# SilverSat Avionics Board

*Software and Command Description*

7 October 2024

Contact: lee@silversat.org

# INTRODUCTION

This document summarizes the hardware and software of the SilverSat Avionics Board. It briefly describes the interfaces with the SilverSat Power Board, Radio Board, Payload Board, and antenna. It also provides an overview of the commands received and sent by the Avionics Board.

## REVISION HISTORY

Version 1.0, Apr 11, 2023

Initial release

Version 1.1 Apr 14, 2023

Added command formats, reorganized sources

Version 1.2

Reformatted command outputs

Version 1.3 October 7, 2024

General update to reflect software changes

# TABLE OF CONTENTS

## OVERVIEW

SilverSat is an educational project for students in middle school and high school. The students are building and programming a 1U cubesat that will take photos from space and transmit them to a widely available media service, currently planned to be Twitter. SilverSat will launch from the International Space Station courtesy of an educational grant from NASA.

## HARDWARE

SilverSat hardware includes an EnduroSat structure surrounded by solar cells and four boards implementing the satellite capabilities. The boards are:

1. An EPS-I power board, also provided by EnduroSat, which stores power from the solar cells surrounding the structure and manages the distribution of power to the satellite
2. An Avionics Board, described in this document, which manages the timing of photo and communications on the Payload Board, processes commands from the ground, and manages the timing and content of beacon broadcasts
3. A Radio Board, which manages communications between the satellite and the ground station
4. A Payload Board, which captures images and communicates them to a media service, currently Twitter.

The Avionics Board, the Radio Board, and the Payload Board are developed and implemented by SilverSat. The antenna used by the satellite is provided by EnduroSat.

This document focuses on the hardware and software located on and accessed by the Avionics Board, including its microcontroller, watchdog timer, realtime clock, inertial measurement unit, and ferroelectric random-access memory. It also describes the interfaces to the Power Board, the Radio Board, the Payload Board, and antenna.

## SOFTWARE

The software running on the Avionics Board implements initialization, services the watchdog, manages beacon timing, executes commands from the ground, and controls the Payload Board. The software is implemented using the Arduino software

environment and is based on the standard setup() and loop() functions provided by Arduino.

### INITIALIZATION

When power is applied to the Avionics Board, the avionics loop software starts the Serial and logging functions used for board software development, monitoring, and testing. These functions may not be used in the flight software, depending on the final checkout procedures selected by the team.

The avionics loop then initializes the Avionics Board and the interfaces to the Power Board, the Radio Board, and the Payload Board.

The Avionics Board then pauses for a predefined deployment delay interval. In the flight software, this delay will be 45 minutes.

After the delay interval, the Avionics Board deploys the antenna, waiting to verify that deployment is successful. In sequence, the Avionics Board attempts two deployments using the I2C interface to the antenna and signals the Radio Board to attempt two deployments using a direct connection to the antenna. Initialization proceeds when the antenna shows that it has been opened successfully.

The Avionics Board then commences its standard process loop.

### PROCESS LOOP

The process loop, instantiated in the Arduino loop() function in avionics_loop.ino, includes the following steps:

1. Determine whether the process is within the watchdog window. If so, trigger the watchdog.
2. Check the IMU to determine whether the satellite is stable.
3. Determine whether the beacon interval has elapsed. If so, send a beacon.
4. Determine whether a command has arrived from the ground or the Radio Board. If so, process the command.
5. Determine whether it is time to take a photo. If so, signal the Payload Board to take the photo.
6. Determine whether the Payload Board is requesting a shutdown. If so, turn off power to the Payload Board after a delay.

The process then repeats.

### SOFTWARE ORGANIZATION

In addition to the overall structure provided by the Arduino setup() and loop() functions in the avionics.ino file, the software is organized into classes providing the required capabilities. Most classes are declared in a header file and defined in a cpp file. The exceptions are the Beacon, Message, and Response classes which are header only.

The PowerBoard, AvionicsBoard, RadioBoard, PayloadBoard, and Antenna functions and data are defined as classes.

Hardware such as the EPS-I power supply and the DS1337 real time clock are defined as classes. Some hardware has an additional abstraction and encapsulation layer, for example the PowerBoard and the ExternalRTC, respectively, for the devices just noted. This is also the case for the ExternalWatchdog and the IMU.

Command processing is controlled by the CommandProcessor class, which acquires commands from the RadioBoard, validates and parses them and retrieves a command object from the CommandWarehouse class. The CommandProcessor then executes the acknowledgement and processing methods in the command object, taking advantage of the virtual function capability provided by the C++ language.

The log_utility class generates standard log entries to support the development process. Logging will not be included in the flight software unless it is required for final testing after the vibration test.

## TESTING INFRASTRUCTURE

The silver-sat/Avionics_Software github repository contains versions of the software used for prototyping, unit testing, and function testing in the AvionicsTesting1, AvionicsTesting2, and AvionicsTesting3 directories. The readme files in those directories provide additional information.

The FlatSat testing version of the software is located in the FlatSat directory in the repository. In addition to the avionics_loop software, it also contains test simulators for the Antenna, the Power Board, and the Payload Board. An extensive set of tests for initialization, board interfaces, commands, and error conditions are located in the test_avionics directory.

The Flight directory contains the flight version of the software, implemented with avionics.ino as the main sketch. Test cases are provided in the test_satellite directory.

These tests are structured to be run as a complete set beginning with board initialization or can be run individually. They are written in Python and use pytest as the test driver. A

common.py module provides convenience functions. Instructions for setting up and executing the tests are in the readme.

Also present in the repository is a Doxygen file that can be used to generate detailed documentation of the software. Again, instructions are in the readme.

## AVIONICS BOARD HARDWARE

The Avionics Board hardware is defined in the Avionics Board repository.

The microcontroller is a Microchip SAMD21G18 32-bit Cortex-M0+ with 256K of flash and 32K of random access memory. The microcontroller runs the avionics loop software in the Arduino runtime environment. The Avionics Board includes an external Texas Instruments TPS3813-Q1 watchdog timer, a Maxim Integrated DS1337 real time clock, an Adafruit MPU-6050 inertial management unit, an Infineon Technologies 32K byte CT15B256J ferroelectric random access memory, and switches to control access to external I2C devices and two serial links. The critical I2C for the realtime clock and the microcontroller serial link are always enabled.

### MICROCONTROLLER

The microcontroller pins are defined in the Arduino variant.h and variant.cpp files in the repository. There is no crystal on the Avionics Board, and the Ardunio environment must be configured to reflect that. The microcontroller software can use a bootloader or can be programmed with a hardware device.

If a bootloader is used, it must be aware of the watchdog and either trigger or disable it during program loading. The load address of the software must be updated to reflect the presence or absence of a bootloader. Recent versions of the software have used a J-Link EDU Mini to program the Avionics Board to avoid problems with the bootloader and watchdog interaction.

A timer in the microcontroller is used for beacon interval timing. The internal real time counter is not used in this capacity, thus enabling beacons before the clock is set. The internal watchdog in the microcontroller is not used.

### WATCHDOG

The external watchdog provides a lower bound and an upper bound for a window and must be triggered within the window to prevent a reset. Because the watchdog is always active, the appropriate resistor must be shorted to enable software loading on the flight

hardware.

### REAL TIME CLOCK

The DS1337 provides real time clock (RTC) services for the satellite, specifically payload photo timing. Since there is no power available when the satellite is in transit to orbit, the clock must be set by a ground command after the satellite is deployed. The SetClock command serves this purpose.

### INERTIAL MANAGEMENT UNIT

The inertial management unit (IMU) is an accelerometer and gyroscope combination. It is used to determine the stability of the satellite. Stability status is transmitted in the beacon and can be retrieved with the GetTelemetry command.

### FRAM

The ferroelectric random access memory (FRAM) is initialized but not used by the Avionics Software.

## POWER BOARD INTERFACE

The Power Board stores, transfers and distributes power generated by the solar panels. The firmware on the Power Board manages the power flow and battery heaters as required. The Avionics Board reads Power Board status via the non-critical I2C bus from the board and transmits it to the ground in response to ground commands (GetPower) and in the power beacon character.

The Power Board has an I2C interface to modify its operation. That interface is use to set the status of some power lines and to cycle power for the Radio Board should the be required. The Power Board also has an interface implementing the EnduroSat satellite management communications protocol. That interface and protocol are not used.

## RADIO BOARD INTERFACE

The Avionics Board communicates with the Radio Board over the Serial1 interface. Avionics  receives commands from the ground, sends local commands to the Radio Board, receives ACKs and RESponses to those local commands, and receives local commands from the Radio on the interface.

This interface is always active and the Avionics Board expects traffic to occur at any time.

Ground commands are collected and the digital signature is validated. The command is then parsed, validated and executed. Some commands generate local traffic with the Radio Board.

The Avionics Board sends a local status request to the Radio Board when a GetComms command is received from the ground station. It sends a local halt command upon receipt of a TweeSlee command.

The Avionics Software also sends beacon local commands to the Radio Board, which adds its status character before transmitting the beacon in morse code. During initialization, the Avionics Software will direct the Radio Board to execute the backup antenna deployment procedure should it be required.

The Avionics Board sends local commands to the Radio Board on receipt of the radio command shown in the table below.

The Radio Board sends a local response to the status request and radio commands. The Avionics Board converts these responses into messages and forwards them to the ground (via the radio).

The Radio can send a local command to Avionics if it requires a power cycle.

## SATELLITE COMMANDS

### GROUND COMMANDS

Commands are transmitted from the ground, through the ground and satellite radios, and then to the Avionics Board via the Serial1 interface.

Commands are KISS encoded and KISS escaped, although the current implementation does not require escapes since commands and arguments are UTF-8 encoded text.

Each ground command arriving at the satellite will have the second byte (after the initial FEND) set to 0xAA to distinguish it from local traffic, which has a second byte of 0x00. Local traffic includes radio responses to commands from the Avionics Board to the Radio Board and the reverse. These commands and responses are described in the Silversat Local Commands document.

Ground commands are case sensitive and shown in the tables below. Arguments are UTF-8 encoded and blank separated.

Ground commands are signed to insure integrity. They consist of a 32-byte HMAC, an 8-byte salt, a 4-byte sequence number, and the command. The HMAC, salt, and sequence are converted to the UTF-8 representation of their hex values for transmission. They double in length under this transformation.

Ground commands sent to the Avionics Board receive an ACK and a RESponse, which is dependent on the command. Invalid and unknown commands, including those with the wrong number or type of parameters, may receive a NACK rather than an ACK and no response. Commands can receive an ACK and fail to execute.

Command length is limited to 256 bytes or the available space in the frame buffer.

## CHANGE SATELLITE STATE

| COMMAND | PARAMETERS | DESCRIPTION |
|---|---|---|
| SetClock | Year, month, day, hour, minute, second: integers in UTF-8 encoding | Set realtime clock |
| BeaconSp | Interval in seconds: integer in UTF-8 encoding | Set beacon spacing |
| PicTimes | Year, month, day, hour, minute, second: integers in UTF-8 encoding | Set time for photos; queue length is 5 |
| ClearPicTimes | | Empty PicTimes queue |
| UnsetClock | | Unset the real time clock |

## GET SATELLITE STATE

| COMMAND | PARAMETERS | DESCRIPTION |
|---|---|---|
| ReportT | | Respond with real time clock setting; subsumes GetTime |

| | | |
|---|---|---|
| GetPicTimes | | Respond with photo schedule |
| GetTelemetry | | Respond with telemetry |
| GetPower | | Respond with power status |
| GetComms | | Respond with Radio Board status |
| GetBeaconInterval | | Respond with beacon interval |

## INVOKE SATELLITE OPERATION

| COMMAND | PARAMETERS | DESCRIPTION |
|---|---|---|
| NoOperate | | Acknowledge; subsumes Ping |
| SendTestPacket | | Reply with test message |
| PayComms | | Start Payload Board in communications mode; subsumes Begin tweet |
| TweeSlee | | Stop Payload Board activity; subsumes Halt |
| Watchdog | | Force watchdog timeout |

## INVALID AND UNKNOWN COMMANDS

| COMMAND | PARAMETERS | DESCRIPTION |
|---|---|---|
| Invalid | | Invalid; used for testing |
| Unknown | | Unknown; command not |

| | | |
|---|---|---|
| | | recognized |

## RADIO COMMANDS

| COMMAND | PARAMETERS | DESCRIPTION |
|---|---|---|
| ModifyMode | Mode index: 1 UTF-8 digit | Modify radio mode |

## DEPRECATED COMMANDS

| COMMAND | PARAMETERS | DESCRIPTION |
|---|---|---|
| Halt | | See TweeSlee |
| s_call_sig | | Set call sign |
| g_call_sig | | Get call sign |
| GetPhotos (GPC) | | Reply with photo count |
| ModifyFrequency | Frequency: 9 UTF-8 digits | Modify radio frequency |
| AdjustFrequency | Frequency: 9 UTF-8 digits | Adjust radio frequency temporarily |
| TransmitCW | Duration: 2 UTF-8 digits | Transmit carrier wave |
| BackgroundRSSI | Duration: 2 UTF-8 digits | Report average RSSI |
| CurrentRSSI | | Report current RSSI |
| SweepTransmitter | Start frequency: 9 UTF-8 digits, stop frequency: 9 UTF-8 digits, steps: 3 UTF-8 digits, dwell: 3 UTF-8 digits | Transmit carrier over range of frequencies |

| | | |
|---|---|---|
| SweepReceiver | Start frequency: 9 UTF-8 digits, stop frequency: 9 UTF-8 digits, steps: 3 UTF-8 digits, dwell: 3 UTF-8 digits | Measure RSSI over range of frequencies |
| QueryRegister | Register: 5 UTF-8 digits | Report register value |

# COMMAND FORMATS

| COMMAND | FORMAT | ACK | RESPONSE |
|---|---|---|---|
| SetClock (SRC) | "SetClock" + ' ' +<br>year + ' ' +<br>month + ' ' +<br>day + ' ' +<br>hour + ' ' +<br>minute + ' ' +<br>second<br><br>Arguments are decimal integers in UTF-8 encoding and extra blanks are ignored | "ACK " + last good command sequence number | "RES SRC" |
| BeaconSp (SBI) | "BeaconSp" + ' ' +<br>interval<br><br>Interval in seconds: decimal integer in UTF-8 encoding | "ACK " + last good command sequence number | "RES SBI" |
| PicTimes (SPT) | "PicTimes" + ' ' +<br>year + ' ' +<br>month + ' ' +<br>day + ' ' +<br>hour + ' ' +<br>minute + ' ' +<br>second<br><br>Arguments are decimal integers in UTF-8 encoding | "ACK " + last good command sequence number | "RES SPT" |
| ClearPicTimes (CPT) | "ClearPicTimes" | "ACK " + last good command | "RES CPT" |

| | | | |
|---|---|---|---|
| | | sequence number | |
| UnsetClock (URC) | "UnsetClock" | "ACK " + last good command sequence number | "RES URC" |
| ReportT (GRC) | "ReportT" | "ACK " + last good command sequence number | "RES GRC " + ISO 8601 timestamp in UTF-8 encoding |
| GetPicTimes (GPT) | "GetPicTimes" | "ACK " + last good command sequence number | "RES GPT " + Zero or more timestamps in UTF-8 encoding |
| GetTelemetry (GTY) | "GetTelemetry" | "ACK " + last good command sequence number | "RES GTY" + " AX " + x-axis acceleration + " AY " + y-axis acceleration + " AZ " + z-axis acceleration + " RX " + x-axis rotation + " RY " + y-axis rotation + " RZ " + z-axis rotation + " T " + temperature<br><br>Values are floating point, UTF-8 encoded. Acceleration is in meters per second per second. Rotation is in radians per second. Temperature is in degrees Celsius. |

| | | | |
|---|---|---|---|
| GetPower (GPW) | "GetPower" | "ACK " + last good command sequence number | "RES GPW" +<br>" BBV " + battery voltage +<br>" BBC " + battery current +<br>" TS1 " + temperature of sensor 1 +<br>" TS2 " + temperature of sensor 2 +<br>" TS3 " + temperature of sensor 3 +<br>" 5VC " + 5 volt current +<br>" H1S " + heater 1 state +<br>" H2S " + heater 2 state +<br>" H3S " + heater 3 state<br><br>Values are floating point, UTF-8 encoded. Voltage is in volts. Current is in Amperes. Temperature is in degrees Celsius. Heater state is true or false. |
| GetComms (GRS) | "GetComms" | "ACK " + last good command sequence number | "RES GRS " + (data to be defined by the Radio Board team) |
| GetBeaconInterval (GBI) | "GetBeaconInterval" | "ACK " + last good command sequence number | "RES GBI " + interval<br><br>Value is an integer. Interval is in seconds. |

| | | | |
|---|---|---|---|
| NoOperate (NOP) | "NoOperate" | "ACK " + last good command sequence number | "RES NOP" |
| SendTestPacket (STP) | "SendTestPacket" | "ACK " + last good command sequence number | "RES STP test" |
| PayComms (PYC) | "PayComms" | "ACK " + last good command sequence number | "RES PYC" |
| TweeSlee (TSL) | "TweeSlee" | "ACK " + last good command sequence number | "RES TSL" |
| Watchdog (WDG) | "Watchdog" | "ACK " + last good command sequence number | "RES WDG" |
| ModifyMode (RMM) | "ModifyMode" + ' ' + index<br><br>Index is 1 decimal UTF-8 digit | "ACK " + last good command sequence number | "RES RMM " + index<br><br>Index is 1 decimal UTF-8 digit |

Extra blanks are ignored in all commands.

All messages are KISS encoded and start and end with a FEND. The command byte is 0xAA for commands received from the ground and 0xAA for ACKs, NACKs, and RESponses sent to the ground. Local commands are described below.

Commands may return a NACK and no response or the error response "RES ERR" if an error is encountered. NACKs are followed by the last good command sequence number if appropriate.

### LOCAL COMMANDS

Local commands are sent from the Avionics Board to the Radio Board and from the Radio Board to the Avionics Board via the Serial1 interface.

Commands are KISS encoded and KISS escaped, although the current implementation does not require escapes since commands and arguments are UTF-8 encoded text.

Each local command will have the second byte (after the initial FEND) set to 0x00 for local ACKs, local NACKs, and local RESponses, or to one of the values in the table below.. These commands and responses and the command arguments are described in detail in the Silversat Local Commands document.

The Avionics Board does not send local ACKs, local NACKs, or local RESponses. It ignores local ACKs and NACKs.

Commands arguments are UTF-8 encoded and blank separated.

Command length is limited to 256 bytes or the available space in the frame buffer.

Local commands are not digitally signed.

| COMMAND | VALUE |
|---|---|
| Local Frame | 0x00 |
| Remote Frame | 0xAA |
| Beacon | 0x07 |
| Release Antenna | 0x08 |
| Get Status | 0x09 |
| Halt | 0x0A |
| Modify Mode | 0x0C |

| | |
|---|---|
| Cycle Radio 5V | 0x0F |

## PAYLOAD BOARD INTERFACE

The Avionics Board communicates with the Payload Board via nine digital IO pins. The three input pins are voted, and the board determines state based on the majority of the pins.

Upon startup, the Avionics Board sets the "payload on" output pins high, or off. The hardware sets the "shutdown" pins high, or shutdown requested.

When the time for a picture arrives, or when the ground sends a command to communicate, the Avionics Board sets the payload "states" pins appropriately and then seat the three payload on pins low.. These pins are then voted on the Payload Board and power is enabled. The Avionics Board then monitors the shutdown lines. When they go low, it is an indication that the Payload Board is in user state.

When the Payload Board has completed its work, it raises the three shutdown pins. Again, the three shutdown pins are majority voted. When the Avionics Board senses the shutdown request, it delays a predetermined interval to allow the Payload Board to complete shutdown, then turn off the payload power.

The Avionics Board monitors for Payload Board sessions exceeding a maximum time and for a Payload over current fault.

## ANTENNA INTERFACE

The Avionics Board communicates with the antenna via the external I2C interface.

During startup, the Avionics Board initializes the satellite, waits for the deployment delay, then executes the antenna deployment protocol. Antenna Algorithm 1 is executed for all doors. The Avionics Board then waits for a predetermined period and reads the status of the doors. If all are open, execution proceeds.

If one or more doors are closed, the Avionics Board executes Antenna Algorithm 2 for all of the doors. It then waits a predetermined interval and reads the status of the doors. If all are open, execution proceeds.

If one or more doors are closed, the Avionics Board sends a local command to the Radio

Board to execute the backup antenna deployment procedure. If one or more doors are closed, the Avionics Board sends a second local command to the Radio Board to execute an alternate backup deployment procedure.

## SOURCES

### TOOLS

Arduino IDE https://www.arduino.cc/en/software

Arduino CLI https://blog.arduino.cc/2020/03/13/arduino-cli-an-introduction/

Visual Studio Code https://code.visualstudio.com/

KiCad https://www.kicad.org/

pytest https://docs.pytest.org/en/7.3.x/

Doxygen https://www.doxygen.nl/

### LIBRARIES

Arduino Log https://github.com/thijse/Arduino-Log

Arduino Cryptography https://rweather.github.io/arduinolibs/crypto.html

### REFERENCES

SilverSat https://silversat.org/

Avionics Board Hardware https://github.com/silver-sat/Avionics_Board

Avionics Board Software https://github.com/silver-sat/Avionics_Software

SilverSat Local Commands https://docs.google.com/document/d/1Vwpk0ab0HoC62mU7A1fQwpmhmtmZO0VwPtXjQipe0v0/edit?usp=sharing

SilverSat github https://github.com/silver-sat

The Power Board and Antenna description documents are held under a non-disclosure agreement with EnduroSat.

KISS Protocol http://www.ax25.net/kiss.aspx

Arduino Language https://www.arduino.cc/reference/en/

SAMD21 Datasheet
https://ww1.microchip.com/downloads/en/DeviceDoc/SAM_D21_DA1_Family_DataSheet_DS40001882F.pdf

Watchdog Datasheet
https://www.ti.com/lit/ds/symlink/tps3813-q1.pdf?ts=1681244431222&ref_url=https%253A%252F%252Fwww.google.com%252F

Real Time Clock Datasheet
https://www.analog.com/media/en/technical-documentation/data-sheets/ds1337-ds1337c.pdf

Inertial Management Unit Overview
https://learn.adafruit.com/mpu6050-6-dof-accelerometer-and-gyro

Ferroelectric Random Access Memory Datasheet
https://www.infineon.com/dgdl/Infineon-CY15B256J-SXE-DataSheet-v04_00-EN.pdf?fileId=8ac78c8c7d0d8da4017d0ee624b26e3a

Wikipedia ASCII Definition https://en.wikipedia.org/wiki/ASCII

CPP Reference https://en.cppreference.com/w/

Python.org https://www.python.org/

## LEARNING

Adafruit https://www.adafruit.com/

Learn CPP https://www.learncpp.com/

CircuitPython https://circuitpython.org/