
   uint64_t updatertc(SFE_UBLOX_GPS &gnss, RTC_DS3231 &rtcd) {
   m.lastProcessIDActive = 0x21;
   if(gnss.getTimeValid() && gnss.getYear() > 2021) setRTCGPS(rtcd
, gnss);
   Serial.print(F("DS3231␣synced␣to␣GPS."));
   return micros64();
   }

   //
////////////////////////////////////////////////////////////////////////////

   //                                restart CPU
              //
   //                                  PID: FF
              //
   //
////////////////////////////////////////////////////////////////////////////

   //Writes a 1 to the SYSRESETREQ register to restart the CPU.
   void restart(void)  {
   m.lastProcessIDActive = 0xFF;
   SYNCFRAM;
   __asm volatile ("cpsid␣i" ::: "memory"); //disable interrupt
reporting
   __asm volatile ("dsb␣0xF":::"memory"); //commit
   SCB->AIRCR = ((0x5FAUL << SCB_AIRCR_VECTKEY_Pos)\
     | SCB_AIRCR_SYSRESETREQ_Msk); //write to system control block
 to reset
   __asm volatile ("dsb␣0xF":::"memory");
   for(;;) __asm volatile("nop");
   }

   //
////////////////////////////////////////////////////////////////////////////

   //                          iso8610 format - struct
              //
   //                                  PID: 23
              //
   //
////////////////////////////////////////////////////////////////////////////
```

```
String struct_iso8610(master &s, bool trailingComma = true) {
m.lastProcessIDActive = 0x23;
String iso8610 = "";
iso8610 += s.lastYear        + (String)"-";
if(s.lastMonth < 10) iso8610 += (String)"0";
iso8610 += s.lastMonth       + (String)"-";
if(s.lastDay < 10)   iso8610 += (String)"0";
iso8610 += s.lastDay         + (String)"T";
if(s.lastHour < 10)  iso8610 += (String)"0";
iso8610 += s.lastHour        + (String)":";
if(s.lastMinute < 10)   iso8610 += (String)"0";
iso8610 += s.lastMinute      + (String)":";
if(s.lastSecond < 10)   iso8610 += (String)"0";
iso8610 += s.lastSecond;
if(trailingComma) iso8610 += ",";
return iso8610;
}
//
//////////////////////////////////////////////////////////////////////////////
//                             discord BOT login
          //
//                                  PID: C0
          //
//
//////////////////////////////////////////////////////////////////////////////
//Discord functions:
bool discord_login(WiFiSSLClient &cl, char *bot_token) {
m.lastProcessIDActive = 0xC0;
return true;
}

//
//////////////////////////////////////////////////////////////////////////////
//                                read SCD30
          //
//                                 PID:  40
          //
//
//////////////////////////////////////////////////////////////////////////////
/*
* Read CO2 sensor samp_times_blocking times. Outputs max, min,
avg, 1-sigma
  * stddev, and read time in usec.
  *
  * @param samp_times_blocking Number of times the sensor is polled.
  * @param CO2Object SCD30 sensor instance, if there are multiple
  * @return String with the output values in the order specified above
  */
   String r_scd30(int samp_times_blocking, Adafruit_SCD30 & CO2Object) {
   m.lastProcessIDActive = 0x40;
   Watchdog.reset();
   //[0]: temperature
   //[1]: RH
   //[2]: CO2 (ppm)
   float max[3] = {-999.};
   float min[3] = {999.};
   float avg[3] = {0.};
   float stddev[3] = {0.};

   uint64_t read_time;

   read_time = micros64(); //set the initial timestamp of read
   //Serial.println((uint32_t)read_time);

   //for max compatibility, an int is passed as input. Error and integrity
   //check is therefore needed. To avoid an exception, we default to 5 reads
   //if an out-of-bounds value is detected.
   if(samp_times_blocking < 2 || samp_times_blocking > 64) {
     //out of bounds is greater than 64 for a good reason
     samp_times_blocking = 5; //least number of points for statistical var
   }

   float latestData[3] = {0,0,0};
   for(int i=0;i<samp_times_blocking;i++) {
     //WAIT FOR VALID DATA:
     while(!CO2Object.dataReady());
     co2Retry:
     if(CO2Object.dataReady()) {
     //Serial.println("data ready");
     Watchdog.reset();
     CO2Object.read();
```

```
      //read latest data:
      latestData[0] = CO2Object.temperature;
      latestData[1] = CO2Object.relative_humidity;
      //sometimes the sensor doesn't read correctly - re-read:
      if(CO2Object.CO2 == 0) goto co2Retry;
      latestData[2] = CO2Object.CO2;
      //populate statistical values:
      for(int j=0;j<3;j++) {
        //max?
        if(max[j] < latestData[j] || max[j] == 0) max[j] = latestData[j];
        //min?
        if(min[j] > latestData[j] || min[j] == 0) min[j] = latestData[j];
        //average accumulator:
        avg[j] += latestData[j];
      }
      }
      else goto co2Retry;
    }
    //compute oneshot statistics:
    for(int j=0;j<3;j++) {
      //average:
      avg[j] /= (float)(samp_times_blocking);
      //unbiased sample variance:
      stddev[j] = sqrt((1.0/((float)samp_times_blocking-1.0))*pow((latestData[j]-avg[j]),2));
    }

    //save to master struct:
    m.scd30_temp_max    = max[0];
    m.scd30_temp_min    = min[0];
    m.scd30_temp_avg    = avg[0];
    m.scd30_temp_stddev = stddev[0];
    m.scd30_rh_max      = max[1];
    m.scd30_rh_min      = min[1];
    m.scd30_rh_avg      = avg[1];
    m.scd30_rh_stddev   = stddev[1];
    m.scd30_co2_max     = max[2];
    m.scd30_co2_min     = min[2];
    w.sci_scd30_co2_prev= m.scd30_co2_avg;
    m.scd30_co2_avg     = avg[2];
    m.scd30_co2_stddev  = stddev[2];
    //Serial.println("complete struct");

    //flags:
```

```
   if(m.scd30_co2_avg > 2000.0f) w.SCI_SCD30_CO2_HIGH_2000_PPM = true;
   else if(m.scd30_co2_avg < 2000.0f) w.SCI_SCD30_CO2_HIGH_2000_PPM = false;

   //save to String:
   String outString = "";
   outString += "0,"; //Sensor ID for CSV readability
   for(int i=0;i<3;i++) {
     outString += max[i] + (String)",";
     outString += min[i] + (String)",";
     outString += avg[i] + (String)",";
     outString += stddev[i] + (String)",";
   }
   m.scd30_micros_op   = (uint32_t)(micros64()-read_time);
   outString += (uint32_t)m.scd30_micros_op + (String)",";
   //Serial.println("complete string");

   SYNCFRAM;
   return outString;
   }

   // //////////////////////////////////////////////////////////////////////////////
   //                               read ADT7410                               //
   //                                 PID: 41                                  //
   // //////////////////////////////////////////////////////////////////////////////
   /*
  * Read ADT7410 sensor samp_times_blocking times at a target target_samp_rate
  * sample rate (in Hertz). Upper limit hard coded to 60 Hz.
  * @param samp_times_blocking Read sensor this many times.
  * @param target_samp_rate Target sample rate in Hz. Not guaranteed due to
  * I2C bus. Hard-coded limit at 60 Hz.
  * @param ADT7410Object ADT7410 sensor instance, if there are multiple.
  * @return Outputs preformatted String with values in the following order:
  * max, min, average, standard deviation, and actual read time in usec. Note
  * that the raw values are outputted to the master struct as well.
  */
   String r_adt7410(int samp_times_blocking, float target_samp_rate,\
   Adafruit_ADT7410 &ADT7410Object) {
   Watchdog.reset();
   m.lastProcessIDActive = 0x41;
   #define CEIL_SAMPS_ADT7410          65535
   #define CEIL_SAMP_RATE_ADT7410      60.0

   //initialize these data points to something impossible to achieve:
   float max = -999.;
   float min = 999.;
   float avg = 0.;
   float stddev = 0.;
   uint64_t read_time;
   read_time = micros64();
   //Serial.println((uint32_t)read_time);

   //for max compatibility, an int is passed as input. Error and integrity
   //check is therefore needed. To avoid an exception, we default to 5 reads
   //if an out-of-bounds value is detected.
   if(samp_times_blocking < 2 || samp_times_blocking > CEIL_SAMPS_ADT7410) {
     samp_times_blocking = 5; //least number of points for statistical var
   }

   //compute the read delay required to meet the frequency specified (micros):
   uint64_t read_delay = (uint64_t)((float)(1.0/target_samp_rate)*1000000.0);
   if(read_delay < 16667ULL) read_delay = 16667ULL; //clamp to 60 Hz max samp_rate
   //Serial.println("reading data...");
   //Serial.println((uint32_t)read_delay);
   float latestData = 0.0;
   uint64_t read_delay_tmp;
   for(int i=0;i<samp_times_blocking;i++) {
     read_delay_tmp = micros64();
     //read latest data:
     latestData = ADT7410Object.readTempC();
```

```
      //populate statistical values:
      //max finder?
      if(max < latestData) max = latestData;
      //min finder?
      if(min > latestData) min = latestData;
      //average accumulator:
      avg += latestData;

      //loop delay:
      //will never be less than zero (fatal if otherwise)
      if(micros64() - read_delay_tmp < read_delay) {
      //proper way of truncating 64 bit to 32 bit (?)
      //mask number
      delayMicroseconds((micros64()-read_delay_tmp+read_delay) & 0xFFFFFFFF);
      }
    }
    //Serial.println("...done.");

    //compute oneshot statistics:
    //average:
    avg /= (samp_times_blocking);
    //unbiased sample variance:
    stddev = sqrt((1.0/((float)samp_times_blocking-1.0))*pow((latestData-avg),2));

    //save to master struct:
    m.adt7410_temp_max    = max;
    m.adt7410_temp_min    = min;
    //save previous temperature for derivative data:
    w.sci_adt7410_avgtmp_prev = m.adt7410_temp_avg;
    m.adt7410_temp_avg    = avg;
    m.adt7410_temp_stddev = stddev;

    //science flags:
    if(m.adt7410_temp_avg > 45.0f) w.SCI_ADT7410_HIGH_TEMP_FLAG = true;
    else if(m.adt7410_temp_avg < 45.0f) w.SCI_ADT7410_HIGH_TEMP_FLAG = false;
    if(abs(m.adt7410_temp_avg - w.sci_adt7410_avgtmp_prev) > 1.0f) w.SCI_ADT7410_LARGE_DT_DT = true;
    else if(abs(m.adt7410_temp_avg - w.sci_adt7410_avgtmp_prev) < 1.0f) w.SCI_ADT7410_LARGE_DT_DT = false;

    //save to String object:
    String outString = "";
```

```
outString += "1,";
outString += max + (String)",";
outString += min + (String)",";
outString += avg + (String)",";
outString += stddev + (String)",";
m.adt7410_micros_op = (uint32_t)(micros64() - read_time);
//beware, this subroutine CANNOT run more than 71.6 minutes, total.
outString += (uint32_t)m.adt7410_micros_op + (String)",";
//return the csv string:
///Serial.println("complete ADT7410");

SYNCFRAM;
return outString;
}

// ////////////////////////////////////////////////////////////////////////////////
//                                read BME680                                    //
//                                  PID:  42                                     //
// ////////////////////////////////////////////////////////////////////////////////
/*
* Read BME680 sensor samp_times_blocking times at a target target_samp_rate
* sample rate (in Hertz). Upper limit hard coded to 4 Hz.
* @param samp_times_blocking Read sensor this many times.
* @param target_samp_rate Target sample rate in Hz. Not guaranteed due to
* I2C bus. Hard-coded limit at 4 Hz.
* @param BME680Object BME680 sensor instance, if there are multiple.
* @return Outputs preformatted String with values in the following order:
* max, min, average, standard deviation, and actual read time in usec. Note
* that the raw values are outputted to the master struct as well.
*/
String r_bme680(int samp_times_blocking, float target_samp_rate,\
Adafruit_BME680 &BME680Object) {
m.lastProcessIDActive = 0x42;
Watchdog.reset();
#define CEIL_SAMPS_BME680 256
#define CEIL_PERIOD_BME680 250000ULL //microseconds, 4 Hz.

//be sure to initialize with arbitrarily large values
//[0]: temperature
//[1]: RH
//[2]: pressure(Pa)
//[3]: gas sensor resistance (ohm)
float max[4] = {-9999999.}; //larger for pressure
float min[4] = {9999999.};
float avg[4] = {0.};
float stddev[4] = {0.};

//be sure to explicitly specify ANY numbers as 123456ULL for uint64_t!!!
uint64_t read_time;
read_time = micros64(); //set the initial timestamp of read


//for max compatibility, an int is passed as input. Error and integrity
//check is therefore needed. To avoid an exception, we default to 5 reads
//if an out-of-bounds value is detected.
if(samp_times_blocking < 2 || samp_times_blocking > CEIL_SAMPS_BME680) {
  samp_times_blocking = 5; //least number of points for statistical var
}

//compute the read delay required to meet the frequency specified (micros):
uint64_t read_delay = (uint64_t)((float)(1.0/target_samp_rate)*1000000.0);
//clamp to 4 Hz max samp_rate:
if(read_delay < CEIL_PERIOD_BME680) read_delay = CEIL_PERIOD_BME680;

float latestData[4] = {0,0,0,0};
uint64_t read_delay_tmp;
for(int i=0;i<samp_times_blocking;i++) {
  read_delay_tmp = micros64();
  Watchdog.reset();
  //WAIT FOR VALID DATA:
```

```
      if(!BME680Object.performReading()) {
      Serial.println("[ERROR]␣Failed␣to␣read␣BME680.");
      };
      //read latest data:
      latestData[0] = bme.temperature;
      latestData[1] = bme.humidity;
      latestData[2] = (float)bme.pressure;
      latestData[3] = (float)bme.gas_resistance;
      //populate statistical values:
      for(int j=0;j<4;j++) {
      //max?
      if(max[j] < latestData[j] || max[j] == 0) max[j] = latestData[j];
      //min?
      if(min[j] > latestData[j] || min[j] == 0) min[j] = latestData[j];
      //average accumulator:
      avg[j] += latestData[j];
      }

      //loop delay:
      //will never be less than zero (fatal if otherwise)
      if(micros64() - read_delay_tmp < read_delay) {
      //proper way of truncating 64 bit to 32 bit (?)
      //mask number
      delayMicroseconds((micros64()-read_delay_tmp+read_delay) & 0xFFFFFFFF);
      }
    }
    //compute oneshot statistics:
    for(int j=0;j<4;j++) {
      //average:
      avg[j] /= (samp_times_blocking);
      //unbiased sample variance:
      stddev[j] = sqrt((1.0/((float)samp_times_blocking-1.0))*pow((latestData[j]-avg[j]),2));
    }

    //save to master struct:
    m.bme680_temp_max =             max[0];
    m.bme680_temp_min =             min[0];
    m.bme680_temp_avg =             avg[0];
    m.bme680_temp_stddev =          stddev[0];
    m.bme680_rh_max =               max[1];
    m.bme680_rh_min =               min[1];
    m.bme680_rh_avg =               avg[1];
```

```
  m.bme680_rh_stddev =           stddev[1];
  m.bme680_prs_max =             max[2];
  m.bme680_prs_min =             min[2];
  m.bme680_prs_avg =             avg[2];
  m.bme680_prs_stddev =          stddev[2];
  m.bme680_gas_res_max =         max[3];
  m.bme680_gas_res_min =         min[3];
  m.bme680_gas_res_avg =         avg[3];
  m.bme680_gas_res_stddev =      stddev[3];

  //save to String (there is a way to do this with a union_t all at once)
  String outString = "";
  outString += "2,"; //Sensor ID for CSV readability
  for(int i=0;i<4;i++) {
    outString += max[i] + (String)",";
    outString += min[i] + (String)",";
    outString += avg[i] + (String)",";
    outString += stddev[i] + (String)",";
  }
  m.bme680_micros_op   = (uint32_t)(micros64()-read_time);
  outString += (uint32_t)m.bme680_micros_op + (String)",";

  SYNCFRAM;
  return outString;
  }

  // //////////////////////////////////////////////////////////////////////////////
  //                               read INA219                                    //
  //                                 PID:  4B                                     //
  // //////////////////////////////////////////////////////////////////////////////

  /*
 * Read INA219 sensor samp_times_blocking times at a target target_samp_rate
 * sample rate (in Hertz). Upper limit hard coded to 100 Hz.
 * @param samp_times_blocking Read sensor this many times.
 * @param target_samp_rate Target sample rate in Hz. Not guaranteed due to
 * I2C bus. Hard-coded limit at 100 Hz.
 * @param obj INA219 sensor instance, if there are multiple.
```

```
  * @return Outputs preformatted String with values in the following order:
  * max, min, average, standard deviation, and actual read time in usec. Note
  * that the raw values are outputted to the master struct as well.
  */
   String r_ina219(int samp_times_blocking, float target_samp_rate,\
   Adafruit_INA219 &BME680Object) {
   m.lastProcessIDActive = 0x4B;
   #define NUM_DATAPOINTS 3
   Watchdog.reset();
   #define CEIL_SAMPS_BME680 256
   #define CEIL_PERIOD_BME680 10000ULL //microseconds, 100 Hz.

   //be sure to initialize with arbitrarily large values
   //[0]: voltage
   //[1]: current
   //[2]: power
   float max[NUM_DATAPOINTS] = {-9999999.}; //larger for pressure
   float min[NUM_DATAPOINTS] = {9999999.};
   float avg[NUM_DATAPOINTS] = {0.};
   float stddev[NUM_DATAPOINTS] = {0.};

   //be sure to explicitly specify ANY numbers as 123456ULL for uint64_t!!!
   uint64_t read_time;
   read_time = micros64(); //set the initial timestamp of read


   //for max compatibility, an int is passed as input. Error and integrity
   //check is therefore needed. To avoid an exception, we default to 5 reads
   //if an out-of-bounds value is detected.
   if(samp_times_blocking < 2 || samp_times_blocking > CEIL_SAMPS_BME680) {
      samp_times_blocking = 5; //least number of points for statistical var
   }

   //compute the read delay required to meet the frequency specified (micros):
   uint64_t read_delay = (uint64_t)((float)(1.0/target_samp_rate)*1000000.0);
```

```
    //clamp to 100 Hz max samp_rate:
    if(read_delay < CEIL_PERIOD_BME680) read_delay = CEIL_PERIOD_BME680;

    float latestData[NUM_DATAPOINTS] = {0,0,0};
    uint64_t read_delay_tmp;
    for(int i=0;i<samp_times_blocking;i++) {
      read_delay_tmp = micros64();
      Watchdog.reset();
      //read latest data:
      latestData[0] = ina.getBusVoltage_V();
      latestData[1] = ina.getCurrent_mA();
      latestData[2] = ina.getPower_mW();
      //populate statistical values:
      for(int j=0;j<NUM_DATAPOINTS;j++) {
      //max?
      if(max[j] < latestData[j] || max[j] == 0) max[j] = latestData[j];
      //min?
      if(min[j] > latestData[j] || min[j] == 0) min[j] = latestData[j];
      //average accumulator:
      avg[j] += latestData[j];
      }

      //loop delay:
      //will never be less than zero (fatal if otherwise)
      if(micros64() - read_delay_tmp < read_delay) {
      //proper way of truncating 64 bit to 32 bit (?)
      //mask number
      delayMicroseconds((micros64()-read_delay_tmp+read_delay) & 0xFFFFFFFF);
      }
    }
    //compute oneshot statistics:
    for(int j=0;j<NUM_DATAPOINTS;j++) {
      //average:
      avg[j] /= (samp_times_blocking);
      //unbiased sample variance:
      stddev[j] = sqrt((1.0/((float)samp_times_blocking-1.0))*pow((latestData[j]-avg[j]),2));
    }

    //save to master struct:
    m.ina_volt_max =            max[0];
    m.ina_volt_min =            min[0];
```

```
  m.ina_volt_avg =            avg[0];
  m.ina_volt_sdv =         stddev[0];
  m.ina_curr_max =            max[1];
  m.ina_curr_min =            min[1];
  m.ina_curr_avg =            avg[1];
  m.ina_curr_sdv =         stddev[1];
  m.ina_powr_max =            max[2];
  m.ina_powr_min =            min[2];
  m.ina_powr_avg =            avg[2];
  m.ina_powr_sdv =         stddev[2];

  //save to String (there is a way to do this with a union_t all at once)
  String outString = "";
  outString += "2,"; //Sensor ID for CSV readability
  for(int i=0;i<NUM_DATAPOINTS;i++) {
    outString += max[i] + (String)",";
    outString += min[i] + (String)",";
    outString += String(avg[i],4) + (String)",";
    outString += stddev[i] + (String)",";
  }
  m.ina_micros_op   = (uint32_t)(micros64()-read_time);
  outString += (uint32_t)m.ina_micros_op + (String)",";

  SYNCFRAM;
  return outString;
  }

  // /////////////////////////////////////////////////////////////////////////////
  //                                read CCS811                                //
  //                                 PID:  43                                  //
  // /////////////////////////////////////////////////////////////////////////////

  /*
 * Read CCS811 sensor samp_times_blocking times at a target target_samp_rate
 * sample rate (in Hertz). Upper limit hard coded to 4 Hz.
 * @param samp_times_blocking Read sensor this many times.
 * @param target_samp_rate Target sample rate in Hz. Not guaranteed due to
 * I2C bus. Hard-coded limit at 4 Hz.
```

```
   * @param obj CCS811 sensor instance, if there are multiple.
   * @return Outputs preformatted String with values in the following order:
   * max, min, average, standard deviation, and actual read time in usec. Note
   * that the raw values are outputted to the master struct as well.
   */
    String r_ccs811(int samp_times_blocking, float target_samp_rate,\
    Adafruit_CCS811 &obj) {
    #define NUM_DATA_OBJS 2 //tvoc, eco2
    #define CEIL_SAMPS_BME680 256
    #define CEIL_PERIOD_BME680 250000ULL //microseconds, 4 Hz.
    m.lastProcessIDActive = 0x43;
    Watchdog.reset();

    //patch 04 May 2022: temperature and humidity compensation for read:
    obj.setEnvironmentalData(m.scd30_rh_avg,m.adt7410_temp_avg);

    //be sure to initialize with arbitrarily large values
    //[0]: tvoc (ppb)
    //[1]: eco2 (ppm derived)
    float max[NUM_DATA_OBJS] = {-9999999.}; //larger for pressure
    float min[NUM_DATA_OBJS] = {9999999.};
    float avg[NUM_DATA_OBJS] = {0.};
    float stddev[NUM_DATA_OBJS] = {0.};

    //be sure to explicitly specify ANY numbers as 123456ULL for uint64_t!!!
    uint64_t read_time;
    read_time = micros64(); //set the initial timestamp of read

    //for max compatibility, an int is passed as input. Error and integrity
    //check is therefore needed. To avoid an exception, we default to 5 reads
    //if an out-of-bounds value is detected.
    if(samp_times_blocking < 2 || samp_times_blocking > CEIL_SAMPS_BME680) {
      samp_times_blocking = 5; //least number of points for statistical var
    }

    //compute the read delay required to meet the frequency
```

```
specified (micros):
  uint64_t read_delay = (uint64_t)((float)(1.0/target_samp_rate)
*1000000.0);
  //clamp to 4 Hz max samp_rate:
  if(read_delay < CEIL_PERIOD_BME680) read_delay =
CEIL_PERIOD_BME680;

  float latestData[NUM_DATA_OBJS] = {0,0};
  uint64_t read_delay_tmp;
  for(int i=0;i<samp_times_blocking;i++) {
    Watchdog.reset();
    read_delay_tmp = micros64();
    //WAIT FOR VALID DATA:
    if(obj.available()) {
    if(!obj.readData()) {
      //read latest data:
      latestData[0] = obj.getTVOC();
      latestData[1] = obj.geteCO2();
      //populate statistical values:
      for(int j=0;j<NUM_DATA_OBJS;j++) {
      //max?
      if(max[j] < latestData[j] || max[j] == 0) max[j] =
latestData[j];
      //min?
      if(min[j] > latestData[j] || min[j] == 0) min[j] =
latestData[j];
      //average accumulator:
      avg[j] += latestData[j];
      }
      //loop delay:
      //will never be less than zero (fatal if otherwise)
      if(micros64() - read_delay_tmp < read_delay) {
      //proper way of truncating 64 bit to 32 bit (?)
      //mask number
      delayMicroseconds((micros64()-read_delay_tmp+read_delay) &
0xFFFFFFFF);
      }
    }
    }
  }
  //compute oneshot statistics:
  for(int j=0;j<NUM_DATA_OBJS;j++) {
    //average:
    avg[j] /= (samp_times_blocking);
    //unbiased sample variance:
    stddev[j] = sqrt((1.0/((float)samp_times_blocking-1.0))*
pow((latestData[j]-avg[j]),2));
   }

   //save to master struct:
   m.ccs811_tvoc_max =            max[0];
   m.ccs811_tvoc_min =            min[0];
   m.ccs811_tvoc_avg =            avg[0];
   m.ccs811_tvoc_stddev =         stddev[0];
   m.ccs811_eco2_max =            max[1];
   m.ccs811_tvoc_min =            min[1];
   m.ccs811_tvoc_avg =            avg[1];
   m.ccs811_tvoc_stddev =         stddev[1];

   //save to String (there is a way to do this with a union_t all
at once)
   String outString = "";
   outString += "3,"; //Sensor ID for CSV readability
   for(int i=0;i<NUM_DATA_OBJS;i++) {
     outString += max[i] + (String)",";
     outString += min[i] + (String)",";
     outString += avg[i] + (String)",";
     outString += stddev[i] + (String)",";
   }
   m.ccs811_micros_op   = (uint32_t)(micros64()-read_time);
   outString += (uint32_t)m.ccs811_micros_op + (String)",";

   SYNCFRAM;
   return outString;
   }

   //
//////////////////////////////////////////////////////////////////////////

   //                                read mag
              //
   //                                PID:  44
              //
   //
//////////////////////////////////////////////////////////////////////////

   /*
  * Read LIS3MDL sensor samp_times_blocking times at a target
target_samp_rate
  * sample rate (in Hertz). Upper limit hard coded to 100 Hz.
  *
  * Magnetometry data, if filtered, can be important in detecting
```

```
solar storm
  * effects, geological changes, and even presence (if the sensor is very well
  * characterized.)
  *
  * @param samp_times_blocking Read sensor this many times.
  * @param target_samp_rate Target sample rate in Hz. Not guaranteed due to
  * I2C bus. Hard-coded limit at 100 Hz.
  * @param obj Magnetometer sensor instance, if there are multiple.
  * @return Outputs preformatted String with values in the following order:
  * max, min, average, standard deviation, and actual read time in usec. Note
  * that the raw values are outputted to the master struct as well.
  */
   String r_mag(int samp_times_blocking, float target_samp_rate,\
   Adafruit_LIS3MDL &obj) {
   m.lastProcessIDActive = 0x44;
   Watchdog.reset();

   //Different from versions above: written to be more general.
   #define NUM_DATA_OBJS 3 //x,y,z
   #define CEIL_SAMPS_SENS 4096
   #define CEIL_PERIOD_SENS 10000ULL //microseconds, 4 Hz.

   //be sure to initialize with arbitrarily large values
   //[0]: mag X
   //[1]: mag Y
   //[2]: mag Z
   float max[NUM_DATA_OBJS] = {-9999999.}; //larger for pressure
   float min[NUM_DATA_OBJS] = {9999999.};
   float avg[NUM_DATA_OBJS] = {0.};
   float stddev[NUM_DATA_OBJS] = {0.};

   //be sure to explicitly specify ANY numbers as 123456ULL for uint64_t!!!
   uint64_t read_time;
   read_time = micros64(); //set the initial timestamp of read

   //for max compatibility, an int is passed as input. Error and integrity
   //check is therefore needed. To avoid an exception, we default to 5 reads
```

```
    //if an out-of-bounds value is detected.
    if(samp_times_blocking < 2 || samp_times_blocking > CEIL_SAMPS_SENS) {
      samp_times_blocking = 5; //least number of points for statistical var
    }

    //compute the read delay required to meet the frequency specified (micros):
    uint64_t read_delay = (uint64_t)((float)(1.0/target_samp_rate)*1000000.0);
    //clamp to 100 Hz max samp_rate:
    if(read_delay < CEIL_PERIOD_SENS) read_delay = CEIL_PERIOD_SENS;

    float latestData[NUM_DATA_OBJS] = {0,0,0};
    uint64_t read_delay_tmp;
    sensors_event_t evt;
    for(int i=0;i<samp_times_blocking;i++) {
      Watchdog.reset();
      read_delay_tmp = micros64();
      //read data
      obj.getEvent(&evt);
      //read latest data:
      latestData[0] = evt.magnetic.x;
      latestData[1] = evt.magnetic.y;
      latestData[2] = evt.magnetic.z;
      //populate statistical values:
      for(int j=0;j<NUM_DATA_OBJS;j++) {
      //max?
      if(max[j] < latestData[j] || max[j] == 0) max[j] = latestData[j];
      //min?
      if(min[j] > latestData[j] || min[j] == 0) min[j] = latestData[j];
      //average accumulator:
      avg[j] += latestData[j];
      }
      //loop delay:
      //will never be less than zero (fatal if otherwise)
      if(micros64() - read_delay_tmp < read_delay) {
      //proper way of truncating 64 bit to 32 bit (?)
      //mask number
      delayMicroseconds((micros64()-read_delay_tmp+read_delay) & 0xFFFFFFFF);
      }
    }
    //compute oneshot statistics:
    for(int j=0;j<NUM_DATA_OBJS;j++) {
      //average:
      avg[j] /= (samp_times_blocking);
      //unbiased sample variance:
      stddev[j] = sqrt((1.0/((float)samp_times_blocking-1.0))*pow((abs(latestData[j]-avg[j])),2));
    }

    //save to master struct:
    m.mag_x_max        = max[0];
    m.mag_x_min        = min[0];
    m.mag_x_avg        = avg[0];
    m.mag_x_stddev     = stddev[0];
    m.mag_y_max        = max[1];
    m.mag_y_min        = min[1];
    m.mag_y_avg        = avg[1];
    m.mag_y_stddev     = stddev[1];
    m.mag_z_max        = max[2];
    m.mag_z_min        = min[2];
    m.mag_z_avg        = avg[2];
    m.mag_z_stddev     = stddev[2];


    //save to String (there is a way to do this with a union_t all at once)
    String outString = "";
    outString += "4,"; //Sensor ID for CSV readability
    for(int i=0;i<NUM_DATA_OBJS;i++) {
      outString += max[i] + (String)",";
      outString += min[i] + (String)",";
      outString += avg[i] + (String)",";
      outString += stddev[i] + (String)",";
    }
    m.mag_micros_op   = (uint32_t)(micros64()-read_time);
    outString += (uint32_t)m.mag_micros_op + (String)",";

    SYNCFRAM;
    return outString;
    }

    //////////////////////////////////////////////////////////////////////////////
    //                                 read gas
```

```
                  //
    //                                 PID:  45                  //
    // /////////////////////////////////////////////////////////////////////////////
    /*
   * Read ADS1115 16-bit ADCs, which are responsible for reading SO2 and H2S,
   * both important indicators of volcanic activity (when in the presence of
   * a volcano).
   *
   * The H2S sensor has a nominal 212 nA/ppm sensitivity, and the SO2 sensor
   * is 25 nA/ppm. Humans are hyper-sensitive to H2S, we can detect in the
   * single digit ppb range. So, if this sensor can detect H2S, there is
   * already too much.
   *
   * Late addition: ozone sensor. It is also 1:1 sensitive with nitrogen
   * dioxide.
   *
   * @param samp_times_blocking Read sensor this many times.
   * @param target_samp_rate Target sample rate in Hz. Not guaranteed due to
   * I2C bus. Hard-coded limit at 100 Hz. Theoretically, the ADS1115 can
   * sample at 740 Hz on a 3.4 MHz I2C bus.
   * @param obj ADS sensor instance, if there are multiple.
   * @return Outputs preformatted String with values in the following order:
   * max, min, average, standard deviation, and actual read time in usec. Note
   * that the raw values are outputted to the master struct as well.
   */
    String r_volc(int samp_times_blocking, float target_samp_rate,\
    Adafruit_ADS1115 &obj) {
    m.lastProcessIDActive = 0x45;
    Watchdog.reset();
    //Different from versions above: written to be more general.
    #define NUM_DATA_OBJS 3 //so2, h2s, o3
    #define CEIL_SAMPS_SENS 4096
```

```
  #define CEIL_PERIOD_SENS 10000ULL //microseconds, 100Hz.
  #define CONST1 1.3568E+3F //COUNTS PER PPM
  #define CONST2 1.60E+2F   //same formula. Using TLE2072IP high speed opamp
  #define CONST3 3.84E+2F   //same formula

  //be sure to initialize with arbitrarily large values
  float max[NUM_DATA_OBJS] = {-9999999.}; //larger for pressure
  float min[NUM_DATA_OBJS] = {9999999.};
  float avg[NUM_DATA_OBJS] = {0.};
  float stddev[NUM_DATA_OBJS] = {0.};

  //be sure to explicitly specify ANY numbers as 123456ULL for uint64_t!!!
  uint64_t read_time;
  read_time = micros64(); //set the initial timestamp of read

  //for max compatibility, an int is passed as input. Error and integrity
  //check is therefore needed. To avoid an exception, we default to 5 reads
  //if an out-of-bounds value is detected.
  if(samp_times_blocking < 2 || samp_times_blocking > CEIL_SAMPS_SENS) {
    samp_times_blocking = 5; //least number of points for statistical var
  }

  //compute the read delay required to meet the frequency specified (micros):
  uint64_t read_delay = (uint64_t)((float)(1.0/target_samp_rate)*1000000.0);
  //clamp to 100 Hz max samp_rate:
  if(read_delay < CEIL_PERIOD_SENS) read_delay = CEIL_PERIOD_SENS;

  float latestData[NUM_DATA_OBJS] = {0,0};
  uint64_t read_delay_tmp;
  for(int i=0;i<samp_times_blocking;i++) {
    Watchdog.reset();
    read_delay_tmp = micros64();
    //read data
    //read latest data:
    latestData[0] = (float)obj.readADC_SingleEnded(0); //h2s
    latestData[1] = (float)obj.readADC_SingleEnded(1); //so2
    latestData[2] = (float)obj.readADC_SingleEnded(2); //o3
```

```
      //populate statistical values:
      for(int j=0;j<NUM_DATA_OBJS;j++) {
      //max?
      if(max[j] < latestData[j] || max[j] == 0) max[j] = latestData[j];
      //min?
      if(min[j] > latestData[j] || min[j] == 0) min[j] = latestData[j];
      //average accumulator:
      avg[j] += latestData[j];
      }
      //loop delay:
      //will never be less than zero (fatal if otherwise)
      if(micros64() - read_delay_tmp < read_delay) {
      //proper way of truncating 64 bit to 32 bit
      //mask number
      delayMicroseconds((micros64()-read_delay_tmp+read_delay) & 0xFFFFFFFF);
      }
    }
    //compute oneshot statistics:
    for(int j=0;j<NUM_DATA_OBJS;j++) {
      //average:
      avg[j] /= (samp_times_blocking);
      //unbiased sample variance:
      stddev[j] = sqrt((1.0/((float)samp_times_blocking-1.0))*pow((abs(latestData[j]-avg[j])),2));
    }

    //save to master struct and convert to ppm
    m.gasv_h2s_max     = max[0] / CONST1;
    m.gasv_h2s_min     = min[0] / CONST1;
    m.gasv_h2s_avg     = avg[0] / CONST1;
    m.gasv_h2s_sdv     = stddev[0] / CONST1;
    m.gasv_so2_max     = max[1] / CONST2;
    m.gasv_so2_min     = min[1] / CONST2;
    m.gasv_so2_avg     = avg[1] / CONST2;
    m.gasv_so2_sdv     = stddev[1] / CONST2;
    m.gasv_o3_max      = max[2] / CONST3;
    m.gasv_o3_min      = min[2] / CONST3;
    m.gasv_o3_avg      = avg[2] / CONST3;
    m.gasv_o3_sdv      = avg[2] / CONST3;
    m.gasv_h2s_av_v    = obj.computeVolts(avg[0]);
    m.gasv_so2_av_v    = obj.computeVolts(avg[1]);
    m.gasv_o3_av_v     = obj.computeVolts(avg[2]);


    //save to String (there is a way to do this with a union_t all at once)
    String outString = "";
    outString += "4,"; //Sensor ID for CSV readability
    outString += dtosstrf(m.gasv_h2s_max,5,6) + (String)",";
    outString += dtosstrf(m.gasv_h2s_min,5,6) + (String)",";
    outString += dtosstrf(m.gasv_h2s_avg,5,6) + (String)",";
    outString += dtosstrf(m.gasv_h2s_sdv,5,6) + (String)",";
    outString += dtosstrf(m.gasv_so2_max,5,6) + (String)",";
    outString += dtosstrf(m.gasv_so2_min,5,6) + (String)",";
    outString += dtosstrf(m.gasv_so2_avg,5,6) + (String)",";
    outString += dtosstrf(m.gasv_so2_sdv,5,6) + (String)",";
    outString += dtosstrf(m.gasv_o3_max,5,6)  + (String)",";
    outString += dtosstrf(m.gasv_o3_min,5,6)  + (String)",";
    outString += dtosstrf(m.gasv_o3_avg,5,6)  + (String)",";
    outString += dtosstrf(m.gasv_o3_sdv,5,6)  + (String)",";
    outString += dtosstrf(m.gasv_h2s_av_v,6,6) + (String)",";
    outString += dtosstrf(m.gasv_so2_av_v,6,6) + (String)",";
    outString += dtosstrf(m.gasv_o3_av_v,6,6)  + (String)",";
    m.gasv_micros_op   = (uint32_t)(micros64()-read_time);
    outString += (uint32_t)m.gasv_micros_op + (String)",";

    SYNCFRAM;
    return outString;
    }

    // ////////////////////////////////////////////////////////////////////////////////
    //                                   read AS7341                                  //
    //                                    PID:  46                                    //
    // ////////////////////////////////////////////////////////////////////////////////

    /*!
   * Read AS7341 light sensor ``samp_times_blocking times``.
   * Due to the large number of data points generated by this sensor, no stats
   * will be collected.
   *
   * This function outputs two sets of data: absolute intensity, and a difference
   * signal between absolute intensity and LED-on intensity. The
```

```
larger the
   * difference, the more the sensors are optically obstructed. For rovers and
   * base stations, this is a good metric to quantify how dirty instruments or
   * solar panels are.
   *
   * The extra code here is to auto-set gain.
   *
   * @param inst Light sensor instance, if there are multiple.
   * @param integrationTime Integration time as defined by the datasheet. Each
   * tick is 2.78 microseconds. Leave the default values, as is.
   * @param integrationStepSize Integration step size as defined by the
   * datasheet. Leave values as-is.
   * @param samp_times Accumulation cycles. Defaults to 16. Clamped to 64.
   * @return Outputs preformatted String with values in the following order:
   * max, min, average, standard deviation, and actual read time in usec. Note
   * that the raw values are outputted to the master struct as well.
   */
    String r_as7341(Adafruit_AS7341 &inst, int integrationTime = 50,\
    int integrationStepSize = 512, int samp_times = 16) {
    m.lastProcessIDActive = 0x46;
    Watchdog.reset();
      //Clamped to 64x because of the uint32_t limitation
    #define MAX_SAMP_TIMES 64

    uint64_t read_time;
    read_time = micros64(); //set the initial timestamp of read

    //clamp samp_times to max if smaller or larger for whatever reason:
    if(samp_times < 2 || samp_times > CEIL_SAMPS_SENS) {
      samp_times = 5; //least number of points for statistical var
    }

    //step 1: determine gain. This takes a relatively long time, if we do it
    //every time we loop... we're gonna sit in this subroutine for 10x longer
    //than necessary.

      inst.setASTEP(integrationStepSize);
      inst.setATIME(integrationTime);
    uint16_t mss_raw[13] = {0,0,0,0,0,0,0,0,0,0,0,0,0}; //temporary storage before processing
      uint32_t integral_accumulator[13] = {0,0,0,0,0,0,0,0,0,0,0,0,0};
      #define TGAIN_CEIL   900
      #define TGAIN_FLOOR  512
    int currentGain = 0; //start at the MINIMUM gain, and work itself down from there...
      do {
        inst.setGain((as7341_gain_t)currentGain);
        currentGain++;
        inst.readAllChannels(mss_raw); //perform reading until sensor is NOT saturated
        if(currentGain > 10) break; //we have reached MAXIMUM ADC gain - 1,024x.
      //This simply means it is reading a very dark environment.
      } while(mss_raw[0] < TGAIN_FLOOR); //read UV channel to determine gain

      //normalize gain with 512x ADC counts.
      uint32_t gainRaw = 0;
      switch(currentGain){
        case 0:
          gainRaw = 1024; break;
        case 1:
          gainRaw = 512;  break;
        case 2:
          gainRaw = 256;  break;
        case 3:
          gainRaw = 128;  break;
        case 4:
          gainRaw = 64;   break;
        case 5:
         gainRaw = 32;   break;
        case 6:
          gainRaw = 16;   break;
        case 7:
          gainRaw = 8;    break;
        case 8:
          gainRaw = 4;    break;
        case 9:
          gainRaw = 2;    break;
        case 10:
          gainRaw = 1;    break;
      }

    //step 2: do first pass for ambient measurement:
    for(int i=0; i<samp_times; i++) {
      inst.readAllChannels(mss_raw);
      //fill integrator vars:
      for(int j=0;j<13;j++) {
      //no undefined behavior
      integral_accumulator[j] += (uint32_t)mss_raw[j];
      }
    }
    //step 2.5: decimate by samp_times
    for(int i=0; i<13; i++) {
      integral_accumulator[i] /= samp_times;
    }
    //step 2.75: save to struct:
    m.lss_415nm_abs = integral_accumulator[0];
    m.lss_445nm_abs = integral_accumulator[1];
    m.lss_480nm_abs = integral_accumulator[2];
    m.lss_515nm_abs = integral_accumulator[3];
    m.lss_555nm_abs = integral_accumulator[6];
    m.lss_590nm_abs = integral_accumulator[7];
    m.lss_630nm_abs = integral_accumulator[8];
    m.lss_680nm_abs = integral_accumulator[9];
    m.lss_wband_abs = integral_accumulator[10];
    m.lss_890nm_abs = integral_accumulator[11];

    Watchdog.reset();
    //step 3: do second pass with light on:
    inst.setLEDCurrent(100); //milliamps
    inst.enableLED(true);
    delay(5); //wait to ensure power is stable before reading
    for(int i=0;i<samp_times; i++) {
      inst.readAllChannels(mss_raw);
      //fill integrator vars:
      for(int j=0;j<13;j++) {
      integral_accumulator[j] += (uint32_t)mss_raw[j];
      }
    }
    inst.enableLED(false);
    for(int i=0;i<13;i++) {
      integral_accumulator[i] /= samp_times;
    }
    //save to struct:
```

```
    //current value minus lower value, should yield positive if obstructed
    m.lss_415nm_diff = integral_accumulator[0]  - m.lss_415nm_abs;
    m.lss_445nm_diff = integral_accumulator[1]  - m.lss_445nm_abs;
    m.lss_480nm_diff = integral_accumulator[2]  - m.lss_480nm_abs;
    m.lss_515nm_diff = integral_accumulator[3]  - m.lss_515nm_abs;
    m.lss_555nm_diff = integral_accumulator[6]  - m.lss_555nm_abs;
    m.lss_590nm_diff = integral_accumulator[7]  - m.lss_590nm_abs;
    m.lss_630nm_diff = integral_accumulator[8]  - m.lss_630nm_abs;
    m.lss_680nm_diff = integral_accumulator[9]  - m.lss_680nm_abs;
    m.lss_wband_diff = integral_accumulator[10] - m.lss_wband_abs;
    m.lss_890nm_diff = integral_accumulator[11] - m.lss_890nm_abs;

    //save to String (there is a way to do this with a union_t all at once)
    String outString = "";
    outString += "5,"; //Sensor ID for CSV readability
    outString += m.lss_415nm_abs + (String)",";
    outString += m.lss_445nm_abs + (String)",";
    outString += m.lss_480nm_abs + (String)",";
    outString += m.lss_515nm_abs + (String)",";
    outString += m.lss_555nm_abs + (String)",";
    outString += m.lss_590nm_abs + (String)",";
    outString += m.lss_630nm_abs + (String)",";
    outString += m.lss_680nm_abs + (String)",";
    outString += m.lss_wband_abs + (String)",";
    outString += m.lss_890nm_abs + (String)",";
    outString += m.lss_415nm_diff + (String)",";
    outString += m.lss_445nm_diff + (String)",";
    outString += m.lss_480nm_diff + (String)",";
    outString += m.lss_515nm_diff + (String)",";
    outString += m.lss_555nm_diff + (String)",";
    outString += m.lss_590nm_diff + (String)",";
    outString += m.lss_630nm_diff + (String)",";
    outString += m.lss_680nm_diff + (String)",";
    outString += m.lss_wband_diff + (String)",";
    outString += m.lss_890nm_diff + (String)",";
    m.lss_micros_op   = (uint32_t)(micros64()-read_time);
    outString += (uint32_t)m.lss_micros_op + (String)",";

    //Save to FRAM:
    SYNCFRAM;

    //finally, we are done here.
    return outString;
    }

    // ///////////////////////////////////////////////////////////////////////////
    //                                      read gps                             //
    //                                      PID: 47                              //
    // ///////////////////////////////////////////////////////////////////////////
    /*
   * Read GPS stats, including H, V, and PDOP. Returns a String with data,
   * and populates the global struct. Data includes max, min, average,
   * and standard deviation.
   *
   * Edit 09 Apr 2022 - reentrancy issue if reading multiple times. Issue
   * unknown. Even debugged the library itself and unable to determine issue.
   * Edit 02 May 2022 - reentrancy highly likely due to stuck I2C bus caused by
   * display. Removed display but will only sample GPS once an hour.
   *
   * @param inst GPS object
   * @param samp_times Number of samples for statistical purposes. Default to 5.
   * @return Preformatted string for concatenation. This data goes before
   * all other data in string because it includes date and time.
   *
   */
    uint64_t gpsStats(SFE_UBLOX_GPS &inst, int samp_times) {
    m.lastProcessIDActive = 0x47;
    SYNCFRAM;
    Watchdog.reset();
    #define CEIL_SAMPS_SENS 16 //limited by int32_t
    #define NUM_DATA_POINTS 5
    uint64_t read_time;
    read_time = micros64(); //set the initial timestamp of read

    //clamp samp_times to max if smaller or larger for whatever reason:
```

```
    if(samp_times < 2 || samp_times > CEIL_SAMPS_SENS) {
      samp_times = 5; //least number of points for statistical var
    }
    //Serial.println("GPSD1");
    //get multi-sample data (int32_t):
    int64_t accumulator[NUM_DATA_POINTS] = {0,0,0,0,0};
    int64_t tmp[NUM_DATA_POINTS] = {0,0,0,0,0};
    int64_t max[NUM_DATA_POINTS] = {0,0,0,0,0};
    int64_t min[NUM_DATA_POINTS] = {0,0,0,0,0};

    //loop has a reentrancy issue
    for(int i=0;i<samp_times;i++){
      Watchdog.reset();
      //read this data only once then do something with it:
      tmp[0] = (int64_t)inst.getPositionAccuracy();
      tmp[1] = (int64_t)inst.getGroundSpeed();
      tmp[2] = (int64_t)inst.getAltitudeMSL();
      tmp[3] = (int64_t)inst.getLatitude();
      tmp[4] = (int64_t)inst.getLongitude();
      //averaging accumulator
      for(int j=0;j<NUM_DATA_POINTS;j++) {
      accumulator[j] += tmp[j];
      //max?
      if(max[j] < tmp[j] || max[j] == 0) max[j] = tmp[j];
      //min?
      if(min[j] > tmp[j] || min[j] == 0) min[j] = tmp[j];
      }
      //Serial.println("GPSD2");
      delay(1000);
    }
    //decimate by samp and save directly to master struct (float):
    m.gps_HDOP_avg    = (float)(accumulator[0] / samp_times);
    m.gps_VDOP_avg    = (float)(accumulator[1] / samp_times);
    m.gps_PDOP_avg    = (float)(accumulator[2] / samp_times);
    m.gps_Lat_avg     = (float)((accumulator[3] / samp_times)/1E7);
    m.gps_Long_avg    = (float)((accumulator[4] / samp_times)/1E7);
   // Serial.println("GPSD3");

    //get stddev and save directly to struct (float):
    m.gps_HDOP_stddev = (float)(sqrt((1.0/((float)samp_times-1.0))*
pow(abs(tmp[0]\
    -m.gps_HDOP_avg),2)));
    m.gps_VDOP_stddev = (float)(sqrt((1.0/((float)samp_times-1.0))*
pow(abs(tmp[0]\
    -m.gps_VDOP_avg),2)));
    m.gps_PDOP_stddev = (float)(sqrt((1.0/((float)samp_times-1.0))*
pow(abs(tmp[0]\
    -m.gps_PDOP_avg),2)));
    m.gps_Lat_stddev  = (float)(sqrt((1.0/((float)samp_times-1.0))*
pow(abs(tmp[0]\
    -m.gps_Lat_avg),2)));
    m.gps_Long_stddev = (float)(sqrt((1.0/((float)samp_times-1.0))*
pow(abs(tmp[0]\
    -m.gps_HDOP_avg),2)));

    //Geofence blacklist (my house and other sensitive locations):
    if((m.gps_Lat_max/1E7) > REDACTED && (m.gps_Lat_max/1E7) <
REDACTED) m.gps_Lat_max = 90.0f;
    if((m.gps_Long_max/1E7) > REDACTED && (m.gps_Long_max/1E7) <
REDACTED) m.gps_Long_max = 0.0f;

    //save max, min:
    m.gps_HDOP_max    = tmp[0];
    m.gps_VDOP_max    = tmp[1];
    m.gps_PDOP_max    = tmp[2];
    m.gps_Lat_max     = tmp[3];
    m.gps_Long_max    = tmp[4];
    m.gps_HDOP_min    = min[0];
    m.gps_VDOP_min    = min[1];
    m.gps_PDOP_min    = min[2];
    m.gps_Lat_min     = min[3];
    m.gps_Long_min    = min[4];

    //get time and save to individual variables
    m.gps_year        = inst.getYear();
    m.gps_month       = inst.getMonth();
    m.gps_day         = inst.getDay();
    m.gps_hour        = inst.getHour();
    m.gps_min         = inst.getMinute();
    m.gps_sec         = inst.getSecond();
    m.gps_msec        = inst.getMillisecond();

    //writeout to String:
    String outString = "";
    outString += "OS*03,"; //Sensor ID for CSV readability
    //include leading zeros in date for proper ISO8610 date:
    outString += m.gps_year         + (String)"-";
    if(m.gps_month < 10) outString += (String)"0";
    outString += m.gps_month        + (String)"-";
    if(m.gps_day < 10)   outString += (String)"0";
    outString += m.gps_day          + (String)"T";
    if(m.gps_hour < 10)  outString += (String)"0";
```

```
outString += m.gps_hour        + (String)":";
if(m.gps_min < 10)   outString += (String)"0";
outString += m.gps_min         + (String)":";
if(m.gps_sec < 10)   outString += (String)"0";
outString += m.gps_sec         + (String)".";
if(m.gps_msec < 100) outString += (String)"0";
if(m.gps_msec < 10)  outString += (String)"0";
outString += m.gps_msec        + (String)",";
outString += m.gps_HDOP_max    + (String)",";
outString += m.gps_VDOP_max    + (String)",";
outString += m.gps_PDOP_max    + (String)",";
outString += m.gps_Lat_max     + (String)",";
outString += m.gps_Long_max    + (String)",";
m.gps_micros_op   = (uint32_t)(micros64()-read_time);
outString += (uint32_t)m.gps_micros_op;
outString += "\r\n";
Serial.print(outString);
//send to SPI RAM:
//Arg1: retrieve head of buffer (size of ram-ram remaining)
//Arg2: typecast to byte
//Arg3: length of String, INCLUDES NULL TERMINATOR!
RAM.writeEnable(true);
RAM.write(m.ram1_sz-m.ram1_mem_free,\
  (const uint8_t *)outString.c_str(),outString.length());
m.ram1_mem_free -= (outString.length()); //bookkeeping
RAM.writeEnable(false);
//destruct and clean up:
//(disabled for now, dangerous operation
//memset((void *)&LTA,'\000',sizeof(LTA));
//memset((void *)&buff,'\000',buff.length());
SYNCFRAM;
return micros64();
}

// ////////////////////////////////////////////////////////////////////////////
//                         read SCL3300 coprocessor                         //
//                                 PID:  48                                 //
// ////////////////////////////////////////////////////////////////////////////
//Read external SCL3300:
//Passes no inputs.
```

```
String scl3300_extern(void) {
m.lastProcessIDActive = 0x48;
SYNCFRAM;
Watchdog.reset();
uint64_t read_time = micros64();
//Send specific command:
Watchdog.reset(); //in case we repeat several times
//watchdog will protect anyways if we goto loop too many times
//and include timeout:
uint64_t timeout = micros64();
sclretry:
Serial1.print("<A01,A3A>RDALL.IIR300,");
//wait for data:
delay(1000);
while(!Serial1.available()) {
  strip.setPixelColor(0,0x000000FF);
  strip.show();
  //try command again
  if((micros64() - timeout) > 3000000ULL) goto sclretry;
}
//read from ring buffer directly

uint32_t msprev = m.scl_ms_reported;
m.scl_ms_reported =  Serial1.readStringUntil(’,’).toInt();
m.scl_sps =          Serial1.readStringUntil(’,’).toInt();
m.scl_xt_decim =     Serial1.readStringUntil(’,’).toDouble();
m.scl_yt_decim =     Serial1.readStringUntil(’,’).toDouble();
m.scl_zt_decim =     Serial1.readStringUntil(’,’).toDouble();
m.scl_temp_raw =     Serial1.readStringUntil(’,’).toDouble();
m.scl_tilt16_max_x = Serial1.readStringUntil(’,’).toFloat();
m.scl_tilt16_min_x = Serial1.readStringUntil(’,’).toFloat();
m.scl_tilt16_avg_x = Serial1.readStringUntil(’,’).toFloat();
//real-time flags:
w.sci_scl3300_sdv_x_prev = m.scl_tilt16_sdv_x;
m.scl_tilt16_sdv_x = Serial1.readStringUntil(’,’).toFloat();
m.scl_tilt16_max_y = Serial1.readStringUntil(’,’).toFloat();
m.scl_tilt16_min_y = Serial1.readStringUntil(’,’).toFloat();
m.scl_tilt16_avg_y = Serial1.readStringUntil(’,’).toFloat();
w.sci_scl3300_sdv_y_prev = m.scl_tilt16_sdv_y;
m.scl_tilt16_sdv_y = Serial1.readStringUntil(’,’).toFloat();
m.scl_tilt16_max_z = Serial1.readStringUntil(’,’).toFloat();
m.scl_tilt16_min_z = Serial1.readStringUntil(’,’).toFloat();
m.scl_tilt16_avg_z = Serial1.readStringUntil(’,’).toFloat();
w.sci_scl3300_sdv_z_prev = m.scl_tilt16_sdv_z;
m.scl_tilt16_sdv_z = Serial1.readStringUntil(’,’).toFloat();

```

```
    //seismometer readback:
    m.scl_sdv_delay_ms = m.scl_ms_reported - msprev;
    m.scl_t16_max_x_mp = Serial1.readStringUntil(',').toFloat();
    m.scl_t16_min_x_mp = Serial1.readStringUntil(',').toFloat();
    m.scl_t16_sdv_mpdx = Serial1.readStringUntil(',').toFloat();
    m.scl_t16_max_y_mp = Serial1.readStringUntil(',').toFloat();
    m.scl_t16_min_y_mp = Serial1.readStringUntil(',').toFloat();
    m.scl_t16_sdv_mpdy = Serial1.readStringUntil(',').toFloat();
    m.scl_t16_max_z_mp = Serial1.readStringUntil(',').toFloat();
    m.scl_t16_min_z_mp = Serial1.readStringUntil(',').toFloat();
    m.scl_t16_sdv_mpdz = Serial1.readStringUntil(',').toFloat();
    m.scl_micros_op = (uint32_t)(micros64()-read_time);
    //clear buffer if there's anything left:
    Serial1.readString(); //read to the void >:)

    //flags section:
    if(m.scl_t16_sdv_mpdx > 0.40f) w.SCI_SCL3300_STDDEV_EXCEEDED_CHX = true;
    else if(m.scl_t16_sdv_mpdx < 0.40f) w.SCI_SCL3300_STDDEV_EXCEEDED_CHX = false;
    if(m.scl_t16_sdv_mpdy > 0.40f) w.SCI_SCL3300_STDDEV_EXCEEDED_CHY = true;
    else if(m.scl_t16_sdv_mpdy < 0.40f) w.SCI_SCL3300_STDDEV_EXCEEDED_CHY = false;
    if(m.scl_t16_sdv_mpdz > 0.60f) w.SCI_SCL3300_STDDEV_EXCEEDED_CHZ = true;
    else if(m.scl_t16_sdv_mpdz < 0.60f) w.SCI_SCL3300_STDDEV_EXCEEDED_CHZ = false;
    //now read to String:
    String outString = "";
    outString += "7,";
    outString += m.scl_sps + (String)",";
    outString += m.scl_xt_decim + (String)",";
    outString += m.scl_yt_decim + (String)",";
    outString += m.scl_zt_decim + (String)",";
    outString += m.scl_temp_raw + (String)",";
    outString += m.scl_tilt16_max_x + (String)",";
    outString += m.scl_tilt16_min_x + (String)",";
    outString += m.scl_tilt16_avg_x + (String)",";
    outString += m.scl_tilt16_sdv_x + (String)",";
    outString += m.scl_tilt16_max_y + (String)",";
    outString += m.scl_tilt16_min_y + (String)",";
    outString += m.scl_tilt16_avg_y + (String)",";
    outString += m.scl_tilt16_sdv_y + (String)",";
    outString += m.scl_tilt16_max_z + (String)",";
    outString += m.scl_tilt16_min_z + (String)",";
```

```
outString += m.scl_tilt16_avg_z + (String)",";
outString += m.scl_tilt16_sdv_z + (String)",";
outString += m.scl_sdv_delay_ms + (String)",";
outString += m.scl_t16_max_x_mp + (String)",";
outString += m.scl_t16_min_x_mp + (String)",";
outString += m.scl_t16_sdv_mpdx + (String)",";
outString += m.scl_t16_max_y_mp + (String)",";
outString += m.scl_t16_min_y_mp + (String)",";
outString += m.scl_t16_sdv_mpdy + (String)",";
outString += m.scl_t16_max_z_mp + (String)",";
outString += m.scl_t16_min_z_mp + (String)",";
outString += m.scl_t16_sdv_mpdz + (String)",";
outString += m.scl_micros_op;
//SPTime(F("[SCL3300_read] "));
Serial.println(outString);
strip.setPixelColor(0 ,0);
strip.show();
return outString;

//that's all the sensors, folks.
}

// ////////////////////////////////////////////////////////////////////////////////
//                              initial memory formatting                //
//                                        PID: E2                        //
// ////////////////////////////////////////////////////////////////////////////////
//only execute whenever chip is swapped out!
uint8_t mem_format(void) {
m.lastProcessIDActive = 0xE2;
uint16_t struct_signature = 0x0;
uint32_t ram0sz_tmp = 0x0;
uint32_t ram1sz_tmp = 0x0;
RAM1.readObject(0x0 , struct_signature);
//reading from a struct in FRAM. Knowledge of struct important!
RAM1.readObject(0x0022 , ram0sz_tmp);
RAM1.readObject(0x0006 , ram1sz_tmp);
//check for struct signature, but also check for struct size. If is 0,
//there's a problem and we need to reformat both memory spaces:
if(struct_signature != SM_SIG ||(ram0sz_tmp == 0x00 ||
ram1sz_tmp == 0x00)) {
    //Watchdog.disable();
    //initial start: populate values
    Serial.print(F("Initial␣start.␣FRAMs␣will␣be␣formatted␣due␣to
␣new␣chip␣or␣fatal␣error.\r"));
    //built-in delay on 32
    dbg_blink(BLINK_INIT_SETUP);
    Serial.println(F("Determining␣FRAM␣sizes,␣please␣wait..."));

    //this section of code: credit Adafruit (Kevin Townsend, Ha
Thach)
    //(has some changes to suit my needs)
    uint32_t size_fram_i2c = 0;
    //figure out size of fram and store to struct later:
    //0x2B is what the memory space is "painted" with.
    RAM1.write(0x0,0x2B);
    uint32_t max_addr;
    for (max_addr = 1; max_addr < 0xFFFFFF; max_addr++) {
    if (RAM1.read(max_addr) != 0x2B)
      continue; //def didnt wrap around yet

    //maybe wrapped? try writing the inverse
    if(!RAM1.write(max_addr, (byte)~0x2B)) {
      Serial.print("Failed␣to␣write␣address␣0x");
      Serial.println(max_addr, HEX);
    }
    uint8_t val0 = RAM1.read(0);
    //re-write the old value
    if (!RAM1.write(max_addr, 0x2B)) {
      Serial.print("Failed␣to␣re-write␣address␣0x");
      Serial.println(max_addr, HEX);
    }
    //check if addr 0 was changed
    if (val0 == (byte)~0x2B) {
      Serial.println("Found␣max␣address");
      break;
    }
    }
    Serial.print(F("FRAM␣A:"));
    Serial.print(max_addr);
    Serial.print(F("B"));
    //Store known size now:
    m.ram0_sz = max_addr;

    //format and read back all mem cells. Stop if any write error
 appears.
```

```
//I2C RAM:
Serial.print(F("FRAM␣A␣(I2C␣memcells)␣formatting␣and␣checking,␣please␣wait..."));
#define RAM_WRITE_ERASE 0x00
for(int i=0;i<m.ram0_sz;i++) {
int retry_cnt = 0;
retry_ram_format:
RAM1.write(i,RAM_WRITE_ERASE);
//readback:
if(RAM1.read(i) != RAM_WRITE_ERASE) {
  //try to write four more times to ensure it wasn't bus noise:
  retry_cnt++;
  if(retry_cnt > 4) {
  Serial.print(F("RAM␣failure␣at␣0x"));
  Serial.print(i, HEX);
  //Do not use this line for flight software. Sub for a bytemask.
  dbg_blink(BLINK_CORE_ERR_UNRECOVERABLE);
  }
  goto retry_ram_format;
}
}
Serial.print(F("done.\r␣FRAM␣A␣formatting␣and␣integrity␣check␣complete.\r"));
//write out struct signature so we don't have to do this again:
struct_signature = SM_SIG;
RAM1.writeObject(0x000000, struct_signature);
m.data_counter_total = 0L;
m.data_counter_lora  = 0L;
w.RES_FRAM_FORMATTED = true;
SYNCFRAM;

//format and check RAM B:
//different method of checking larger memsize:
Serial.print(F("Getting␣size␣of␣FRAM␣B..."));

max_addr = 0;
while (readBack(max_addr, max_addr) == max_addr) {
max_addr += 256;
}

Serial.print(max_addr);
Serial.println(F("B"));
m.ram1_sz = max_addr;
```

```
      //store to FRAM struct
      SYNCFRAM;

      Serial.print(F("Formatting␣FRAM␣B,␣please␣wait..."));
      //format and integrity check:
      RAM.writeEnable(true);
      for(int i=0;i<m.ram1_sz;i++) {
      int retry_cnt = 0;
      retry_ramb_format:
      RAM.write8(i,RAM_WRITE_ERASE);
      if(RAM.read8(i) != RAM_WRITE_ERASE) {
        retry_cnt++;
        if(retry_cnt > 4) {
        Serial.print(F("RAM␣failure␣at␣0x"));
        Serial.println(i, HEX);
        Serial.println(F("For␣flight␣software,␣do␣NOT␣suspend.␣The␣mission␣must␣continue␣under␣any␣circumstance."));
        dbg_blink(BLINK_CORE_ERR_UNRECOVERABLE);
        //end of execution, cannot continue with corrupt RAM
        }
        goto retry_ramb_format;
      }
      }
      RAM.writeEnable(false);
      m.ram1_mem_free = m.ram1_sz;
      Serial.println(F("complete."));
      Serial.println(F("FRAM␣init␣complete.␣Restarting..."));
      delay(2000);
      //record and sync to struct:
      Serial.print(F("Synchronizing␣struct..."));
      m.lastProcessIDActive = 0xE2;
      m.lastRecordedExecState = 0x08; //formatted
      SYNCFRAM;
      //restart to ensure uncorrupted memory space.
      Serial.println(F("done."));
      delay(1);
      Serial.println(F("MACH␣Reboot"));
      restart();
      for(;;);
      return 0;
    }
    else {
      m.lastProcessIDActive = 0xE2;
      m.lastRecordedExecState = 0x01; //data already exists
      //SYNCFRAM;

      //debug: hexdump FRAM object after every restart. Helps with debug of a
      //troublesome piece of code. Dump to WiFi as well when implemented.
      //malloc() to ensure this is returned to the heap later.
      //sanity check: the implementation of malloc works. First real-world
      //implementation of it.
      char *dump = (char *)malloc(sizeof(m));
      RAM1.read(0x000002,(uint8_t *)dump,sizeof(m));
      Serial.print(F("FRAM␣hexdump:\r\n"));
      #define BYTESPACING 16
      int j=0;
      for(int i=0;i<sizeof(m);i++) {
      if(dump[i] < (byte)0x10) Serial.print("0");
      Serial.print(dump[i],HEX);
      Serial.print(F("␣"));
      j++;
      if(j==BYTESPACING) {
        Serial.print("\r\n");
        j=0;
      }
      }
      free(dump);
      return 0;
    }
    }

    // ////////////////////////////////////////////////////////////////////////////
    //                             start routine                             //
    //                                PID: E1                                //
    // ////////////////////////////////////////////////////////////////////////////
    //returns 0 for success, > 0 for failure code. Do not execute more than once.
    uint8_t start(void) {
    digitalWrite(RFM69_RST, HIGH);
    delay(50);
    digitalWrite(RFM69_RST, LOW);

    rf69.init();
```

```
    rf69.setFrequency(922.5);
    rf69.setModemConfig(RH_RF69::ModemConfigChoice::GFSK_Rb2_4Fd4_8
);
    //2.4 ksps, 4.8kHz bandwidth
    SPLTime(F("GFSK␣command␣uplink␣started␣with␣2.4kSPS␣data␣rate,␣
4.8kHz␣bandwidth."));
    //serials
    delay(9950); //wait for user to connect to serial port
    //serial speed doesn't matter when
    Serial.begin(1000000);
    Watchdog.enable(16000);
    SPLTime(F("[init]␣Early␣watchdog␣started"));
    //while(!Serial);
    SPLTime(F("Init"));
    SPLTime(F("while(!Serial)␣user␣hold␣DISABLED"));
    SPLTime(F("Init␣Serial1"));
    Serial1.begin(9600); //serial to inclinometer
    SPLTime(F("Wavefront␣v6.3"));
    SPLTime(F("Stanley␣M.␣Krzesniak"));
    SPLTime(F("Copyright␣(C)␣2020-2022␣Stanley␣M.␣Krzesniak"));
    SPLTime(F("All␣rights␣reserved."));

    //Watchdog.enable(16000);
    delay(500);

    SPTime(F("Resetting␣I2C␣lines␣just␣in␣case..."));
    //LCD is a POS. Has a stuck bus issue and I don't have an
option to power it
    //off. Toggle I2C a few dozen times to free it:
    //Clock data line as well
    //27 April: Permanently removed LCD. Bus gets stuck.
    for(int i=0; i<64; i++) {
      digitalWrite(27, LOW);
      digitalWrite(26, LOW);
      delayMicroseconds(10);
      digitalWrite(27, HIGH);
      digitalWrite(26, HIGH);
      delayMicroseconds(10);
    }
    Serial.println(F("done."));


    strip.begin();
    strip.setBrightness(100);
    strip.setPixelColor(0,0x00FFFFFF);
    strip.show();
    SPLTime(F("LED␣begin"));
    SPTime(F("Waiting␣for␣RAM..."));

    if(!RAM.begin()) {
      Serial.println(F("FATAL:␣SPI␣RAM␣failed␣to␣start."));
      dbg_blink(255);
    } else SPLTime(F("SPI␣FRAM␣ok.")); //2Mbit FRAM - SPI
    if(!RAM1.begin()) {
      Serial.println(F("FATAL:␣I2C␣RAM␣failed␣to␣start."));
    } else SPLTime(F("I2C␣FRAM␣ok.")); //256Kbit FRAM - I2C.
    //Shadow copy of RAM

    //COPY FRAM TO RAM IMMEDIATELY:
    DLFRAM;
    SPLTime(F("Copied␣FRAM␣A␣to␣SRAM"));
    SPTime(F("Free␣FRAM:␣"));
    Serial.print(m.ram1_mem_free);
    Serial.println(F("␣bytes"));
    SPLTime(F("If␣this␣number␣is␣zero,␣bug␣present␣in␣mem␣mgmt."));
    w.RES_FIRST_LOOP = true;
    SPLTime(F("[init]␣First␣loop␣flag␣set"));
    sched_blks[TASK_LOOP_SCANLORA]  = 1000000ULL;
    sched_blks[TASK_LOOP_MACHSTATS] = 300000000ULL;
    sched_blks[TASK_LOOP_MAINSENSE] = 120000000ULL;
    sched_blks[TASK_LOOP_WIFIDELEG] = 30000000ULL;
    w.RES_FRAM_FULL = false;
    SYNCFRAM;

    if(w.RES_FRAM_INTEG_CHK_FAILED == true) {
      SPLTime(F("WARNING:␣PREVIOUS␣BOOT\
␣␣␣␣␣␣␣FRAM␣CHECK␣FAILED!␣MEMORY␣MAY␣BE␣CORRUPT!"));
      w.RES_FRAM_INTEG_MSG_FIRED = false;
      w.RES_FRAM_INTEG_CHK_FAILED = false;
      //reset for now, do something with this later like starting the formatter
    }

    //return the last time this unit was active:
    SPTime(F("Last␣recorded␣time␣in␣struct:␣"));
    Serial.print(struct_iso8610(m,false));
    Serial.println(F("Z"));
    Watchdog.reset();

    if(!gps.begin()) {
      SPLTime(F("FATAL:␣GPS␣failed␣to␣start."));
    } else Serial.println(F("GPS␣start␣ok."));
```

```
  gps.setI2COutput(COM_TYPE_UBX); //set as ublox frame output,
  //no NMEA "noise"
  //DO NOT write to Flash. This instruction is already in THIS processor!
  //GPS.saveConfiguration();
  uint32_t tm = millis();
  SPTime(F("Waiting␣for␣valid␣time␣from␣GPS..."));
  while(!gps.getTimeValid() && (millis() <= tm + GPS_INIT_TIMEOUT)) {
    dbg_blink(BLINK_LINK_ACQUISITION);
  }
  Serial.println(F("done."));

  if(!rtc.begin()) {
    SPLTime(F("FATAL:␣DS3231␣failed␣to␣start."));
    strip.setPixelColor(0,0x00FF0000);
    strip.setBrightness(16);
    if(F_CPU == 48000000L);
    strip.show();
    dbg_blink(BLINK_CORE_ERR_UNRECOVERABLE);
    //we need the RTC in order to function...
  } else {
    SPLTime(F("DS3231␣started."));
  }
  m.lastProcessIDActive = 0xE1;
  //bug: executed before memory management routine.
  //SYNCFRAM;

  rf95.init();
  rf95.setFrequency(915.0);
  rf95.setModemConfig((RH_RF95::ModemConfigChoice)0x60); //61.25 kHz with
  //4:5 interleave
  SPLTime(F("LoRa␣node␣downlink␣started␣with␣19.2kSPS␣data␣rate,␣61.25kHz␣bandwidth."));

  SPLTime(F("Init␣1␣complete."));
  return 0;
  }

  // //////////////////////////////////////////////////////////////////////////////
  //                               start routine 2                               //
  //                                   PID: E3
```

```
              //
  //
//////////////////////////////////////////////////////////////////////////////

  //science instruments enable and check. Do not halt if
instrument is fouled,
  //just skip with zeroes in string return.

  uint8_t startsci(void) {
  m.lastProcessIDActive = 0xE3;
  SYNCFRAM;

  //start INA219:
  ina.begin();
  ina.setCalibration_16V_400mA();

  //start ADT7410:
  pts.begin();
  Serial.println();
  Serial.println(F("[SCI_S]␣PTS..."));

  //start CO2 sensor:
  co2.begin();
  Serial.println(F("[SCI_S]␣CO2..."));

  //start and configure BME680:
  //do NOT filter or oversample! my algos will do that already.
  //increases read speed.
  bme.begin();
  bme.setTemperatureOversampling(BME680_OS_1X);
  bme.setHumidityOversampling(BME680_OS_1X);
  bme.setPressureOversampling(BME680_OS_1X);
  bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
  bme.setGasHeater(320, 150); // 320*C for 150 ms
  Serial.println(F("[SCI_S]␣BME..."));

  //start and configure ADS1115:
  ads.begin(0x4A);
  ads.setGain(GAIN_SIXTEEN); //256 mV full scale
  Serial.println(F("[SCI_S]␣ADS..."));

  //set up ccs811 gas sensor
  //assumed that it is properly wired, etc.
  ccs.begin();
  ccs.setDriveMode(CCS811_DRIVE_MODE_250MS); //continuous
  while(!ccs.available()); //wait for sensor ready
  Serial.println(F("[SCI_S]␣CCS..."));

  //set up magnetometer:
  mag.begin_I2C();
  mag.setPerformanceMode(LIS3MDL_ULTRAHIGHMODE);
  mag.setOperationMode(LIS3MDL_CONTINUOUSMODE);
  mag.setDataRate(LIS3MDL_DATARATE_155_HZ);
  mag.setRange(LIS3MDL_RANGE_4_GAUSS);
  Serial.println(F("[SCI_S]␣MAG..."));

  //start light sensor and init to default values
  as.begin();
  as.setATIME(100);
  as.setASTEP(999);
  as.setGain(AS7341_GAIN_256X);
  Serial.println(F("[SCI_S]␣AS7..."));

  //arm and enable watchdog
  SPLTime(F("[watchdog]␣Enabling␣watchdog,␣16s␣timeout"));
  Watchdog.enable(16000);

  //GPS time sync if lock available:
  if(gps.getTimeValid()) {
    updatertc(gps,rtc);
    dbg_blink(BLINK_GPS_T_SYNC);
    SPTime(F("GPS␣Time␣valid.␣Current␣time:␣"));
    Serial.print(rtc_iso8610(rtc,false,false));
    Serial.println(F("␣Z"));
  } else {
    SPTime(F("GPS␣Time␣INVALID.␣Current␣RTC␣DS3231␣time:␣"));
    Serial.print(rtc_iso8610(rtc, false,false));
    Serial.println(F("␣Z"));
  }

  SPLTime(F("Science␣init␣complete."));

  return 0;
  }

  //
  /////////////////////////////////////////////////////////////////////////////

  //                  WiFi start and overall integrity check
             //
  //                             No  associated PID
             //
```

```
// ////////////////////////////////////////////////////////////////////////////////
void startwifi(void) {
SPLTime(F("Starting␣WiFi..."));
//WiFi.setPins(SPIWIFI_SS, SPIWIFI_ACK, ESP32_RESETN, ESP32_GPIO0, &SPI);
WiFi.setPins(ATWIFI_SS,ATWIFI_ACK,ATWIFI_RST);
wifi_delegate(WiFi.status());
status = WiFi.status();
if(WiFi.status() == WL_CONNECTED) {
  SPTime(F("Sending␣beacons␣to␣Discord..."));
  //sendDiscordCSV(DISCORD_ALERTS_URL,"Boot OK","Wavefront 6.4",INGENUITY_pfp,false);
  //parse compile time into machine-readable date and time to determine if
  //we either crashed or we just uploaded new code:
  DateTime compileTime(F(__DATE__), F(__TIME__));

  String se = "Boot␣OK.␣Firmware␣compiled␣";
  se += rtcd_iso8610(compileTime,false,true);
  se += "␣last␣active␣PID␣(dec):␣";
  se += m.lastProcessIDActive; se += ",";
  se += "␣FRAM␣free:␣";
  se += m.ram1_mem_free;
  se += "␣Bytes.␣Connected␣to␣";
  se += String(WiFi.SSID());
  se += ",␣RSSI␣";
  se += String(WiFi.RSSI());
  se += "␣dBm.";
  sendDiscordCSV(DISCORD_ALERTS_URL,se,"Wavefront␣6.4", INGENUITY_pfp,false);
  //test if restart was due to user or hang:
  //predicated on whether RTC successfully started
  DateTime now = rtc.now();
  DateTime mdt(m.lastYear,m.lastMonth,m.lastDay,
       m.lastHour,m.lastMinute,m.lastSecond);
  se = "";
  int64_t time_compare = (int64_t)compileTime.unixtime()+25200LL-(int64_t)now.unixtime();
  if(time_compare > 900LL) {
  se = "[startwifi]␣*Processor␣crashed␣or␣WDT␣timed␣out.*";
  SPLTime(F("Processor␣crashed␣message"));
  sendDiscordCSV(DISCORD_ALERTS_URL,se,"Wavefront␣6.4", INGENUITY_pfp,false);
  } else {
        se = "[startwifi]␣*New␣firmware␣loaded.";
        w.RES_NEW_FIRMWARE = true;
        sendDiscordCSV(DISCORD_ALERTS_URL,se,"Wavefront␣6.4",INGENUITY_pfp,false);
        }
        if(w.RES_FRAM_FORMATTED == true && w.RES_NEW_FIRMWARE == true) {
        sendDiscordCSV(DISCORD_ALERTS_URL,"[mem_init]␣NVRAM␣variable␣space␣updated.␣FRAM␣A␣and␣FRAM␣B␣formatted.","Wavefront␣6.4",INGENUITY_pfp,false);
        //clear bit:
        w.RES_FRAM_FORMATTED = false;
        w.RES_NEW_FIRMWARE = false;
        }
        else if(w.RES_FRAM_FORMATTED == true && w.RES_NEW_FIRMWARE == false) {
        String str = "[mem_init]␣**ATTENTION**:␣NVRAM␣REFORMATTED␣AUTOMATICALLY␣AFTER␣REBOOT␣OR␣CRASH.␣INTERNAL␣I/O␣ERROR␣LIKELY.";
        sendDiscordCSV(DISCORD_ALERTS_URL,str,"Wavefront␣6.4",INGENUITY_pfp,false);
        //clear bit:
        w.RES_FRAM_FORMATTED = false;
        }
        else {
        //clear it if for whatever reason it didn't clear:
        w.RES_FRAM_FORMATTED = false;
        w.RES_NEW_FIRMWARE = false;
        }
        //Hexdump:
        sendDiscordCSV(DISCORD_PASTEBIN_URL,framhexdump(RAM1,true),"Wavefront␣6.4",INGENUITY_pfp,true);
    }
    }
    // ////////////////////////////////////////////////////////////////////////////
    //                              main setup prototype                  //
    //                               No  associated PID                   //
    // ////////////////////////////////////////////////////////////////////////////
    void setup() {
    start(); //start core peripherals, PID E1
    mem_format(); //check if formatting is needed (almost never),
    PID E2
    startsci(); //start science instruments
    startwifi(); //start wifi chip

    Watchdog.reset();
    strip.setPixelColor(0,0);
    strip.show();
    Serial.println(PriUint64<DEC>(micros64()));
    Serial.println(F("Setup␣100%␣complete."));
    }

    // ////////////////////////////////////////////////////////////////////////////
    //                                      main science loop                                    //
    //                                            PID: 0E                                        //
    // ////////////////////////////////////////////////////////////////////////////
    #define SER_STR_ADDR "0S*01,"
    //High level function:
    uint64_t loop_MainSense(void) {
    //identify packet type to make it easier to process serialized FRAM buffer
    m.lastProcessIDActive = 0x0E;
    SYNCFRAM;
    String serialSend = SER_STR_ADDR;
    //get time from RTC:
    serialSend += rtc_iso8610(rtc, true,false);
    uint32_t currentMicros[2] = {0,0};
    currentMicros[0] = micros64() >> 32;
    currentMicros[1] = (uint32_t)micros64();
    serialSend += (currentMicros[0]) + (String)",";
    serialSend += (currentMicros[1]) + (String)",";
    serialSend += r_ina219(30,60,ina);
    serialSend += r_scd30(6,co2);
    serialSend += r_adt7410(30,60,pts);
    serialSend += r_bme680(8,4,bme);
    serialSend += r_ccs811(10,4,ccs);
    serialSend += r_mag(128,60,mag);
    serialSend += r_volc(128,60,ads);
    serialSend += r_as7341(as,50,512,8);
    serialSend += scl3300_extern();
    serialSend += "\r\n";
```

```
//send to RAM (serialize):
//(make function later)
m.lastProcessIDActive = 0x0E;
SYNCFRAM;
RAM.writeEnable(true);
RAM.write(m.ram1_sz-m.ram1_mem_free,\
  (const uint8_t *)serialSend.c_str(),serialSend.length());
m.ram1_mem_free -= serialSend.length(); //bookkeeping
m.data_counter_total += serialSend.length();
SYNCFRAM;
RAM.writeEnable(false);
//Send to raw data channel:
if(WiFi.status() == WL_CONNECTED) sendDiscordCSV(DISCORD_RAWDATA_URL,serialSend,"Wavefront 6.4",INGENUITY_pfp,true);
Serial.print(serialSend);
return micros64();
}


// /////////////////////////////////////////////////////////////////////////////
//                        Deserializer to Serial port                                   //
//                                 PID: C3                                              //
// /////////////////////////////////////////////////////////////////////////////
//26 April. HOLD THE DESERIALIZER OFF UNTIL I FIGURE EVERYTHING ELSE OUT.
//REDACTED
//REALLY difficult.
//28 April. Everything else figured out - clear to go.
//Note 9 April 2023 - never finished this deserializer due to maintenance error.
/*
uint64_t deser_discord_serial(void) {
m.lastProcessIDActive = 0xC3;
SYNCFRAM;
//initially just serial port - need a flag to tell it to bypass discord if
//the WiFi physical layer isn't connected (lost connection, etc)

```

```
    //strategy: Packet reassembly like IP layer. Read in blocks
    //Find OS* or LN*:
    char *last_sid;
    char sid[3] = "";
    int last_did = -1;
    char block_buf[256] = {'\000'}; //block buffer. Read in 256-byte sectors
    bool prd = false; //packet currently being read?

    //while we are reading data:
    for(uint32_t sz=0;sz<m.ram1_sz-m.ram1_mem_free;) {
      //read sector:
      RAM.read(sz,(uint8_t *)block_buf,sizeof(block_buf)-1);
      //find the first valid beginning of a frame:
      //("*" only occurs in the SID.)
      last_sid = strstr((const char *)block_buf,"*");
      if(last_sid != NULL) {
      prd = true; //start of frame
      //if we find the start of a frame, identify the SID:
      memcpy((void *)sid,(const void *)last_sid[-2],2);
      }
      //start copying to serial port:
      for(int j=0;j<sizeof(block_buf);j++) {

      }

      if(String(block_buf).substring(0,3)) {
      prd = true; //start of frame, read packet

      }
      sz += sizeof(block_buf);

    }

    return micros64();
    }
    */


    // ////////////////////////////////////////////////////////////////////////////
    //                            SPI FRAM dump to Serial                    //
    //                                    PID: C4                            //
```

```
// /////////////////////////////////////////////////////////////////////////////
uint64_t dump_fram_serial(bool erase) {
m.lastProcessIDActive = 0xC4;
SYNCFRAM;

char block_buf[256] = {'\000'}; //block buffer. Read in 256-byte sectors
//no third arg, increment manually
//READ ONE BYTE AT A TIME FOR INTEGRITY:
for(uint32_t sz=0;sz<(m.ram1_sz-m.ram1_mem_free);) {
  RAM.read(sz,(uint8_t *)block_buf,sizeof(block_buf)-1);
  Serial.print(String(block_buf));
  sz += sizeof(block_buf);
}
Serial.println();
//Erase if commanded to do so:
if(erase == true) {
  RAM.writeEnable(true);
  //wipe:
  for(uint32_t i=0;i<m.ram1_sz;i++) {
  RAM.write8(i,'\000');
  }
  RAM.writeEnable(false);
}
m.ram1_mem_free = m.ram1_sz; //set to start
SYNCFRAM;
strip.setPixelColor(0,0x00FFFFFF);
strip.show();
delay(500);
strip.setPixelColor(0,0);
strip.show();
SPTime(F("FRAM␣dumped␣to␣Serial.␣FRAM␣B␣"));
if(erase == true) Serial.println(F("erased."));
else Serial.println(F("NOT␣erased."));
return micros64();
}

// /////////////////////////////////////////////////////////////////////////////
//                                 mach  stats                                //
//                                   PID: 01                                  //
```

```
//
/////////////////////////////////////////////////////////////////////////////

uint64_t loop_machstats(void) {
Watchdog.reset();
m.lastProcessIDActive = 0x02;
SYNCFRAM;
String tmp = "OS*02,";
tmp += rtc_iso8610(rtc, true,false);
tmp += m.ramfree; tmp += ",";
tmp += m.data_counter_total; tmp += ",";
tmp += m.data_counter_lora; tmp += ",";
tmp += m.ram1_mem_free; tmp += "\r\n";
SYNCFRAM;
RAM.writeEnable(true);
RAM.write(m.ram1_sz-m.ram1_mem_free,\
  (const uint8_t *)tmp.c_str(),tmp.length());
m.ram1_mem_free -= tmp.length(); //bookkeeping
m.data_counter_total += tmp.length();
DateTime now = rtc.now();
m.lastYear = now.year();
m.lastMonth = now.month();
m.lastDay = now.day();
m.lastHour = now.hour();
m.lastMinute = now.minute();
m.lastSecond = now.second();
SYNCFRAM;
RAM.writeEnable(false);
Serial.print(tmp);
return micros64();
}

//
/////////////////////////////////////////////////////////////////////////////

//                                 GFSK modem
              //
//                                   PID: 07
              //
//
/////////////////////////////////////////////////////////////////////////////

//Handles basic commands on GFSK radio.
String gfsk_handler(void) {
String str = "";
uint8_t bbuf[61];
```

```
   if(rf69.available()) {
     rf69.recv((uint8_t *)&bbuf,(uint8_t *)61);
     str = String(*bbuf);
   }
   return str;
   }

   //
////////////////////////////////////////////////////////////////////////////

   //              command uplink handler - serial and GFSK modem
              //
   //                                   PID: 07
              //
   //
////////////////////////////////////////////////////////////////////////////

   //for configuration, etc. Discord bot will have a different
command handler.
   uint64_t sercmd_handler(String input = "\t") {
   Watchdog.reset();
   //list of top-level commands:
   //read already protected by internal timeout
   bool CMDHDL_GFSK = false;
   String r = "";
   //If our input is from Serial:
   //aka input is default:
   if(input == "\t") {
     r = Serial.readStringUntil('\r');
     Serial.print("[CMD]␣");
     Serial.println(r);
     //dump everything else after carriage return:
     while(Serial.available()) {
     char dump = Serial.read();
     }
   }
   //if NOT default (from external command seq, like GFSK)
   else {
     r = input;
     //conditional to transmit via GFSK
     CMDHDL_GFSK = true;
   }

   //System management:
   if(r.substring(0,7) == "restart") {
     SPTime(F("[CMD]␣This␣will␣restart␣the␣system.␣Are␣you␣sure?␣"
));
      #define SP_TIMEOUT 30000000
      uint64_t timeout = micros64();
      //Disable watchdog, we are in a loop larger than the timeout:
      Watchdog.disable();
      while(Serial.available() == 0 && (micros64() < timeout +\
     (uint64_t)SP_TIMEOUT));
      //if serial is still not available, leave loop
      if(Serial.available() == 0) {
      SPLTime(F("\r\n[CMD]␣Timed␣out,␣leaving."));
      } else {
      char c = Serial.read();
      if(c == 'Y') {
        Serial.println(c);
        SPLTime(F("Restarting..."));
        SPLTime(F("MACH␣Reboot"));
        rf69.send((const uint8_t *)"REBOOT",7);
        restart();
      }
      else if(c == 'N') {
        Serial.println(c);
        SPLTime(F("[CMD]␣Aborted."));
        //flush buffer
        while(Serial.available()) Serial.read();
      }
      }
    }
    else if(r.substring(0,14) == "nuke47f10a8813") {
      sendDiscordCSV(DISCORD_ALERTS_URL,"[kern]␣Nuke␣command␣
received.␣Restarting␣and␣reformatting␣all␣NVRAM.","Wavefront␣6.4",
INGENUITY_pfp,false);
      RAM1.write(0x0000,0x0); //Invalidate RAM.
      restart();
    }
    //FRAM management:
    else if(r.substring(0,12) == "fram-manage␣") {
      r.remove(0,12);
      if(r.substring(0,11) == "dumpserial␣") {
      r.remove(0,11);
      if(r.substring(0,5) == "erase") {
        r.remove(0,5);
        rf69.send((const uint8_t *)"ok\r\n",5);
        dump_fram_serial(true);
      }
      else if(r.substring(0,8) == "no-erase") {
        r.remove(0,8);
```

```
          rf69.send((const uint8_t *)"ok\r\n",5);
          dump_fram_serial(false);

        }
        else if(r.substring(0,10) == "dumpstruct") {
          r.remove(0,10);
          char *dump = (char *)malloc(sizeof(m));
          RAM1.read(0x000002,(uint8_t *)dump,sizeof(m));
          Serial.print(F("FRAM␣hexdump:\r\n"));
          #define BYTESPACING 16
          int j=0;
          for(int i=0;i<sizeof(m);i++) {
          if(dump[i] < (byte)0x10) Serial.print("0");
          Serial.print(dump[i],HEX);
          Serial.print(F("␣"));
          j++;
          if(j==BYTESPACING) {
            Serial.print("\r\n");
            j=0;
          }
          }
          free(dump);
          Serial.println("");
        }
        }
        else if(r.substring(0,7) == "ramfree") {
        r.remove(0,7);
        SPTime(F("[CMD]␣FRAM␣free:␣"));
        Serial.print(m.ram1_mem_free);
        Serial.println(F("␣bytes"));
        rf69.send((uint8_t *)String(m.ram1_mem_free).c_str(),7);
        }
        SPLTime(F("[CMD]␣ok"));
      }
      //Discord commands:
      else if(r.substring(0,8) == "discord␣") {
        r.remove(0,8);
        if(r.substring(0,9) == "dumpfram␣") {
        r.remove(0,9);
        if(r.substring(0,6) == "erase␣") {
          r.remove(0,6);
          int ff = r.toInt();
          if(status == WL_CONNECTED)
dumpframdiscord(DISCORD_RAWDATA_URL\
          ,INGENUITY_pfp,true,ff);
          else SPLTime(F("[discord]␣Error:␣no␣WiFi␣connection␣
  available."));
          SPLTime(F("[discord]␣ok"));
        }
        else if(r.substring(0,9) == "no-erase␣") {
          r.remove(0,9);
          int ff = r.toInt();
          if(status == WL_CONNECTED)
  dumpframdiscord(DISCORD_RAWDATA_URL\
          ,INGENUITY_pfp,false,ff);
          else SPLTime(F("[discord]␣Error:␣no␣WiFi␣connection␣
  available."));
          SPLTime(F("[discord]␣ok"));
        }
        }
        else if(r.substring(0,6) == "atest␣") {
        r.remove(0,6);
        int fd = r.toInt();
        sendDiscordFileRNG(DISCORD_RAWDATA_URL,"
  testaaa1234567890jsonstri\
  ␣␣␣␣␣␣␣ngify","Wavefront",INGENUITY_pfp,false,fd);
        SPLTime(F("[discord][ATEST]␣ok"));
        }
      }
      //HTTP Netcat send (no encryption):
      else if(r.substring(0,5) == "http␣") {
        r.remove(0,5);
        if(r.substring(0,9) == "dumpfram␣") {
        r.remove(0,9);
        if(r.substring(0,5) == "erase") {
          r.remove(0,5);
          if(status == WL_CONNECTED) {dumpframhttp(true);
          sendDiscordCSV(DISCORD_ALERTS_URL,"**DEBUG**␣FRAM␣uploaded␣
  to\
  ␣␣␣␣␣␣␣␣␣REDACTED,␣erased.","Wavefront␣6.4",INGENUITY_pfp,false);}
          else SPLTime(F("[http]␣Error:␣no␣WiFi␣connection␣available.
  "));
          SPLTime(F("[http]␣ok"));
        }
        else if(r.substring(0,8) == "no-erase") {
          r.remove(0,8);
          if(status == WL_CONNECTED){ dumpframhttp(false);
          sendDiscordCSV(DISCORD_ALERTS_URL,"**DEBUG**␣FRAM␣uploaded␣
  to\
  ␣␣␣␣␣␣␣␣␣REDACTED,␣not␣erased.","Wavefront␣6.4",INGENUITY_pfp,false)
  ;}
          else SPLTime(F("[discord]␣Error:␣no␣WiFi␣connection␣
  available."));
          SPLTime(F("[http]␣ok"));
        }
        }
      }
      else if(r.substring(0,10) == "dumpdelays") {
        r.remove(0,10);
        SPLTime(F("[cmd]␣Reading␣RTOS␣delay␣table:"));
        for(int i=0;i<NUMTASKS;i++) {
        SPTime(F("[cmd]␣Task␣"));
        Serial.print(i);
        Serial.print(F(":␣"));
        Serial.print(PriUint64<DEC>(sched_blks[i]));
        Serial.println(F("␣microsec"));
        }
      }
      else {
        Serial.println(("[CMD]␣Invalid␣command␣or␣line␣endings␣not␣
  set␣to␣\\r."));
      }
      return micros64();
      }

      //Note: can only detect single-bit errors.
      //Manchester (???) or Diffie-Hellman can detect and CORRECT
  multi-bit errors.
      uint64_t fram_integrity_check(void) {
      //check master:
      uint32_t cksum1 = CRC32::calculate((uint8_t *)&m,sizeof(m)-1);
      //check working flags:
      uint32_t cksum2 = CRC32::calculate((uint8_t *)&w,sizeof(w)-1);
      c.crc_master = cksum1;
      c.crc_workng = cksum2;
      //upload FRAM:
      SYNCFRAM;
      //DOWNLOAD FRAM:
      DLFRAM;
      //if ANYTHING changed, this should be detected:
      //check master:
      cksum1 = CRC32::calculate((uint8_t *)&m,sizeof(m)-1);
      //check working flags:
      cksum2 = CRC32::calculate((uint8_t *)&w,sizeof(w)-1);
      //compare:
      if(cksum1 != c.crc_master) w.RES_FRAM_INTEG_CHK_FAILED = true;
      if(cksum2 != c.crc_workng) w.RES_FRAM_INTEG_CHK_FAILED = true;
      //What to do about a checksum failure? Bytemask the error out.
```

```
Prohibitively
   //complicated, but doable. Not a priority for now.
   return micros64();
   }

   //Mercalli index:
   //
////////////////////////////////////////////////////////////////////////////////

   //                                   Mercalli index
             //
   //                                      PID: N/A
             //
   //
////////////////////////////////////////////////////////////////////////////////

   String Mercalli(float raw_scl_val_LSB) {
   String str = "";
   float v = (1./12000.)*raw_scl_val_LSB;
   //based on USGS information on Modified Mercalli intensity
index:
      if(v < 0.000464) str = "I";
   else if(v >= 0.000464 && v < 0.0015)   str = "II";
   else if(v >= 0.0015   && v < 0.00297)  str = "III";
   else if(v >= 0.00297  && v < 0.0276)   str = "IV";
   else if(v >= 0.0276   && v < 0.115)    str = "V";
   else if(v >= 0.115    && v < 0.215)    str = "VI";
   else if(v >= 0.215    && v < 0.401)    str = "VII";
   else if(v >= 0.401    && v < 0.747)    str = "VIII";
   else if(v >= 0.747    && v < 1.39)     str = "IX";
   else if(v >= 1.39)                     str + "X+";
   return str;
   }

   //
////////////////////////////////////////////////////////////////////////////////

   //                              state machine transition
             //
   //                                      PID: 01
             //
   //
////////////////////////////////////////////////////////////////////////////////

   //the MOST important piece of code in this program. It is what
makes this whole
```

```
  //microcontroller run. Essentially the "tier 1" outer loop program scheduler.
  //(Tier 0 is the inner loop scheduler.)
  uint64_t loop_mach(void) {
  Watchdog.reset();
  m.lastProcessIDActive = 0x01;
  //get free RAM (do something with this parameter later):
  m.ramfree = (uint16_t)freeRam();
  if(m.ramfree < 4096) {
    sendDiscordCSV(DISCORD_ALERTS_URL,F("[mach]␣**Notice**:␣Available␣SRAM␣less␣than␣4K␣due␣to␣long-term␣fragmentation.␣Restarting␣system."),"Wavefront␣6.4",INGENUITY_pfp,false);
    restart();
  }
  //this instruction takes a while
  SYNCFRAM;
  //check if we have a message waiting in serial:
  if(Serial.available()) sercmd_handler();

  //do we have a message waiting in the GFSK buffer?
  //Calls a periodic function
  if(rf69.available()) sercmd_handler(gfsk_handler());

  /* State machine transition section*/
  //Science:
  if(w.SCI_SCL3300_STDDEV_EXCEEDED_CHX == true && w.SCI_SCL3300_CHX_FIRED == false) {
    w.SCI_SCL3300_CHX_FIRED = true;
    if(status == WL_CONNECTED)
    sendDiscordCSV(DISCORD_ALERTS_URL,"[sci]␣X-axis␣stddev_t␣exceeded.␣Value:␣" + String(m.scl_t16_sdv_mpdx) + "␣LSB,␣" +\
  String((double)((1./12000.)*m.scl_t16_sdv_mpdx),5) + "*g*␣(MMI␣" + Mercalli(m.scl_t16_sdv_mpdx) + ")", "Wavefront␣6.4", INGENUITY_pfp,false);
  }
  else if(w.SCI_SCL3300_STDDEV_EXCEEDED_CHX == false) w.SCI_SCL3300_CHX_FIRED = false;

  if(w.SCI_SCL3300_STDDEV_EXCEEDED_CHY == true && w.SCI_SCL3300_CHY_FIRED == false) {
    w.SCI_SCL3300_CHY_FIRED = true;
    if(status == WL_CONNECTED)
    sendDiscordCSV(DISCORD_ALERTS_URL,"[sci]␣Y-axis␣stddev_t␣exceeded.␣Value:␣" + String(m.scl_t16_sdv_mpdy) + "␣LSB,␣" +\
  String((double)((1./12000.)*m.scl_t16_sdv_mpdy),5) + "*g*␣(MMI␣" + Mercalli(m.scl_t16_sdv_mpdy) + ")", "Wavefront␣6.4",
INGENUITY_pfp,false);
    }
    else if(w.SCI_SCL3300_STDDEV_EXCEEDED_CHY == false) w.SCI_SCL3300_CHY_FIRED = false;

    if(w.SCI_SCL3300_STDDEV_EXCEEDED_CHZ == true && w.SCI_SCL3300_CHZ_FIRED == false) {
      w.SCI_SCL3300_CHZ_FIRED = true;
      if(status == WL_CONNECTED)
      sendDiscordCSV(DISCORD_ALERTS_URL,"[sci]␣Z-axis␣stddev_t␣exceeded.␣Value:␣" + String(m.scl_t16_sdv_mpdz) + "␣LSB,␣" +\
    String((double)((1./12000.)*m.scl_t16_sdv_mpdz),5) + "*g*␣(MMI␣" + Mercalli(m.scl_t16_sdv_mpdz) + ")", "Wavefront␣6.4", INGENUITY_pfp,false);
    }
    else if(w.SCI_SCL3300_STDDEV_EXCEEDED_CHZ == false) w.SCI_SCL3300_CHZ_FIRED = false;

    if(w.SCI_ADT7410_HIGH_TEMP_FLAG == true && w.SCI_ADT7410_HTF_FIRED == false) {
      w.SCI_ADT7410_HTF_FIRED = true;
      if(status == WL_CONNECTED)
      sendDiscordCSV(DISCORD_ALERTS_URL,"[sci]␣**ALERT**␣Enclosure␣temperature␣high!␣Value:␣" + String(m.adt7410_temp_avg) + "␣*C","Wavefront␣6.4",INGENUITY_pfp,false);
    }
    else if(w.SCI_ADT7410_HIGH_TEMP_FLAG == false) { w.SCI_ADT7410_HTF_FIRED = false;}

    if(w.SCI_SCD30_CO2_HIGH_2000_PPM == true && w.SCI_SCD30_CO2H_FIRED == false) {
      w.SCI_SCD30_CO2H_FIRED = true;
      if(status == WL_CONNECTED)
      sendDiscordCSV(DISCORD_ALERTS_URL,"[sci]␣**ALERT**␣Ambient␣CO2␣high!␣Value:␣" + String(m.scd30_co2_avg) + "␣ppm","Wavefront␣6.4",INGENUITY_pfp,false);
    }
    else if(w.SCI_SCD30_CO2_HIGH_2000_PPM == false && w.SCI_SCD30_CO2H_FIRED == true) {
      w.SCI_SCD30_CO2H_FIRED = false;
      sendDiscordCSV(DISCORD_ALERTS_URL,"[sci]␣**ALERT**␣Ambient␣CO2␣high!␣Value:␣" + String(m.scd30_co2_avg) + "␣ppm","Wavefront␣6.4",INGENUITY_pfp,false);
    }

    //MACH Prio 0: check for FRAM SPI memory usage:
```

```
    //if(m.ram1_mem_free < 5000) deser_discord_serial();
    //if nearly out of FRAM, send to riv6n.net:
    if(m.ram1_mem_free < 5000) {
      w.RES_FRAM_FULL = true;
      //if there is no WiFi, do NOT collect data!
      if(status == WL_CONNECTED && dumpframhttp(true))
sendDiscordCSV(\
      DISCORD_ALERTS_URL,"FRAM␣successfully␣uploaded␣to␣riv6n.net.\
r\n\
␣␣␣␣␣‘‘Network␣metrics:␣high:␣" + String(w.ping_high) + "ms,␣low:␣"
+\
    String(w.ping_low) + "ms,␣avg:␣" + String(w.ping_avg) + "ms,␣
pings:␣" +\
    String(w.rpt_times) + "/8␣times.","Wavefront␣6.4",INGENUITY_pfp
,false);
      else if(status == WL_CONNECTED)
sendDiscordCSV(DISCORD_PASTEBIN_URL,\
    "[mach]␣Network␣conditions␣currently␣unsatisfactory␣to␣upload␣
FRAM.\
␣␣␣␣␣Retrying␣in␣60␣seconds...","Wavefront␣6.4",INGENUITY_pfp,false)
;


    }
    //Power state flag to Discord
    if((m.ina_volt_avg < 10.30 && m.ina_volt_avg > 9.41) &&
(m.powerState > 0)) {
      m.powerState = 1;
      if(status == WL_CONNECTED) sendDiscordCSV(DISCORD_ALERTS_URL,
"**Attention**:␣Battery␣low!␣Voltage:␣" + String(m.ina_volt_avg) +
"V.","Wavefront␣6.4",INGENUITY_pfp,false);
    }
    else if((m.ina_volt_avg < 9.40 && m.ina_volt_avg > 2.00 &&
(m.powerState == 0 || m.powerState == 1))) {
      m.powerState = 8;
      if(status == WL_CONNECTED) sendDiscordCSV(DISCORD_ALERTS_URL,
F("**Attention!**:␣Battery␣voltage␣below␣9.40V.␣Undervoltage␣
protection␣will␣activate␣soon␣and␣system␣will␣shut␣down."),"
Wavefront␣6.4",INGENUITY_pfp,false);
    }
    else if(m.ina_volt_avg > 10.30 && m.powerState > 0)
m.powerState = 0;
    else if(m.ina_volt_avg < 1.99 && m.powerState != 2) {
      m.powerState = 2;
      if(status == WL_CONNECTED) sendDiscordCSV(DISCORD_ALERTS_URL,
F("**ALERT**:␣Buck␣regulator␣NOT␣connected␣to␣system!␣Floating␣
ground␣present!"),"Wavefront␣6.4",INGENUITY_pfp,false);
```

```
    }
    //Integrity check:
    if(w.RES_FRAM_INTEG_CHK_FAILED == true && w.RES_FRAM_INTEG_MSG_FIRED == false) {
      if(status == WL_CONNECTED) sendDiscordCSV(DISCORD_ALERTS_URL, F("**ALERT**:␣FRAM␣integrity␣check␣failed:␣CRC32␣mismatch␣detected!␣FRAM␣might␣be␣corrupt!"),"Wavefront␣6.4",INGENUITY_pfp,false);
      w.RES_FRAM_INTEG_MSG_FIRED = true;
    }

    //WiFi roaming and connection retry:
    if(w.RES_CONNECTED_TO_WIFI == false) {
      //retry this every 5 minutes for roaming handler
      sched_blks[TASK_LOOP_WIFIDELEG] = 300000000ULL;
      //if we have no connection, but we are not full, slow down data collection!
      if(w.RES_FRAM_FULL == false) {
      sched_blks[TASK_LOOP_MAINSENSE] = 600000000ULL; //10 minutes
      }
      else {
      //if we are full, do NOT collect ANY data. Drop LoRa as well.
      sched_blks[TASK_LOOP_SCANLORA] = 429648582218ULL;
      sched_blks[TASK_LOOP_MACHSTATS]= 429648582218ULL;
      sched_blks[TASK_LOOP_MAINSENSE]= 492648582218ULL;
      }
    }
    else {
      //reset all loop settings if they were set and we are now connected and
      //uploaded:
      sched_blks[TASK_LOOP_SCANLORA]  = 1000000ULL;
      sched_blks[TASK_LOOP_MACHSTATS] = 300000000ULL;
      sched_blks[TASK_LOOP_MAINSENSE] = 120000000ULL;
      sched_blks[TASK_LOOP_WIFIDELEG] = 30000000ULL;
    }
    return micros64();
    }

    ////////////////////////////////////////////////////////////////////////////////
    //                           lora scan, for nodes                           //
    //                                  PID: F6                                 //
    //
```

```
////////////////////////////////////////////////////////////////////////////////
//adapted from standalone version of code
//also follows up with TRANSMIT ack, prototype for param upload
//follow up with 1k2 bps 4/8 FEC NOT 19k2 4/5!
//returns execution time
uint64_t loop_scanLora(void) {
//if not available, return and do something else
m.lastProcessIDActive = 0xF6;
SYNCFRAM;
Watchdog.reset();
if(rf95.available()) {
  //Serial.println(F("lora available"));
  //ADR feedback algorithm parameters
  //Negotiation will substantially increase program complexity from systems
  //engineering perspective
  bool adr = false; //ADR supported?
  int32_t relfreqErr = -915000000; //bs initialization
  int16_t rssi = -32767;
  int8_t  snr = -127;
  //check RAM availability (it is the function caller's responsibility to
  //not overflow the RAM)
  //Allow mach to send prio to mem management.
  if(m.ram1_mem_free < (uint32_t)1024) return micros64();
  //if available, DETERMINE the struct type. Addresses < 0x20 are type A,
  //0x21 to 0x3F are type B.
  //Need to translate struct to serialized plaintext.
  if(rf95.headerFrom() > 0x00 && rf95.headerFrom() < 0x21) {
  //constructor
  uint8_t datlen = sizeof(LTA);
  //decode to struct
  if(rf95.recv((uint8_t *)&LTA, &datlen));
  strip.setPixelColor(0,0x00FF00FF);
  strip.show();
  //populate to intermediate buffer
  String buff = "";
  //data type preamble (for deserializer):
  buff += "LN*";
  //preamble address in DEC
  buff += rf95.headerFrom(); buff += ",";
  //time of receipt (no fram in nodes):
  buff += rtc_iso8610(rtc, true,false);
  //last RSSI:
  rssi = rf95.lastRssi();
  buff += rssi; buff += ",";
  //last SNR:
  snr = rf95.lastSNR();
  buff += snr; buff += ",";
  //last frequency error:
  relfreqErr = rf95.frequencyError();
  buff += relfreqErr; buff += ",";
  //RX time in local millis()
  buff += millis(); buff += ",";
  //scientific data
  buff += LTA.tilt32_x;       buff += ",";
  buff += LTA.tilt32_y;       buff += ",";
  buff += LTA.tilt32_z;       buff += ",";
  buff += LTA.tilt32_max_x;   buff += ",";
  buff += LTA.tilt32_min_x;   buff += ",";
  buff += LTA.tilt32_max_y;   buff += ",";
  buff += LTA.tilt32_min_y;   buff += ",";
  buff += LTA.tilt32_max_z;   buff += ",";
  buff += LTA.tilt32_min_z;   buff += ",";
  buff += LTA.meas_humidity;  buff += ",";
  buff += LTA.scltemp;        buff += ",";
  buff += LTA.adt7410;        buff += ",";
  buff += dtosstrf(LTA.sysvolt,4,5); buff += ",";
  buff += LTA.meas_pressure; buff += ",";
  buff += LTA.btmp_lower;    buff += ",";
  buff += LTA.spect[0];      buff += ",";
  buff += LTA.spect[1];      buff += ",";
  buff += LTA.spect[2];      buff += ",";
  buff += LTA.spect[3];      buff += ",";
  buff += LTA.spect[6];      buff += ",";
  buff += LTA.spect[7];      buff += ",";
  buff += LTA.spect[8];      buff += ",";
  buff += LTA.spect[9];      buff += ",";
  buff += LTA.spect[10];     buff += ",";
  buff += LTA.spect[11];     buff += ",";
  buff += LTA.spect[12];     buff += "\r\n";
  //deserialization token is \r.

  //send to SPI RAM:
  //Arg1: retrieve head of buffer (size of ram-ram remaining)
  //Arg2: typecast to byte
  //Arg3: length of String, INCLUDES NULL TERMINATOR!
  RAM.writeEnable(true);
  RAM.write(m.ram1_sz-m.ram1_mem_free,\
    (const uint8_t *)buff.c_str(),buff.length());
```

```
     m.ram1_mem_free -= (buff.length()); //bookkeeping
     RAM.writeEnable(false);
     //destruct and clean up:
     //(disabled for now, dangerous operation
     //memset((void *)&LTA,'\000',sizeof(LTA));
     //memset((void *)&buff,'\000',buff.length());
     //m.ram1_mem_free -= buff.length();
     m.data_counter_lora += buff.length();
     SYNCFRAM;
     Serial.print(buff);
     strip.setPixelColor(0,0x0);
     strip.show();
     }
     else if(rf95.headerFrom() >= 0x21 && rf95.headerFrom() < 0x40) {
     //Unimplemented as of April 2022. Node still needs to be designed.
     }
   }
   //Serial.println(F("lora not available, exiting..."));
   return micros64();
   }

   // ////////////////////////////////////////////////////////////////////////////////
   //                            Operational Summary, 4h                                          //
   //                                   PID: F4                                                   //
   // ////////////////////////////////////////////////////////////////////////////////
   uint64_t bihourly_summ(void) {
   String su = "Wavefront␣Top-level␣Summary␣as␣of␣";
   su += rtc_iso8610(rtc,false,false);
   su += "␣Z:\r\n";
   su += "Total␣data␣processed:␣␣"; su += String(m.data_counter_total/1024.0,2);\
    su += "kB\r\n";
   su += "LoRa␣data␣forwarded:␣␣␣"; su += String(m.data_counter_lora/1024.0,2);\
    su += "kB\r\n";
   su += "Stack␣RAM␣free:␣␣␣␣␣␣␣␣"; su += String(m.ramfree/1024.0,3); su +=\
    "kB\r\n";
    su += "FRAM␣free:␣␣␣␣␣␣␣␣␣␣␣␣␣"; su +=
String(m.ram1_mem_free/1024.0 ,2);\
     su += "kB\r\n";
    su += "Last␣CO2␣average:␣␣␣␣␣␣"; su += m.scd30_co2_avg; su += "
ppm\r\n";
    su += "Last␣ambient␣temp:␣␣␣␣␣"; su += m.adt7410_temp_avg; su +
= "*C\r\n";
    su += "Last␣int.␣humidity:␣␣␣␣"; su += m.bme680_rh_avg; su += "
%RH\r\n";
    su += "Last␣batt.␣voltage:␣␣␣␣"; su += m.ina_volt_avg; su += "V
\r\n";
    su += "Last␣hyd␣sulfide:␣␣␣␣␣␣"; su += String(m.gasv_h2s_avg
*1000,1);\
     su += "ppb\r\n";
    su += "Last␣sulf␣dioxide:␣␣␣␣␣"; su += String(m.gasv_so2_avg
*1000,1);\
     su += "ppb\r\n";
    su += "Last␣ozone/nitrous␣ox:␣"; su += String(m.gasv_o3_avg
*-1,3);\
     su += "ppm\r\n";
    su += "Last␣VOC␣content:␣␣␣␣␣␣"; su += m.ccs811_tvoc_avg; su +=
 "ppb\r\n";
    su += "Last␣X␣tilt:␣␣␣␣␣␣␣␣␣␣␣"; su +=
String(atan2(m.scl_zt_decim ,\
    m.scl_xt_decim)*(360/(2*PI))*60,2); su += "␣MOA\r\n";
    su += "Last␣Y␣tilt:␣␣␣␣␣␣␣␣␣␣␣"; su +=
String(atan2(m.scl_zt_decim ,\
    m.scl_yt_decim)*(360/(2*PI))*60,2); su += "␣MOA\r\n";
    su += "Last␣checksum␣A:␣␣␣␣␣␣␣"; su += "0x"; su +=
String(c.crc_master ,\
    HEX); su += "\r\n";
    su += "Last␣checksum␣B:␣␣␣␣␣␣␣"; su += "0x"; su +=
String(c.crc_workng ,\
    HEX); su += "\r\n";
    //su += "Last reported location:"; su += "\r\n";
    //if(m.gps_Lat_max == 0)            su += "Not available\r\n";
    //else if(m.gps_Lat_max >= 89.9f && m.gps_Lat_max < 90.1f) su +
=
    //"REDACTED\r\n";
    //else {su += "  Lat: "; su += (m.gps_Lat_max/1E7); su += "\r\n
";}
    //if(m.gps_Long_max == 0)           su += "Not available\r\n";
    //else if(m.gps_Long_max >= -0.1f && m.gps_Long_max <= 0.1f) su
 +=
    // "REDACTED\r\n";
    //else {su += "  Long: "; su += (m.gps_Long_max/1E7); su += "\r
```

```
\n";}
   sendDiscordCSV(DISCORD_ALERTS_URL,su,"Wavefront␣6.4",
INGENUITY_pfp,true);
   return micros64();
   }


   void exec(int pid) {
   switch(pid) {
     case 0:
     //MACH calls one-time, deterministic programs
     lprc[0] = loop_mach(); break;
     case 1:
     lprc[1] = loop_scanLora(); break;
     case 2:
     lprc[2] = loop_MainSense(); break;
     case 3:
     lprc[3] = loop_machstats(); break;
     case 4:
     lprc[4] = wifi_delegate(WiFi.status()); break;
     case 5:
     lprc[5] = bihourly_summ(); break;
     case 6:
     lprc[6] = fram_integrity_check(); break;
     //case 7:
     //lprc[7] = gpsStats(gps,8); break;
     default:
     break;
   }
   }

   //do not change func name!!!
   //THE scheduler
   void tifa(void) {
   while(true) {
     for(int p=0;p<NUMTASKS;p++) {
     if(w.RES_FIRST_LOOP == true) {
       exec(p);
     }
     else if((micros64() >= lprc[p] + sched_blks[p])) {
       //Serial.println(micros());
       exec(p);
     }
     Watchdog.reset();
     }
     w.RES_FIRST_LOOP = false;
```

```
}
}


EXEC
```

5. Gateway Code - Inclinometer and Strong-motion Seismometer

```
#include <Arduino.h>
#include <Wire.h>
#include <Adafruit_DotStar.h>
#include <SCL3300.h>
//To make this code work on a Raspberry Pi RP2040, I WILL need the Arduino
  //base framework because of Strings. Shouldn't be too much of a problem,
  //except for pin assignments depending on the variant.h defined. The
  //PIO state logic will be a whole other beast, as that's programmed
  //in Assembly.

  //C is the best systems programming language, hands down.

  //yes thats right
  #define main(void) setup(void)

  //networking
  #define SER_MASTER    "A01"
  #define SER_ADDR      "A3A"
  #define HPIN          13
  static char *cmd_table[] {
    "RD1",
    "RD2",
    "RD3",
    "RDALL.INST",
    "RDALL",
    "PING",
    "IIRSET", //set IIR coefficients, returns new and old values
    "SPS"
  };

  #define NUMPIXELS      1
  #define DATAPIN       41
  #define CLOCKPIN      40
  Adafruit_DotStar strip(NUMPIXELS, DATAPIN, CLOCKPIN, DOTSTAR_BRG);

  //Third order IIR filter coefficients - initialized
  //adjustable via command
  //By default, extremely stiff
  #define IIR_A1       0.98
  #define IIR_A2       0.018
  #define IIR_A3       0.002

  //Hard coded 5th order temperature compensation coefficients.
  //Experimentally determined through highly controlled
  //enviromment.
  //Loaded in via FRAM or mission computer by network command.
  #define TCT_A0       0.0
  #define TCT_A1       0.0
  #define TCT_A2       0.0
  #define TCT_A3       0.0
  #define TCT_A4       0.0

  SCL3300 scl0;
  SCL3300 scl1;
  SCL3300 scl2;

  //decimal to Arduino string (allocated to stack, not for use on ATMega series.)
  String dtosstrf(double val, signed char width, unsigned char prec) {
    asm(".global␣_printf_float");
    char sout[64];
    char fmt[20];
    sprintf(fmt, "%%%d.%df", width, prec);
    sprintf(sout, fmt, val);
    String ssout(sout);
    return ssout;
  }

  //Writes a 1 to the SYSRESETREQ register to restart the CPU.
  void restart(void)  {
    __asm volatile ("cpsid␣i" ::: "memory"); //disable interrupt reporting
    __asm volatile ("dsb␣0xF":::"memory"); //commit
    SCB->AIRCR = ((0x5FAUL << SCB_AIRCR_VECTKEY_Pos) | SCB_AIRCR_SYSRESETREQ_Msk); //write to system control block to reset
    __asm volatile ("dsb␣0xF":::"memory");
    for(;;) __asm volatile("nop");
  }

  //Hardcoded. ALWAYS *to* the master controller. May change if I do a masterless bus.
  void send_header(String address) {
    Serial1.print("<A01," + (String)address + (String)">");
  }

  //Special print function to send with TX line brought high
  void printAddr(uint32_t MASTxr, String address, String message) {
    digitalWrite(MASTxr, HIGH);
    //potential to bring this time down to 10 uS, but being extremely
    //conservative for now
    delay(1);
    send_header(address);
    Serial1.println(message);
    delay(1);
    digitalWrite(MASTxr, LOW);
  }

  //struct to hold all program data
  //edited 040522 to include 16-bit statistical data with this
  //significant expansion
  //050322: demote ALL double prec floads to singles. takes 4x as long to process
  //a double compared to a float.
  struct seismoStruct {
    //baseline data:
    int32_t tiltos32_0x; int32_t tiltos32_0y; int32_t tiltos32_0z;
    int32_t tiltos32_1x; int32_t tiltos32_1y; int32_t tiltos32_1z;
    int32_t tiltos32_2x; int32_t tiltos32_2y; int32_t tiltos32_2z;
    //supplementary seismo data:
    //added 050322:
    //reset registers after every read
    float t16_max_x_mp; float t16_min_x_mp; float t16_sdv_mpdx;
    float t16_max_y_mp; float t16_min_y_mp; float t16_sdv_mpdy;
    float t16_max_z_mp; float t16_min_z_mp; float t16_sdv_mpdz;
    //statistics - max32f:
    float tilt16_max_x; float tilt16_max_y; float tilt16_max_z;
    //statistics - min32f:
    float tilt16_min_x; float tilt16_min_y; float tilt16_min_z;
    //statistics - avg32f:
    float tilt16_avg_x; float tilt16_avg_y; float tilt16_avg_z;
    //statistics - stddevf:
    float tilt16_sdv_x; float tilt16_sdv_y; float tilt16_sdv_z;
    //3rd order IIR
    double iirf_x[3]; double iirf_y[3]; double iirf_z[3];
    double xiir; double yiir; double ziir;
    uint32_t sps; //samples per sec
    uint32_t samp_time; //initialize to 2000msec
    double iir_coeffs[3];
    uint32_t tilt0_temp; uint32_t tilt1_temp; uint32_t tilt2_temp;
    double iirf_t[3]; //temperature long period
    double tiir;
    double temperature; //drift correction - ADT7410 controlled
    uint32_t readComplete;
  }; static seismoStruct ss;

  //Restart the CPU if a rollover will happen.
  //Critical due to timing sensitive routines.
  //Fixable in the future with
  uint32_t clk_wd(uint32_t delays) {
    if(millis() > 4200000000) restart();
    else return millis();
  }

  //this is a very loaded process, requires 99% of cpu time to be effective
  //will result in highly delayed responses on the serial network
  //i will not be able to read the sensors at 2,000 Hz, unless I can drive the sensors
  //at 8 MHz. I will try. Edit: can only drive sensors at 333 Hz @ 8 MHz.
  //Ideally, should read at 2,000 Hz to achieve the lowest noise density possible

  /*
   * Control system:
   * Some explanataion is needed here so I don't forget what I did, and so I
   * can document it in the report.
   *
   * This computationally inefficient, discrete-time filter depends on a
   * priori state information, as any filter does. (Side note, this could be
   * made a Kalman filter if I add a predictor step, but that's useless in the
   * case of an ultra precision inclinometer.)
   *
   * 1. The input data (in int16) is integrated over 2000 msec into an int32
   * accumulator variable from three precision MEMS accelerometers.
   *
   * 2. The current state of all sensor axes is averaged to create one 3D
   * vector.
   *
   * 3. The second order value is an average of the a priori state and the
   * current state.
   *
   * 4. The third order value is the a priori state only.
   *
   * 5. All values are summed with their corresponding weights to form the final
   * filtered output value.
   *
   * 6. The current state is then transferred to the a priori state variable.
   *
   * With a second order filter, the frequency response rolls off at 20 dB/octave
   * then 40 dB/octave.
   */
  uint32_t reads(uint32_t delay_input) {
    //for oscilloscope timing analysis
    digitalWrite(12, LOW);
    uint32_t startread = millis();
    uint32_t samps_taken = 0; //total samples taken
    //RESET INTEGRATOR!
    ss.tiltos32_0x = 0;
    ss.tiltos32_0y = 0;
    ss.tiltos32_0z = 0;
    ss.tiltos32_1x = 0;
    ss.tiltos32_1y = 0;
    ss.tiltos32_1z = 0;
    ss.tiltos32_2x = 0;
    ss.tiltos32_2y = 0;
    ss.tiltos32_2z = 0;
    ss.tilt0_temp = 0;
    ss.tilt1_temp = 0;
    ss.tilt2_temp = 0;

    //reset stats as well!
    ss.tilt16_max_x = -99999999.0; ss.tilt16_max_y = -99999999.0; ss.tilt16_max_z = -99999999.0;
    ss.tilt16_min_x = 99999999.0; ss.tilt16_min_y = 99999999.0; ss.tilt16_min_z = 99999999.0;
    ss.tilt16_avg_x = 0.0; ss.tilt16_avg_y = 0.0; ss.tilt16_avg_z = 0.0;
    ss.tilt16_sdv_x = 0.0; ss.tilt16_sdv_y = 0.0; ss.tilt16_sdv_z = 0.0;

    while(millis() <= startread + ss.samp_time) {
    //obtain sample:
    if(scl0.available()) {
      //accumulate to oversampling buffer and convert to int32_t
      ss.tiltos32_0x += (int32_t)scl0.sclData.AccX;
      ss.tiltos32_0y += (int32_t)scl0.sclData.AccY;
      ss.tiltos32_0z += (int32_t)scl0.sclData.AccZ;
      ss.tilt0_temp  += (uint32_t)scl0.sclData.TEMP;
    }
    if(scl1.available()) {
      ss.tiltos32_1x += (int32_t)scl1.sclData.AccX;
      ss.tiltos32_1y += (int32_t)scl1.sclData.AccY;
      ss.tiltos32_1z += (int32_t)scl1.sclData.AccZ;
      ss.tilt1_temp  += (uint32_t)scl1.sclData.TEMP;
    }
    if(scl2.available()) {
      ss.tiltos32_2x += (int32_t)scl2.sclData.AccX;
      ss.tiltos32_2y += (int32_t)scl2.sclData.AccY;
      ss.tiltos32_2z += (int32_t)scl2.sclData.AccZ;
      ss.tilt2_temp  += (uint32_t)scl2.sclData.TEMP;
    }

    //estimated 2.7 uS to execute all if statements
    //concatenate and select maximum values of ALL sensors for this read cycle:
    //max:
    if(scl0.sclData.AccX < scl1.sclData.AccX) ss.tilt16_max_x = scl1.sclData.AccX;
    if(scl1.sclData.AccX < scl2.sclData.AccX) ss.tilt16_max_x = scl2.sclData.AccX;
    if(scl2.sclData.AccX < scl0.sclData.AccX) ss.tilt16_max_x = scl0.sclData.AccX;
    if(scl0.sclData.AccY < scl1.sclData.AccY) ss.tilt16_max_y = scl1.sclData.AccY;
    if(scl1.sclData.AccY < scl2.sclData.AccY) ss.tilt16_max_y = scl2.sclData.AccY;
    if(scl2.sclData.AccY < scl0.sclData.AccY) ss.tilt16_max_y = scl0.sclData.AccY;
    if(scl0.sclData.AccZ < scl1.sclData.AccZ) ss.tilt16_max_z =
scl1.sclData.AccZ;
    if(scl1.sclData.AccZ < scl2.sclData.AccZ) ss.tilt16_max_z =
scl2.sclData.AccZ;
    if(scl2.sclData.AccZ < scl0.sclData.AccZ) ss.tilt16_max_z =
scl0.sclData.AccZ;

    //min
    if(scl0.sclData.AccX > scl1.sclData.AccX) ss.tilt16_min_x =
scl1.sclData.AccX;
    if(scl1.sclData.AccX > scl2.sclData.AccX) ss.tilt16_min_x =
scl2.sclData.AccX;
    if(scl2.sclData.AccX > scl0.sclData.AccX) ss.tilt16_min_x =
scl0.sclData.AccX;
    if(scl0.sclData.AccY > scl1.sclData.AccY) ss.tilt16_min_y =
scl1.sclData.AccY;
    if(scl1.sclData.AccY > scl2.sclData.AccY) ss.tilt16_min_y =
scl2.sclData.AccY;
    if(scl2.sclData.AccY > scl0.sclData.AccY) ss.tilt16_min_y =
scl0.sclData.AccY;
    if(scl0.sclData.AccZ > scl1.sclData.AccZ) ss.tilt16_min_z =
scl1.sclData.AccZ;
    if(scl1.sclData.AccZ > scl2.sclData.AccZ) ss.tilt16_min_z =
scl2.sclData.AccZ;
    if(scl2.sclData.AccZ > scl0.sclData.AccZ) ss.tilt16_min_z =
scl0.sclData.AccZ;
    samps_taken++;
    }
    ss.sps = samps_taken/2;

    //avg and stddev:
    ss.tilt16_avg_x = (float)ss.tiltos32_0x/(float)samps_taken;
    ss.tilt16_avg_y = (float)ss.tiltos32_0y/(float)samps_taken;
    ss.tilt16_avg_z = (float)ss.tiltos32_0z/(float)samps_taken;
    ss.tilt16_sdv_x = sqrt((1.0/((float)samps_taken-1.0))*
pow((abs((float)scl0.sclData.AccX-ss.tilt16_avg_x)),2));
    ss.tilt16_sdv_y = sqrt((1.0/((float)samps_taken-1.0))*
pow((abs((float)scl0.sclData.AccY-ss.tilt16_avg_y)),2));
    ss.tilt16_sdv_z = sqrt((1.0/((float)samps_taken-1.0))*
pow((abs((float)scl0.sclData.AccZ-ss.tilt16_avg_z)),2));

    //long-term maxima register
    if(ss.t16_max_x_mp < ss.tilt16_max_x) ss.t16_max_x_mp =
ss.tilt16_max_x;
    if(ss.t16_max_y_mp < ss.tilt16_max_y) ss.t16_max_y_mp =
ss.tilt16_max_y;
    if(ss.t16_max_z_mp < ss.tilt16_max_z) ss.t16_max_z_mp =
ss.tilt16_max_z;
    //long-term minima register
    if(ss.t16_min_x_mp > ss.tilt16_min_x) ss.t16_min_x_mp = ss.tilt16_min_x;
    if(ss.t16_min_y_mp > ss.tilt16_min_y) ss.t16_min_y_mp = ss.tilt16_min_y;
    if(ss.t16_min_z_mp > ss.tilt16_min_z) ss.t16_min_z_mp = ss.tilt16_min_z;
    //long-term standard deviation register
    if(ss.t16_sdv_mpdx < ss.tilt16_sdv_x) ss.t16_sdv_mpdx = ss.tilt16_sdv_x;
    if(ss.t16_sdv_mpdy < ss.tilt16_sdv_y) ss.t16_sdv_mpdy = ss.tilt16_sdv_y;
    if(ss.t16_sdv_mpdz < ss.tilt16_sdv_z) ss.t16_sdv_mpdz = ss.tilt16_sdv_z;

    //decimate by oversamp_time/samp_time (no way around a floating pt)
    //(AFTER oversampled data is taken)
    //actually what am I thinking? this can be done without floating point numbers
    //Oscilloscope debug: 5.1msec to execute. (old info as of 050322)

    //X filtered:
    ss.xiir      = ss.iir_coeffs[2]*((double)(ss.tiltos32_0x+ss.tiltos32_1x+ss.tiltos32_2x)/3.0) +\
           ss.iir_coeffs[1]*(ss.iirf_x[2]+((double)(ss.tiltos32_0x+ss.tiltos32_1x+ss.tiltos32_2x)/3.0))/2.0 +\
           ss.iir_coeffs[0]*(ss.iirf_x[2]);
    ss.iirf_x[2] = ss.xiir;

    //Y filtered:
    ss.yiir      = ss.iir_coeffs[2]*((double)(ss.tiltos32_0y+ss.tiltos32_1y+ss.tiltos32_2y)/3.0) +\
           ss.iir_coeffs[1]*(ss.iirf_y[2]+((double)(ss.tiltos32_0y+ss.tiltos32_1y+ss.tiltos32_2y)/3.0))/2.0 +\
           ss.iir_coeffs[0]*(ss.iirf_y[2]);
    ss.iirf_y[2] = ss.yiir;

    //Z filtered:
    ss.ziir      = ss.iir_coeffs[2]*((double)(ss.tiltos32_0z+ss.tiltos32_1z+ss.tiltos32_2z)/3.0) +\
           ss.iir_coeffs[1]*(ss.iirf_z[2]+((double)(ss.tiltos32_0z+ss.tiltos32_1z+ss.tiltos32_2z)/3.0))/2.0 +\
           ss.iir_coeffs[0]*(ss.iirf_z[2]);
    ss.iirf_z[2] = ss.ziir;

    //Temperature filtered (REQUIRED for precision corrections!):
    ss.tiir      = ss.iir_coeffs[2]*((double)(ss.tilt0_temp+
ss.tilt1_temp+ss.tilt2_temp)/3.0) +\
           ss.iir_coeffs[1]*(ss.iirf_t[2]+((double)(ss.tilt0_temp+
ss.tilt1_temp+ss.tilt2_temp)/3.0))/2.0 +\
           ss.iir_coeffs[0]*(ss.iirf_t[2]);
    ss.iirf_t[2] = ss.tiir;

    digitalWrite(12, HIGH);
    return millis();
  }

  //command and data handling
  uint32_t sercom(uint32_t delay_input) {
    //there is a packet in the buffer waiting
    if(Serial.available()) {
    //buffer the string from the hardware serial buffer into RAM
buffer:
    String buf = "";
    buf = Serial.readStringUntil(',');
    //if the message is to itself (should never happen),
    if(buf == ("<" + (String)SER_ADDR)) return millis();
    //digest the next portion of the packet:
    buf = Serial.readStringUntil('>');
    //if it is destined to us,
    if(buf == SER_ADDR) {
      strip.setPixelColor(0,0x00FFFFFF);
      strip.show();
      buf = Serial.readStringUntil(',');
      //read last recorded tilt value on sensor 1
      if(buf == cmd_table[0]) {
      //for type conversion so I don't have to make an overloaded
function
      String tmp;
      tmp = (String)millis() + "," +\
          (String)ss.tiltos32_0x + "," +\
          (String)ss.tiltos32_0y + "," +\
          (String)ss.tiltos32_0z + "," +\
          (String)ss.tilt0_temp;
      printAddr(HPIN, SER_MASTER, tmp);
      }
      //read last recorded tilt value on sensor 2
      else if(buf == cmd_table[1]) {
      String tmp;
```

```
        tmp = (String)millis() + "," +\
            (String)ss.tiltos32_1x + "," +\
            (String)ss.tiltos32_1y + "," +\
            (String)ss.tiltos32_1z + "," +\
            (String)ss.tilt1_temp;
        printAddr(HPIN, SER_MASTER, tmp);
        }
        //read last recorded tilt value on sensor 3
        else if(buf == cmd_table[2]) {
        String tmp;
        tmp = (String)millis() + "," +\
            (String)ss.tiltos32_2x + "," +\
            (String)ss.tiltos32_2y + "," +\
            (String)ss.tiltos32_2z + "," +\
            (String)ss.tilt2_temp;
        printAddr(HPIN, SER_MASTER, tmp);
        }
        //read last recorded tilt value averaged among all sensors
        //include internal temperatures to provide calibration
        else if(buf == cmd_table[3]) {
        String tmp;
        tmp = (String)millis() + "," +\
            (String)((ss.tiltos32_0x+ss.tiltos32_1x+ss.tiltos32_2x)
/3.0) +\
            "," +\
            (String)((ss.tiltos32_0y+ss.tiltos32_1y+ss.tiltos32_2y)
/3.0) +\
            "," +\
            (String)((ss.tiltos32_0z+ss.tiltos32_1z+ss.tiltos32_2z)
/3.0);
        printAddr(HPIN, SER_MASTER, tmp);
        }
        //read last 300 second IIR-averaged filter, with all
necessary data to
        //fully characterize timing, filtering, and temperature

        //edit 040522: everything including the kitchen sink to
provide fine-
        //grained metrics and possible earthquake and strong-motion
ident.
        //edit 050322: the kitchen sink wasn't enough, need the
whole house
        //to determine earthquakes (registers for max over
accumulated period)
        else if(buf == cmd_table[4]) {
        String tmp;
        tmp = (String)millis() + "," +\
            (String)ss.sps + "," +\
            (String)ss.xiir + "," +\
            (String)ss.yiir + "," +\
            (String)ss.ziir + "," +\
            (String)ss.tiir + "," +\
            (String)ss.tilt16_max_x + "," +\
            (String)ss.tilt16_min_x + "," +\
            (String)ss.tilt16_avg_x + "," +\
            (String)ss.tilt16_sdv_x + "," +\
            (String)ss.tilt16_max_y + "," +\
            (String)ss.tilt16_min_y + "," +\
            (String)ss.tilt16_avg_y + "," +\
            (String)ss.tilt16_sdv_y + "," +\
            (String)ss.tilt16_max_z + "," +\
            (String)ss.tilt16_min_z + "," +\
            (String)ss.tilt16_avg_z + "," +\
            (String)ss.tilt16_sdv_z + "," +\
            (String)ss.t16_max_x_mp + "," +\
            (String)ss.t16_min_x_mp + "," +\
            (String)ss.t16_sdv_mpdx + "," +\
            (String)ss.t16_max_y_mp + "," +\
            (String)ss.t16_min_y_mp + "," +\
            (String)ss.t16_sdv_mpdy + "," +\
            (String)ss.t16_max_z_mp + "," +\
            (String)ss.t16_min_z_mp + "," +\
            (String)ss.t16_sdv_mpdz;

        //060422 - USBSerial diversion. Most sensors and board fried.
        //Need to finish off the main instrument: seismometer, which
        //survived, connected to Raspberry Pi.
        //Serial1.print(tmp);
        Serial.println(tmp);
        //NOW FINALLY RESET THE BUFFERS:
        ss.t16_sdv_mpdx = 0.0;
        ss.t16_sdv_mpdy = 0.0;
        ss.t16_sdv_mpdz = 0.0;
        ss.t16_max_x_mp = -99999999.0;
        ss.t16_max_y_mp = -99999999.0;
        ss.t16_max_z_mp = -99999999.0;
        ss.t16_min_x_mp = 99999999.0;
        ss.t16_min_y_mp = 99999999.0;
        ss.t16_min_z_mp = 99999999.0;
        //printAddr(HPIN, SER_MASTER, tmp);
        }
```

```
    //ping command, are we ok?
    else if(buf == cmd_table[5]) {
    printAddr(HPIN, SER_MASTER, "ok");
    }
    else if(buf == cmd_table[6]) {
    //incremental read of buffer...
    double tmp[3];
    //NO ERROR CHECKING! BEWARE!
    buf = Serial.readStringUntil(',');
    tmp[0] = buf.toDouble();
    buf = Serial.readStringUntil(',');
    tmp[1] = buf.toDouble();
    buf = Serial.readStringUntil('\n');
    tmp[2] = buf.toDouble();
    //the only check is if it sums to 1.0:
    if((tmp[0] + tmp[1] + tmp[2]) >= 0.999999 && \
       (tmp[0] + tmp[1] + tmp[2]) <= 1.000001) {
      memcpy((void *)ss.iir_coeffs,(const void *)tmp, sizeof(ss.iir_coeffs));
      printAddr(HPIN, SER_MASTER, "ok");
    } else printAddr(HPIN, SER_MASTER, "ERROR␣CMD␣6␣-␣INVALID␣IIR␣FILTER␣COEFFICIENTS");
    }
    else if(buf == cmd_table[7]) {
    String t = (String)ss.sps;
    printAddr(HPIN, SER_MASTER, t);
    }
  }
  }
  strip.setPixelColor(0,0x0);
  strip.show();
  return millis();
}

//Super basic scheduler:
//globals
#define NUM_PROCESSES   3

uint32_t delays[] = {
  500,    //Serial communication task  |  PID 0
  5,      //Murata read - synchronous  |  PID 1
  60000   //Clock watchdog             |  PID 2
};
uint32_t last_proc_run_time[NUM_PROCESSES] = {
  0,
  0,
  0
};

//Executive "lookup table".
void exec(int pid) {
  switch (pid) {
  case 0:
  last_proc_run_time[0] = sercom(delays[0]);
  break;
  case 1:
  last_proc_run_time[1] = reads(delays[1]);
  break;
  case 2:
  last_proc_run_time[2] = clk_wd(delays[2]);
  default:
  //other instructions take too much time
  asm("nop");
 }
}

void scheduler_basic(uint32_t delays_in[]) {
  //scan if timer on any process is zero:
  //this will help when there are concurrent processes waiting to be executed
  //IMPORTANT: p<NUM_PROC NOT p<=NUM_PROC
  for(int p=0;p<NUM_PROCESSES;p++) {
  if(millis() >= last_proc_run_time[p] + delays[p]) {
    exec(p);
  }
  }
}

void main() {
  //Remember to EXPLICITLY DECLARE MOSFET PIN AS OUTPUT!
  pinMode(HPIN, OUTPUT);
  strip.begin();
  strip.setBrightness(64);
  strip.setPixelColor(0,0x00FF0000);
  scl0.begin(A5);
  scl1.begin(A4);
  scl2.begin(A3);
  //struct initializers
  ss.iir_coeffs[0] = IIR_A1;
  ss.iir_coeffs[1] = IIR_A2;
  ss.iir_coeffs[2] = IIR_A3;
  ss.samp_time = 2000;
  //bulk initialize to 0.0f;
  memset(ss.iirf_x,'\x00',sizeof(ss.iirf_x));
  memset(ss.iirf_y,'\x00',sizeof(ss.iirf_y));
  memset(ss.iirf_z,'\x00',sizeof(ss.iirf_z));
  Serial1.begin(9600);
  Serial.begin(1000000);
  while(true) {
  scheduler_basic(delays);
  }
}

void loop() {}
```

6. Node Code - Node Version 2

```
#define LTA_NODE_ID    0x03
/*
 * NINES Node - Serpac RB22
 * Code v1.1.7 [Deployed]
 * Stanley Krzesniak
 * This work is a part of my masters project.
 *
*/
#include <Arduino.h>
#include <nines.h> //yes that's right, my own library cause its high time I have one
#include <RH_RF95.h>
#include <Adafruit_MS8607.h>
#include <Adafruit_Sensor.h>
#include <Adafruit_BusIO_Register.h>
#include <SPI.h>
#include <SCL3300.h>
#include "Adafruit_ADT7410.h"
#include <Adafruit_SleepyDog.h>
#include <Adafruit_AS7341.h>

Adafruit_MS8607 meas;
Adafruit_ADT7410 ts;
SCL3300 inclinometer;

Nines n;
Adafruit_AS7341 spect;

#define RFM95_CS A1
#define RFM95_RST A3
#define RFM95_INT A2

RH_RF95 rf95(RFM95_CS, RFM95_INT);

int gain = 0;
double euclidean;
int iters = 0;



void setup() {
  //delay(6000);
```

```
//Serial.begin(9600);
pinMode(PIN_SERIAL1_TX, OUTPUT);
pinMode(A0, INPUT);

//Arduino libraries are lying to me. According to the Atmel SAMD21 datasheet, the ADCs are
//capable of 16 bit resolution. Good riddance ADS1115.
//Values are straight from the chip support headers, from Atmel.
analogReadResolution(12);
/*
ADC->CTRLB.bit.RESSEL = ADC_CTRLB_RESSEL_16BIT_Val;
while(ADC->STATUS.bit.SYNCBUSY == 1); //sync

//Even more insult to injury, THE ADC HAS WIDER GAIN THAN THE ADS1115, NOT TO MENTION THE
//OPTION TO **SELECT** A GAIN.
ADC->INPUTCTRL.bit.GAIN = ADC_INPUTCTRL_GAIN_1X_Val; //write to ADC block, 1x gain
//ADC->REFCTRL.bit.REFSEL = ADC_REFCTRL_REFSEL_AREFA_Val; //select ADC Ref pin on ItsyBitsy, if needed.
while (ADC->STATUS.bit.SYNCBUSY == 1); //sync
*/

digitalWrite(PIN_SERIAL1_TX, LOW); //this pin drives the sleep enable timer
//float volt = ((analogRead(A0)*(1/0.7297297)*3.3)/pow(2.,12.));
//Serial.println(volt);
//if(volt <= 3.35) { //if battery is empty, shut down immediately.
//  digitalWrite(PIN_SERIAL1_TX, HIGH);
//  while(1);
//}
//Serial.begin(9600);
//while(!Serial);

if (inclinometer.begin(PIN_SERIAL1_RX) == false) {
Serial.println("Murata SCL3300 inclinometer not connected.");
while(1); //Freeze
}
if(!ts.begin()) {
Serial.println("Couldn't find ADT7410!");
while(1);
}
if(!meas.begin()) {
Serial.println("Failed to find MS8607 chip");
while(1);
}
```

```
  if (!spect.begin()){
  Serial.println("Could␣not␣find␣AS7341");
  while (1) { delay(10); }
  }
  meas.setHumidityResolution(MS8607_HUMIDITY_RESOLUTION_OSR_12b);
  pinMode(RFM95_RST, OUTPUT);
  digitalWrite(RFM95_RST, HIGH);
  delay(10);
  digitalWrite(RFM95_RST, LOW);
  delay(10);
  digitalWrite(RFM95_RST, HIGH);
  delay(100);
  if (!rf95.init()) {
  Serial.println("LoRa␣init␣failed.␣Check␣your␣connections.");
  while (true);                       // if failed, do nothing
  }
  Watchdog.enable(90000);
  spect.setLEDCurrent(10);
  rf95.setFrequency(922.00625);
  rf95.setModemConfig(RH_RF95::Bw31_25Cr48Sf4096); //61.25 kHz
  //rf95.setLowDatarate();
  rf95.setTxPower(0,false);
  //BEFORE UPLOADING, CHANGE THIS VARIABLE!!
  rf95.setHeaderFrom(LTA_NODE_ID);
  n.LTA.node_id = LTA_NODE_ID;
  for(int i=0;i<13;i++) n.LTA.spect[i] = 0; //initialize so we don't integrate on junk from the heap
}

void loop() {
  Watchdog.reset();
  //Watchdog.sleep(32000);
  spect.powerEnable(true);
  sensors_event_t temp, pressure, humidity;
  meas.getEvent(&pressure, &temp, &humidity);
  inclinometer.setFastReadMode(); //set fast read mode, 2000 Hz
  //Serial.print("X Tilt: ");
  n.LTA.tilt32_x = 0;
  n.LTA.tilt32_y = 0;
  n.LTA.tilt32_z = 0;
  n.LTA.tilt32_max_x = -32767;
  n.LTA.tilt32_max_y = -32767;
  n.LTA.tilt32_max_z = -32767;
  n.LTA.tilt32_min_x = 32767;
  n.LTA.tilt32_min_y = 32767;
  n.LTA.tilt32_min_z = 32767;
```

```
n.LTA.sysvolt      = 0.0;
//For next version of code, use struct to pack more data.
//Incorporate min and max detector, and 16-bit average.
//Sets the stage for use as a seismometer in continuous mode.
//Done, 111321.
//Serial.println(F("preloop"));
for(int i=0;i<2000;i++) {
  if(inclinometer.available()) {
  n.LTA.tilt32_x += (int32_t)inclinometer.sclData.AccX;
  if(inclinometer.sclData.AccX < n.LTA.tilt32_min_x) n.LTA.tilt32_min_x = (uint32_t)inclinometer.sclData.AccX;
  if(inclinometer.sclData.AccX > n.LTA.tilt32_max_x) n.LTA.tilt32_max_x = (uint32_t)inclinometer.sclData.AccX;
  n.LTA.tilt32_y += (int32_t)inclinometer.sclData.AccY;
  if(inclinometer.sclData.AccY < n.LTA.tilt32_min_y) n.LTA.tilt32_min_y = (uint32_t)inclinometer.sclData.AccY;
  if(inclinometer.sclData.AccY > n.LTA.tilt32_max_y) n.LTA.tilt32_max_y = (uint32_t)inclinometer.sclData.AccY;
  n.LTA.tilt32_z += (int32_t)inclinometer.sclData.AccZ;
  if(inclinometer.sclData.AccZ < n.LTA.tilt32_min_z) n.LTA.tilt32_min_z = (uint32_t)inclinometer.sclData.AccZ;
  if(inclinometer.sclData.AccZ > n.LTA.tilt32_max_z) n.LTA.tilt32_max_z = (uint32_t)inclinometer.sclData.AccZ;

  //Subject to change. Make function later so there is no ambiguity with pow(2., N.).
  //Also, the internal ADC in this mode is operating on freerun mode - CTRLB.FREERUN is probably set to 1.
  //Can decrease system load by async sampling, while the inclinometer is sampling.
  n.LTA.sysvolt  += (float)((analogRead(A0)*(1/0.7297297)*3.3)/pow(2.,12.));
  delayMicroseconds(120);
  }
}
//Serial.println(F("postloop"));
inclinometer.stopFastReadMode();
n.LTA.scltemp = inclinometer.getCalculatedTemperatureCelsius();
n.LTA.adt7410 = ts.readTempC();
//Serial.println(F(" SCL3300, ADT7410 temperature (C)"));
n.LTA.sysvolt /= 2000.0;
n.LTA.meas_pressure = pressure.pressure;
n.LTA.btmp_lower    = temp.temperature;
n.LTA.meas_humidity = humidity.relative_humidity;
n.AS_AGI(spect,1500,255,n.LTA.spect,2); //20 iterations
n.LTA.spect[12] = spect.detectFlickerHz();
```

```
spect.enableLED(true);
delay(10);
spect.enableLED(false);
rf95.setTxPower(10);
delay(25); //wait for the radio to wake up!
//Serial.println(F("complete data"));
/*
String discordSend = "";
discordSend = "";
  discordSend += millis(); discordSend += ",";
  discordSend += n.LTA.tilt32_x; discordSend += ",";
  discordSend += n.LTA.tilt32_y; discordSend += ",";
  discordSend += n.LTA.tilt32_z; discordSend += ",";
  discordSend += n.LTA.tilt32_max_x; discordSend += ",";
  discordSend += n.LTA.tilt32_min_x; discordSend += ",";
  discordSend += n.LTA.tilt32_max_y; discordSend += ",";
  discordSend += n.LTA.tilt32_min_x; discordSend += ",";
  discordSend += n.LTA.tilt32_max_z; discordSend += ",";
  discordSend += n.LTA.tilt32_min_z; discordSend += ",";
  discordSend += n.LTA.scltemp; discordSend += ",";
  discordSend += n.LTA.adt7410; discordSend += ",";
  discordSend += n.LTA.sysvolt; discordSend += ",";
  discordSend += n.LTA.meas_pressure; discordSend += ",";
  discordSend += n.LTA.btmp_lower; discordSend += ",";
  discordSend += n.LTA.spect[0]; discordSend += ",";
  discordSend += n.LTA.spect[1]; discordSend += ",";
  discordSend += n.LTA.spect[2]; discordSend += ",";
  discordSend += n.LTA.spect[3]; discordSend += ",";
  discordSend += n.LTA.spect[6]; discordSend += ",";
  discordSend += n.LTA.spect[7]; discordSend += ",";
  discordSend += n.LTA.spect[8]; discordSend += ",";
  discordSend += n.LTA.spect[9]; discordSend += ",";
  discordSend += n.LTA.spect[10]; discordSend += ",";
  discordSend += n.LTA.spect[11]; discordSend += ",";
  discordSend += n.LTA.spect[12];
  Serial.println(discordSend);
*/
rf95.send((uint8_t*)&(n.LTA),sizeof(n.LTA));
rf95.waitPacketSent(2000); //wait for the packet to be sent to prevent collision
//Serial.println(F("ping"));
//sleep sensors:
spect.powerEnable(false);
rf95.sleep();
inclinometer.powerDownMode();
//Serial.println(spect.getGain());
```

```
  digitalWrite(PIN_SERIAL1_TX, HIGH); //we are done here, turn off
  while(1); //await power down...
}
```

7. ADCS Development - Noise Characterization

```
#include <Arduino.h>
#include <ADIS16460.h> //Analog Devices library provided, has a lot of bugs and had to modify library to get it towork.
//#include <Adafruit_SleepyDog.h>

#define DTR   9u //data ready pin
#define CSR   7u //chip select pin
#define RST   2u //reset pin

ADIS16460 IMU; //IMU object
uint16_t  MSC,FLTR,DECR = 0; //register readback variables
//int16_t  *burstData; //dynamically allocated var for readback information
//int16_t burstChecksum = 0; //Checksum explicit storage
float AXS, AYS, AZS, GXS, GYS, GZS, TEMPS = 0.0; //IMU variables
uint16_t printCounter = 0; //microseconds, DO NOT USE FOR FLIGHT PURPOSES!
//variable will roll over after 71.6 minutes and result in undesirable operation!!!
//all variables MUST be initialized to 0, or else a NaN in a divide-by-zero error will occur.
int16_t GX, GY, GZ, AX, AY, AZ, XDANGL, YDANGL, ZDANGL, XDVEL, YDVEL, ZDVEL, TEMP = 0;
uint32_t samps = 0;

float ax_stat, ay_stat, az_stat = 0.0; //accumulator
float gx_stat, gy_stat, gz_stat = 0.0; //accumulator
//Correction/offset factor.
float AX_zero, AY_zero, AZ_zero = 0.0;
float GX_zero, GY_zero, GZ_zero = 0.0;
//Sample epoch, to start from zero milliseconds in CSV/DAT file.
uint32_t samp_epoch_ms = 0;

//if we have a problem or we want to restart the program (the CPU), reset.
//This code is functionally equivalent to pressing the hardware RESET button.
void restart(void) {
  //See the 1,200 page interface control document on ATSAMD21G for more information,
  //section on system control block AIRCR register bank.

  //Unfortunately, the only way to do this requires assembly language.
  __asm volatile ("cpsid␣i" ::: "memory"); //disable interrupt reporting
  __asm volatile ("dsb␣0xF":::"memory"); //commit
  SCB->AIRCR = ((0x5FAUL << SCB_AIRCR_VECTKEY_Pos)\
  | SCB_AIRCR_SYSRESETREQ_Msk); //write to system control block 0x5FA to reset
  __asm volatile ("dsb␣0xF":::"memory");
  for(;;) __asm volatile("nop"); //do nothing until system controller reads the SCB state to restart
}


void scaleData() {
  GXS = IMU.gyroScale(GX); //Scale X Gyro
  GYS = IMU.gyroScale(GY); //Scale Y Gyro
  GZS = IMU.gyroScale(GZ); //Scale Z Gyro
  AXS = IMU.accelScale(AX); //Scale X Accel
  AYS = IMU.accelScale(AY); //Scale Y Accel
  AZS = IMU.accelScale(AZ); //Scale Z Accel
  TEMPS = IMU.tempScale(TEMP); //Scale Temp Sensor
}

//interrupt service routine (ISR), fires on the instant when the imu is ready to readback
void grabData() {
  //we only need to read what is necessary (the 6-axes for rate and acceleration), but all the derived velocities and
  //angular positions are read out to showcase what this IMU is capable of (it is a true IMU, not just a rate gyro
  //with accelerometers.)
  GX = IMU.regRead(X_GYRO_OUT);
  GY = IMU.regRead(Y_GYRO_OUT);
  GZ = IMU.regRead(Z_GYRO_OUT);
  AX = IMU.regRead(X_ACCL_OUT);
  AY = IMU.regRead(Y_ACCL_OUT);
  AZ = IMU.regRead(Z_ACCL_OUT);
  //XDANGL = IMU.regRead(X_DELT_ANG);
  //YDANGL = IMU.regRead(Y_DELT_ANG);
  //ZDANGL = IMU.regRead(Z_DELT_ANG);
  //XDVEL = IMU.regRead(X_DELT_VEL);
  //YDVEL = IMU.regRead(Y_DELT_VEL);
  //ZDVEL = IMU.regRead(Z_DELT_VEL);
  //TEMP = IMU.regRead(TEMP_OUT); //NOT calibrated!
  scaleData(); // Scale data acquired from the IMU
  //Print ms from epoch(must be initialized):
  Serial.print(micros() - samp_epoch_ms);
  Serial.print(",");
  // Print scaled gyro data
  Serial.print(String(GXS-GX_zero,2)); Serial.print(",");
  gx_stat += GXS-GX_zero;
  Serial.print(String(GYS-GY_zero,2)); Serial.print(",");
  gy_stat += GYS-GX_zero;
  Serial.print(String(GZS-GZ_zero,2)); Serial.print(",");
  gz_stat += GZS-GZ_zero;
  // Print scaled accel data"
  Serial.print(String(AXS-AX_zero,3)); Serial.print(",");
  ax_stat += AXS-AX_zero;
  Serial.print(String(AYS-AY_zero,3)); Serial.print(",");
  ay_stat += AYS-AY_zero;
  Serial.print(String(AZS-AZ_zero,3)); Serial.print("\r\n");
  az_stat += AZS-AZ_zero;
  ++samps;
}

void setup() {
  //pinMode(RST,OUTPUT);
  //digitalWrite(RST, HIGH); //enable the IMU
  digitalWrite(LED_BUILTIN, LOW);
  Serial.begin(1000000); //max USB Serial speed, bit/s
  while(!Serial); //wait for serial port
  Serial.println(F("START"));
  //Watchdog.enable(6000); //We have a hardware watchdog to ensure if we freeze, we immediately reset
  //In the real world, we need something as critical as an IMU to recover from a frozen state as
  //quickly as possible.
  IMU.configSPI(); //set bit order and speed
  IMU.begin(CSR, DTR, RST);
  delay(1000);
  IMU.resetDUT(255);
  delay(1000);
  IMU.regWrite(MSC_CTRL, 0xC1); delay(20); //enable data ready, set polarity
  IMU.regWrite(FLTR_CTRL, 0x500); delay(20);
  IMU.regWrite(DEC_RATE, 1); delay(20); //disable decimation (discrete-time decimation)
  MSC  = IMU.regRead(MSC_CTRL);
  FLTR = IMU.regRead(FLTR_CTRL);
  DECR = IMU.regRead(DEC_RATE);
  Serial.print(F("MSC_CTRL: ")); Serial.println(MSC, HEX);
  Serial.print(F("FLTR_CTRL:␣")); Serial.println(FLTR, HEX);
  Serial.print(F("DEC_RATE:␣")); Serial.println(DECR, HEX);
  delay(2000);

  //manually run the interrupt to gather time-averaged data to assess if we need to run the autoleveler:
  #define AUTOLEVEL_SAMPS           1024 //samples
  //define a temporary float vector to collect integrated data (i.e. large numbers happen when integrating)
  float gyro_integrated[3]  = {0}; //initialize to zero to prevent NaN
  float accel_integrated[3] = {0};
  //int actual_samps = 0;
  //integrate for AUTOLEVEL_SAMPS
  for(int z=0;z<AUTOLEVEL_SAMPS;z++) {
  gyro_integrated[0]  += IMU.regRead(X_GYRO_OUT); //typecast to floating point from int
  gyro_integrated[1]  += IMU.regRead(Y_GYRO_OUT); //this is so we can decimate the sample next
  gyro_integrated[2]  += IMU.regRead(Z_GYRO_OUT);
  accel_integrated[0] += IMU.regRead(X_ACCL_OUT);
  accel_integrated[1] += IMU.regRead(Y_ACCL_OUT);
  accel_integrated[2] += IMU.regRead(Z_ACCL_OUT);
  //wait until the next interrupt is triggered:
  while(!digitalRead(DTR)); //wait for a state change signifying the new data is ready
  }

  //decimate by AUTOLEVEL_SAMPS to get time-averaged data:
  gyro_integrated[0]  = gyro_integrated[0] / AUTOLEVEL_SAMPS;
  gyro_integrated[1]  = gyro_integrated[1] / AUTOLEVEL_SAMPS;
  gyro_integrated[2]  = gyro_integrated[2] / AUTOLEVEL_SAMPS;
  accel_integrated[0] = accel_integrated[0] / AUTOLEVEL_SAMPS;
  accel_integrated[1] = accel_integrated[1] / AUTOLEVEL_SAMPS;
  accel_integrated[2] = accel_integrated[2] / AUTOLEVEL_SAMPS;

  //additional setup: initial autoleveling:
  #define AUTOLEVEL_GYRO_LIM        1.00 //deg
  #define AUTOLEVEL_ACCEL_LIM_X_Y   0.05 //G
  //if any of the conditions are exceeded:
  if((abs(gyro_integrated[0])  >    AUTOLEVEL_GYRO_LIM)       ||
   (abs(gyro_integrated[1])  >    AUTOLEVEL_GYRO_LIM)       ||
   (abs(gyro_integrated[2])  >    AUTOLEVEL_GYRO_LIM)       ||
   (abs(accel_integrated[0]) > (  AUTOLEVEL_ACCEL_LIM_X_Y)) ||
   (abs(accel_integrated[1]) > (  AUTOLEVEL_ACCEL_LIM_X_Y)) ||
   (abs(accel_integrated[2]) > (1-AUTOLEVEL_ACCEL_LIM_X_Y))) { //z-
  axis
  //perform autoleveling
  Serial.println(F("Level␣ground␣criteria␣not␣met,␣autoleveling...."));
  //we just did all the hard work, why waste time and sample again? assuming we are standing still,
  //(we DEFINITELY should), use the values we just got as the correction factor:
  AX_zero = accel_integrated[0] * 0.00025;
  Serial.print(F("AX_ZERO:␣")); Serial.print(AX_zero); Serial.print(F(","));
  AY_zero = accel_integrated[1] * 0.00025;
  Serial.print(F("AY_ZERO:␣")); Serial.print(AY_zero); Serial.print(F(","));
  AZ_zero = accel_integrated[2] * 0.00025;
  Serial.print(F("AZ_ZERO:␣")); Serial.print(AZ_zero); Serial.print(F(","));
  GX_zero = gyro_integrated[0]  * 0.005;
  Serial.print(F("GX_ZERO:␣")); Serial.print(GX_zero); Serial.print(F(","));
  GY_zero = gyro_integrated[1]  * 0.005;
  Serial.print(F("GY_ZERO:␣")); Serial.print(GY_zero); Serial.print(F(","));
  GZ_zero = gyro_integrated[2]  * 0.005;
  Serial.print(F("GZ_ZERO:␣")); Serial.println(GZ_zero);
  Serial.println(F("Done."));
  }
  else {
  Serial.println(F("Level␣ground␣criteria␣met.␣Proceeding."));
  }
  delay(1000);
  Serial.println(F("If␣you␣do␣not␣have␣a␣serial␣terminal␣logging␣program␣like␣PuTTY␣open,"));
  Serial.println(F("unplug␣the␣device␣now␣and␣start␣a␣PuTTY␣session␣with␣logging."));
  Serial.println(F("Please␣see␣https://cdn.discordapp.com/attachments/677058803681722389/1049778607477751848/image.p
␣for␣an␣example."));

  //we are finished initializing.
  //attachInterrupt(DTR, grabData, RISING); //specifies the ISR (interrupt service routine) to get the IMU data for that frame

}

#define BFL1     0x500 //1 tap
```

```
#define BFL2     0x501 //2 taps
#define BFL3     0x502 //4 taps
int filtctrl = 0x500;


void loop() {
  //make a user interface to streamline collection
  //Use a very simple method: one letter command. Should be machine and human friendly.
  if(Serial.available()) { //if user has input something into serial port,
  detachInterrupt(DTR); //DO NOT PROCESS IMU DATA IF SOMETHING IS AVAILABLE FROM USER!!
  char c = Serial.read(); //read a singular character from the ring buffer to operate on it
  switch(c) { //decide between all the possible inputs:
    case 'r': //"reset"
    Serial.println(F("Restarting␣device.␣You␣will␣need␣to␣reselect␣the␣serial␣port"));
    Serial.println(F("to␣restart␣this␣UI."));
    delay(1000);
    restart();
    break; //for the formality so compiler doesn't throw a warning.
    case 'a': //"activate", start collection
    Serial.println(F("starting␣collection..."));
    Serial.println(F("=============================================="));
    Serial.println(F("TIME(ms),XGYRO,YGYRO,ZGYRO,XACCEL,YACCEL,ZACCEL"));
    samp_epoch_ms = micros(); //advance counter to start time
    gx_stat = 0.0; gy_stat = 0.0; gz_stat = 0.0;
    ax_stat = 0.0; ay_stat = 0.0; az_stat = 0.0;
    attachInterrupt(DTR, grabData, RISING);
    break;
    case 's': //"stop" stop collection
    Serial.println(F("stopped␣collection."));
    //decimate by samples taken:
    ax_stat /= samps;
    ay_stat /= samps;
    az_stat /= samps;
    gx_stat /= samps;
    gy_stat /= samps;
    gz_stat /= samps;
    //print statistics:
    Serial.print(F("ax␣avg:␣")); Serial.println(ax_stat);
    Serial.print(F("ay␣avg:␣")); Serial.println(ay_stat);
    Serial.print(F("az␣avg:␣")); Serial.println(az_stat);
    Serial.print(F("gx␣avg:␣")); Serial.println(gx_stat);
    Serial.print(F("gy␣avg:␣")); Serial.println(gy_stat);
    Serial.print(F("gz␣avg:␣")); Serial.println(gz_stat);
    break;
    case '1': //filter control, tap #
    IMU.regWrite(FLTR_CTRL,BFL1); delay(20);
    break;
    case '2': //filter control, tap #
    IMU.regWrite(FLTR_CTRL,BFL2); delay(20);
    break;
    case '4': //filter control, tap #
    IMU.regWrite(FLTR_CTRL,BFL3); delay(20);
    break;
    default:
    break;
  }
  }

  //detachInterrupt(2);

  //attachInterrupt(DTR, grabData, RISING);
}
```

8. SEEDS-A Assembly and Deployment Manual

INSTRUCTION MANUAL FOR ASSEMBLY AND DEPLOYMENT OF NETWORK

ADAPTED FROM

APOLLO 15-17 ALSEP-MT-03, SECTION 4

8.1 Instrument Description

Table 8.1: Gateway

Item	Description
Node Enclosure (Box 1)	Grey polycarbonate case with TTL Serial JPEG Camera affixed to top. Contents include: StratoPi, 12V, and 3.3V converter Command Data and Handling (CDH) brick, Li-Po battery, battery charger
Solid State Drive (Box 1)	1 TB Samsung solid state drive
30 W solar panel + Connectors (Box 1)	Renogy 30W 12V Monocrystalline Solar Panel for Node and Cable
BGAN + Cable (Box 1)	Inmarsat compact unit comprising of transceiver and antenna
Wifi antenna (Box 1)	Black 915 MHz Whip Tilt Antenna
Mast (Box 3)	Steel Tripod Assembly for Gateway
Wind Sonic & Cable (Box 4)	Black polycarbonate exterior, 2-axis ultrasonic wind sensor and cable connector
Sensor Head Ring + Cable (Box 4)	3D-printed PLA 3-Tiered Ring
Grounding cable Kit (Box 2)	50'Guy Wire, 3 S Hooks, Guy Wire Ring, 6 Clamps

Table 8.2: Node

Item	Description
Node Enclosure (Box 2)	Grey polycarbonate case with pod bricks (3) affixed to the top. Contents include: 12V and 3.3V converter, CDH brick, Li-Po battery, battery charger.
10 W solar panel + Connectors (Box 2)	Renogy 10W 12V Monocrystalline Solar Panel with connectors
Wifi antenna (Box 1)	Black 915 MHz Whip Tilt Antenna
Spare Hardware	Spare washers and mounting stakes for node

8.2 Post-Arrival Conditions and Deployment Risks.

Conditions at Cerro Aguas Calientes and Laguna Lejía impose constraints on the hardware and deployment crew.

The altitude of Laguna Lejía and base camp is 4,100 meters, which, after adequate acclimation, should present few risks. Due to this, deployment of Node 042 can occur at any time during the day that is convenient for the expedition leader and allows for up to two hours of anticipated total deployment time.

The constraints of Cerro Aguas Calientes are severe, and present a high risk. The altitude of the summit is 5,900 meters, and presents a large contribution to risk factor. Any crew summiting Aguas Calientes shall acclimate at the altitude of Laguna Lejía for at least one week, or at least three days with 125 mg acetazolamide. The physical condition of the summit crew shall be evaluated a day before attempting a summit. All crew must take 125 mg acetazolamide before summiting; failure to do so will result in a substantially higher risk of nausea, fatigue, or death.

Aguas Calientes is a stratovolcano, and due to geologically recent eruptions from itself and the neighboring Lascar volcano, loose igneous rocks on slopes of up to 45 degrees make up the terrain. This presents a risk in terms of safe navigability.

During the day and during high sun angles, extreme UV exposure will occur. All crew must cover as much skin as possible, especially the face and eyes.

Due to the need for on-site modifications and firmware updates, a crew member with significant firmware engineering experience and complete familiarity with the system internals is needed. This member shall rehearse as many expected scenarios as possible and familiarize themselves with this deployment manual before deployment to the Atacama Desert as the highly rarefied atmosphere at Cerro Aguas Calientes will result in significant mental and visual impairment.

Network deployment-related events that take place after adequate acclimation are presented here in the order in which they will be accomplished by the crew:

A. Survey deployment sites to allow for secure equipment mounting to the ground,

B. Verify equipment layout so as to allow for maximum exposure to solar panels,

C. Deploy equipment in surveyed locations.

8.2.1 Tools used in Deployment.

Table 8.3: Gateway

Tool	**Purpose**
Allen Key Set	BGAN Mount
1 kg Hammer	to hammer stakes into the ground
6 helical A2 Stainless Stakes	To firmly mount antenna tripod to ground composed of small igneous rock
24 Straight A2 Stainless Stakes with Loops	To mount pod, node, and solar panels to ground
1 Key Switch Key	To turn on units
Philips and Flathead Screwdriver Set	To tighten internal power blocks and other components if/as necessary
Soldering Kit	To repair or replace wiring to node sensor head, and to repair electronics in the field if necessary
Portable Battery with AC Inverter	To power soldering iron while in the field
Foldable Standard-Issue Army Shovel	To dig out mounting areas as necessary
22AWG Solid-Core Wire	To repair electronics as necessary

8.2.2 Repair Procedures.

This procedure assumes that the node head is fully tested and expedition qualified. If an RJ-11 terminated cable is not included with this node sensor head, please consult the expedition firmware engineer for assistance in completing this repair in the field. Additionally, **this procedure must be performed at the base camp and NOT at the summit. Physiological limitations prevent long-duration stays at the summit.**

1. Identify the node sensor head by looking for a white, 3D-printed cylinder with several electronic components in it. Leave this sensor head in its antistatic packaging for now.

2. Find the tripod with the attached wind sensor. Clipped to the top of it is the two-part sensor package. Using a small flathead screwdriver, wedge the screwdriver between the sensor head cap and the mounting base. Work the screwdriver around the gap until the sensor head cap comes off.

3. Unplug the internal RJ-11 connector and place the sensor head cap into the Nanuk hard case.

4. Take the replacement sensor head cap and remove the antistatic packaging.

5. Take the roll of gum tape and carefully unroll 600 mm inside the hard case, which is used to shield the tape from gusts of wind that might carry dust.

6. Stick the end of the tape to the case and use scissors to cut the tape. It might take a few attempts to cut the tape. 7. Carefully take the stretched out tape to the sensor head base and wrap it around the circumference of the base. Apply pressure around the entire tape to secure it.

8. Using the blade of the scissor, peel off the backing slowly from a corner. Once a large enough surface is peeled with the scissors, use fingers to slowly peel the wax paper.

9. Once it is fully peeled off, take the scissors and cut the excess tape from the upper lip of the sensor head base.

10. Take the new sensor head and plug the RJ-11 terminated cable into the AI04 jack. Be sure this is seated firmly before proceeding.

11. Seat the sensor head on the base. Slowly rotate while applying firm downward pressure on the sensor head.

12. When the sensor head cap cannot go down anymore even after applying a modest amount of pressure, stop. Take the scissors to pry off any excess gum around the circumference of the sensor head.

8.2.3 Site survey.

The terrain comprises porous igneous rock, which may largely be aggregate bigger than pebbles especially at the summit. This will significantly hinder the ability to mount the equipment to the ground securely.

At the summit of Cerro Simba, two crew members shall walk the entire crater rim, carefully noting the aggregate size and being especially mindful of the steep slopes. To aid in choosing a site, one member shall take pictures of a candidate location, with the lake in the background as a radial reference. At each candidate location, a crew member shall use a shovel to qualitatively judge the ease of digging. The most suitable candidate locations shall be marked with a small red survey flag, so that the crew can come back to the location to deploy the node.

8.2.4 Deployment procedures.

These procedures assume that Boxes 1-4 are fully unpacked.

8.2.4.1 LAKE LEJIA POD.

1. Open the Nanuk hard case and remove *Unit 042*, which is the larger light gray polycarbonate case with two metal latches. It does not have a fisheye lens protruding from the center of the case.

2. Remove eight 300 mm A2 stainless stakes.

3. Remove one 10-watt solar panel, which is labeled “Renogy” on the rear of the unit. The black anodized aluminum stands should already be fastened to the solar panel unit, and the 3-meter red and black flexible wire pair should already be fastened.

a. If flexible wire pair is *not* attached, find the rectangular end of the cable, and find the rectangular connector on the solar panel. Observe the connector polarity and plug the connectors in.

4. Using predetermined sunrise and sunset bearings, take the solar panel, four A2 stainless stakes, and a 1 kg hammer to the site chosen from 1.3 Site Survey. The solar panel azimuth must be aligned to the median of sunrise and sunset, and the elevation of the solar panel must be elevated 30 degrees from the horizon to allow for maximum solar exposure throughout the entire year.

a. Once aligned, take a stake and place it in the hole on the L-bracket of the lower left leg of the solar panel, marked “1”. Taking care to not strike the solar panel surface, gently hammer the stake into the ground until the loop of the stake reaches the surface of the L-bracket.

b. Repeat the previous step going counterclockwise until the solar panel is firmly secured to the ground.

5. Take the flexible cable pair and string it to the pod site chosen.

6. Take four stakes, the node, and a hammer to the pod site. Level the soil so that the node unit will be parallel with the horizon. Ensure that there are no large rocks below the mounting holes in the L-brackets.

a. Once the pod is placed and leveled, start at the lower-left corner L-bracket marked “1”. Place the stake into the L-bracket hole. Taking the hammer and taking care not to strike the pod box, gently hammer the stake into the ground until the loop of the stake reaches the surface of the L-bracket.

b. Proceed counterclockwise in the same manner until the node is firmly secured to the ground.

7. With the solar panel and pod secured to the ground, uncap the round plug on the pod and uncap the cable. Put the caps in the assembler's pocket, and plug the cable into the pod.

8. Open the node by unlatching both steel latches, and take the key taped to the lower-left corner wall. Replace the tape and close the pod. Insert the key into the key switch and turn it clockwise 90 degrees. This will turn the pod on. Remove the key and place it into the assembler's pocket.

9. Make sure the antenna is perpendicular to the ground before leaving the site.

8.2.4.2 SUMMIT PODS AND NODE.

Assembly team: PLEASE review this section several times before summiting; the procedure should be memorized and all should be familiar with all components of the system.

- NODE:

1. Near the chosen site for the node, find a relatively flat location for equipment staging. Open the Nanuk hard case and remove the Node server unit, which can be identified as the white polycarbonate case with a fisheye lens in the middle of the lid.

2. Remove 8 straight spikes and 6 helical spikes from the hard case.

3. Take a smartphone and open a compass app. Calibrate the magnetometer and set the phone inside the hard case. This will allow the phone to obtain a GPS lock to identify the true North correction.

4. Take the gold-colored tripod containing the attached Explorer 540 modem and Windsonic wind sensor and move it to the chosen site. If it is very windy, carefully lay the tripod down, ensuring that the wind sensor does not come into

contact with the ground unless it is still wrapped in bubble wrap. If it is not windy, proceed to the next step.

5. Begin deploying the tripod by pulling the legs apart. The tripod is fully deployed when the beams attached to each leg and the center collar form a flat plane perpendicular to the tripod mast.

6. If the bubble wrap has not been removed from the Windsonic wind sensor, remove it now and place it in the Nanuk hard case. On the Windsonic, identify the red triangle underneath the lip of the circular reflector. This indicates true North. Have a crew member retrieve the smartphone from the hard case to find true North. Now rotate the tripod assembly until the red triangle is facing true North.

7. With the tripod correctly aligned, the tripod must now be levelled and secured. In the intended tripod leg locations, dig out rocks underneath each leg at up to 15 cm below the surface. Place the tripod into the holes. If the tripod is not level, remove rocks from each hole until the tripod is level.

7.1. Take one stainless helical spike and insert it into the mounting hole. While a crew member holds the tripod, rotate the spike until the end of the spike is reached. Repeat this process until all spikes are secured. If more leverage is necessary due to tough soil, take the large Vise-Grip pliers and attach it to the spike to use as a lever. Apply downward force while rotating.

7.2. Replace the rocks into the holes to cover the tripod legs and spikes.

8. Because the Inmarsat 4 geosynchronous satellite is almost directly above the Atacama, pointing is very straightforward. Using an M5 Allen key, loosen the nuts on the elevation plate until the antenna is loose. Point the antenna as far up as allowable and secure the elevation plate.

9. Take a break for at least 10 minutes before continuing assembly. This will allow you to regain focus if necessary. Ensure you are not feeling drowsy and/or

nauseous; if you feel any of these now or at any time during the installation process, notify your crew members immediately so a portable compression chamber can be deployed.

10. Take the node server unit and place it 60 cm away from the center of the mast, in between two of the legs opposite the Inmarsat antenna. The cable ports must be facing the mast.

11. A cabling fit check is now necessary. Ensure the cable bundle from the Inmarsat antenna and white sensor head will reach the server unit with some slack. The cable bend radius on the Inmarsat modem shall not be smaller than 15 cm; if it is smaller than this, there is not enough slack. Move the server unit closer to the mast. Do NOT plug anything in during this step.

12. Take the gray, stiff cable labeled “Ethernet” and remove the rubber connector guard. Plug the cable into the jack labeled “Ethernet”.

13. Take the white cable labeled “Sensor Head” and remove the rubber connector guard. Plug the cable into the jack labeled “Sensor Head”.

14. Retrieve the large, 30W solar panel. Do not drag the power cable on the ground. Using predetermined sunrise and sunset bearings and the predetermined solar panel location, align the solar panel to the median of sunrise and sunset. Use a compass app as necessary. Adjust the elevation of the solar panel by moving the legs so that the elevation is 30 degrees from the horizon. This will allow for maximum solar exposure throughout the entire year.

14.1. Once aligned, take a stake and place it in the hole on the L-bracket of the lower left leg of the solar panel, marked “1”. Taking care to not strike the solar panel surface, gently hammer the stake into the ground until the loop of the stake reaches the surface of the L-bracket.

14.2. Repeat the previous step going counterclockwise until the solar panel is firmly secured to the ground.

14.3. Stretch the solar panel power cable to the gateway server. Find the final remaining plug labelled “Solar Input” and plug the cable in.

15. Using the key from the Lake Lejia pod, insert it into the key switch and rotate 90 degrees. Remove the key. This will turn the system on. Do not perform post-deployment system checks until at least one hour has passed - this will allow the system to partly recharge the batteries.

16. The setup is now complete. Ensure all components of the system are securely fastened to the ground.

- SUMMIT PODS:

Assembler notes: These are the "Block 1.5" pods, assembled post-delivery.

1. Open the Nanuk hard case and remove the small gray polycarbonate boxes. Each has ABS bricks mounted on the top of their covers.

2. Remove eight stakes, solar panels, hammer, and shovel. Do not drag the solar panel power cables on the ground.

3. While the site location for these pods are determined at the mission director’s discretion, locations should be selected that allow maximum solar exposure, especially if deeper in the crater. This entails placing them on the sunward side of the crater. Take each pod set to their respective chosen locations.

4. Since the crater walls are very steep and the site is prone to earthquakes, some preparatory work may be necessary. For pod(s) placed on the steepest portions of the crater wall, a method needed to protect the pod from rock slides will need to be developed. Using a shovel, flatten out an area large enough to fit the node, where it is level with the horizon. Do not throw the soil downslope.

5. Place the pod into the flattened area, and fill the pod in up to the surface of the polycarbonate lid with the dirt that was placed aside.

6. Take the solar panel and identify the small black box on the rear of the solar panel. This is the junction box. Place the solar panel on the ground, with the short end down on top of the base of the pod, and the junction box furthest away from the ground. The solar panel should have a 45 degree angle relative to the horizon; adjust the angle of the solar panel using the end on the ground as a pivot point and the solar panel legs to adjust the angle. Make sure the mounting brackets on the base and legs do not get buried.

6.1. This setup is intended to shield small rocks and gravel from sliding onto the surface of the sensors in the event of an earthquake or nearby volcanic eruption. If in the unlikely case that the crater slope is greater than 45 degrees, a keep-out zone of shallower slope will need to be made. Dig out an area behind the solar panel of at most a 22 degree slope of 1 meter length from the base of the solar panel. **Proceed slowly**, as one will be easily fatigued at this altitude. If fatigued and/or nauseous, notify crew immediately.

7. With the ground prepared, take a stake and insert it into the L-bracket hole. Hammer the stake into the ground until the loop is reached. Be careful not to hit the solar panel while hammering. Repeat for all stakes until the solar panel is firmly staked to the ground.

8. To allow rocks to flow around the solar panel and node in a rock slide, fill rocks completely underneath the solar panel. **DO NOT** tamp down the rocks, as it may loosen the stakes from the ground.

9. For node(s) that are placed near the shore of the crater lake, the same procedure can be applied. Note that the lake level changes due to snow and evaporation; node(s) should be placed far away enough that the lake level will not reach the node equipment.

10. Take the same key that was used to power on the node and Lake Lejia pod

and turn each pod on. This completes the installation.

8.2.4.3 Post-deployment system Check

This procedure in the field will nesure the system is fully operations and is taking in-situ data. Communication with mission control (MC) at SETI and San Jose State University is required.

1. When ready, contact the base camp via walkie-talkie to send a text message via satellite phone or shortwave-band digital to MC to enable the BGAN modem remotely.

2. After no more than 15 minutes of sending an "on" command to the modem via Inmarsat, the yellow status light should turn solid on. Communicate this event back to MC.

3. MC should text back "VPN GREEN" within 10 to 20 minutes of the modem powering on. When this occurs, wait for MC to text back a full data string to the base camp from all pods and the node.

4. At the base camp, read the data string. **If there are any instances of 65535.00 or any negative values that are not temperature, notify the firmware engineer at the summit IMMEDIATELY.**

5. If the data string test has passed, the system is fully operational. Text "STATUS 0" to MC to indicate completion of test.

6. Ensure one last time that all systems are fastened on the mountain securely.