Power-Efficient Software Architecture for a CubeSat System

Samuel H. Russ
Department of Electrical and Computer Engineering
University of South Alabama
Mobile, AL, USA
sruss@southalabama.edu

Edmund Spencer
Department of Electrical and Computer Engineering
University of South Alabama
Mobile, AL, USA
espencer@southalabama.edu

Todd Scroggins
Department of Electrical and Computer Engineering
University of South Alabama
Mobile, AL, USA
tgs1621@jagmail.southalabama.edu

Saeed Latif
Department of Electrical and Computer Engineering
University of South Alabama
Mobile, AL, USA
slatif@southalabama.edu

***Abstract*— A 2U CubeSat, (20cm by 10 cm by 10 cm) is being developed to make impedance measurements of plasma in the ionosphere at 400 km altitude. Because the new instrument is able to make measurements rapidly, and because of the severe power and size constraints of the satellite, special care is needed in storing the measurement data and transmitting it to Earth. For both the measurements and for general housekeeping functions, the software of the CubeSat must be able to respond rapidly to events and operate in a manner that is not only flexible and extensible but also power efficient. This work outlines the process by which the software was designed to satisfy the constraints of the mission.**

Keywords—CubeSat, Software architecture, embedded systems, power-aware computing, energy-aware computing

I. Introduction

Designing an instrument to measure the properties of the ionosphere requires design of a space-borne instrument which, in turn, imposes severe limitations on size, power, communications bandwidth, and processor capability. Conversely, there is a clear scientific motivation for the instrument to gather as much data as possible. The trade-off of the two competing sets of goals drives the design of a suitable system.

A team of students and faculty at the University of South Alabama is designing and constructing a small-form-factor satellite or "CubeSat" to make the measurements. Specifically, the satellite will debut a new instrument called a Time Domain Impedance Probe (TDIP) which is designed to measure the impedance versus frequency of ionospheric plasma. [1]

This paper describes the software architecture of the satellite that was optimized for power efficiency and ease of development, and analyzes some of the data flows inside the satellite.

II. Review of Literature

A. Time Domain Impedance Probe

The goal of the instrument being designed is to make measurements of absolute electron density and electron neutral collision frequency, both properties of the plasma in Earth's ionosphere, at an extremely fine spatial resolution. The core of the technique is to make a single impulsive measurement, relying on the property that an impulse in the time domain contains frequency content over a broad range of frequencies [1].

As opposed to older techniques that sweep across frequencies (*i.e.* make a series of measurements at different frequencies), the new time-domain impedance probe (TDIP) samples the neighboring plasma in a single measurement taking 100 μs [1].

At the sampling speed employed by the TDIP, the satellite is able to make a measurement every 0.7 meters, at an orbital velocity of 7 km/s. Thus there is a substantial gap between the instrument's ability to gather data and the satellite's ability to send the data back to Earth. Further, the small size of the satellite constrains the area available for solar power, and that, in turn, imposes severe constraints on the power and energy available both for the instrument and for the computing infrastructure needed to operate the satellite autonomously.

B. Small satellites

A popular method for cost-effective deployment of space-borne instruments is to use small satellites known as "CubeSats" [2,3]. A typical CubeSat is a 10 cm cube, and many are larger. Often CubeSats are launched in connection with commercial satellite launches, and so the incremental cost to launch a CubeSat is nearly zero. The CubeSat being designed in this project is a 2U Cubesat, which means that it is the size of two stacked 10-cm. cubes (20 cm x 10 cm x 10 cm).

The unique problems of space systems, and especially the issue of transmitting large volumes of scientific data over relatively slow channels, has been widely studied [4].

III. Description of Instrument and Satellite

The TDIP is designed to be housed in the CubeSat, and the CubeSat will provide the systems necessary for the TDIP to function, including power and communications.

This work was funded by the NASA Undergraduate Student Instrument Project (USIP) Grant Number NNX16AK75A.

978-1-7281-6861-6/20/$31.00 ©2020 IEEE

A. *Time Domain Impedance Probe*

The new version of the TDIP is designed to make a measurement in 100 µs by sampling data at 40 MS/s. (The version described in the literature sampled at 5 MS/s [1] and future enhancements are being considered [5]).

This is accomplished by having two digital FIFOs feeding two 12-bit DACs (to create the waveform applied to the probe and to create a reference waveform) and having one digital FIFO at the output of a 12-bit ADC to capture the resulting signal. Since the signal being captured is the difference of the signal at the spacecraft probe and the reference signal, only one signal needs to be captured.

At 40 MS/s, the measurement requires the capture of 4000 digital samples, each 12 bits in size. Thus a single measurement is 6000 bytes large.

In a single orbit, if the instrument is operated continuously, there will be 61 million measurements. This amount of data will be too large to telemeter in a reasonable amount of time to ground. Thus the TDIP will only be operated in the dusk side between 1700 to 1900 MLT (mean local time), and at selected intervals during a month.

The complete TDIP consists of two main components. The first is a circuit board to house the analog circuitry, the FIFO chips, the ADCs and DAC, a microcontroller, and glue logic. The second is the actual probe antenna, which is designed to unfold in space.

B. *CubeSat Architecture*

The remainder of the CubeSat is designed to house the circuit board and probe and provide all of the necessary support functions.

Power is provided by a combination of solar cells, batteries, and a charging circuit. The board housing the circuit is called "EPS."

The satellite will need to operate at a known orientation and at known locations. These functions will be provided by an attitude determination and control system (ADCS) which will use magnetorquers (coils that provide torque in the Earth's magnetic field) and reaction wheels to orient the craft. The board housing the circuitry is called "ADCS."

The satellite will need a control system to maintain a schedule, monitor power and orientation, coordinate making measurements, and coordinate sending the data back to Earth. The board housing the microcontroller that performs these tasks is called Command and Data Handling or "C&DH". Since C&DH will manage the schedule, it has an on-board SD card to maintain it. (An SD card is preferred in case the satellite loses power.)

Communications will be provided through a third-party company. The software to establish and use a downlink to Earth runs on a Beagle Bone Black board, and the board runs Linux and has access to its own SD card for storage.

The communications system includes a low-bitrate omnidirectional simplex system, which functions almost like a text-messaging system, and a high-bitrate directional duplex system. The latter system can send data down to Earth at data rates up to 256 kilobits per second (kbit/s). While the simplex system operates in the L-band, the high-speed duplex system works in the S-band. Both systems use planar printed antennas without requiring any deployment mechanism. Radios will use Globalstar's 32 LEO satellites. Through the duplex system, the CubeSat will be able to communicate with three satellites at a time.

The architecture of the satellite is shown below in Fig. 1.

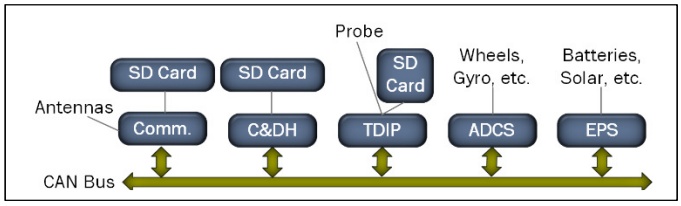

Fig. 1. Block diagram of the complete CubeSat. Note that both C&DH and the TDIP board have access to non-volatile storage.

For most of the communications between boards, the industry-standard CAN bus has been selected. CAN provides layers of error detection and correction, and has been widely adopted by the automotive industry [6]. It is also widely used in CubeSats because of its robustness [7].

As will be explained below, there are other communications options available for board-to-board communications, including CAN, SPI, I^2C, and UARTs. The Kinetis family of low-power microcontrollers was selected for the mission, and all four of these interfaces are supported.

C. *Design Constraints*

The satellite's design imposes several constraints.

One of the more obvious constraints is that of power and energy. The total energy is limited by the energy storage of the batteries, and the power is limited by the capacity of the on-board DC-DC converters.

Since the satellite is solar powered, tasks that require more power than the solar panels can provide and tasks that must be performed in the Earth's shadow must use the batteries. In such cases, the batteries will need to charge between tasks, but the duty cycle (the time between tasks) can be spaced out as needed to recharge.

One subtle power constraint is the Beagle Bone Black that runs the communications software. The board runs Linux and uses DRAM and, as a result, consumes 1-2 orders of magnitude more power than the other boards. (Another reason for the disparity is that the Kinetis family of microcontrollers is optimized for extremely low power consumption.)

Another constraint is space. The complete satellite will be a "2U" design, meaning it is the size of 2 10-cm cubes placed side-by-side, and so each circuit board is roughly a 10-cm square. As a result, the TDIP board does not have enough room for its own SD card.

Another constraint has to do with making measurements and communicating. The satellite probes should be perpendicular to

the direction of motion in order to make the TDIP measurement. In order to transmit, however, the satellite needs to be oriented with the antenna facing away from Earth. In other words, between making the measurement and transmitting the results, the satellite needs to change its orientation. Since physical motion requires energy, it is possible that the satellite will lose power during the time it is being reoriented. This, in turn, creates a requirement that the data be stored in a non-volatile medium while the satellite is being reoriented.

The only way to satisfy these constraints is to store the TDIP measurement data on the TDIP for subsequent transfer to the BeagleBoneBlack to Earth..

The remainder of the paper explains how the software was designed with the constraints in mind, and how the TDIP / BBB interface was designed and optimized.

IV. Software Architecture

The architecture of the JagSat software stack began with the design of the C&DH software.

A. *C&DH Software Architecture*

The original goal of the software architecture was to develop the software for the Command and Data Handling board. Since the C&DH is designed to manage the overall schedule of the satellite, it needs to receive updates from all of the other boards, maintain a schedule, and issue commands to all of the other boards. For example, it must notify TDIP when to make a measurement, notify ADCS went to re-orient to transmit scientific data, and notify TDIP and the BBB when it is time to transfer the data and send it to Earth.

The first observation was that DRAM is relatively power-hungry, and so the software was designed to fit inside the footprint of the Kinetis microcontroller's on-board SRAM. The second observation was that the C&DH must respond to a variety of events, and that the nature of the events may change as the software and hardware design proceeds. That is, the input must be flexible and extensible. The third observation was that scheduled events could be mapped in such a way to appear to be just like all other real-time events.

The software was then structured as follows.

First, each of the interrupt handlers for input events, such as the CAN interrupt handler, was designed to place incoming messages into a single, central queue of incoming events. The data structure of the elements of the queue was developed to include both the payload of the message and sufficient metatdata about the origin of the message to permit the rest of the software to understand the message's origin and significance.

Second, a periodic interrupt was created to poll the calendar. The calendar is the primary list of scheduled events, and a polling event entails a simple check to see if the time of the next event has arrived. If so, the event is then added to the central incoming-event queue. That is, the event is handled in a manner identical with all other events. This enables the rest of the software to process scheduled events in a manner completely uniformly with treatment of real-time events.

Third, a central dispatcher is being created to dequeue input events and route each to its appropriate handler. This enables the software developers to create and add handlers as needed, and to map key functions of the satellite to appropriate handlers. For example, one handler might be dedicated to determining the overall operational state of the satellite, while another might be dedicated to managing the batteries and solar panels. The handlers will require shared access to a global data structure that represents the overall running state of the C&DH board, and will have read and write access to the state. As another example, a message from the duplex communications system could require an update to the onboard schedule of events.

Fourth, an output queue can be created of outgoing commands and messages. These are the commands that each handler issues in response to state updates. For example, a scheduled event to make a scientific measurement will be processed by the schedule handler and result in a command to the TDIP board to perform a measurement cycle.

Fifth, the queue can be connected to another dispatcher that routes the outgoing messages to the correct communications interface. In other words, this dispatcher is the dual of the incoming-event dispatcher. One of the interfaces is a calendar-update handler, and so this is how the calendar is updated.

This architecture is illustrated below in Fig. 2.

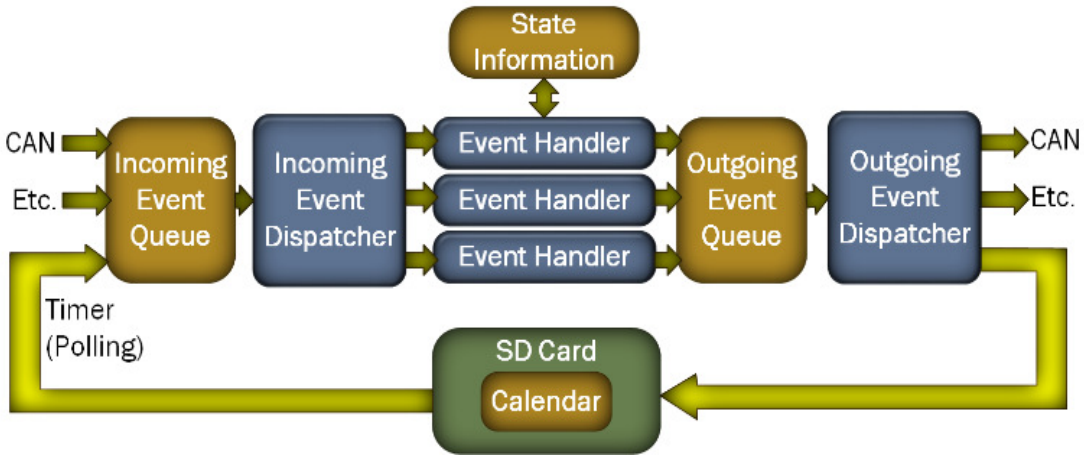

Fig. 2. Overall architecture of the C&DH software. This implements an efficient real-time control system.

One interface that is not shown in Fig. 2 is the interface to the GPS board. The GPS board will be mounted onto the C&DH board, and the interface to it will be via a serial port. The C&DH will be able to poll the GPS and update the last known location in the state information.

The resulting architecture is both extremely efficient, in terms of memory footprint, and extensible. New handlers can be added, and new communication paths can be enabled.

B. *Remaining Boards*

Once the design of the C&DH software was completed, the observation was made that essentially the same architecture could be used on all of the remaining boards.

Consider ADCS. The periodic-interrupt facility can be used to create a polling loop for the main control loop of the satellite. The loop will likely be based on an Enhance Kalman Filter, which performs best if updated in regular and periodic intervals. Once it reaches the time for an update, the timer handler for ADCS can poll all of the motion- and position-sensing elements, update its estimate of orientation, and issue commands as needed to actuators. This is illustrated below in Fig. 3.

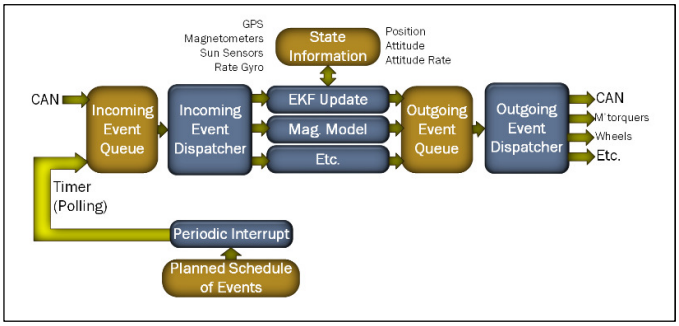

Fig. 3. Example of architecture ported to ADCS board. The state information includes position and orientation estimates, and some of the outgoing interfaces are directly to actuators to change position, such as magnetorquers and reaction wheels.

The exact rate gyro for the ADCS is still being determined, as testing results from the model initially selected was less than desirable. This is an example of the advantage of the current software approach – the handler for polling the rate gyro can be updated, without requiring any other software changes. Likewise, the selection of a control loop only affects a single software entity.

In the case of EPS, the polling facility can be used to update charge-status information and broadcast it to the rest of the satellite via CAN. It can also respond to commands from C&DH (received via CAN) to power down unneeded functions.

In the case of TDIP, it can respond to commands from C&DH to make (and store) scientific measurements and, when ready, send the measurements to the BBB for transmission to Earth.

With the architecture in place, it remains to be decided how to route the scientific data inside the satellite.

V. Handling Scientific Data

The process of handling the data acquired by the TDIP was designed taking the constraints into account.

A. *Board-to-Board Communications*

Communications between boards can be accomplished via one of four different interfaces.

The Serial Peripheral Interface (SPI) is a popular method for connecting embedded systems [8]. For example, NAND flash often supports a SPI interface, as do many magnetometers and accelerometers.

The basic data format is usually a single-byte address followed by some bytes of data, although formats vary considerably. For the purpose of this analysis, it is assumed that there is a one-byte address (or possibly data sequence number) followed by 16 bits of data.

The transmission speed of SPI can be relatively high (on the order of megabits per second), but the fact that the data is traveling between boards lowers the practical data rate. A speed of 200 kbit/s is assumed.

The Inter-IC (IIC or I^2C) bus is another popular embedded interface, often found with sensors. Its data format is one byte of device address, one byte of register address, and one byte of data. Because I^2C uses an open-drain driver, the speed is usually constrained to be on the order of 100 kbit/s, especially when traveling off-board [8].

The Controller Area Network (CAN) is found often in automotive applications, and was selected here as the primary control bus because of its robustness. A typical CAN frame is about 14 bytes in size and carries 8 bytes of data. Typical CAN speeds are 100 kbit/s to about 1 Mbit/s, and, since CAN was designed for automobiles, it can run considerable distances robustly [6]. A data rate of 200 kbit/s is assumed here.

Finally, RS-232 communications, via dedicated UARTs, is another option [8]. RS-232 typically only operates up to about 115 kbit/s. It natively has very low overhead, but also has no mechanisms for error detection or recovery. Here it is assumed a protocol like Kermit is in use, which has about 14% overhead [9].

The selection of a communications bus must take into account both the data rate and the overhead of the bus. For example, I^2C transmits 3 bytes on the bus for every 1 byte of data. The communication options are summarized below in Table 1.

The message size is the size of a single message on the bus. The payload size is the amount of data each message can carry, and the efficiency is the ratio of payload to message. The size of data, as dictated by TDIP, is 6000 bytes. Each bus then carries the combination of the 6000-byte payload and the overhead. The total time to transmit the payload can then be calculated.

SPI results in the fastest data transmission time. Its relatively high overhead is offset by its faster data rate. The UART option is quite fast – its relatively slow 115 kbps data rate compares well to other embedded standards – but CAN has an almost identical data-transmission time because its faster rate offsets its

higher overhead. Since CAN is already in use, and since CAN offers multiple levels of error detection and recovery (unlike a UART), its use should also be considered.

TABLE I. COMPARISON OF COMMUNICATION BUSES

Property	Communication Bus			
	SPI	*I²C*	*CAN*	*UART*
Message Size (Bytes)	3	3	14.375	1.16
Data Payload Size (Bytes)	2	1	7	1
Efficiency	66.7%	33.3%	48.7%	86.0%
Data Rate (bits/s)	200,000	100,000	200,000	115,200
Size of Data (Bytes)	6000	6000	6000	6000
Size of Message (Bytes)	9000	18000	12322	6977
Board-to-board transmit time (s)	0.360	1.440	0.493	0.484

B. Storage

There are two options for storing the data. The first is to store it temporarily on the TDIP board SD card, and the second is to store it on the Comms Beagle Bone Black SD card. While storing it on the Beagle Bone Black seems obvious – it also manages communications and so the data would be ready for transmission – the Beagle Bone Black also takes quite a bit of time to boot up and consumes quite a bit more current than the other boards. The trade-off of boot time and power consumption will be detailed below.

C. Data Transmission to Earth

The final step is the transmission of the data to Earth. At 256 kbit/s, the 6000-byte payload will take about 0.2 seconds to transmit. The radio will consume about 5 Watts of power while transmitting. This step is invariant – it must be taken regardless of the data storage method that is selected.

Data compression has been considered, but will likely not be implemented. Since this is the first time a TDIP will have been flown in orbit, the artifacts created by data compression are unknown, and there is a clear concern that the compression may average out the features that the probe detects.

Simpler operations, such as averaging results from multiple TDIP measurements, may be performed in flight. There is a trade-off that needs to be peformed, that of performing the calculations on the relatively slower, but much lower-powered, Kinetis microcontroller versus the much faster, but higher-powered, Beagle Bone Black. Benchmarking of the software to perform the averaging will be needed to determine which option consumes lesser energy (computation time times power consumption while computing).

D. Energy Budget and Selection of Architecture

There are three options for handling the scientific data. The first is to store the data on the TDIP board right after it is acquired, and then transfer it to the communications board at downlink time. The second is to transfer the data to the communications board right after it is acquired, then power down the communications board and reboot it at downlink time. The third is to transfer the data to the communications board right after it is acquired, then transmit the data soon afterwards. However, the satellite must be reoriented between data acquisition and transmission, so this option cannot be used.

In comparing the first two options, the difference between them lies in the board that stores the data. In the first option, the TDIP board stores the data, goes to sleep, wakes up, and transfers the data for downlink. In the second option, the communications board is awakened, stores the data, goes to sleep, wakes up, and transfers the data for downlink. In other words, the difference lies in the amount of energy required to operate the board used for temporary data storage (and to store the data temporarily).

The Kinetis-based TDIP board consumes about 20 mW of power and, because it is running a very lean run-time kernel, “boots” (that is, wakes up) in a few ms.

The communications board (which is a Beagle Bone Black embedded computer) requires about 10 seconds to boot and consumes 2.3 Watts while booting.

Thus the communications board expends about 23 Joules of energy to boot, while the TDIP board expends a few mJ. Thus using TDIP is clearly the preferred option.

Another clear advantage of the approach is that, because the communications system only has to be booted once, the energy to boot is amortized over both the data-transfer process from TDIP and the data transfer to Earth.

VI. CONCLUSIONS AND FUTURE WORK

Based on the analysis, the SPI interface offers the prospect of the fastest data rate. However, CAN offers layers of error detection and recovery and also has an advantage that it has already been selected for board-to-board communications.

Future work will be needed to address the downlink to Earth, taking into account the actual data rate of the link, the directionality of the pointing requirements needed to attain data rates, and the possibility of data compression.

ACKNOWLEDGMENT

The authors would like to thank the NASA Undergraduate Student Instrument Project (USIP) for funding this work.

REFERENCES

[1] E. Spencer et al., "First results from a time domain impedance probe for measuring plasma properties in the ionosphere," *2017 IEEE 60th*

International Midwest Symposium on Circuits and Systems (MWSCAS), Boston, MA, 2017, pp. 1284-1287.

[2] M. N. Sweeting, "Modern Small Satellites-Changing the Economics of Space," in *Proceedings of the IEEE*, vol. 106, no. 3, pp. 343-361, March 2018.

[3] Y. Miyazaki, "Deployable Techniques for Small Satellites," in *Proceedings of the IEEE*, vol. 106, no. 3, pp. 471-483, March 2018.

[4] S. Gao, Y. Rahmat-Samii, R. E. Hodges and X. Yang, "Advanced Antennas for Small Satellites," in *Proceedings of the IEEE*, vol. 106, no. 3, pp. 391-403, March 2018.

[5] E. Spencer, D. Clark, R. Gollapalli, S. Russ and B. Kerrigan, "A System On Chip design for fast time domain impedance spectroscopy," *2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)*, Boston, MA, 2017, pp. 124-127.

[6] S. Corrigan, *Introdcution to the Controller Area Network (CAN),* Texas Instruments Application Report SLOA101A, July 2008.

[7] E. Çınar and O. Turhan, "Command and data handling subsystem of GÖKTÜRK-2 flight model*," 2015 7th International Conference on Recent Advances in Space Technologies (RAST),* Istanbul, 2015, pp. 379-384.

[8] Y. Wang, J. Chen, and Y. Jiang, *Embedded Systems Design and Applications with Coildfire*, Accessed via https://community.nxp.com/docs/DOC-94931.

[9] F. da Cruz, "The Truth About Kermit File Transfer Performance", *Kermit News Number 5,* June 1993. Accessed via http://www.columbia.edu/kermit/perf.html