BAMA-1 Software Architecture, Implementation, and Testing

Jonathan S. Martini* and Alexandra L. Boehm.†
University of Alabama College of Engineering, Tuscaloosa, Al., 35401

CubeSats are small satellites of standardized dimensions that can be launched en masse from a single payload. They provide a platform for smaller research programs to investigate various topics in a space environment at significantly lower costs than was previously available. NASA encourages student teams to utilize this technology through their ELaNa (Educational Launch of Nanosatellites) program and selected the University of Alabama BAMA-1 CubeSat for the 41st mission. The BAMA-1 (Ballistic Atmospheric Measurement Apparatus) CubeSat was designed and developed for a demonstration of drag sail technology for deorbitting from an initial 500 km, 41° inclination orbit. The scope of this paper will cover the development of the BAMA-1 flight software suite, which includes communication, control, and process hypervisor software subsystems. The rapid development of the BAMA-1 flight software suite was successful due to the focus on iterative design and code re-usability. The processes which were used in development, the testing performed, and the final implementation of the flight software are presented in the work. This will offer a baseline model and starting point for future student CubeSat design teams.

I. Introduction

The BAMA-1 (Ballistic Atmospheric Measurement Apparatus) CubeSat was designed and developed to test drag sail technology and to support efforts of building a refined atmospheric model for altitudes around 500 km. Following payload separation, BAMA-1 will enter a circular orbit at an inclination of 41° and an altitude of 500 km. The drag sail will be deployed remotely after detumbling, and the CubeSat will record information about the orbit degeneration due to the drag sail. The data it gathers will be sent to the ground station in the form of two line elements (TLE) recorded from NORAD.

Drag sails offer a solution to the rising problem of space debris, as they allow satellites to rapidly and reliably deorbit. This clears orbits of satellites that have completed their missions and lowers risk for operational satellites, space stations, and crewed vehicles. BAMA-1 was intended to perform a drag sail technology demonstration to validate the effectiveness of the system, as well as to promote student participation in STEM projects.

Over three years, UASpace, an undergraduate student design team at the University of Alabama, created BAMA-1, from mission design to full product. This paper describes the development and final implementation of the BAMA-1 flight software, which consists of communication, control, and process hypervisor software subsystems that work in tandem with the electrical power subsystem and an ultra-high frequency radio. The simulation, implementation, and testing of the software suite are discussed in depth. This work will offer a starting point for future student design teams who are interested in creating and launching small, low-cost CubeSats.

II. Simulation

The BAMA-1 simulation consisted of a full digital twin of the CubeSat software that tested and proved the communications and control subsystems. The flexibility of the software was achieved through an extremely modular design that leveraged test-based iteration. Part of this digital twin was a model of the space environment. Initially, the space environment model was proven in MATLAB Simulink, but then re-implemented in Python3 to utilize ZMQ message passing for easier software development and compatibility. The digital twin test suite used CubeSat Protocol 1.4's ZMQ backend to facilitate rapid communication subsystem development, testing, and simulation without flight hardware. This simulation was then used for partial hardware in the loop (HIL) testing that validated the full mission configuration.

The dynamics of the CubeSat were modelled in 6DOF with 2-body orbital motion and rigid body rotational motion. The position $\vec{r} = [x, y, z]^T$ was determined by the two-body equation of motion, Eq. (1), where μ is the gravitational

*Undergraduate Student, Mechanical Engineering, UASpace Lead Software Developer
†Undergraduate Student, Mechanical Engineering, UASpace Flight Software Team Member

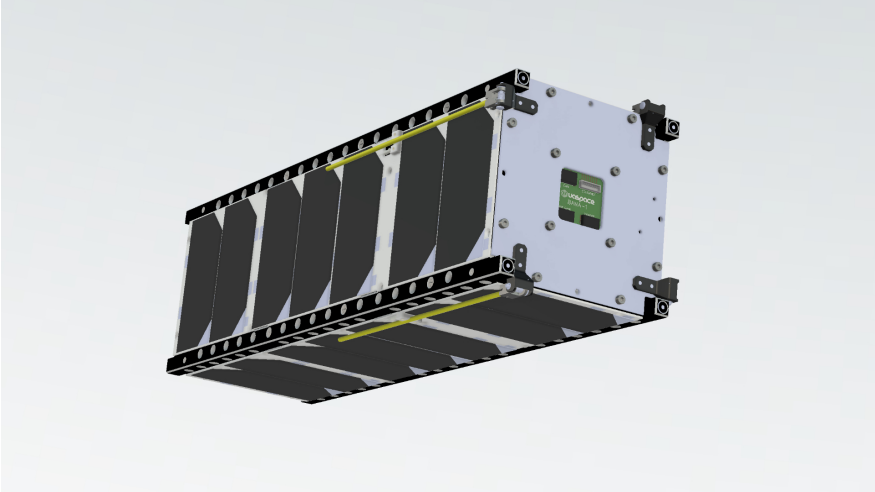

Fig. 1 BAMA-1 in the stowed hardware configuration.

parameter.

$$\ddot{\vec{r}} = -\frac{\mu}{r^2}\hat{r} \tag{1}$$

The rotational dynamics of the rigid CubeSat around its center of mass are modeled with Euler's equations, Eq. (2), where $\mathbf{I}$ is the inertia matrix and $\vec{\tau}_{ext}$ are the total external moments applied to the body in the body frame.

$$\mathbf{I}\dot{\vec{\omega}} = -\vec{\omega} \times \mathbf{I}\vec{\omega} + \vec{\tau}_{ext} \tag{2}$$

$\vec{\omega} = [\omega_x, \omega_y, \omega_z]^T$ is the vector of rotational rates of the body relative to the inertial frame expressed in components of the body frame. The external moments are composed of disturbance torques affecting the body and the control torque used to detumble the CubeSat from the initial rotation rates it experiences upon ejection from the rest of the payload.

The magnetic field of the Earth, as measured by the four on-board magnetometers, was simulated using the IGRF geomagnetic field model [1] with the most recent IGRF magnetic field values [2]. The geocentric Cartesian position coordinates from the two-body dynamics is mapped to the WGS84 geodetic coordinates for the IGRF model input. Radius from the center of the Earth, latitude, and longitude in degrees are are shown in Eqs. (3) to (5)

$$\rho = |\vec{r}| \tag{3}$$

$$LAT = 90° - \arccos\frac{z}{\rho} \tag{4}$$

$$LON = \arctan\frac{y}{x} \tag{5}$$

The IGRF model outputs the magnetic field vector in the North, East, Down (NDE) coordinate frame, which must be converted into the inertial frame with a 3-2-1 Euler rotation sequence:

$$\vec{b}_{xyz} = \mathbf{R}(\phi, \theta, \psi)\vec{b}_{NED} \tag{6}$$

$$\phi = 0 \tag{7}$$

$$\theta = \arccos\frac{z}{\rho} + \pi \tag{8}$$

$$\psi = \arctan\frac{y}{x} \tag{9}$$

The model, shown in Fig. 2, was implemented as a server application in Python that served IGRF sensor data to the control subsystem software. The control system served the requested magnetic moment to the simulation server which was then integrated into the next step and state of the model. This process was repeated indefinitely to test the

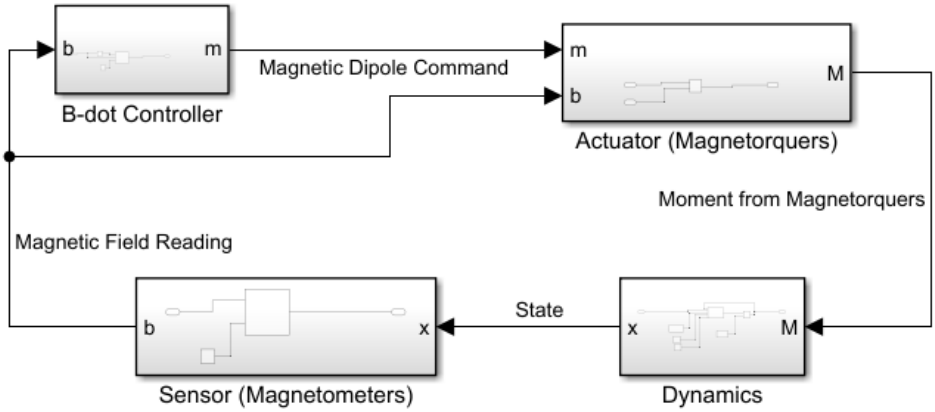

Fig. 2 BAMA-1 MATLAB Simulink Model

control system hardware and software with simulated values that reflect those measured in space. The following inertial matrix, Eq. (10), from the BAMA-1 CAD model was used in the simulation. The full view of the digital twin with the communications and control system simulations is displayed in Fig. 3.

$$I_{CAD} = \begin{bmatrix} 49194924.31 & 496.00 & -64338.02 \\ 496.00 & 48959062.48 & -183051.05 \\ -64338.02 & -183051.05 & 30754798.26 \end{bmatrix} gmm^2 \tag{10}$$

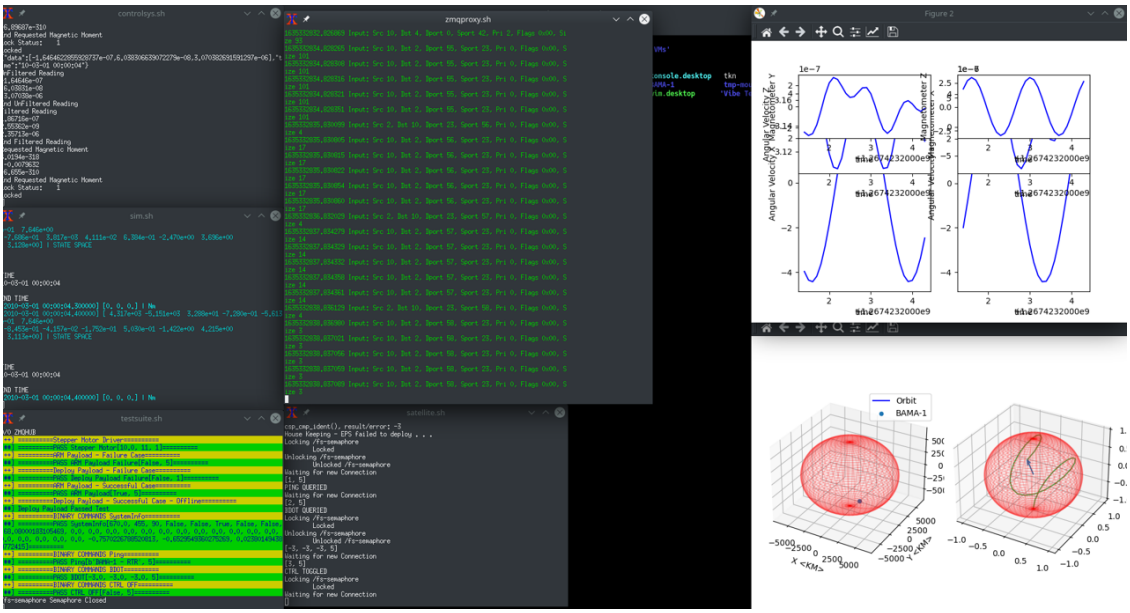

Fig. 3 Screen Capture of the digital twin running on non-flight hardware

In addition to the simulated space environment, the communications subsystem was tested through the CSP ZMQ proxy that simulated the embedded network. This allowed for offline communications development and validation. A test suite was developed with correct and erroneous command data to find flaws and edge case logic bugs that would cause the system to stall. This communications simulation was critical to proving a robust and reliable communications subsystem. It also made iterative development and extensions quick and easy to implement.

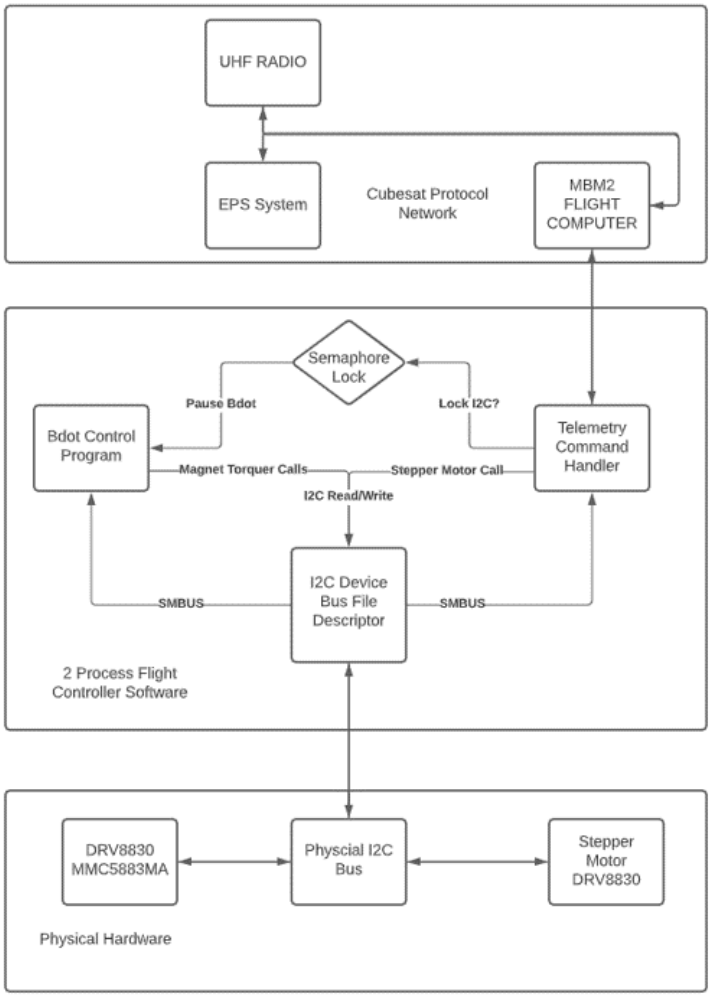

Fig. 4 BAMA-1 Hardware/Software Interaction Flowchart

III. Implementation

The implementation of BAMA-1 included developing the B-Dot attitude controller and communications subsystems to work in tandem through running blocking and non-blocking IO through a single I2C hardware bus used for sensor data acquisition and magnetorquer actuation. Both subsystems ran as separate processes under a failure watchdog hypervisor that executed a system reboot of the Pumpkin Aerospace MBM2 embedded Linux flight computer if failure arose in either of the processes. The B-Dot controller was written in C++ and the communications subsystem leveraged the Python3 C-bindings of Cubesat Protocol 1.4. Both processes accessed devices on a single shared I2C bus and were synchronized using an semaphore spin lock implemented in the B-Dot controller program, which was accessed through the communications subsystem program. This was required for this application because the communication subsystem featured a significant amount of blocking IO functionalities while the B-Dot controller was implemented as an asynchronous process with non-blocking IO. The interactions between both subsystems is illustrated in Fig. 4.

A. Communications Subsystem

The communications subsystem relied on Cubesat Protocol 1.4 (CSP) [3] for embedded system networking and message passing. CSP implemented a custom 5-bit classless interdomain routing (CIDR) system with the satellite segment spanning node addresses 0-15 and the ground segment spanning 16-31. Incoming packets were then routed to nodes based on the routing table of each receiving node by filtering the target address by matching most significant

(MSB) bits [3] demonstrated by Table 1. Packets continued through the network until they hit the specified node or were dropped.

MSB Bit Specifier	Addresses Routed
/0	All 32 Addresses Routed
/1	16 Addresses Routed
/2	8 Addresses Routed
/3	4 Addresses Routed
/4	2 Addresses Routed
/5	1 Address Routed

Table 1 Table of MSB bits with corresponding addresses routed.

The BAMA-1 communications subsystem was inspired by RESTFUL web service backends where each incoming request is routed to a session handler function. BAMA-1 utilized these features through filtering and passing incoming session connections to session handlers based on the incoming packet's target port as displayed in Fig. 5. For the design of BAMA-1 this included services to arm, deploy, and ping in addition to sub-services based on binary commands such as system information collection, control system toggling, rebooting, and other easily extendable functionalities that did not require standalone session handler functions. The session handler functions mimicked remote procedure call style of API where packet data is handled as function arguments. These session handler functions wrapped the functionality of the hardware or software procedure with the necessary formatting and I/O to perform the requested functionality. The full logic is demonstrated by the block diagram in Fig. 5 which shows how the communication subsystem initializes and handles connections.

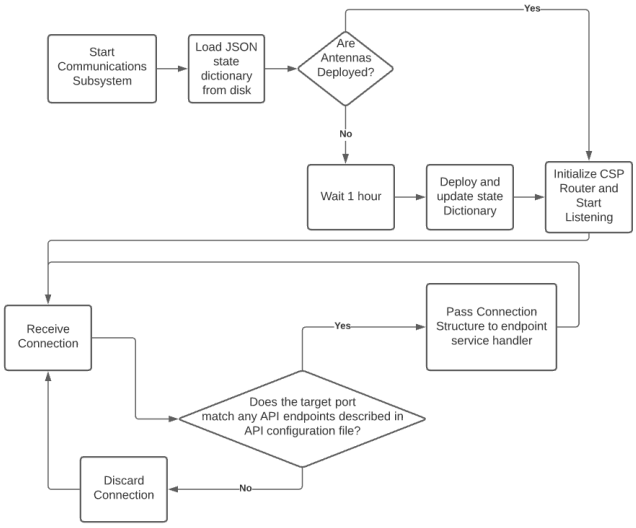

Fig. 5 BAMA-1 Communication Handler Flow Chart

For design purposes, CSP macros were developed to extend the usability of the CSP C Python bindings. The goal of this design choice was to enforce easy code modification and extension by handling all packet assembly and disassembly through single read and send functions. This allowed for faster iteration in designing the connection handler functions for each endpoint. It also allowed for simple debugging by minimizing the location of all data processing bugs to only two functions.

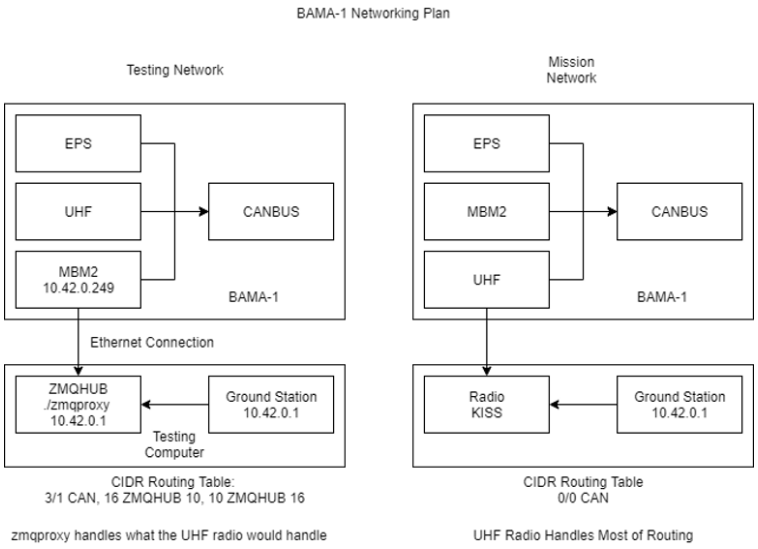

Fig. 6 BAMA-1 Networking Configurations

The digital twin and production system had two individual routing tables. Due to the timeline of the project, this was necessary as the radio subsystem was not ready until the final stage of development and testing. This allowed for flight software development and validation independent of the NanoAvionics UHF radio subsystem. The networking configuration is described in Fig. 6.

B. Control Subsystem

BAMA-1 uses a fully magnetic attitude control system to detumble the satellite from the high rotational rates it experiences following payload separation to much lower rates suitable for mission operations. This is accomplished using an implementation of the weighted B-dot controller [4], which seeks to counteract the rotation of the satellite by applying opposing torques.

The sensing suite consists of four magnetometers that measure the strength and direction of the Earth's magnetic field. This sensor data is filtered with a weighted moving average to smooth out sensor noise. The controller infers the rotation of the CubeSat from the changing magnetometer readings and calculates a dipole moment command for the magnetorquers as shown in Eq. (11).

$$\vec{m} = -\begin{bmatrix} 0.3 & 0 & 0 \\ 0 & 0.3 & 0 \\ 0 & 0 & 0.34 \end{bmatrix} \frac{\vec{\dot{b}}}{|\vec{b}|} \tag{11}$$

where $\vec{b}$ is the Earth magnetic field vector expressed in the body frame. The gains matrix was based on the maximum dipole the magnetorquers were rated to output. The magnetic dipole created by a magnetorquer is given by Eq. (12).

$$\vec{m} = nI\vec{A} \tag{12}$$

where n is the number of turns of the coil in the magnetorquer, I is the current through the wire, and $\vec{A}$ is the vector area of the coil. The direction of $\vec{A}$, thus the direction of the dipole moment, is determined by the direction of current through the coil as given by the right hand rule. The magnetorquers in the CubeSat lie along each major body axis, so each element of the magnetic dipole vector corresponds with the command for one magnetorquer.

This dipole moment will interact with the Earth's magnetic field and produce a control torque opposing rotational motion. The torque produced by the magnetometers on the CubeSat body is given (in the body frame) by Eq. (13).

$$\vec{\tau_m} = \vec{m} \times \vec{b} \tag{13}$$

The B-dot controller is implemented on the BAMA-1 CubeSat as a variable output controller. The derivative term of the B-dot controller was multiplied against the a diagonal matrix of the maximum magnetic dipole output possible. A

weighted moving average filter was also applied to the controller to allow for a smoother rate of change in the output command. Using the magnetorquer data sheet, the commanded dipole moments were matched to a voltage command for each magnetorquer.

The magnetic-only sensing and actuation system is offers limited precision, so the controller can only decrease the rotation of the spacecraft in the body frame to an angular rate equal to twice the mean motion of the its orbit [4]. This provides sufficient detumbling for the objectives of the BAMA-1 mission. This is evident in Fig. 7, which shows that even at high initial rotation rates, all axes rotating at $60°/s$, the control system was able to converge to a rate close to twice the orbital rate.

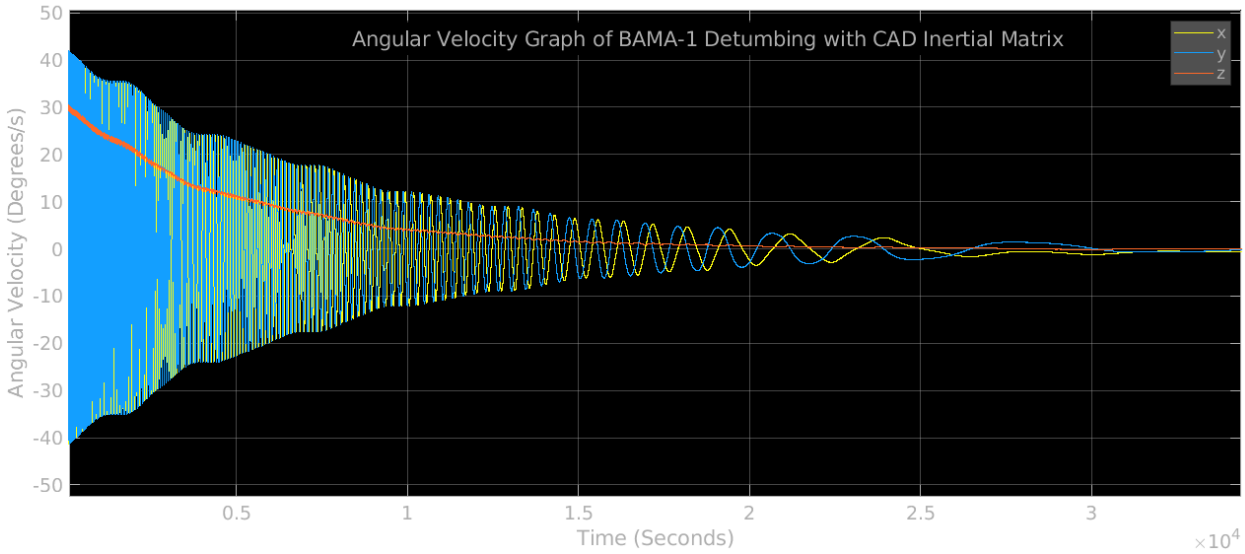

Fig. 7 BAMA-1 Detumbling Simulation with an initial angular rate of 60 deg/s in all axes.

C. Process Hypervisor and Process Failure Watchdog

The process hyper-visor is a simple Python script run as a Linux system daemon that has administrative root reboot privileges. It initializes each process and checks the process's PID status to confirm that the processes are still running. Both the control system and communications subsystems are infinite loops, so it can be assumed that any reason for the processes to die is the result of some system crash or failure and requires a reboot. All subsystem programs were designed to die on failure, which would invoke the hypervisor to initiate a reboot. A 5 minute window between failure detection and reboot was chosen to allow for easy debugging and recovery for testing, as well as allowing the ground station a chance to reconfigure the software states to prevent failures from occurring in the future. The block flowchart for this simple process is displayed in Fig. 8.

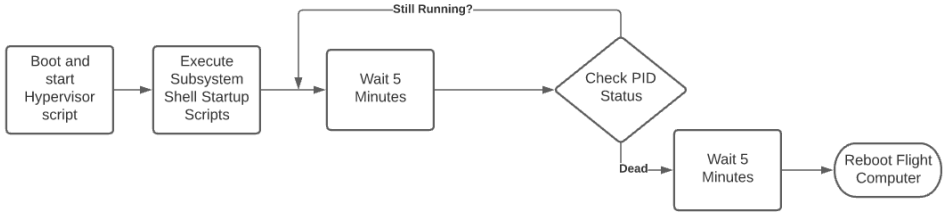

Fig. 8 BAMA-1 Hypervisor and Process Monitoring Flow Chart

IV. Testing

The software for the control system and communications system was designed with quick iteration and code re-usability in mind. For example, the control system leveraged a design pattern called dependency injection in C++. Dependency injection is when you create a class or function around a templated type that is expected to have the some implementation of standard methods and private variables. This allows for different test classes to be injected that perform tests or change the data input. The control system had two of these classes, one for the hardware input and one for the simulator input. This isolated the control logic from the input logic of the software which minimized the possibility for bugs when carrying over code from the simulator based controller to the final control subsystem software.

In addition, the digital twin allowed for fast implementation and validation of features. The test suite for the communications subsystem and control system simulator allowed for bugs to be found fast and changes to be implemented at a fast rate. For validation of flight software, not only was the digital twin used, but a several full mission simulations were done that tested the full extent of the flight software on mission hardware. This validated the work done and proved BAMA-1 before final integration.

V. Conclusion

The BAMA-1 CubeSat was successfully designed, developed, and manufactured over three years using the software implementation outlined in this work. By creating a digital twin of the CubeSat, its software was able to be tested as if it were in operation in order to verify and validate functionality. The B-dot controller for the detumbling system was also tested with a simulated space environment. The success of the software testing continued into successful hardware-in-the-loop testing for all software components, magnetometers, and magnetorquers. These systems, in conjunction with the success of the other satellite subsystems, would have allowed the CubeSat to reach a stable orbit after ejection from the rest of the ELaNa 41 payload.

On February 10, 2022, ELaNa 41 was launched at Cape Canaveral. Due to an issue during stage separation, the launch failed, and the payload never made it to orbit. Although BAMA-1 will not fulfill its mission objectives of testing drag sails for deorbiting, it has become a baseline model for future satellites and spacecraft projects from the University of Alabama and other student design teams.

References

[1] Compston, D., "International Geomagnetic Reference Field (IGRF) Model," *MATLAB Central File Exchange*, 2019. URL `https://www.mathworks.com/matlabcentral/fileexchange/34388-international-geomagnetic-reference-field-igrf-model`.

[2] Alken, P., Erwan, T., Beggan, C., Amit, H., Aubert, J., Baerenzung, J., Bondar, T., Brown, W., Califf, S., Chambout, A., Chulliat, A., Cox, G., Finlay, C., Fournier, A., Gillet, N., Grayver, A., Hammer, M., Holschneider, M., Huder, L., and Zhou, B., "International Geomagnetic Reference Field: the thirteenth generation," *Earth, Planets and Space*, Vol. 73, 2020. https://doi.org/10.1186/s40623-020-01288-x.

[3] GOMspace, "CubeSat Space Protocol (CSP) Network-Layer delivery protocol for CubeSats and embedded systems," , 2011. URL https://bytebucket.org/bbruner0/albertasat-on-board-computer/wiki/1.%20Resources/1.1.%20DataSheets/CSP/GS-CSP-1.1.pdf?rev=316ebd49bed49fdbb1d74efdeab74430e7cc726a.

[4] Fonod, R., and Gill, E., "Magnetic Detumbling of Fast-tumbling Picosatellites," 2018. URL https://www.iac2018.org/, 69th International Astronautical Congress, IAC 2018 ; Conference date: 01-10-2018 Through 05-10-2018.