Tartan Artibeus: A Batteryless, Computational Satellite Research Platform

Bradley Denby* Emily Ruppel* Vaibhav Singh Shize Che Chad Taylor Fayyaz Zaidi
Swarun Kumar Zac Manchester Brandon Lucia
{bdenby, eruppel, vaibhav3, sche, cataylor, fhz, swarunk, zmanches, blucia}@andrew.cmu.edu
Carnegie Mellon University

Abstract

Tartan Artibeus (TA1) is the first batteryless, computational pocketqube satellite; its open-source hardware and software launched into low-Earth orbit (LEO) in January 2022. TA1 is a 1p (125 cm^3) pocketqube built around the Tartan Artibeus Bus (TAB), which connects independently-designed modules into a batteryless, computational satellite. In TA1, TAB incorporates an electrical power supply (EPS) module that harvests solar energy into a supercapacitor, a fault-tolerant command and data handling (C&DH) module, a radio-communication module, and a configurable computational payload module.

The open-source hardware [17] and software [18] of TA1 supports independently designed modules oblivious to the batteryless nature of the power system via adherence to TAB's well-defined communication protocol serviced by a C&DH board. TAB allows the C&DH board to manage independent subsystems for power savings and to provide isolation for reduced impact of faults. The C&DH software supports frequent power cycles via task-based, intermittent execution. These features guarantee forward program progress and free subsystem developers to focus on each payload application.

To evaluate the computational nanosatellite design TAB enables, we integrate many subsystems, including a radio module, a GNSS module, and a computing payload. The radio, which is based on OpenLST [33] hardware and software, demonstrates the ease of using existing modules with TAB. To the best of our knowledge, TA1 is the first 1p pocketqube to have a GPS module without COCOM limits. The computing payload includes hardware to accelerate machine inference and can be reprogrammed in orbit. The TA1 mechanical, hardware, and software designs are open source to reduce the barrier to entry for orbital edge computing (OEC) research.

1 Introduction

Large, expensive, monolithic satellites dominate low-Earth orbit (LEO) Earth observation. Space vehicles (SVs) like WorldView-3 [22], Earth-Observing 1 [50], and Landsat-8 [39] cost hundreds of millions of dollars each [26] and require over a dozen years of "arduous" development [39]. To justify such time and expense, these SVs must operate for decades. Extremely high cost demands extremely low risk; as a result, designers often select satellite subsystems for their "flight heritage" and not for their cutting-edge capabilities. By the end of a satellite mission, some components may be more than a quarter century behind the state-of-the-art.

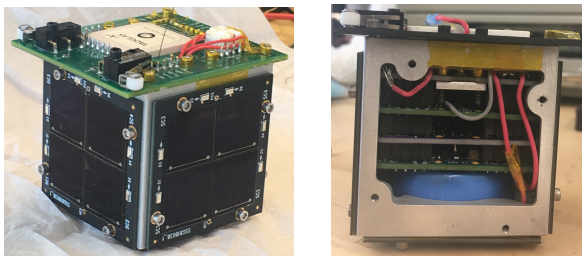
Monolithic, expensive satellites usually take this approach. For these satellites, a *ground segment* closely manages operations via a *bent pipe* [35]: ground stations transmit commands to the satellite, and the satellite responds with raw, unprocessed sensor data. Innovation often concentrates around the sensor payload of these monolithic, expensive satellites. Generally, these systems perform minimal onboard computing [1,7,11] and instead focus on reliable remote control from the ground.

Recently, there has been a proliferation of LEO launches with *nanosatellites*. A typical nanosatellite is four orders of magnitude cheaper, three orders of magnitude less massive, and four orders of magnitude smaller than a typical, monolithic satellite. Rather than operate for decades, a nanosatellite operates for a few years at most or a few weeks at least. Lower costs and shorter missions reduce per-device risk and support use of cutting-edge, commercial, off-the-shelf (COTS) hardware. If hardware lacking flight heritage fails, the nanosatellite can be quickly replaced.

Higher per-device risk tolerance of nanosatellites provides opportunities to deploy more advanced subsystems to orbit compared to expensive, monolithic satellites. However, most nanosatellites still adhere to the same concept of operations (CONOPS) as monolithic satellites; i.e., a bent pipe [20]. New Earth-observation capabilities enabled by large constellations — e.g., daily global coverage — are limited by continued adherence to a bent-pipe CONOPS [37]. Additionally, new challenges — e.g. effectively managing a large constellation by remote control [37] — arise in a more crowded LEO.

Recent work observes, enumerates, and characterizes some

*Both authors contributed equally to this work



(a) **TA1 Assembled.** (b) **TA1 Board Stack.**

Figure 1: **TA1 Satellite.** TA1's modular stacking design (visible at right in a prototype) easily integrates new payloads.

of these challenges [15, 16, 40]. For example, a proliferated LEO exacerbates the *downlink bottleneck*: Earth-observation satellites and constellations observe much more data than can be downlinked per orbit revolution. These challenges are addressed by *orbital edge computing* (OEC): colocating computing resources with sensors on orbit to process data before transmission. OEC makes better use of limited communication opportunities by pre-processing and selecting valuable data for transmission. However, this technique generally requires significant energy for computation on the satellite.

Thus, the volume and surface area constraints of nanosatellites make OEC particularly challenging to deploy. Limited surface area and low-cost requirements prevent nanosatellites from leveraging over-provisioned solar panels. Limited volume constrains the amount of energy that can be stored inside the device (i.e., the capacity of the *energy buffer*). In this work, we import techniques of *intermittent computing* from cutting-edge computer systems research to reconcile the benefits of OEC with the constraints of a nanosatellite platform. We buffer energy with a supercapacitor instead of a battery — a choice that is further motivated in Section 2.2.

Because evaluating all proposed OEC schemes in orbit is not practical, previous work developed the `cote` software library to assess proposals in simulation [16]. TA1 integrates with `cote` and any other space simulation environment via TAB to provide a hardware-in-the-loop evaluation platform. In this work, we present a low-cost, open-source [17, 18] satellite to augment simulated OEC evaluation with both hardware-in-the-loop (before launch) and in orbit (after launch).

We present Tartan Artibeus (TA1), an open-source hardware [17] and software [18] pocketqube satellite for OEC research. Figure 1 provides exterior and interior views of TA1. The TA1 command and data handling (C&DH) module serves as an arbiter between standard subsystem modules and the *batteryless* electrical power system (EPS), which harvests energy into a supercapacitor. To abstract the batteryless, intermittent operation from other subsystems, TA1 introduces the Tartan Artibeus Bus (TAB): a framework for integrating independently-developed satellite subsystem modules. TAB extends the open-source OpenLST [33] serial communication protocol to support OEC operations. The C&DH module uses TAB to operate satellite subsystems oblivious to the batteryless, intermittent nature of the satellite. Further, TAB supports integration with external simulation tools like `cote` for hardware-in-the-loop evaluation. TA1 leverages TAB to provide an open source platform for OEC research.

In summary, Tartan Artibeus provides the following contributions to computational space systems research:

- TA1 complements software-based OEC simulation with a batteryless, hardware platform for real-world evaluation.
- TAB provides a communication framework for easy integration of flight hardware with both software simulation environments and new hardware research modules (e.g. experimental payloads).
- TA1 is an open-source hardware [17] and software [18] reference implementation of a flight-ready picosatellite for in-situ OEC evaluation.
- TA1 software includes three reference apps that exercise TAB and provide a model for OEC research CONOPS.

2 Background

LEO satellite systems proliferate as new launch services [24] provide increasing access to space. Nanosatellite deployments in particular exhibit rapid growth [62] since the standardization of the "cubesat" form factor [49]. Satellites continue to reduce in size with the recent proposal of the "pocketqube" standard [52] and deployments of "chipsats" [70]. We provide an overview of existing LEO nanosatellites, opportunities to leverage batteryless and intermittent computing in space, and the emerging field of orbital edge computing (OEC).

2.1 Existing Nanosatellites

Smaller satellite sizes and costs support deployments of large numbers of devices to LEO. For the first time, satellite constellations consist of hundreds of devices [10, 37] instead of dozens. Now, some of the largest LEO constellations consist almost entirely of nanosatellites. A *nanosatellite* masses between 1 kg and 10 kg. Often, a nanosatellite adheres to the cubesat standard [49] to leverage COTS components and launch services. Momentum toward even smaller designs, like the pocketqube standard [52], promise even lower unit costs.

While a cubesat consists of integer multiples of 10 cm×10 cm×10 cm ("1U") volumes, and each 1U volume must mass no more than 1.33 kg, a pocketqube consists of integer multiples of 5 cm×5 cm×5 cm ("1p") volumes. Each 1p volume of a pocketqube must mass no more than 0.25 kg (i.e., 250 g). Thus, even a 3p pocketqube cannot meet the 1 kg threshold to be considered a nanosatellite and might more accurately be described as a *picosatellite*. Satellites smaller than a picosatellite — i.e., a *chipsat* — may be even cheaper to

launch in high quantity, but they do not yet enjoy the benefits of standardized COTS components and launch services.

For decades, universities and other education-focused organizations launched and operated the majority of LEO nanosatellites. More recently, multiple business organizations [10, 37] have deployed large nanosatellite constellations for commercial Earth-observation. These constellations enable new geospatial applications — e.g., observing the entire Earth in a single day [37]. However, effectively managing satellites by remote control — i.e., via a bent-pipe — grows increasingly difficult with larger constellation populations [37].

The smaller size of emerging satellites creates new challenges. Nanosatellites face exacerbated downlink bottlenecks [16], limited data quality due to volume constraints [15], and less harvested power [40]. In contrast, monolithic satellites enjoy large engineering margins for power. These systems may harvest more than 3 kW via solar arrays. A 3U nanosatellite with low-risk, body-mounted solar panels harvests less than 10 W of power — three orders of magnitude less than a large SV. Picosatellites and chipsats face more stringent power budgets. As satellites become smaller, their design must address increasingly strict operating constraints.

2.2 Batteryless and Intermittent Computing: Failure *is* an Option

As nanosatellites reduce in size and mass, multiple challenges emerge for supporting batteries. First, batteries begin to occupy an outsized fraction of the SV body. Second, as thermal mass declines, the EPS experiences wider temperature swings that may damage rechargeable batteries [34]. At low temperatures energy and power delivery reduce [47], and at high temperatures thermal runaway is a risk [6]. Finally, the EPS must carefully manage battery depth of discharge to guarantee operation over deployments spanning even just three to five years [9]. Supercapacitors avoid these concerns with higher power density [32], wider operating temperature ranges [31], and higher cyclability [71] with the primary concession that supercapacitors exhibit lower energy density (and total energy storage capacity) than batteries of the same volume. TA1 therefore introduces a *batteryless* nanosatellite design that buffers harvested energy in one or more supercapacitors.

Smaller satellites also limit the surface area of low-risk, body-mounted solar panels, constraining access to harvested power. A batteryless nanosatellite draws power stored in its energy buffer when operating power exceeds harvested power. When satellite subsystems deplete the energy buffer, power *fails* and the SV must power down while the buffer refills. Power failures typically spell disaster for a satellite, but batteryless satellites continue execution across power failures using techniques from the field of *intermittent computing* [41]. Intermittent computing strategies preserve program state by strategically writing to non-volatile memory preserved across power failures [4, 5, 12, 27, 30, 42–44, 53, 55, 59–61].

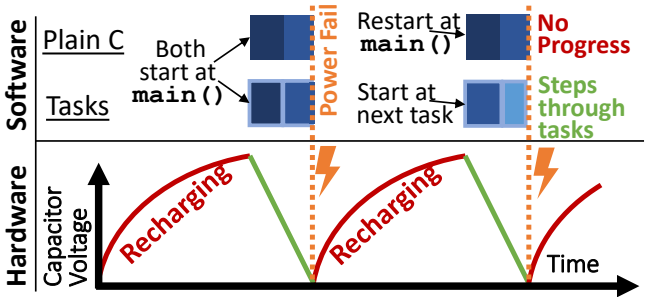


Figure 2: **Intermittent Execution.** Hardware turns on and off as the energy buffer drains and refills. These power failures interrupt software execution. In C-language programs, such power failures prevent forward program progress. Task-based code preserves progress after each task completes.

Figure 2 illustrates the operating states of hardware and software as a device executes intermittently. After refilling the energy buffer, the device turns on and begins running code until the buffer depletes and the device turns off to await the next operating period. Figure 2, top, demonstrates the problem of running unmodified code on an intermittently-powered device. After a power failure, execution restarts from the beginning of the program and all progress is lost. In contrast, programmers may use intermittent *tasks* to decompose a program into a sequence of atomic code regions to preserve progress at task boundaries [12, 27, 41–43, 68]. On reboot, the program begins at the start of the failed task. One challenge under an intermittent execution model involves extended periods with no power because little or no energy can be harvested (e.g. in eclipse). Programmers may extend operation in eclipse by reducing the frequency and power of expensive operations [45].

By framing nanosatellite operation as intermittent, developers ensure a more resilient and lower-risk design. Many tools and programming models exist to scale with capacitor size [5, 12, 13, 30, 43, 44]. Intermittent tasks are robust to reboots, regardless of the cause. Tasks are idempotent by design, so re-executing a task due to errors like a watchdog timeout allows the program to continue operating and only re-run the failed task. Further, tasks allow programs to tolerate power system degradation because the entire program need not complete at once for the mission to succeed. One can effectively derate for capacitor aging (i.e. reduced capacitance over time) by breaking code into smaller tasks. An intermittent execution model allows system developers to aggressively increase computing load since the energy buffer will recharge and execution will resume if a power failure occurs.

2.3 Orbital Edge Computing

To address the exacerbation of the downlink bottleneck due to increasing constellation populations, orbital edge computing colocates computational hardware with sensors in satellites

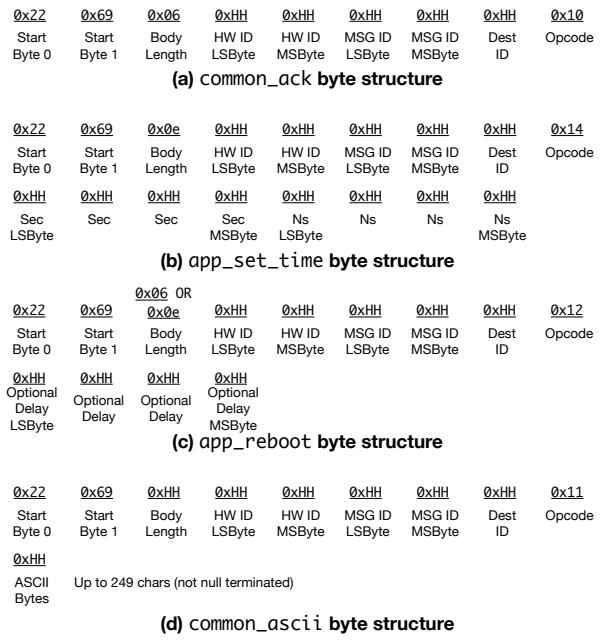


Figure 3: (a) The byte structure of the `common_ack` TAB command. The header bytes compose the entire command. (b) The byte structure of the `app_set_time` command with both header bytes and payload bytes. (c) The byte structure of the `app_reboot` command, which includes optional payload bytes. (d) The byte structure of the `common_ascii` command, which contains a variable number of payload bytes.

to process data at the edge [15, 16, 40]. While monolithic satellites eschew such hardware to avoid increased risk of failure, nanosatellites are free to use more recent computational devices. For example, the lower cost and more frequent replacement rate of nanosatellites reduce the risk of deploying computational hardware that is merely "radiation tolerant" instead of "radiation hardened."

An OEC satellite collects data, *processes data at the orbital edge*, and uses the results to intelligently transmit information to the ground. For example, many Earth-observation satellites collect image data. These satellites capture a sequence of images along the satellite's *ground track*. Each image in this sequence is a *ground track frame* (GTF), often consisting of a large geographic region. Under a bent pipe, a satellite attempts to downlink as many GTFs to the ground segment as the limited communication opportunities support. These frames are then split into hundreds or thousands of smaller *tiles* (each consisting of a smaller geographic region) for analysis. An OEC satellite instead processes tiles at the orbital edge, supporting more intelligent use of the limited downlink.

To evaluate the efficacy of OEC proposals, our prior work developed the `cote` simulation environment [16]. This software tool models satellite orbital mechanics, rotation of the Earth and ground stations, and satellite subsystem characteristics such as harvested and stored energy, data collection, computation, communication, and radio bitrates. Researchers used this tool to propose and evaluate the *computational nanosatellite* and *computational nanosatellite pipelines* (CNPs) [16]. A CNP distributes computational tasks across computational nanosatellites in a constellation. With a sufficient number of satellites, a CNP completes processing of all tiles in a frame before the *ground track frame period* (GTFP): the time between observation of new ground track frames.

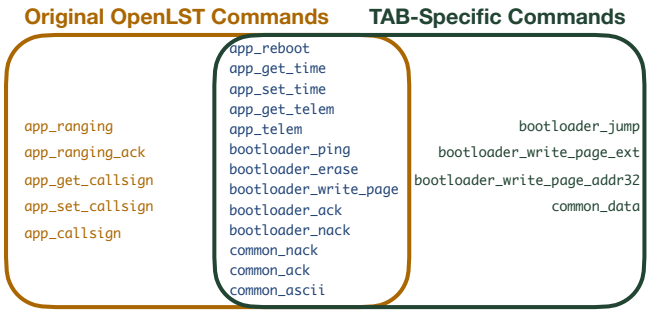


Figure 4: Left: Commands unique to the original OpenLST software. Middle: Commands common to both the original OpenLST software and TAB. Right: Commands unique to TAB and aimed at supporting OEC research.

In this work, we present Tartan Artibeus (TA1), a low-cost, fully open-source hardware [17] and software [18] satellite for use as an OEC research and evaluation platform. Using the Tartan Artibeus Bus (TAB), researchers easily integrate independently-designed hardware and software modules with TA1. TAB allows TA1 to integrate with `cote` or other software for hardware-in-the-loop simulation. TA1 can also be deployed to orbit for in-situ evaluation of OEC proposals.

3 TAB: The Tartan Artibeus Bus

The Tartan Artibeus Bus (TAB) accomplishes three main goals: (i) augmenting software simulation of OEC proposals with hardware-in-the-loop, (ii) integrating unmodified COTS subsystems into an intermittent, batteryless satellite, and (iii) operating in LEO as a proof of concept. TAB uses a standardized communication protocol for transferring commands and data among independent satellite modules. This serial communication protocol consists of 17 commands that adhere to a well-defined message structure. Users may easily extend the protocol with additional commands. TAB implements a core subset of OpenLST [33] commands and an additional set of new commands to better support OEC research.

Unlike the OpenLST communication protocol, which supports development and operation of the OpenLST radio, TAB aims to facilitate (i) interaction between software simulation environments and research hardware (hardware-in-the-loop); (ii) integration between independently-designed satellite sub-

common_ack

Required payload:
None
Optional payload:
None
Reply:
common_ack

app_set_time

Required payload:
Seconds and nanoseconds
Optional payload:
None
Reply:
If success: common_ack
If failure: common_nack

Figure 5: Left: The specification of the common_ack TAB command. This command contains no optional or required payload bytes and always elicits a common_ack reply. Right: The specification of the app_set_time TAB command. This command requires seconds and nanoseconds payload bytes. The command recipient sends a common_ack to indicate success or a common_nack to indicate failure.

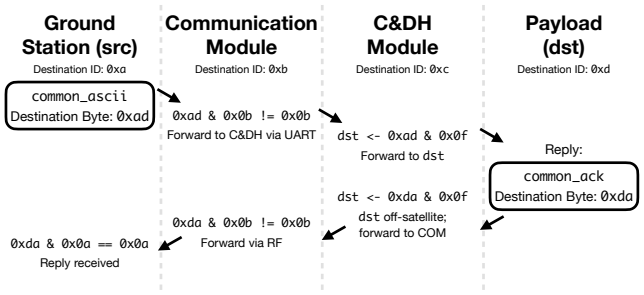


Figure 6: A ground station sends an RF command. The TA1 radio module receives the command and forwards it to the C&DH module, which parses the destination and delivers the message to the computational payload. The reply follows the same sequence in reverse.

systems; and (iii) in-flight operation of deployed satellites. We greatly appreciate the open-source release of OpenLST hardware and software and, in the same spirit, release TAB for open-source use in space and computer systems research at http://intermittent.systems. We provide reference TAB implementations both as C header and implementation files and as a single Python script to allow straightforward integration with other projects.

In Section 3.1, we describe the anatomy of a TAB command and compare the smaller, more general TAB command set to the larger, application-specific set of OpenLST commands. We highlight commands unique to TAB that aim to better support OEC research goals. We describe the TAB communication protocol in Section 3.2. In Section 3.3, we illustrate the versatility of TAB for hardware-in-the-loop simulation, seamless subsystem integration, and in-flight operation of deployed satellites.

3.1 TAB Commands

Every TAB command consists of two sections: a header and a payload. The header contains the start bytes, length, hardware ID, message ID, destination ID, and the command "opcode." Thus, the TAB header consists of a constant number of 9 bytes. The payload contents vary by command. Command payloads range from 0 to 249 bytes.

Figure 3 (a) illustrates the structure of the common_ack command, which consists entirely of header bytes. To preserve compatibility with the OpenLST protocol, we retain the two start byte values of 0x22 and 0x69. The length byte indicates the number of remaining bytes in the command and therefore always takes a value between 0x06 and 0xff, inclusive. The hardware ID, which consists of two bytes with the least-significant byte first, indicates the ID of the satellite or device targeted by the command. The message ID, which also consists of two bytes with the least-significant byte first, acts similarly to a nonce and allows a reply to be paired with its initiating command. TAB uses the upper and lower nibble of the destination byte to support intra-satellite communication among independent subsystems. Command recipients parse the opcode byte to determine the proper TAB reply.

Figure 3 (b), (c), and (d) present three examples of TAB commands with payload bytes. The app_set_time command consists of a constant number of payload bytes. The first four payload bytes represent the seconds since the J2000 [46] epoch (least-significant byte first), and the next four payload bytes represent the remaining nanoseconds (again, least-significant byte first). This command allows an external source to set the real-time clock (RTC) of a TAB module. The app_reboot command exemplifies a payload with optional bytes. The four payload bytes (least-significant byte first) are optional — their presence or absence is indicated by the length byte in the header — and represent the delay before executing a reboot procedure. Finally, the common_ascii command consists of a variable number of payload bytes. The command, which should be used for debugging messages (see common_data for generic data transfer), contains 0 to 249 ASCII-encoded character bytes as indicated by the header length byte (and, thus, requires no null-character termination).

We include a full list of TAB commands in Figure 4. For reference, we illustrate the larger set of application-specific OpenLST commands, the overlap between these commands and the core TAB commands, and additional TAB commands that support OEC research. In keeping with the OpenLST convention, TAB commands are categorized by app-focused commands, bootloader-focused commands, and commands common to both domains. For specification of the remaining commands, see our open-source reference implementations in C [18] and Python [19] and the accompanying documentation.

3.2 TAB Protocol

Under the TAB protocol, every command elicits a single reply. The reply varies by command and by command content.

Commands and replies are paired by the header message ID bytes; every reply mirrors the message ID of the initiating command. Commands are addressed by a hardware ID, which indicates the destination device, and the destination ID, which supports communication among satellite submodules.

The TAB protocol augments the function of the destination ID byte compared to the OpenLST protocol in a backwards-compatible manner. In TAB, the upper nibble of the destination ID indicates the originating module, and the lower nibble of the destination ID indicates the target module. Thus, the upper and lower nibble of the destination ID in a reply are swapped when compared to the upper and lower nibble of the destination ID of the initiating command.

We adhere to a single-command, single-reply protocol to encourage simple command logic and predictable behavior. A command-reply structure fosters intuitive operation and makes debugging straightforward. Such a protocol naturally limits the scope of command-triggered logic, because any such logic must complete swiftly to produce a timely reply. This point is particularly important in the energy-constrained regime of small satellites. Further, limited command-triggered logic supports portable protocol implementations without the need for integration with MCU-specific interrupts.

Figure 5, left, illustrates the specification for a `common_ack` command, which always elicits a `common_ack` reply. This example demonstrates the expected behavior of an in-flight basic check from a ground station to a deployed satellite. See Section 5.1 for a more detailed description of this use case.

Figure 5, right, specifies the `app_set_time` command, which elicits either a `common_ack` reply or a `common_nack` reply depending on the command result. This example illustrates how a software simulation package can initialize the RTC of a hardware-in-the-loop satellite module during pre-flight testing and evaluation. If the simulator receives a `common_ack`, then the simulation can continue. If the simulator instead receives a `common_nack`, then the simulation can be aborted with an error message for the operator.

Figure 6 traces multiple "hops" of a `common_ascii` command. This example demonstrates use of the destination ID byte to support communication among independently-designed satellite modules. A computational payload board requires an ASCII-encoded TLE to calculate the satellite position given its RTC time. This TLE, which is generated on the ground, must be received by the radio module, forwarded to C&DH, and then delivered to the computational payload. See Section 5.3 for a more detailed description of this use case.

3.3 TAB Use Cases

Because the TAB specification is agnostic to the physical layer, it can be deployed regardless of the communication medium. TAB aims to facilitate interaction between software simulation environments and research hardware (hardware-in-the-loop), integration between independently-designed satellite subsystems, and in-flight operation of deployed satellites.

For hardware-in-the-loop simulations, we connect a satellite submodule to a software simulation package via USB-to-serial. Both the satellite submodule and the simulation software leverage reference implementations of TAB. The satellite submodule implements TAB over its UART pins, and the simulation software implements TAB over its serial port. Independently-designed satellite subsystems in TA1 communicate using TAB via connected UART pins. For in-flight operation of deployed satellites, TAB commands are encoded for RF transmission and decoded by a receiving radio. See Section 4.5 for a more detailed description of this use case.

4 Tartan Artibeus

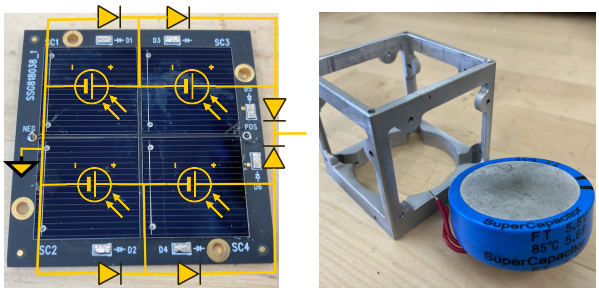
Tartan Artibeus (TA1) is the first batteryless, computational pocketqube satellite; its open-source hardware and software launched into low-Earth orbit (LEO) in January 2022. TA1 is a 1p ($125\,\mathrm{cm}^3$) pocketqube built around the Tartan Artibeus Bus (TAB), which connects independently-designed modules into a batteryless, computational satellite. TA1 incorporates an electrical power supply (EPS) module that harvests solar energy into a supercapacitor, a fault-tolerant command and data handling (C&DH) module, a radio communication module, and a configurable computational payload module. We present an overview of the major components of TA1.

4.1 Energy Harvesting and Storage

TA1 mounts five solar panel PCBs to five of the six pocketqube faces. The sixth pocketqube face is reserved for the baseplate and is described in Section 4.6. The solar panels are electrically connected to the internal electrical power system (EPS) PCB. The power module conditions the solar panel voltages and currents and stores the harvested energy in a 5.6 F supercapacitor. We provide additional details for the solar panels, EPS, and energy storage.

Solar Panels: A custom solar panel PCB covers five of the six $25\,\mathrm{cm}^2$ faces of the TA1 pocketqube. Each solar panel PCB contains four square solar cells. A single solar cell measures 1.88 cm on each side. We first connect two cells in series to increase panel voltage, and we then connect two pairs of cells in parallel to increase panel current. Figure 7a provides a diagram of the solar panel PCB design that illustrates the solar cell connections and the corresponding bypass diodes.

Using the standardized value for solar spectral irradiance of $1366.1\,\mathrm{W/m}^2$ [2] and the datasheet efficiency of 29.4% at the time of manufacture [3], each solar cell exhibits a maximum power (MP) voltage of 2.441 V and an MP current of 58.88 mA. Thus, a single solar panel provides an MP voltage of 4.882 V and an MP current of 117.8 mA. Therefore, each solar panel provides up to 0.5751 W of power to the internal EPS PCB module. After an extended deployment [3], MP voltage falls to 2.246 V and efficiency falls to 26.5%. Thus,



(a) **TA1 Custom Solar Panel.** (b) **TA1 Supercapacitor.**

Figure 7: Figure 7a shows an annotated picture of an assembled solar panel. Bypass diodes allow current to flow while protecting against the possibility of inoperable solar cells. Figure 7b contains an image of the supercapacitor used to buffer energy in TA1 next to the satellite chassis for scale.

MP current falls to 56.97 mA. For each solar panel, these values correspond to an MP voltage of 4.492 V and an MP current of 113.9 mA with an overall MP of 0.5116 W.

Internal EPS PCB Module: Figure 8 shows a high-level schematic of the TA1 power system, which uses a dual input-output booster design [14, 40, 51]. The input booster charges the capacitor up to 5.5 V, at which point the voltage supervisor enables the output booster. The output booster provides a stable 3.3 V to the VDD rail and discharges the capacitor as low as 2.0 V before the supervisor disables the output booster. The lower threshold of 2.0 V avoids inefficient charging from the input booster's "cold-start" region.

To reduce the effort of building flexible applications for TA1, the EPS design takes on some complexity to shield application subsystems. First, the dual booster system supports peripherals that cannot tolerate the full voltage range experienced by the capacitor. This approach simplifies peripheral subsystem hardware development — the developers can expect a 3.3 V input to the subsystem that is stable until the device powers down. The separate power rails for each peripheral subsystem minimize the effort on the part of the C&DH module to disable peripherals to save power. Features like the hardware-defined hysteresis thresholds also reduce the flight software complexity by removing the need for software intervention [14, 44, 45]. Finally, the EPS includes onboard measurement hardware to capture load current, harvested power, and capacitor voltage without involving off-board subsystems. A four-port op-amp buffers the voltage measurements and passes the results to a 16-bit ADC that can be accessed by any subsystem via I2C. The ADC and op-amp are both powered by VDD, so when VDD is low, both are powered off. However, the op-amp will sink current from its inputs if they are higher voltage than VDD, which would reduce the system efficiency during charging. High impedance voltage dividers between the measured voltages and the op-amp prevent extra current from draining.

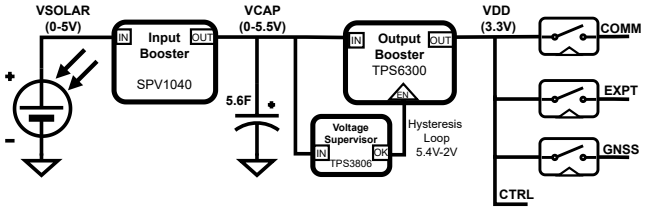


Figure 8: **TA1 Power System.**

Supercapacitor: On TA1, the EPS buffers energy in a 5.6 F supercapacitor, the FT0H565ZF (FTH) [31]. With a total energy storage capacity of over 80 J and a maximum power output of more than 8 W, the FTH approaches the maximum energy and power that can be stored in a 1p pocketqube form factor with a capacitor. We also select the FTH for its manageable effective series resistance (ESR) over a wide temperature range. We expect the FTH's ESR to increase to no more than 4.2 Ω at −40 °C and no more than 600 mΩ at 85 °C. ESR limits the power a supercapacitor can supply, so lower is better. We perform altitude testing to confirm that the capacitor performance is not impacted by vacuum.

4.2 Command and Data Handling

TA1 aims to minimize the hardware and software components responsible for persisting execution context across power failures (see Section 2.2). Thus, TA1 concentrates flight control and persistency management software onto the C&DH module. Hardware and software design choices allow C&DH to coordinate peripheral subsystems via TAB while maintaining a large degree of isolation among these subsystems. Figure 9 shows the design of the C&DH module PCB.

The C&DH module uses intentional hardware choices to provide power-failure awareness at little cost to the peripheral subsystems. First, the C&DH MCU is an MSP430FR5994 with 256 kB of byte-addressable, non-volatile FRAM (NVM) [66]. Byte-addressable NVM reduces the cost of persisting state compared to technologies like Flash that can only be read/written at a page-granularity [12]. Second, the C&DH module uses level shifters [65] to electrically isolate peripheral subsystems from the C&DH module. This isolation prevents current from leaking across the peripheral connections when they are unpowered. Finally, the C&DH module contains two critical sensors that provide telemetry and position data accessible by peripheral subsystems: a 9-DoF IMU [58] and a GPS unit (with COCOM limits removed) [29]. The C&DH module hardware simplifies application development on TA1, and the design of the programming interface is just as important.

Figure 10 shows an example program written with tasks (in green) that handles interrupts from the TAB (in orange), supported by the failure-aware programming interface (keywords shown in blue) . The application atomically walks through

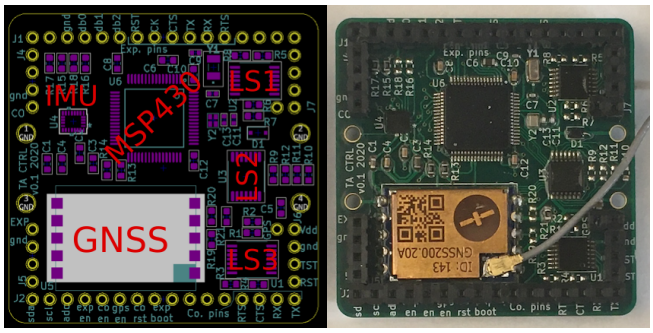


Figure 9: **C&DH module hardware.** The C&DH MCU, the MSP430, gathers telemetry from the GNSS and IMU. Using level shifters (LS1-3), the C&DH MCU communicates over the TAB to the attached subsystems.

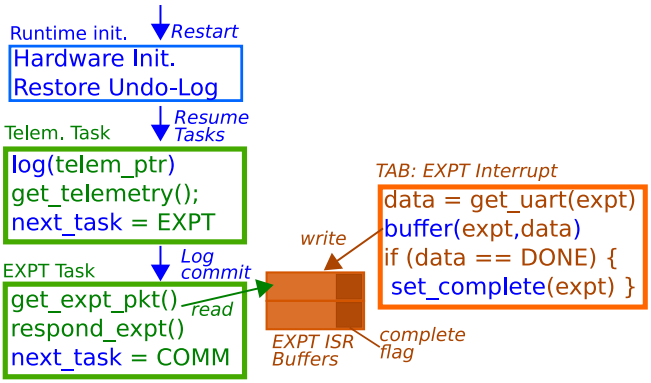


Figure 10: **Power-failure-aware program.** C&DH programs are written as a series of tasks (in green). The power-failure-aware runtime components (in blue) correctly restart the program after a power-failure. Interrupts from the TAB (in orange) share data with tasks through a managed buffer (center).

tasks by persistently updating the active task. If power fails in the middle of a task, execution will resume from the start of the task on the next boot. Because TA1 is designed to support compute-bound applications, the code does not contain any low-power wait states. Tasks execute opportunistically until power fails.

To execute correctly, intermittent tasks must handle write-after-read (WAR) data accesses when a failed task is re-executed [41]. Effectively, the dynamic execution gets out of sync with the persistent state and leads to corrupted values. To prevent WAR bugs, programmers use the C&DH logging interface to write variables involved in WAR conflicts to an undo-log [43]. On reboot, the log is replayed to restore the state of non-volatile memory to the start of the failed task. Programmers may use a variety of tools to *identify* variables involved in WAR conflicts [41, 42, 59], but we recommend that they avoid using the modified code these tools produce. The performance gap between code produced by the production MSP430 GCC implementation [67] and LLVM-based compiler tooling [36] is prohibitive for real systems.

The C&DH board's non-blocking TAB implementation relies on interrupt service routines (ISRs) to process and store packets that arrive on its UART ports. However, managing concurrent access to shared data is a challenge for any embedded system. Intermittent execution exacerbates the problem by introducing the possibility of a power failure during an interrupt leaving partial updates to shared state. To overcome this limitation, the C&DH software minimizes the interface between application-defined tasks and peripheral-triggered interrupts [55], illustrated on the right side of Figure 10. The C&DH software defines a restricted set of buffers that each interrupt may write into and tasks may read from. Interrupts mark the buffers as "ready" when they may be read by tasks, at which point tasks may extract data from the buffers and mark them as empty. Consecutive interrupts can add to the same buffer of data, but interrupts may not read from this buffer nor any other persistent data. Instead, ISRs share data via the buffer interface, and use *static* variables for state that should be retained across invocations of the ISR. Following the task/interrupt interfacing rules prevents power failures from corrupting shared data. By providing the underlying persistence and initialization guarantees, the C&DH board supports a wide range of application payloads.

4.3 Computational Payload

The computational payload board consists of a custom-designed PCB around and an STM32L496RGT3 microcontroller unit (MCU). This MCU contains an ARM Cortex-M4 core: a well-documented [69], high-performance (80 MHz) MCU with extensive open-source software support (e.g., `libopencm3` [38]). The STM32L4 series supports multiple levels of ultra low-power modes, while STM32L496 MCUs offer the maximum amounts of integrated SRAM (320 kB) and Flash (1 MB). We select the STM32L496RGT3 for its tolerance of extended temperature ranges (−40 °C to 125 °C) and industrial applications rating. Additionally, the LQFP64 package supports reliable hand-soldering. We illustrate the custom PCB design and an assembled board in Figure 11.

The ARM Cortex-M4 includes a single-precision hardware floating point unit (FPU), hardware support for digital signal processing (DSP) instructions, hardware support for single-instruction, multiple data (SIMD) multiplication, and multiply-accumulate (MAC) instructions [57]. As a result, the MCU core supports hardware acceleration of machine inference. The MCU also includes an internal real-time clock (RTC) with support for operation in a low power (320 nA) state while persisting 32 backup registers. Both the SRAM and Flash provide hardware error detection.

In the payload PCB, we populate both the high-speed and low-speed crystal oscillator pins with components rated for the extreme temperatures of space. We also populate the

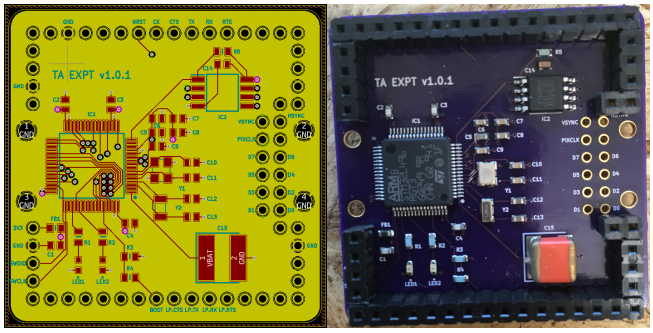


Figure 11: Left: The computational payload board design. Right: An assembled computational payload board.

backup power pin with a supercapacitor containing enough capacitance to support RTC persistence for hours. We break out the DCMI digital camera interface pins for future use with satellite sensors. We attach a high-capacity (16 MB) external Flash storage chip with an optional quad-SPI high-speed interface for storage of machine learning models and captured data. The PCB breaks out UART pins for use with TAB.

Because space systems require accurate timekeeping for task scheduling, event timestamps, and navigation, the TA1 computational payload board tightly integrates operation of the on-chip RTC with its CONOPS. The RTC includes a date register and a time register. The date register stores year, month, and day by encoding the digits in the BCD format, and the time register stores hour, minute, and second in the same format. The date and time registers start ticking after initialization and are handled completely by hardware logic. To ensure the registers tick at the correct frequency, the RTC clock source (either LSE, LSI, or HSE) generates a 1 Hz clock signal by prescaling its frequency.

TA1 uses the on-chip RTC peripheral to keep time for module applications. When the computational payload module receives power, MCU software initializes the RTC peripheral with the LSI clock source. Upon receipt of an `app_set_time` TAB command, the MCU software initializes the date and time registers. Because the `app_set_time` TAB command uses the Julian day format and the RTC registers use the Gregorian format, the MCU software calculates a conversion. The MCU software implements a well-known procedure for this conversion [23].

4.4 Radio Communication

The TA1 radio communication module takes inspiration from the OpenLST [33] radio hardware and software. The original OpenLST hardware measures 6 cm × 5 cm — too large to fit into a pocketqube form factor. In addition to using a few difficult-to-source or deprecated components, the original OpenLST hardware makes use of three voltage domains (5 V, 3.6 V, and 3.3 V) To simplify assembly, reduce cost, and update the design for integration with other TA1 modules, we heavily modify the OpenLST hardware design. We illustrate the custom PCB design and an assembled board in Figure 12.

The TA1 radio communication module measures 43 mm × 43 mm so that it easily fits inside of a pocketqube. We replace the deprecated power amplifier (PA) with a mass-market alternative. Following recommendations provided by the documentation of this alternative PA, we omit the difficult-to-source SAW filter of the original OpenLST hardware and instead deploy a network of discrete components for filtering. These changes eliminate all voltage domains except for 3.3 V.

To support rapid testing, the radio communication board includes two signal path options between the PA and the antenna. Soldering one capacitor selects the "test" signal path, which terminates at a u.FL connector for use with a COTS 50 Ω impedance antenna. Because no COTS antenna exists in a form-factor compatible with the pocketqube standard, the radio communication board also supports a second, "flight" signal path. Soldering the selection capacitor to a different set of pads completes this "flight" signal path, which terminates at a solder point preceded by pads for an L-matching network. TA1 uses a custom-designed, nitinol antenna. After matching the antenna length to the half-wavelength of the center operating frequency, the appropriate passive components match the antenna impedance to the signal path.

Despite significant hardware changes, the OpenLST software remains largely unchanged. As in OpenLST, the TA1 radio leverages the CC1110 RF MCU. We modify GPIO software logic to support an alternative RF switch, and we modify frequency specification variables to reflect the 401.82 MHz at which the board is licensed to operate. TAB makes integration of pre-existing modules seamless and straightforward.

4.5 Radio Link Analysis

Under a bent-pipe CONOPS, RF communication plays a key role in the satellite mission. Even under OEC, RF communication occupies an indelible position in satellite operations: delivering processed data to the ground segment. Therefore, the RF CONOPS requires sufficient link budget and appropriate error correction in the packet design. Error correction plays a particularly important role in satellite missions due to artifacts inherent to long range, high velocity satellite communications such as the doppler effect. These effects are less prevalent in terrestrial communication scenarios.

Because the small form factor and associated power constraints of TA1 prevent boosting transmit power arbitrarily high (within the FCC regulations), we use a modified version of OpenLST [33] based on the popular CC1110 [25] radio transceiver platform for low power operation of the RF subsystem. We describe the packet structure, link budget, and the ground station demodulation and decoding pipeline of TA1.

Packet Structure and Transmit Pipeline: Figure 13(a) shows the packet structure in the radio communication mod-

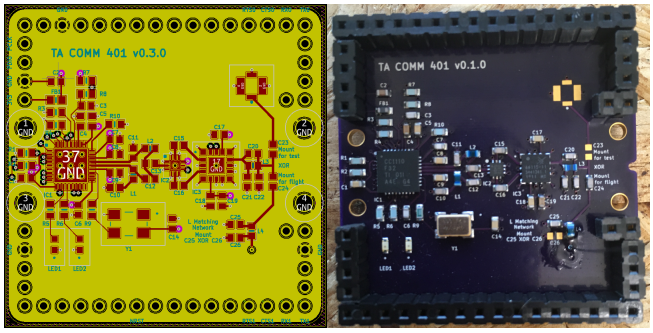


Figure 12: Left: The radio communication board design. Right: An image of an assembled communication board.

ule, which is based on the CC1110 pipeline. It contains a 4 byte *preamble* consisting of an alternating sequence of 1010... bits and a 2 byte *sync word* that can be set by the user. The *preamble* sequence helps in packet detection and the *sync word* aids in byte-level synchronization as well differentiating packets from other sources transmitting using the same platform in the absence of the address field. This sequence is followed by the *length* and *payload* fields. The former indicates the length in bytes of the payload. The maximum payload length is 255 bytes. Finally, a 16-bit *CRC* concludes the payload to ensure bit error detection and correction.

After packet creation, the bits undergo appropriate coding and modulation before being transmitted over the air. Figure 13(b) describes the bits-to-symbol encoding and modulation pipeline. Bits are first *whitened* using a 9-bit pseudo-random sequence [64] defined by the generator polynomial $x^9+x^5+x^0$. Data whitening ensures bit randomization, which protects against DC bias introduced in the transmit signal due to a burst of continuous 0s or 1s. The whitened data is then convolutionally encoded [54] using a half rate encoder defined by the generator polynomials $g_1 : x^3+x^1+x^0$ and $g_2 : x^3+x^2+x^1+x^0$, respectively, to ensure robustness to channel-induced errors. Post encoding, the data is interleaved using a 4×4 interleaver to ensure robustness to burst errors that can be introduced due to channel fading. The interleaved bits are encoded in 2-FSK modulation and transmitted over the air using the baud rate of approximately 7.5 Kbaud. Both the modulation format and baud rate are user-specific fields that can be modified based on need. It should be noted that the *preamble* and *sync word* fields do not undergo the whitening, encoding, and interleaving transformations, while the rest of the fields in the packet do.

RF Characteristics and Link Budget: The 2-FSK modulation essentially encodes bits 0 and 1 using two frequencies, $-f/2$ and $f/2$ respectively, separated by a two-sided bandwidth of f (approximately 7.5 kHz in TA1). This fact can be seen as two prominent peaks in the frequency spectrum of the received signal. The link budget analysis can be broken down as follows:

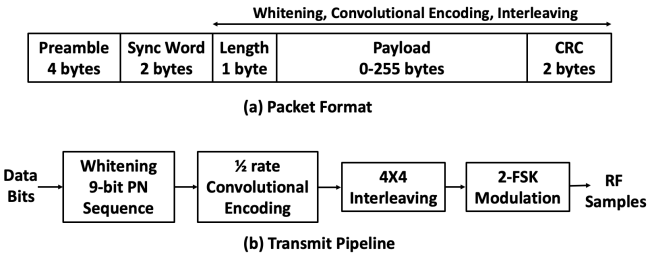


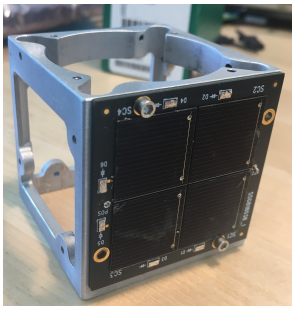
Figure 13: TA1 radio communication module (a) packet format, and (b) transmit pipeline

1. *Transmitter:* The RF signal is transmitted from the satellite using a maximum transmit power of 1 W (30 dBm). This transmit power can be throttled based on the power availability on the satellite. Assuming unity gain antenna at the satellite, these factors provide an Equivalent Isolated Radiated Power (EIRP) of 30 dBm.

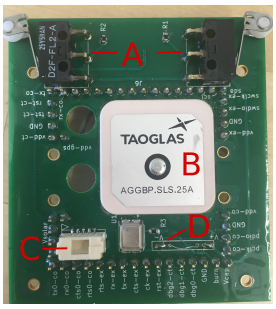
2. *Path loss:* The orbital altitude of 500 km along with transmit frequency of 401.82 MHz results in a free space path loss of 138.5 dB. Note that the actual distance can vary, depending on the exact relative distance between the satellite and the ground station, which varies as the satellite moves in its orbit.

3. *Receiver:* Any SDR-based ground station would receive the signal at a bandwidth slightly higher than the transmitter to ensure decent oversampling factor that can be used to average out noise. Receiving the satellite signal at a $5\times$ oversampling results in a noise floor of -128 dBm. Accounting for a 6 dB noise factor introduced by any hardware, we can safely assume an effective receiver sensitivity of -116 dBm. In order to ensure a good signal-to-noise-ratio (SNR) of 10 dB for 2FSK modulation, any received signal with power over -106 dBm should be decodeable. Removing path loss from EIRP and accounting for a 10 dB antenna gain at the ground station, the received signal power at the ground station comes out to be -98.8 dBm, which in turn results in a SNR of approximately 18 dB. This SNR is more than sufficient to ensure successful decoding. While the numbers used in this analysis are typical values, in case of higher path loss or higher bandwidth (noise floor) or lower transmit power, a higher gain antenna with a low noise amplifier can be used at the ground station to ensure sufficient SNR for successful decoding.

Demodulation and Decoding Pipeline: While on-chip decoders use the exact same parameters like bandwidth, gain, etc. for demodulation and decoding, employing an SDR-based ground station provides more flexibility to record at higher gain and debug in low SNR scenarios. We create a demodulation and decoder script to process the received signal from any SDR operating at any acceptable receiver bandwidth ($>2\times$



(a) **TA1 chassis.**



(b) **TA1 baseplate.**

Figure 14: **TA1 Mechanical Components.** The TA1 chassis (left) and baseplate (right) comply with the pocketqube standard and electrically attach key components.

the transmit bandwidth). Using a higher bandwidth at the SDR provides a higher oversampling factor that gives two benefits: a larger number of RF samples to yield a higher *preamble* and *syncword* detection gain, and a larger number of samples to disambiguate the FSK symbol frequency while demodulation. Our packet detection and demodulation benefits from these two aspects. We first detect the packet using a large number RF samples to improve the chance of detection and then demodulate to bits. In comparison, an on-chip system first demodulates to bits and then detects the packet using much fewer bits thus reducing the chance of detection. Upon demodulation to bits, we reverse the steps described in Figure 13(b) to recover the transmitted message.

4.6 Baseplate, Chassis, and Mechanics

The TA1 stacked-PCB design inside the pocketqube form factor eliminates many mechanical questions encountered by cubesats. For TA1, the mechanical design effort concentrates in the chassis, the sliding baseplate required by the pocketqube standard, and compliance testing required by the launch provider. Figure 14a shows the 115 g aluminum chassis, co-designed with the solar panel PCBs also pictured. The in-house mechanical design group TechSpark at Carnegie Mellon University modeled and manufactured the chassis. The chassis connects to the baseplate using four standard M2 screws, and custom aluminum standoffs affix the stack of modules to the baseplate to provide an electrical chassis ground to the modules.

The baseplate is a critical component of the TA1 design that resolves mechanical, electrical and compliance requirements. Figure 14b shows a labeled image of the baseplate. In addition to the connection to the chassis, the baseplate electrically connects to all of the pins used by TAB to communicate between modules and exposes them for testing, debugging, and pocketqube standard compliance. In the pocketqube standard, the baseplate is primarily defined as the interface to the pocketqube launcher. In addition to adhering to the required dimensions, the baseplate must provide mechanical "kill switches" that keep the satellite powered off until deployment. Leveraging the baseplate module connections, the kill switches short the supercapacitor voltage to chassis ground when the satellite is inside the launcher, depressing the switches. Connecting the supercapacitor to ground over a low resistance (100 Ω) prevents any errant harvested energy from accruing in the supercapacitor before launch. Beyond its required functionality, TA1 utilizes the baseplate to mechanically secure the GNSS antenna [63](Figure 14b, B), provide a convenient connection to the solar panels (Figure 14b, C), and hold the antenna release mechanism (Figure 14b, D). The radio module nitinol antenna is secured before launch by wrapping it against the baseplate and tying it to a nichrome wire [48] with fishing line [8]. On first boot, the C&DH module activates a high power transistor that shorts the supercapacitor to ground over the nichrome wire. The resulting current flow causes the nichrome wire to heat up and melt away the monofilament fishing line, releasing the antenna.

Once the pocketqube standards are met, compliance with the launch provider must be satisfied. To mechanically test TA1, we carry out qualification level testing on the final TA1 flight unit for random vibration and sine burst testing [21]. No damage was incurred on TA1 throughout these tests, and the unit functioned as expected afterwards. Additionally, TA1 must meet outgassing requirements set by the launch provider [56]. We apply a conformal coating to both sides of each module and the baseplate to prevent substantial outgassing from any of the electrical components. Interestingly, the conformal coating resulted in an unspecified conflict with the pocketqube standard. The coating increased the friction between the edge of the TA1 baseplate and the pocketqube launcher rail [52] and prevented TA1 from sliding freely. To launch, the coating had to be removed along the 2 mm overlap between the baseplate and the rail. Future revisions to the pocketqube standard should specify the acceptable coefficient of friction between the baseplate and the launcher rail.

5 TAB Reference Applications

To evaluate the TA1 implementation of TAB, we present three reference applications. These applications illustrate the concept of operations (CONOPS) of TA1. Each application makes use of TAB. The first application, "ping," exercises the TA1 radio module. The second application, "data," exercises both the radio module and the C&DH module. TAB messages are received by the radio module, parsed, and forwarded to the C&DH module destination. The third application, "`cote`," exercises the radio module, the C&DH module, and the computational payload module. TAB messages are received by the radio module, parsed, and forwarded to the C&DH module for delivery to the computational payload module.

5.1 Application 1: Ping

The "ping" application consists of a single command, which generates a single reply. A ground station emits a `common_ack` via RF (see Section 4.5) for reception by TA1. To be accepted, the hardware ID of the `common_ack` must match the hardware ID of TA1. All TA1 modules have been programmed with this hardware ID.

This application aims to elicit a reply from the radio module. Therefore, the `common_ack` destination ID contains the radio module ID as the lower nibble. To indicate that the reply should be transmitted via RF, the upper nibble contains the ground station ID. Upon reception, the radio module generates a `common_ack` response. The upper and lower nibbles of reply destination ID are swapped as described in Section 3.2.

5.2 Application 2: Data

The "data" application demonstrates the ability of TAB to support communication between modules. This application requests telemetry stored on the C&DH module. Every minute, TA1 collects data from the IMU on the C&DH board as well as the time, date, position, and status information from the GNSS and power introspection information from the ADC on the EPS. These data are pushed onto a stack until a telemetry request command arrives. In response, the C&DH module pops the most recent telemetry for reply.

5.3 Application 3: `cote`

The "`cote`" application consists of a single command that ultimately results in a single reply. However, this application exercises three TA1 modules: the radio module, the C&DH module, and the computational payload module. This "`cote`" application implements a limited version of the open-source `cote` simulator for computational nanosatellites [16]. Specifically, we port the `cote` implementation of SGP4 [28] from C++ to C for execution on the computational payload MCU. Given a TLE via a `common_ascii` command, the payload board leverages the MCU RTC to calculate the current position of the satellite and generate a message containing these coordinates. See Figure 6 for a diagram of this application.

6 Additional TAB Applications

TAB supports over-the-air upload of new software applications after deployment to orbit. Post-deployment updates make TA1 a flexible orbital edge computing research platform. In the lab, we have demonstrated this upload mechanism and the ability to time-multiplex several programs on a single payload board. This feature provides a first step toward dynamic multitenancy and new, flexible "software-defined nanosatellite constellations."

The application upload mechanism delivers new programs to the satellite via RF; upon delivery, these programs are stored for execution. Our implementation uses Intel's line-oriented HEX format, which includes a per-line start code, byte count, address, record type, data, and checksum. The start code specifies the record type and, if the line start code indicates a data record, the TA1 ground support software [19] parses data from that line for transmission. After collecting all data lines, the data are split into 128-byte "chunks" for use in the `bootloader_write_page` TAB command. These commands are sent to the computational payload board, where the program data are written into storage.

Simultaneous deployment of multiple programs requires a more general TAB command than the backwards-compatible `bootloader_write_page` command. Specifically, in order to make use of the entire address range of the program storage, TAB introduces the `bootloader_write_page_addr32` command. This new, more general command writes the 128 byte TAB payload starting at the address in memory specified in the command. In the lab, we exercise this new, more general TAB command by demonstrating a computational payload board with three user programs. Each user program executes in response to a TAB `bootloader_jump` command.

7 Conclusion

We present Tartan Artibeus, an open source hardware [17] and software [18] 1p pocketqube satellite launched into LEO in January 2022. Built around the Tartan Artibeus Bus (TAB), TA1 integrates independently-designed modules into a batteryless, computational satellite. TAB positions TA1 as a platform for computational nanosatellite research by supporting (i) interaction between software simulation environments and research hardware (hardware-in-the-loop); (ii) integration between independently-designed satellite subsystems; and (iii) in-flight operation of deployed satellites.

We provide in-depth descriptions of the satellite mechanics, PCB hardware design, software, system CONOPS, and radio communication to serve as a blueprint for other organizations working in computational space systems. Detailed bills of material (BOMs), MCU software, communication protocol and CONOPS documentation, and ground station software are available publicly. TA1 serves as a low-cost, compliance-tested research platform and satellite for LEO operations.

Acknowledgments

We thank members of the CMU ABSTRACT research group for feedback and discussion on the topic of this paper. This work was generously funded by the Kavčić-Moura Endowment Fund and National Science Foundation Award #2111751.

References

[1] J. Andersson, M. Hjorth, F. Johansson, and S. Habinc. Leon processor devices for space missions: First 20 years of leon in space. In *SMC-IT*. IEEE, 2017.

[2] ASTM. Standard extraterrestrial spectrum reference e-490-00. Technical report, American Society for Testing and Materials, 2000.

[3] Azur Space Solar Power. Triple junction solar cell 3g30c, 2018.

[4] D. Balsamo, A. S. Weddell, A. Das, A. R. Arreola, D. Brunelli, B. M. Al-Hashimi, G. V. Merrett, and L. Benini. Hibernus++: a self-calibrating and adaptive system for transiently-powered embedded devices. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 35(12):1968–1980, 2016.

[5] D. Balsamo, A. S. Weddell, G. V. Merrett, B. M. Al-Hashimi, D. Brunelli, and L. Benini. Hibernus: Sustaining computation during intermittent supply for energy-harvesting systems. *IEEE Embedded Systems Letters*, 7(1):15–18, 2015.

[6] T. M. Bandhauer, S. Garimella, and T. F. Fuller. A critical review of thermal issues in lithium-ion batteries. *Journal of the Electrochemical Society*, 158(3):R1, 2011.

[7] R. Berger, S. Chadwick, E. Chan, R. Ferguson, P. Fleming, J. Gilliam, M. Graziano, M. Hanley, A. Kelly, M. Lassa, et al. Quad-core radiation-hardened system-on-chip power architecture processor. In *Aerospace Conference*. IEEE, 2015.

[8] Berkley. FireLine Original. https://www.berkley-fishing.com/products/fireline-original-1316955, 2022.

[9] R. Buckle. Life testing of cots cells for optimum battery sizing. In *2019 European Space Power Conference (ESPC)*, pages 1–7. IEEE, 2019.

[10] J. Cappaert. Building deploying and operating a cubesat constellation-exploring the less obvious reasons space is hard. In *Proc. AIAA/USU Conf. Small Satellites*, 2018.

[11] S. Chien, R. Sherwood, D. Tran, B. Cichy, G. Rabideau, R. Castano, A. Davies, R. Lee, D. Mandl, S. Frye, et al. The eo-1 autonomous science agent. In *Joint Conference on Autonomous Agents and Multiagent Systems*, 2004.

[12] A. Colin and B. Lucia. Chain: tasks and channels for reliable intermittent programs. In *Proceedings of the 2016 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*, pages 514–530. ACM, 2016.

[13] A. Colin and B. Lucia. Termination checking and task decomposition for task-based intermittent programs. In *Proceedings of the 27th International Conference on Compiler Construction*, CC 2018, pages 116–127, 2018.

[14] A. Colin, E. Ruppel, and B. Lucia. A reconfigurable energy storage architecture for energy-harvesting devices. In *Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '18, pages 767–781, New York, NY, USA, 2018. ACM.

[15] B. Denby and B. Lucia. Orbital edge computing: Machine inference in space. *IEEE Computer Architecture Letters*, 2019.

[16] B. Denby and B. Lucia. Orbital edge computing: Nanosatellite constellations as a new class of computer system. In *Architectural Support for Programming Languages and Operating Systems*, 2020.

[17] B. Denby and E. Ruppel. Tartan artibeus hardware. https://github.com/cmuabstract/tartan-artibeus-hw, 2022.

[18] B. Denby, E. Ruppel, S. Che, C. Taylor, and F. Zaidi. Tartan artibeus software. https://github.com/cmuabstract/tartan-artibeus-sw, 2022.

[19] B. Denby, E. Ruppel, and C. Taylor. Tartan artibeus ground support. https://github.com/cmuabstract/tartan-artibeus-gnd-sw, 2022.

[20] K. Devaraj, R. Kingsbury, M. Ligon, J. Breu, V. Vittaldev, B. Klofas, P. Yeon, and K. Colton. Dove high speed downlink system. In *Proc. AIAA/USU Conf. Small Satellites*, 2017.

[21] EXO Luanch. Mechanical Testing Falcon 9 Rideshare, 2020.

[22] W. Ferster. Digitalglobe adding infrared capability to worldview-3 satellite. *Space News, https://spacenews.com/digitalglobe-adding-infrared-capability-worldview-3-satellite/*, 2012.

[23] H. F. Fliegel and T. C. Van Flandern. Letters to the editor: a machine algorithm for processing calendar dates. *Communications of the ACM*, 1968.

[24] W. Frick and C. Niederstrasser. Small launch vehicles-a 2018 state of the industry survey. In *Proc. AIAA/USU Conf. Small Satellites*, 2018.

[25] Grant Christiansen, Texas Instruments. CC1110-CC1111 Datasheet. https://www.ti.com/lit/ds/swrs033h/swrs033h.pdf, 2010.

[26] W. Harwood. Nasa launches usd 855 million landsat mission. *CBS News, https://www.cbsnews.com/news/nasa-launches-855-million-landsat-mission/*, 2013.

[27] J. Hester, K. Storer, and J. Sorber. Timely execution on intermittently powered batteryless sensors. In *Proceedings of the 15th ACM Conference on Embedded Network Sensor Systems*. ACM, 2017.

[28] F. R. Hoots and R. L. Roehrich. Models for propagation of norad element sets: spacetrack report no. 3. Technical report, Aerospace Defense Command, Peterson AFB, Office of Astrodynamics, 1980.

[29] Hyperion Technologies. Gnss200. https://space-for-space.com/wp-content/uploads/2020/04/HT_GNSS200_v2.1-flyer.pdf, 2019. Accessed: 2022-05-25.

[30] H. Jayakumar, A. Raha, and V. Raghunathan. Quickrecall: A low overhead hw/sw approach for enabling computations across power cycles in transiently powered computers. In *VLSI Design and 2014 13th International Conference on Embedded Systems, 2014 27th International Conference on*, pages 330–335. IEEE, 2014.

[31] Kemet Electronics Corporation. Supercapacitors FT Series. https://content.kemet.com/datasheets/KEM_S6014_FT.pdf, 2020.

[32] B. K. Kim, S. Sy, A. Yu, and J. Zhang. Electrochemical supercapacitors for energy storage and conversion. *Handbook of Clean Energy Systems*, pages 1–25, 2015.

[33] B. Klofas. Planet releases openlst, an open radio solution. *https://www.planet.com/pulse/planet-openlst-radio-solution-for-cubesats/*, 2018.

[34] V. Knap, L. K. Vestergaard, and D.-I. Stroe. A review of battery technology in cubesats and small satellite solutions. *Energies*, 13(16):4097, 2020.

[35] W. J. Larson and J. R. Wertz. *Space mission analysis and design*. Microcosm, 1992.

[36] C. Lattner and V. Adve. Llvm: A compilation framework for lifelong program analysis & transformation. In *International Symposium on Code Generation and Optimization, 2004. CGO 2004.*, pages 75–86. IEEE, 2004.

[37] L. Leung, V. Beukelaers, S. Chesi, H. Yoon, D. Walker, and J. Egbert. Adcs at scale: Calibrating and monitoring the dove constellation. In *Proc. AIAA/USU Conf. Small Satellites*, 2018.

[38] LibOpenCM3. Libopencm3. *http://libopencm3.org/*, 2022.

[39] T. R. Loveland and J. R. Irons. Landsat 8: The plans, the reality, and the legacy. *Remote Sensing of Environment*, 2016.

[40] B. Lucia, B. Denby, Z. Manchester, H. Desai, E. Ruppel, and A. Colin. Computational nanosatellite constellations: Opportunities and challenges. *GetMobile: Mobile Computing and Communications*, 2021.

[41] B. Lucia and B. Ransford. A simpler, safer programming and execution model for intermittent systems. In *ACM SIGPLAN Notices*, volume 50, pages 575–585. ACM, 2015.

[42] K. Maeng, A. Colin, and B. Lucia. Alpaca: Intermittent execution without checkpoints. In *Proceedings of the 2017 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM, 2017.

[43] K. Maeng and B. Lucia. Adaptive dynamic checkpointing for safe efficient intermittent computing. In *Proceedings of the 12th USENIX conference on Operating Systems Design and Implementation*, pages 129–144. USENIX Association, 2018.

[44] K. Maeng and B. Lucia. Supporting peripherals in intermittent systems with just-in-time checkpoints. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1101–1116, 2019.

[45] K. Maeng and B. Lucia. Adaptive low-overhead scheduling for periodic and reactive intermittent execution. In *Proceedings of the 41st ACM SIGPLAN Conference on Programming Language Design and Implementation*, pages 1005–1021, 2020.

[46] D. D. McCarthy and P. K. Seidelmann. *Time: from Earth rotation to atomic physics*. Cambridge University Press, 2018.

[47] B. McKissock, P. Loyselle, and E. Vogel. Guidelines on lithium-ion battery use in space applications. https://ntrs.nasa.gov/api/citations/20090023862/downloads/20090023862.pdf, 2009.

[48] McMaster-Carr. Easy-to-Form Nickel Chromium. https://www.mcmaster.com/8880K82/, 2022.

[49] A. Mehrparvar, D. Pignatelli, J. Carnahan, R. Munakat, W. Lan, A. Toorian, A. Hutputanasin, and S. Lee. Cubesat design specification rev. 13. Technical report, California Polytechnic State University, San Luis Obispo, 2014.

[50] E. M. Middleton, S. G. Ungar, D. J. Mandl, L. Ong, S. W. Frye, P. E. Campbell, D. R. Landis, J. P. Young, and N. H. Pollack. The earth observing one (eo-1) satellite mission: Over a decade in space. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 2013.

[51] M. Nardello, H. Desai, D. Brunelli, and B. Lucia. Camaroptera: A batteryless long-range remote visual sensing system. In *Proceedings of the 7th International Workshop on Energy Harvesting & Energy-Neutral Sensing Systems*, ENSsys'19, page 8–14, New York, NY, USA, 2019. Association for Computing Machinery.

[52] S. Radu, M. Uludag, S. Speretta, J. Bouwmeester, A. Dunn, T. Walkinshaw, P. Kaled Da Cas, and C. Cappelletti. The pocketqube standard issue 1. Technical report, TU Delft, 2018.

[53] B. Ransford, J. Sorber, and K. Fu. Mementos: System support for long-running computation on rfid-scale devices. *Acm Sigplan Notices*, 47(4):159–170, 2012.

[54] Robin Hoel, Texas Instruments. FEC Implementation. https://www.ti.com/lit/an/swra113a/swra113a.pdf, 2007.

[55] E. Ruppel and B. Lucia. Transactional concurrency control for intermittent, energy harvesting, computing systems. In *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. ACM, 2019.

[56] SpaceX. Rideshare Payload User's Guide. https://storage.googleapis.com/rideshare-static/Rideshare_Payload_Users_Guide.pdf, 2022.

[57] ST. Pm0214 programming manual: Stm32 cortex-m4 mcus and mpus. Technical report, ST, 2020.

[58] StMicroelectronics. 9-axis inemo inertial module (imu): 3d magnetometer, 3d accelerometer, 3d gyroscope with i2c and spi. https://www.st.com/en/mems-and-sensors/lsm9ds1.html, 2015. Accessed: 2022-05-25.

[59] M. Surbatovich, L. Jia, and B. Lucia. I/o dependent idempotence bugs in intermittent systems. In *Proceedings of the 2019 ACM SIGPLAN International Conference on Object-Oriented Programming, Systems, Languages, and Applications*. ACM, 2019.

[60] M. Surbatovich, L. Jia, and B. Lucia. Automatically enforcing fresh and consistent inputs in intermittent systems. In *Proceedings of the 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation*, PLDI 2021, page 851–866, New York, NY, USA, 2021. Association for Computing Machinery.

[61] M. Surbatovich, B. Lucia, and L. Jia. Towards a formal foundation of intermittent computing. *Proceedings of the ACM on Programming Languages*, 4(OOPSLA), Nov. 2020.

[62] M. Swartwout. A statistical history of university-class satellites. 2018.

[63] Taoglas. AGGBP.SLS.25A – GPS/GLONASS/Galileo/BeiDou 25mm Active Patch with SAW/LNA/SAW. https://www.taoglas.com/product/aggbp-sls-25a-gps-glonass-galileo-beidou-25mm-active-patch-with-saw-lna-saw/, 2022.

[64] Texas Instruments. Data Whitening and Random TX Mode. https://www.ti.com/lit/an/swra322/swra322.pdf, 2022.

[65] Texas Instruments Inc. 4-bit bidirectional multi-voltage level translator for open-drain & push- pull. https://www.ti.com/product/LSF0204, 2021. Accessed: 2022-05-25.

[66] TI Inc. Products for msp430frxx fram. http://www.ti.com/lsds/ti/microcontrollers-16-bit-32-bit/msp/ultra-low-power/msp430frxx-fram/products.page, 2017. Accessed: 2017-04-08.

[67] TI Inc. User's guide msp430 gcc toolchain. https://www.ti.com/lit/ug/slau646f/slau646f.pdf, 2020. Accessed: 2022-05-25.

[68] K. S. Yıldırım, A. Y. Majid, D. Patoukas, K. Schaper, P. Pawelczak, and J. Hester. Ink: Reactive kernel for tiny batteryless sensors. In *Proceedings of the 16th ACM Conference on Embedded Networked Sensor Systems*, pages 41–53. ACM, 2018.

[69] J. Yiu. *The Definitive Guide to ARM Cortex-M3 and Cortex-M4 Processors*. Newnes, 2013.

[70] Zac Manchester. KickSat. http://zacinaction.github.io/kicksat/, 2015.

[71] D. Zogbi. Supercapacitors: A 25-year market review. https://www.tti.com/content/ttiinc/en/resources/marketeye/categories/passives/mezogbi-20200806.html, 2020. Accessed: 2021-11-11.