Real-time Embedded Mission Science Payload for Miniaturized Space Research Applications

Author:
David E. Smith
des460@nyu.edu

In development with:

NYU CubeSAT
Matt Cocca, Dymytro Moyseyev, Abhimanyu Ghosh, Danny Chiang, David E. Smith

1.1 Forward This paper is produced as part of an interdisciplinary collaboration amongst five engineering students as part of the NYU CubeSAT design project at Tandon School of Engineering. The work in this paper is the sole work of the author, unless otherwise noted. Where included, all works by team members who are not the author can be found directly on the page, appropriately next to their contribution. This paper is limited in scope only to include the parts of project which the author himself worked on, and will only briefly detail portions outside of that scope as needed. For more information on those portions not detailed within, please contact the appropriate team member.

All code written for this project can be found on the projects github at http://github.com/EasonNYC/

1.2 Abstract With the shrinking of digital electronics and an active internet of resources connecting students can collaborate with planetary scientists and space-faring engineers, the ability for universities and even high school students to design and send meaningful scientific experiments up into space at significantly reduced cost has become a 21st Century reality. In 1999, a collaboration between California Polytechnic State University (Cal Poly) and Stanford University led to the creation of the CubeSAT specification.[2] A CubeSAT is a 10cm x 10cm x 10cm (1-U sized) satellite which has a mass of up to 1.33 Kg. [3] The CubeSAT platform was designed to promote and develop the launching of small satellites into low earth orbit (LEO) by both large and small academic, commercial and even amateur parties. As part of NASA's own CubeSat Launch Initiative, NASA provides an avenue for small CubeSAT payloads built by universities, high schools and non-profit organizations to fly and deploy on upcoming rocket launches.[1] This project is intended to be a low cost implementation of a mission science payload which might one day fly on one of those rockets, as part of the NYU CubeSAT design implementation.

2. Introduction The scope of this project covers the hardware and software integration of two PCB-based mission science modules which are to be implemented on NYU CubeSAT, a small 1-U Sized CubeSAT built at NYU Tandon School of Engineering. The mission science payload modules: Referred to throughout as modules "A" and "B" are located as the two middle PCB sections of the satellite. This area was chosen because it was equally close to both the battery and radio modules, because it would offer better placement of an Inertial Measurement Unit within the center of the satellite, and for maximal natural shielding from radioactive particle collisions within a highly active aerospace environment.

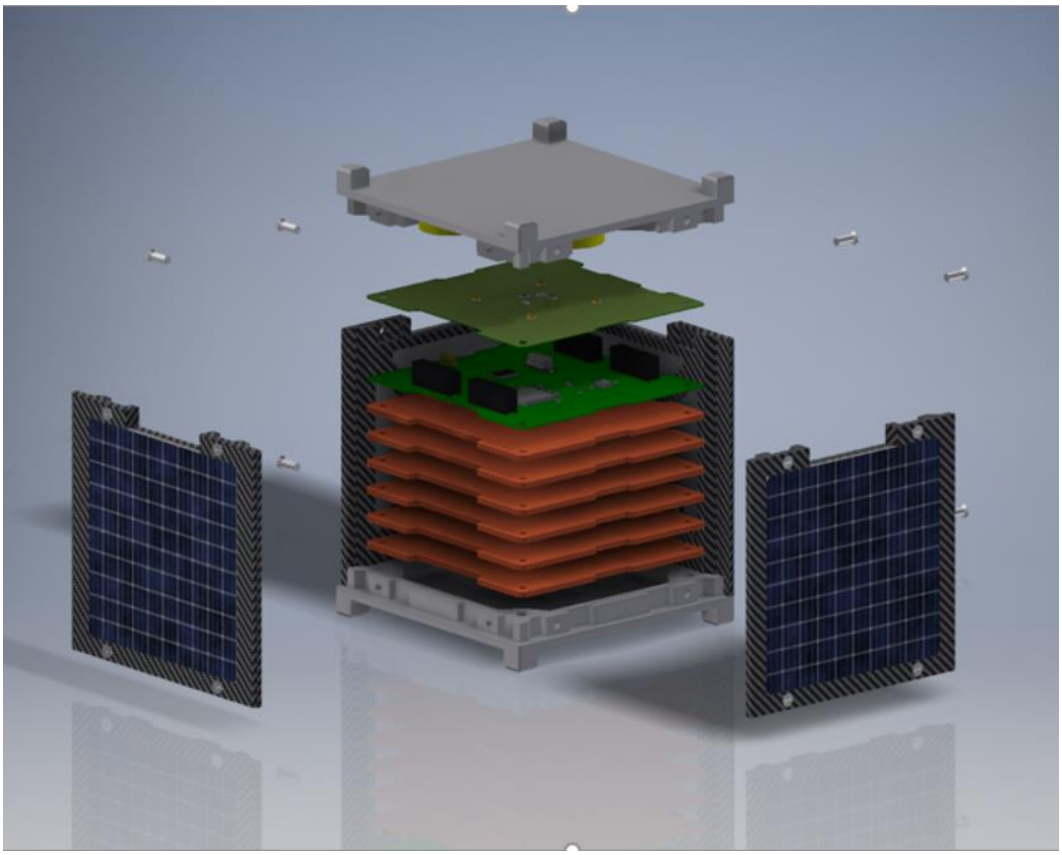


Figure 1. 3D rendering of NYU CubeSAT. Structural design and photo by Matt Cocca

2.1 Design Constraints In order to collaborate effectively, a few key design constraints were developed by the team and followed during the development of mission science payload modules A and B. Notable constraints followed were:

Voltage rail: VBattfor use by Payload Modules:
Vin Max: 16.8V
Vin Min: 14.8V
Max Current Draw: 3A

Each module on the CubeSAT, including the MS payload, is based from the same “standard” schematic and PCB footprint. This standard was developed by Matt Cocca, with coordination and input from all members as the development process continued. The standard was designed so that the PCB modules form interlocking “stack” which routes the major communication and power arteries throughout the satellite, limiting opportunities for a routing conflict between sub designs and modules. The schematic and operating PCB from which the mission science payload is based can be seen in the figures below:

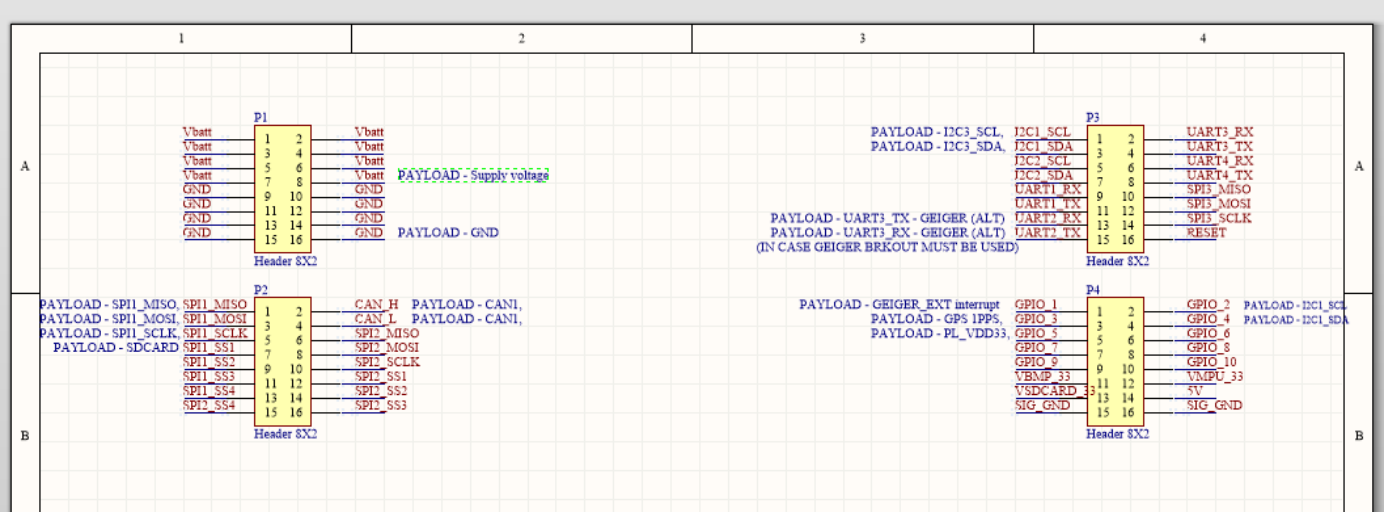


Figure 2. Schematic of the PCB Standard developed by Matt Cocca. Annotations by David E Smith.

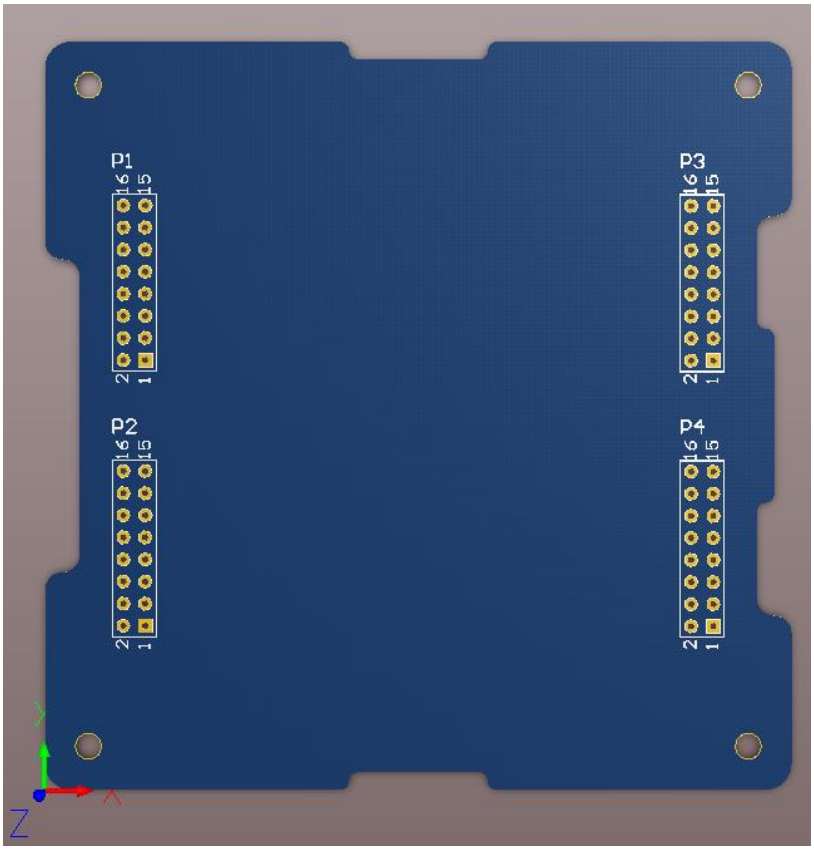


Figure 3. Base layout of the PCB Standard. Design by M. Cocca

2.2 Development Strategy Because of the broad scope of hardware and software requirements for this project and it's short development timeframe, "development speedup" and "existing support" were the two largest consideration factors when coming up with a development plan. This would impact both hardware and software development, as well as component choice and selection going forward. Below is a brief listing of the early design guideposts followed for the MS Payload modules and informed development of the rest of this project:

- Select sensors, parts, and IC's with existing open source COTS devkits for maximum support, reliability, and to minimize development costs
- Use ST HAL library and code generation software for speedup in chip and peripheral configuration and software development
- Use a Real-time OS to synchronize sensor data flow and maximize sample rate/measurement frequency
- Debug with a HW debugger with multithreading support
- Use spice modeling on high voltage part of circuit design for safety and mitigating risk to low level electronics

3. Software Organization

3.1 Development Environment This project was developed in C using the eclipse IDE environment. In the early stages of software design, it used the ARM-GCC toolchain with personally edited makefiles for linking and compilation. It also was based on ST's now discontinued Standard Peripheral Library hardware abstraction layer (HAL). However, after switching to HAL's currently supported peripheral library (CubeMX), the decision was made to switch to let eclipse manage the makefile of the project. This decision to switch to a managed project was largely due to convenience after finally switching to a working implementation of the CubeMX library. Both the old and new environments can be found on the project github.

3.2 Debugging Environment In order to debug in a multithreaded environment, a Segger J-Link EDU debugger was used. The Eclipse IDE was used as the base debugging environment alongside Segger's own GDB variant in order to help visualize what was happening within each thread at a given point in time.

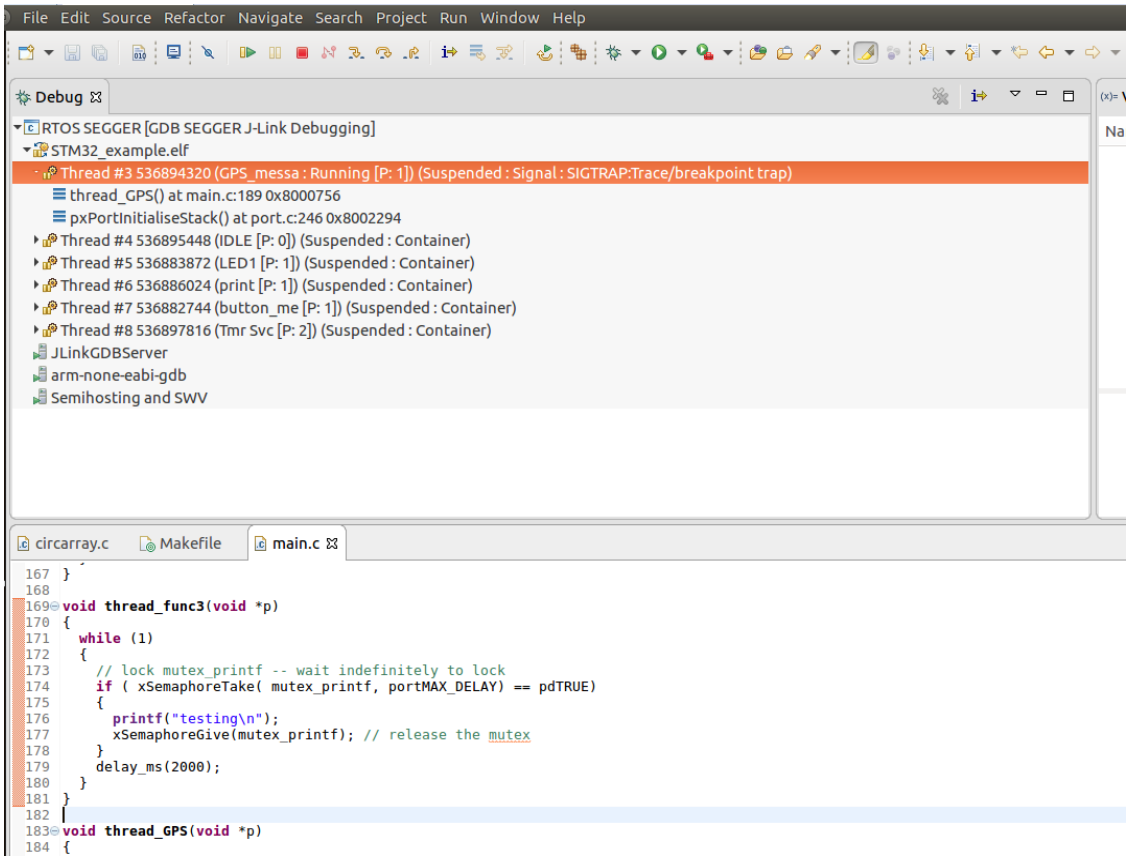


Figure 4. Multi-threaded view of an eclipse debugging session using a Segger J-LINK.

Figure 5. Jlink EDU debugger via segger.com

The JLINK debugger supplies a 20pin JTAG header and is compatible with Serial Wire Debug (SWD), a debugging interface which typically uses 6 wires but minimally may use only two. SWD was used for this project.

3.3 Real Time Operating System (RTOS) Due to the high amount of sensor data which would need to be interpreted, processed, and potentially sent off to the Radio OBC for transmission, a real time operating system would be required to manage these priorities, resources, interrupts and time constraints. FreeRTOS (an open-source real time operating system) was selected for inclusion in this project based on its large support base, availability for the selected microcontroller the project was using, and easy to read documentation and example code. It also was compatible with the Segger JLINK debugger, with Segger going as far as creating offering their own FreeRTOS visualization utilities on their website.[5]

3.4 Hardware Abstraction Layer (HAL) Developing code which interacts and configures a microcontroller chip, turning on and off various peripheral buses, pins, clocks, and onboard peripherals can be a long and meticulous starting task for an embedded software developer working with a new chip. To aid in this process, ST has developed the STCubeMX code generation utility to allow the developer to quickly configure a chip using visual drop down elements, and with the click of a button the project code environment is generated (as well as other helpful documentation). This greatly reduced development time of the configuration part of the project because if a peripheral (say the CAN1 bus) or pin assignment needed to ever change, it was a few simple drop-down selections away. One could even switch out the chip the project was based around for another larger or smaller offering and it would only require minimal changes to the existing codebase after clicking regenerate. This is the sort of speedup which turns what would be hours into minutes, making the STCubeMX utility and STCube HAL ideal for use with this project.

3.5 Software Organization Because the eclipse environment was used in a configuration which managed the makefile for the project, all header and source files for the project were kept separated in order to aid in their discovery during the compilation and linking process. The default locations for the peripheral libraries (including FreeRTOS) were kept in the System folder, which is the default location used by the Gnu-ARM Eclipse toolchain plugin. The Basic layout of the main project files, ST Cube library, and FreeRTOS can be seen in the figures below:

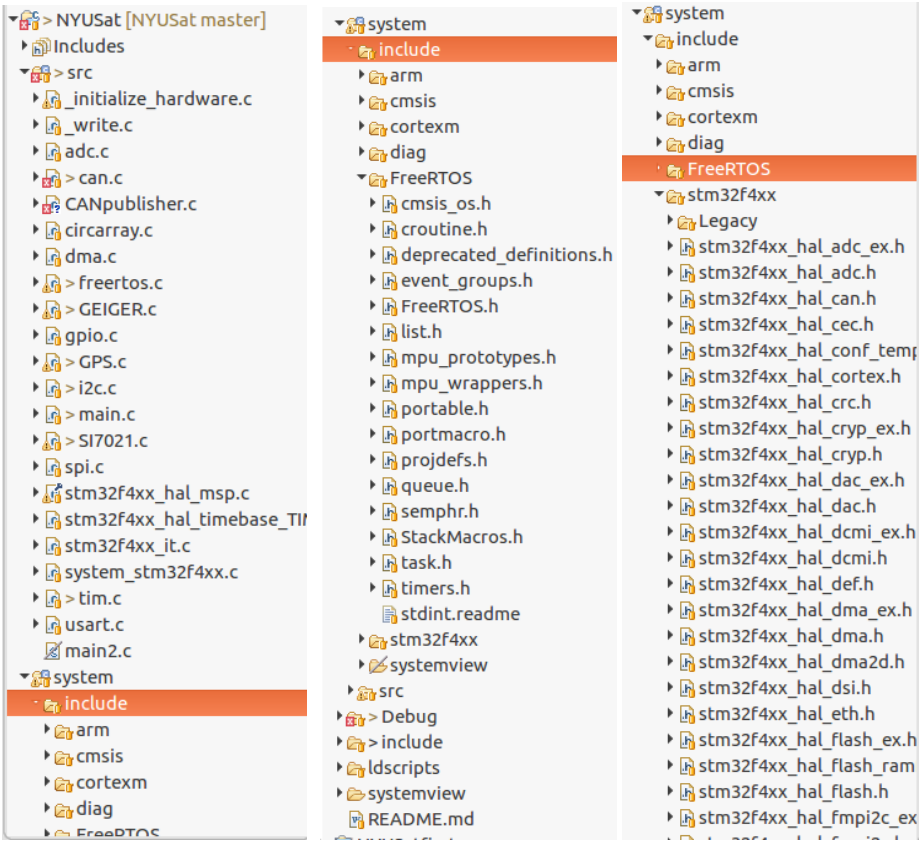


Figure 7. : Core source files, FreeRTOS, and Cube HAL organization stucture

4. Hardware Organization

4.1 Development Environment To create and organize the PCB and schematics for this project, Altium Designer was used. Altium allows multiple PCB's to be sourced within an overall design, as well as 3D cad view as well as part creating utilities. It is also able to generate a working BOM and link chosen parts libraries to Digikey orders. For modeling and simulation, LTSpice was used due largely to its ease of use and large built-in electrical model database.

4.2 Debugging Tools In general for this project, an oscilloscope was the most often tool used to look at signals passing to and from the selected sensors. A "Bus Pirate" communications protocol "sniffer" was additionally used at various points to interact with various sensors and confirm readings taken by code running on the OBC. Twin XBee modules were initially included for the wireless UART communication with a GPS modules, so that the GPS unit could be placed far away near a window (allowing it to pick up a solid satellite fix) and coding and reception of that data could happen at a more ideal desktop/workbench in a

central location. Finally, A breadboard was bought to house the COTS development kits and breakout boards which would encompass the project and connect to the target device.

4.3 Project Organization The schematics were organized so that they had an overall “reference” design which could be changed permanently, and two child PCB’s which referenced the core schematic and would update automatically if the core reference ever changed. This allowed the creation of two separate PCB module designs, sharing the same core foundation schematics seen below.

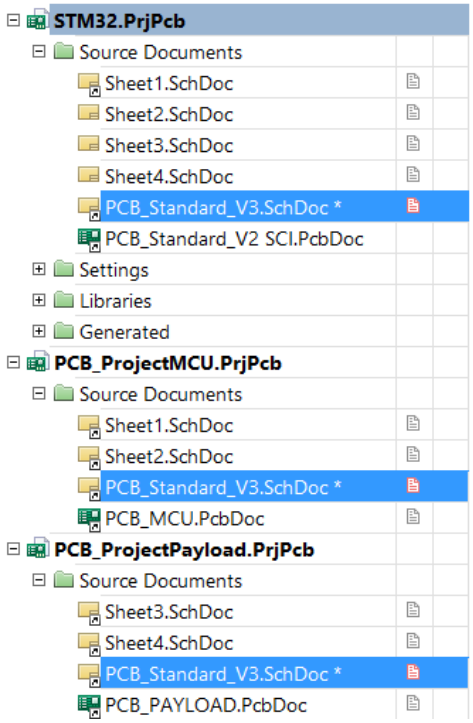


Figure 8. : In the photo above, Sheets 1 and 2 as well as the PCB Standard make up module A. Sheets 3,4 and the PCB Standard make up module B.

4.4 Schematics The current schematics for the project can be seen below. Sheet 4, which houses the Geiger Counter Voltage Boost Circuit was generated using LTSpice and is to be implemented in Altium at a later date.

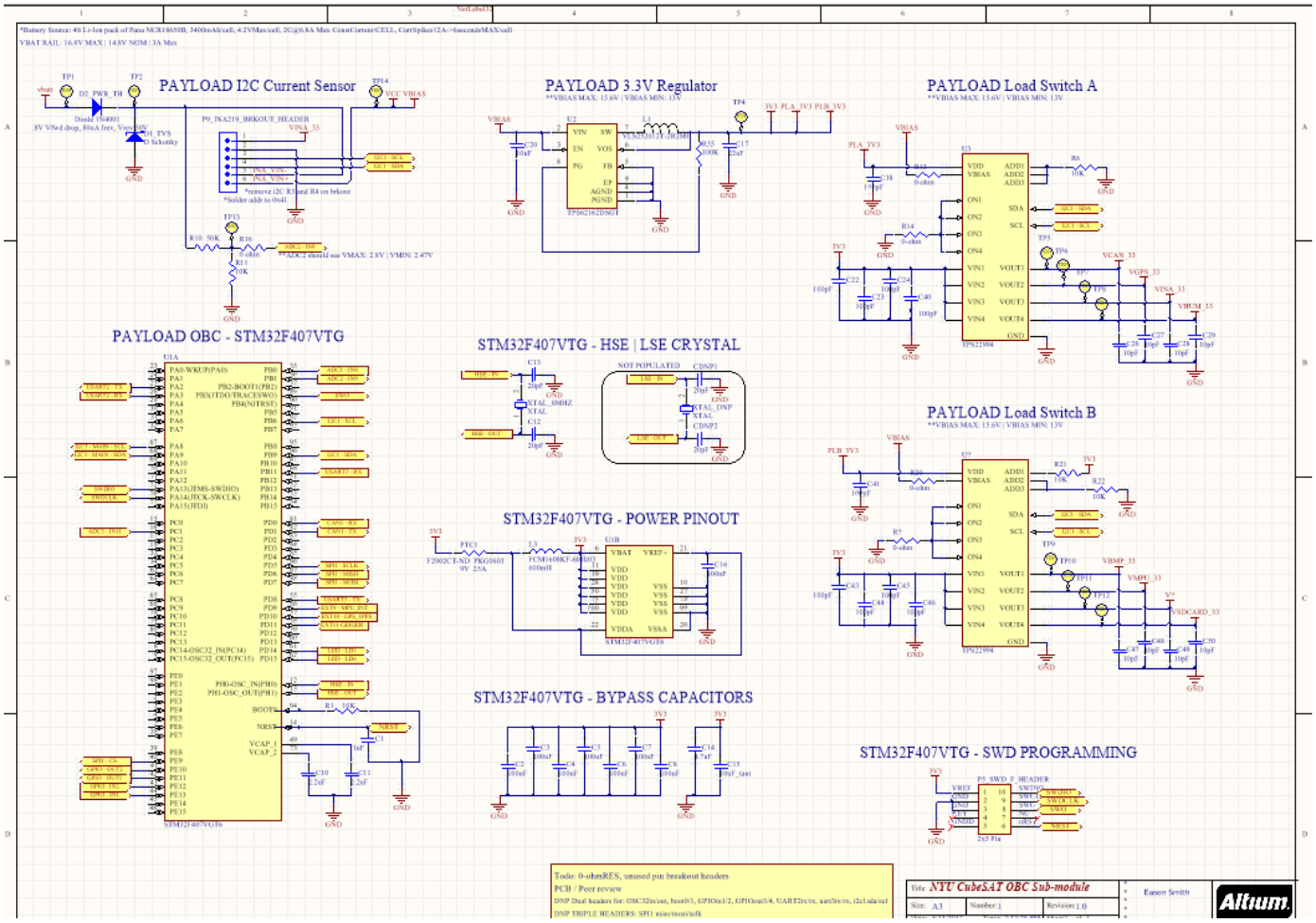


Figure 9. :Schematic Sheet 1. Payload OBC chip, Power Regulation and Sequencing and I2C Current Sensor on Module A

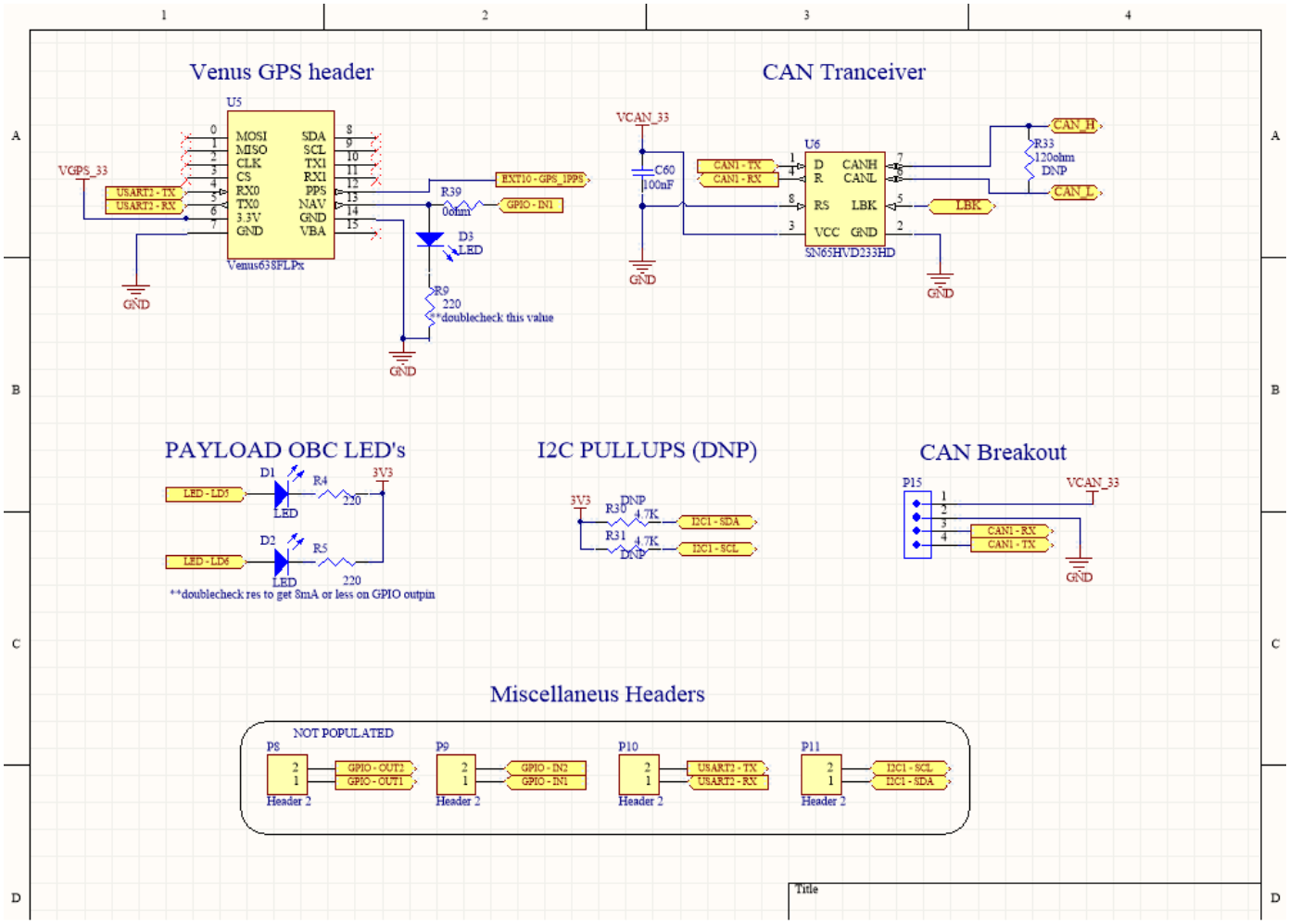


Figure 10. :Schematic Sheet 2. GPS Sensor Can Transceiver, and miscellaneous breakout headers for Module A.

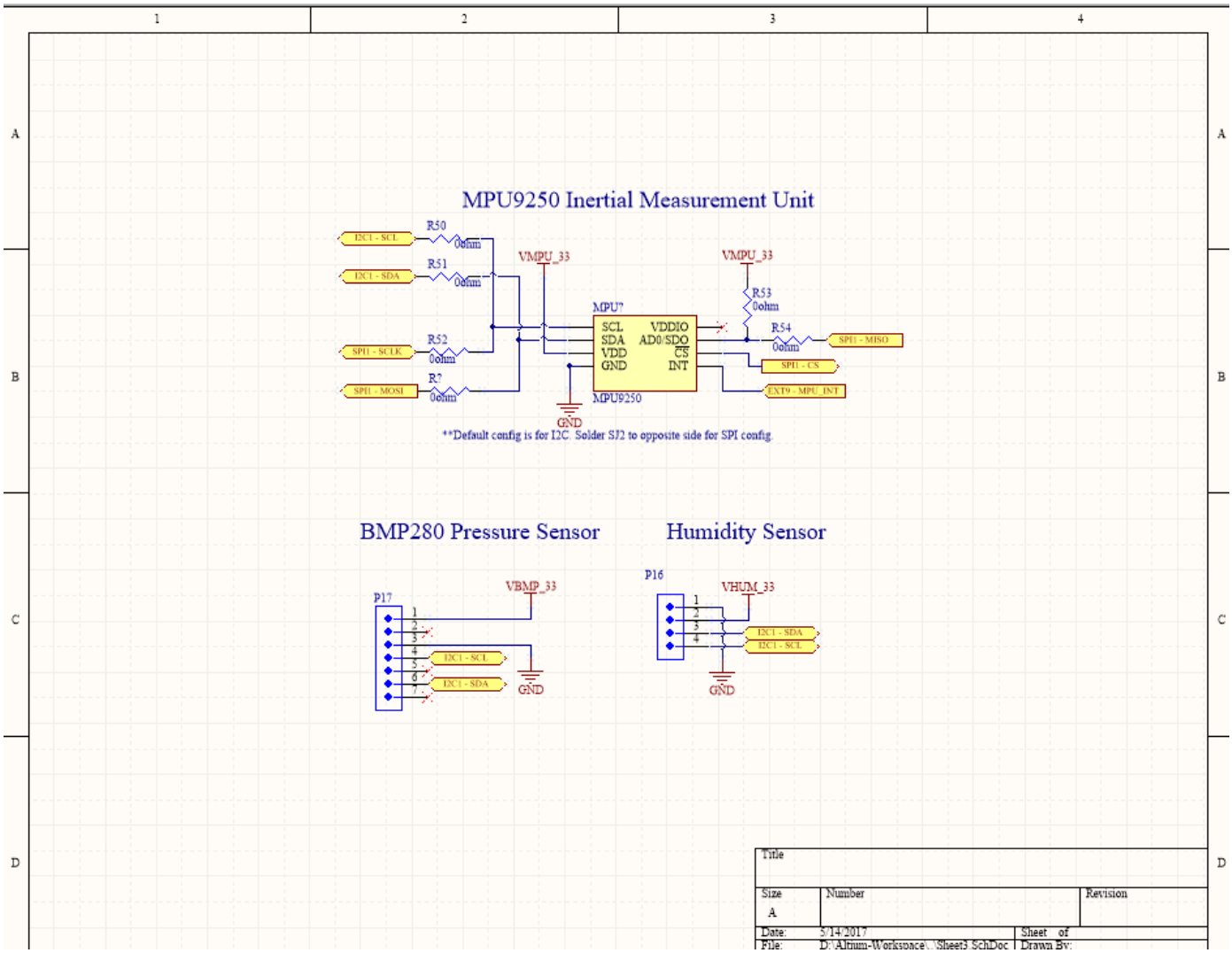


Figure 11. :Schematic Sheet 3. Inertial Measurement unit, Pressure Sensor and Humidity Sensor for Module B.

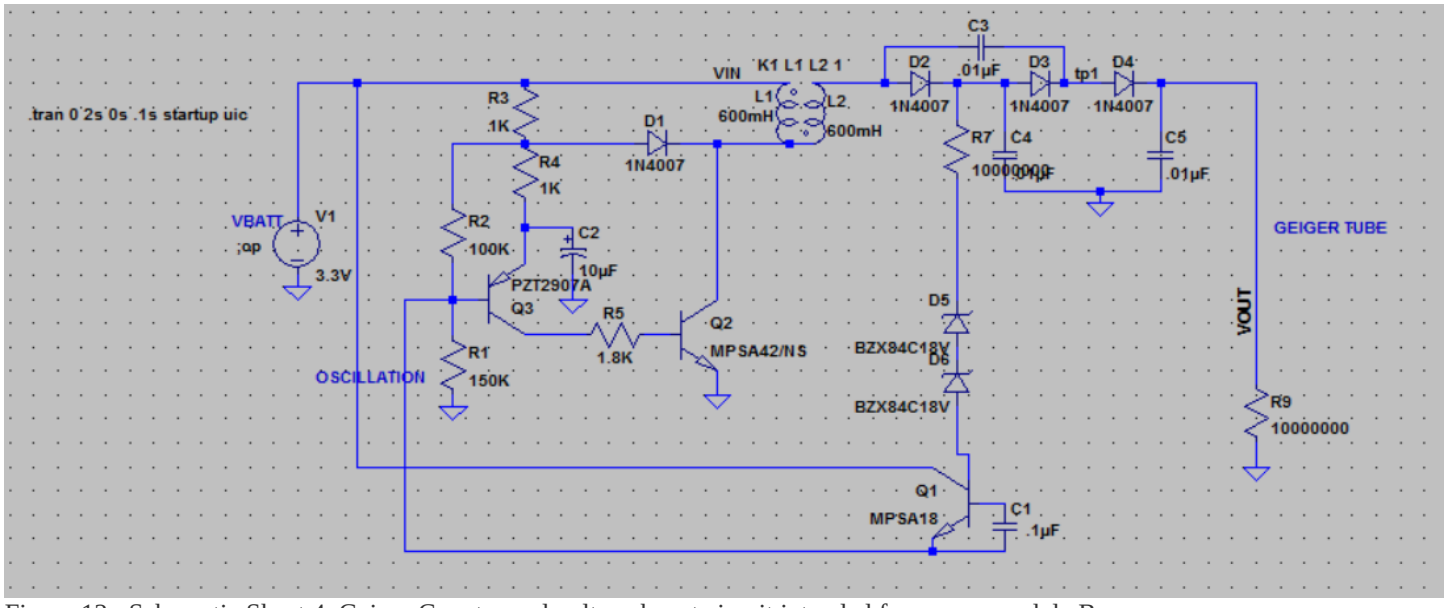


Figure 12. :Schematic Sheet 4. Geiger Counter and voltage boost circuit intended for use on module B.

4.5 PCB Layout As mentioned, the schematics are grouped into two module designs. Module A is centered around critical components like the OBC and CAN transceiver modules, while Module B is centered around the high voltage Geiger Counter as well as several other science instruments. Because of an issue brought to light by the spice simulation of the Geiger Counter voltage boost circuit, only the PCB for module A has been started. Below is the current state of the layout of the PCB for module A.

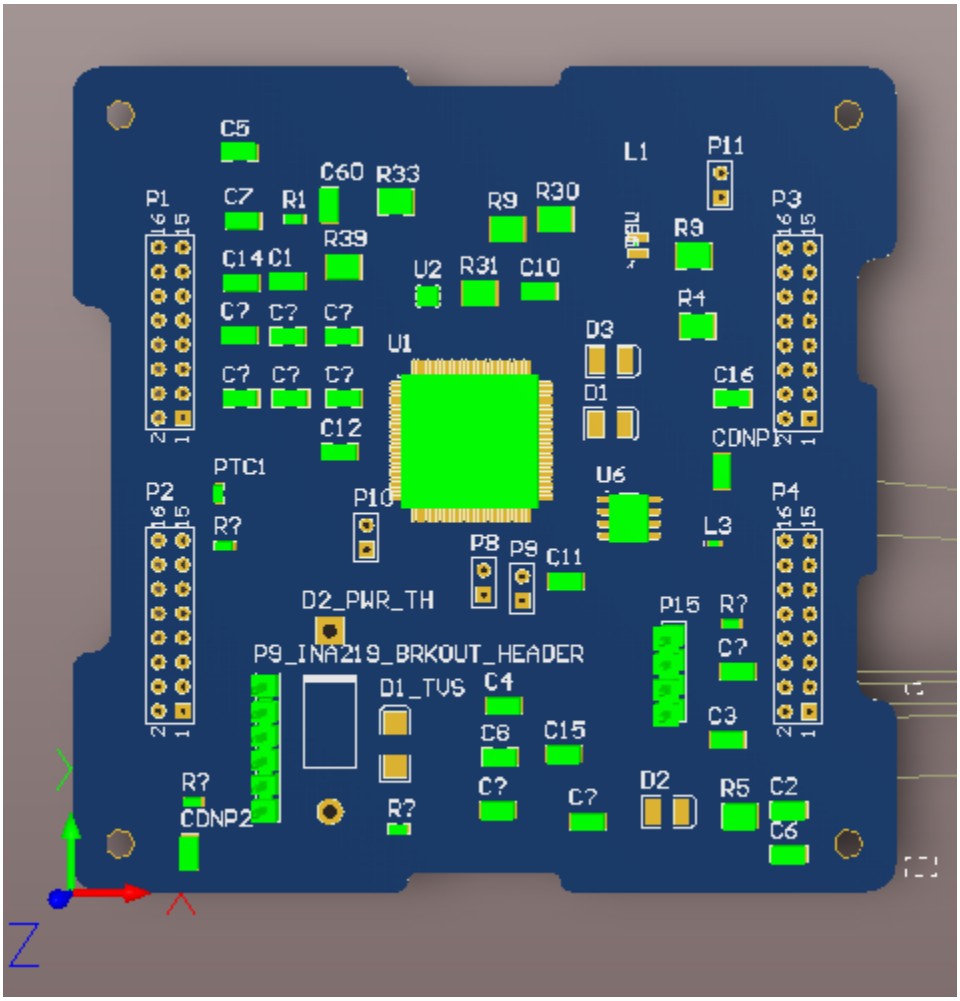


Figure 13.: PCB Layout Module A

6. On Board Computer (OBC) The chip chosen for development with this project is the ARM based STM32F407VTG. This chip was chosen due to the authors familiarity and experience working with the chip, as well as its suitable peripheral interface for the various communication bus's needed. It also has a large pre-existing support base, as well as a low cost easy to obtain COTS development kit. The large majority of this project was designed while connected to an STM32F4 Discovery board development kit acting as the base OBC until at a later time the PCB could be manufactured. Some of the key features of this chip are:

- 32-bit ARM® Cortex® -M4 with FPU core
- 168 MHz Clock Rate
  LQFP100 package
- 1-Mbyte Flash memory
- 192-Kbyte RAM
- Support for USB, CAN, SPI, I2C, UART

6.1 Pin Mapping Diagram
Below is the current pin mapping diagram for the OBC, as produced by the STCubeMX software. All active pins are highlighted in green.

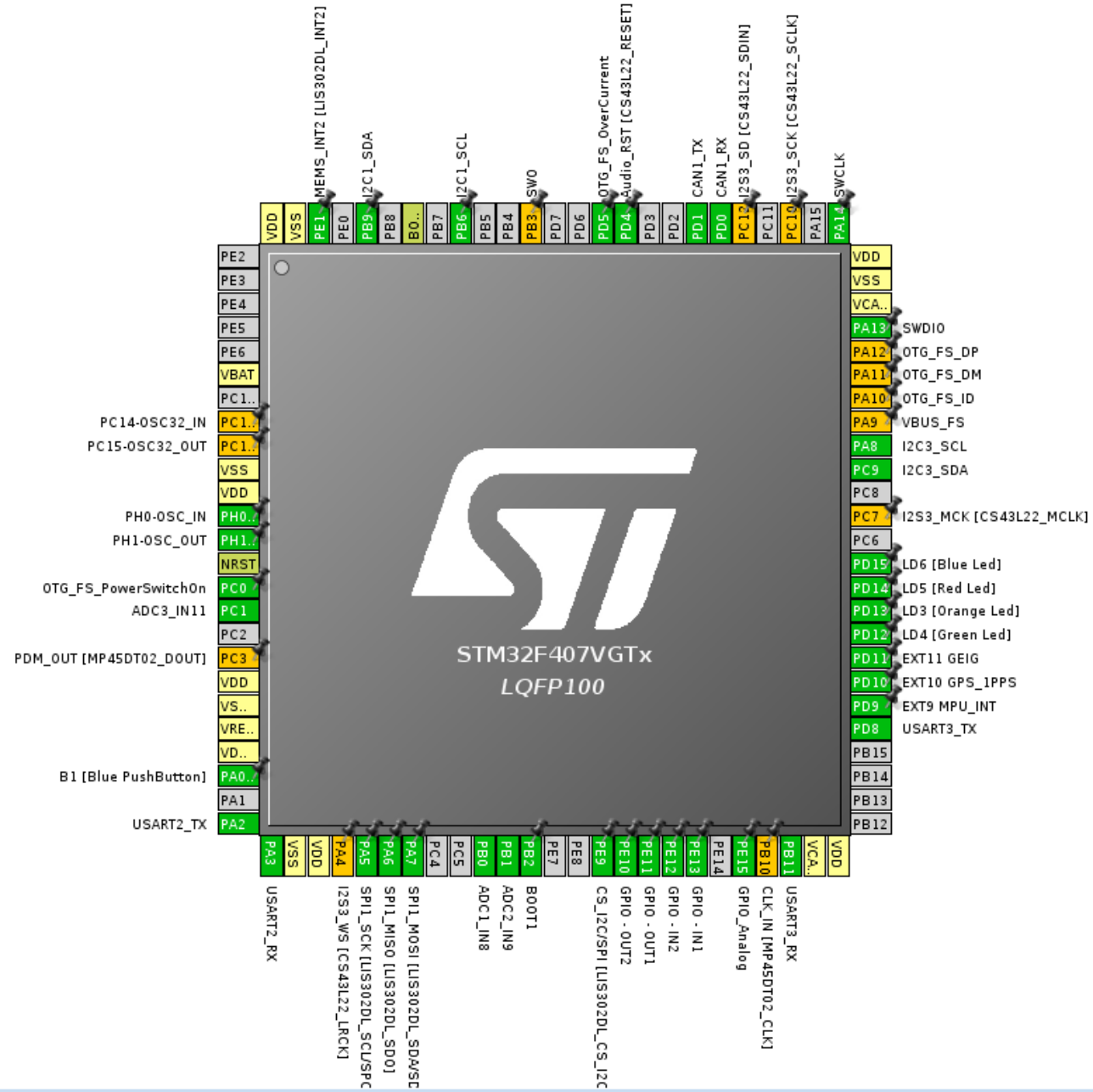


Figure 14. : STM32F4 Pin Mapping Diagram produced by the STCubeMX utility

6.2 OBC Clock Configuration Settings

The STM32F4 was configured using the STCubeMX utility to run at 100Mhz sourced from an 8MHz external crystal. The 100Mhz (reduced) speed was chosen in order to reduce power consumption but still maintain a considerable speed for the various communication busses while maximizing battery life. The final clock speed can be seen as the result “FCLK” below.

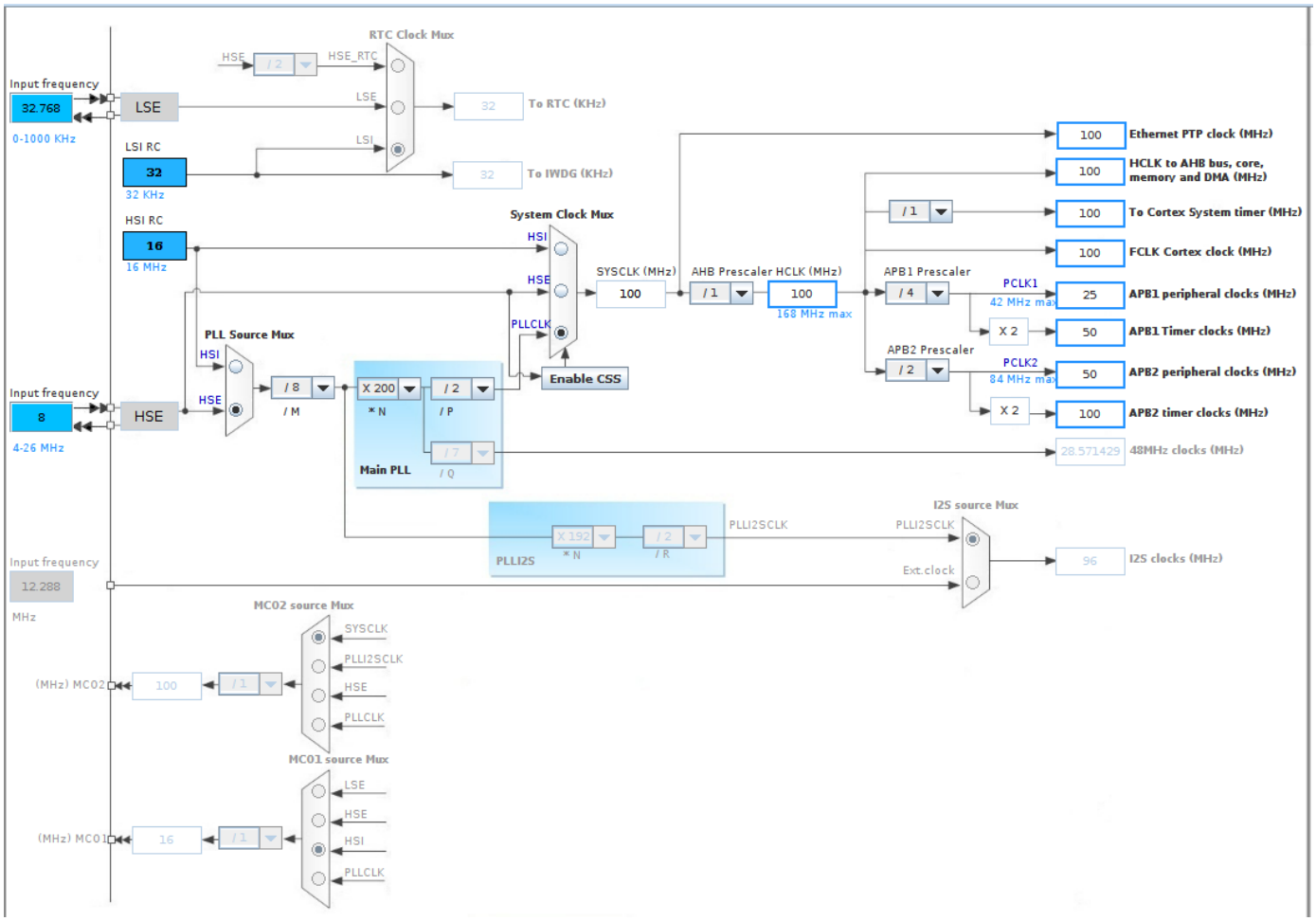


Figure 15. : Clock Settings used for the STM32F407VTG OBC

6.3 Power Management and Load Sequencing

The STM32F4 and all other sensors within the mission science payload are designed to run at 3.3V, with the exception of the Geiger counter circuit, which is designed to run at 5V. As the battery voltage is fed into OBC module A from one of the 4 satellite bus headers, it is split between a voltage divider circuit (for reading the battery voltage) and an INA219 I2C current sensor placed just after a power blocking diode (for reading the module load current). When the voltage exits the Current Sensor, it is passed through a TPS62162 Voltage Regulator, which outputs the primary 3.3V feed used by every other component in the mission science payload.

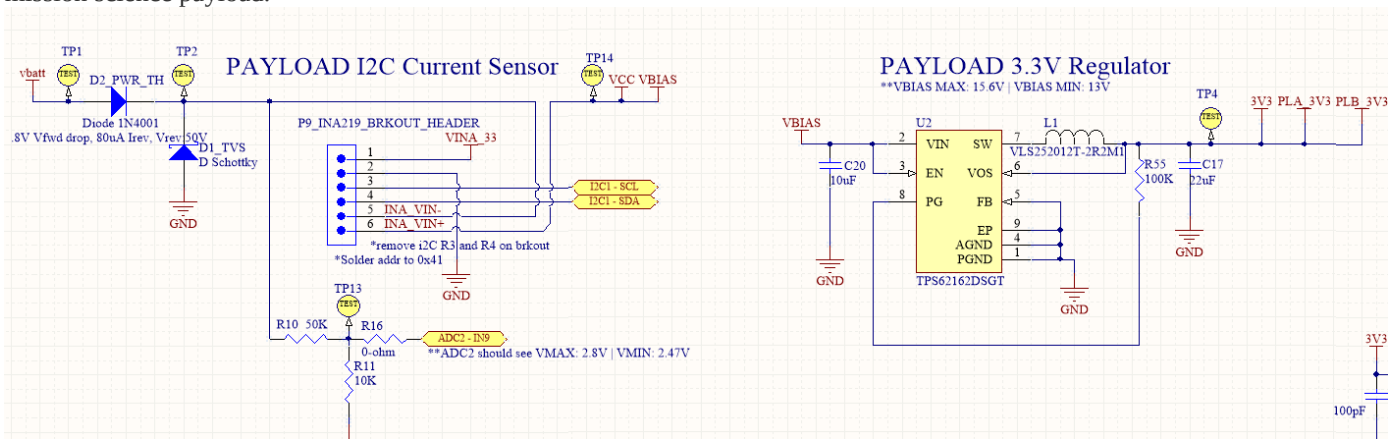


Figure 16. : Power Regulation Circuit

Due to the phenomenon of “in-rush current” that happens when all of the peripherals, microcontrollers, and IC’s in a circuit turn on simultaneously, a simple voltage regulator will not be enough to protect the spacecraft circuitry from brownouts when these

devices are powered on together by a voltage regulator. So a dual I2C based Load Sequencer (TPS22994) was found and included, outputting power rails for each respective sensor on both modules and controllable through I2C, as seen below.

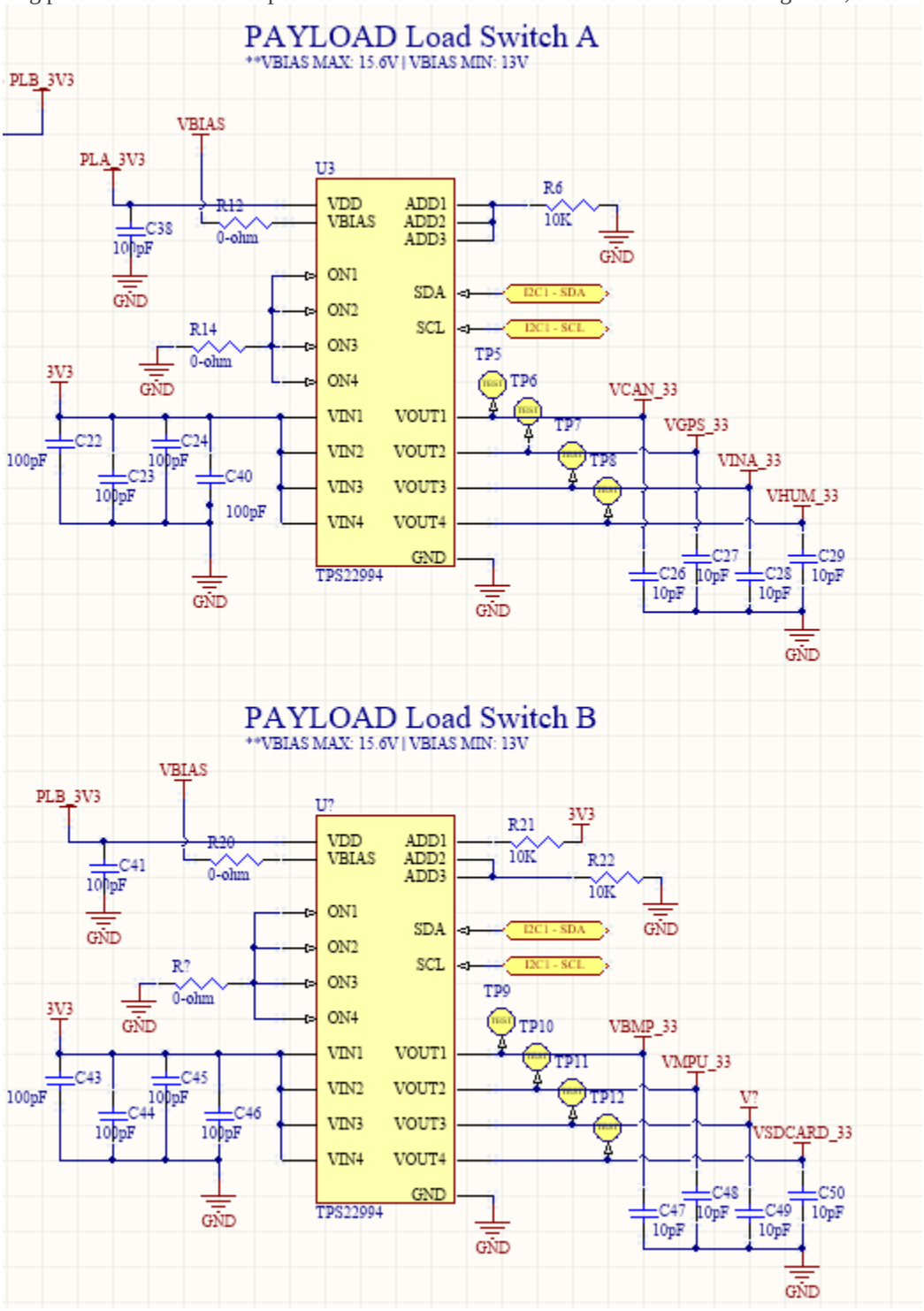


Figure 17. : Power Sequence IC circuit for modules A and B

7. Mission Science Payload

The mission science payload selected for this satellite includes a Geiger Counter, Humidity and Temperature sensor, GPS device, as well as a Pressure Sensor and Inertial Measurement Unit (IMU). The Pressure sensor and IMU have been included in the schematics and design files of this project however they have not yet been implemented or tested in software. Below is a hardware and software overview of the sensors included with the science payload, all of which have been tested and implemented into the project.

7.1 GPS Unit

Figure 18: Venus GPS breakout from Sparkfun.com

The GPS Unit used on the CubeSAT is a Venus638FLPx GPS Sensor with SMA antenna. This unit runs at 3.3V and uses UART communication to send and receive geolocation data to the OBC from signal timings sent by orbiting satellites. In addition to outputting data, it also outputs a 1PPS signal (active only when it has a fix) which can generate extremely accurate measurement timings. The 1PPS pin is connected to an interrupt on the OBC.

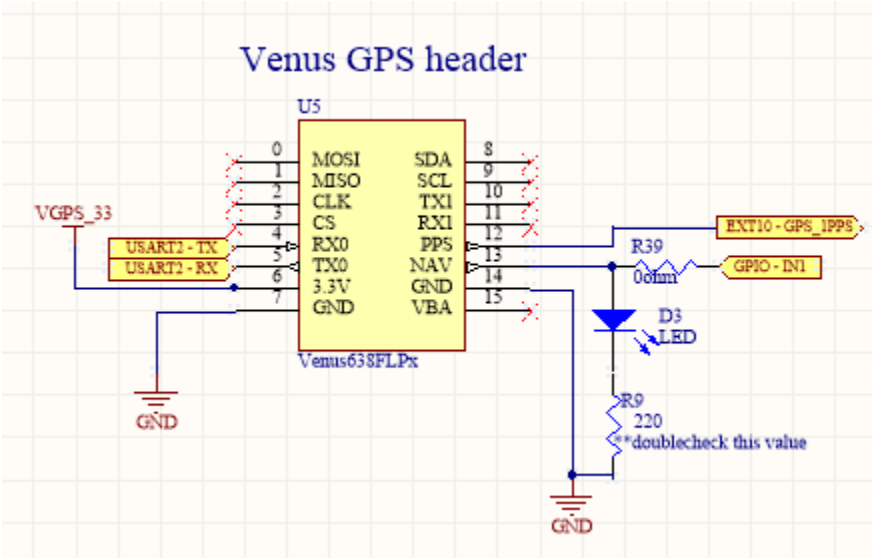


Figure 19. : The Venus GPS unit schematic used by module A.

The GPS unit was run in binary mode so that message length could be shortened. In order to keep track of incoming data, a state machine is used to process the bytes. Below is a rough sketch of the states diagram used in development:

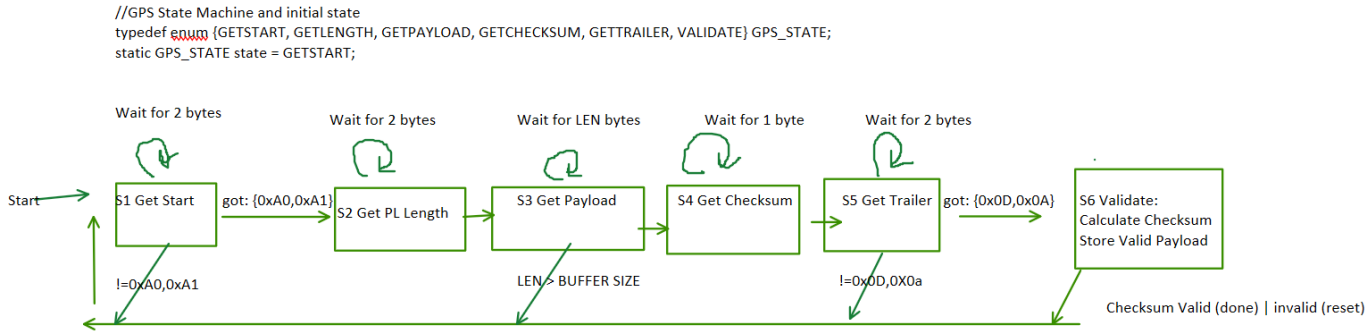


Figure 20. : GPS process state diagram for incoming GPS bytes.

In order to integrate the GPS module with FreeRTOS synchronization, A basic UPPER half/LOWER half thread and interrupt implimentation was used to synchronize the real time constraints required by this device. The synchronization strategy used is depicted in the context transition diagram below.

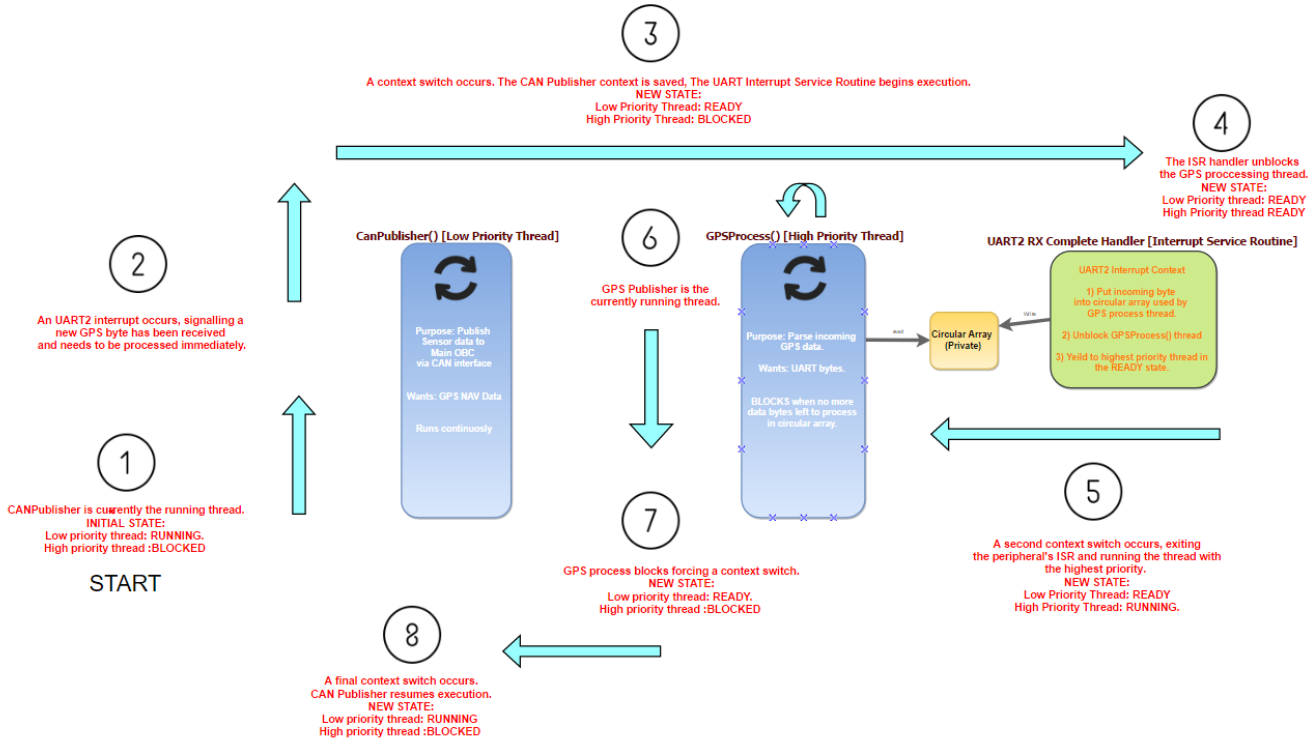


Figure 21. : GPS Process Thread Synchronization Routine / Context transition diagram

7.2 Humidity Sensor

SI7021 humidity sensor from Sparkfun.com

For humidity and Temperature readings, the SI7021 Humidity Sensor from Sparkfun was selected. This module communicates using I2C and uses the same basic threading strategy as GPS.

Due to it's small size, the Sparkfun breakout board for this device was chosen for inclusion with the overall spacecraft design, as opposed to building it into the PCB module itself. The schematic housing the breakout for this device can be seen below:

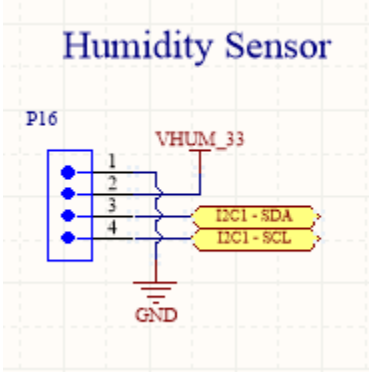


Figure 22. : SI7021 Breakout

The RTOS thread synchronization strategy for this device is largely the same one used for the GPS module, except because the I2C bus is shared amongst multiple devices, a special I2C mutex is used to keep other I2C devices from interrupting the current I2C transmission in the case of a context switch.

The results of testing the SI7021 can be seen in the figure below:

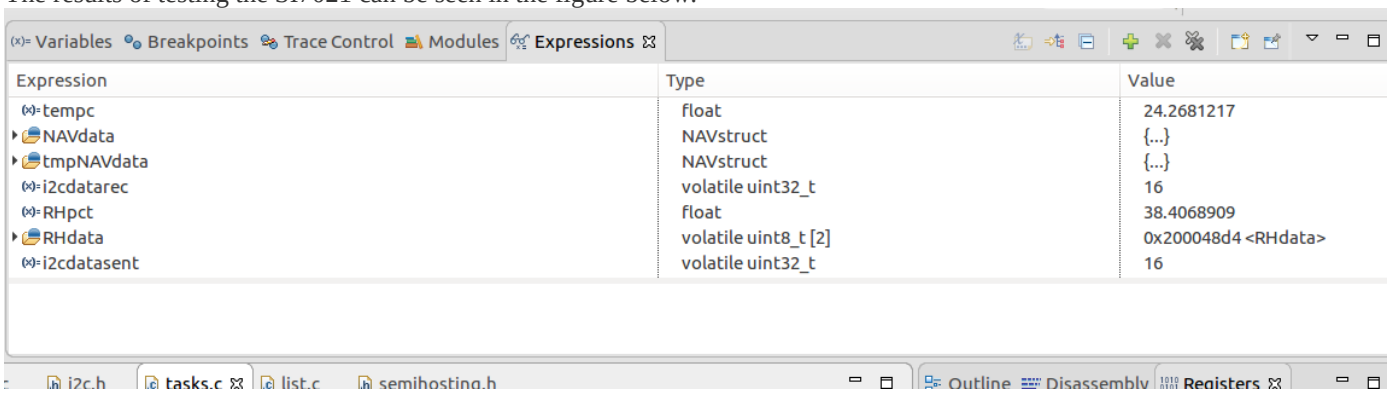
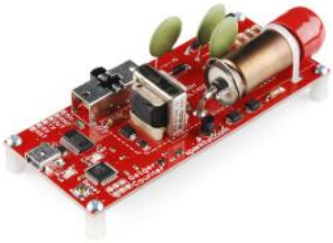


Figure 23. : SI7021 test data. (TempC) Temperature is 24.26 degrees Celsius. Relative Humidity (RHData) is 38.4 percent.

7.3 Geiger Counter



Geiger Counter breakout from Sparkfun.com

The Geiger Counter was the most complex instrument developed for on the MS Payload. Not only is the tube very large in physical size, but in order for the Geiger tube to function, a complex voltage boost circuit must be created to raise the on board voltage on the PCB to above 320V (ideally 450V).[4] In addition, the RTOS software implimentation interacting with the unit would require not just a process thread but also an external interrupt handler as well as an interrupt timer. These would be needed to process and maintain the current Clicks per second (CPS) and also clicks per minute (CPM), using a sliding window circular array.

Below is an oscilloscope capture of a “click” (radiation event), produced by the tube:

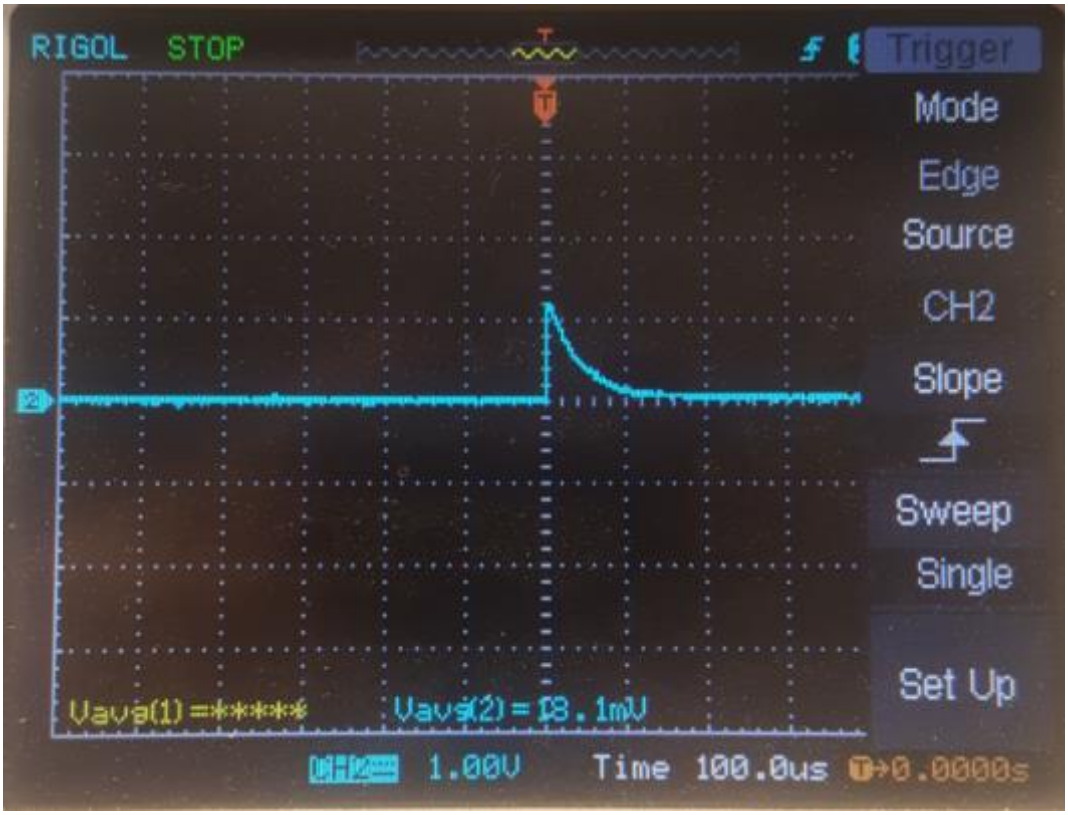


Figure 24. : Oscilloscope output of a Geiger tube “Event”.

The thread synchronization routine used for this device on the spacecraft can be seen below:

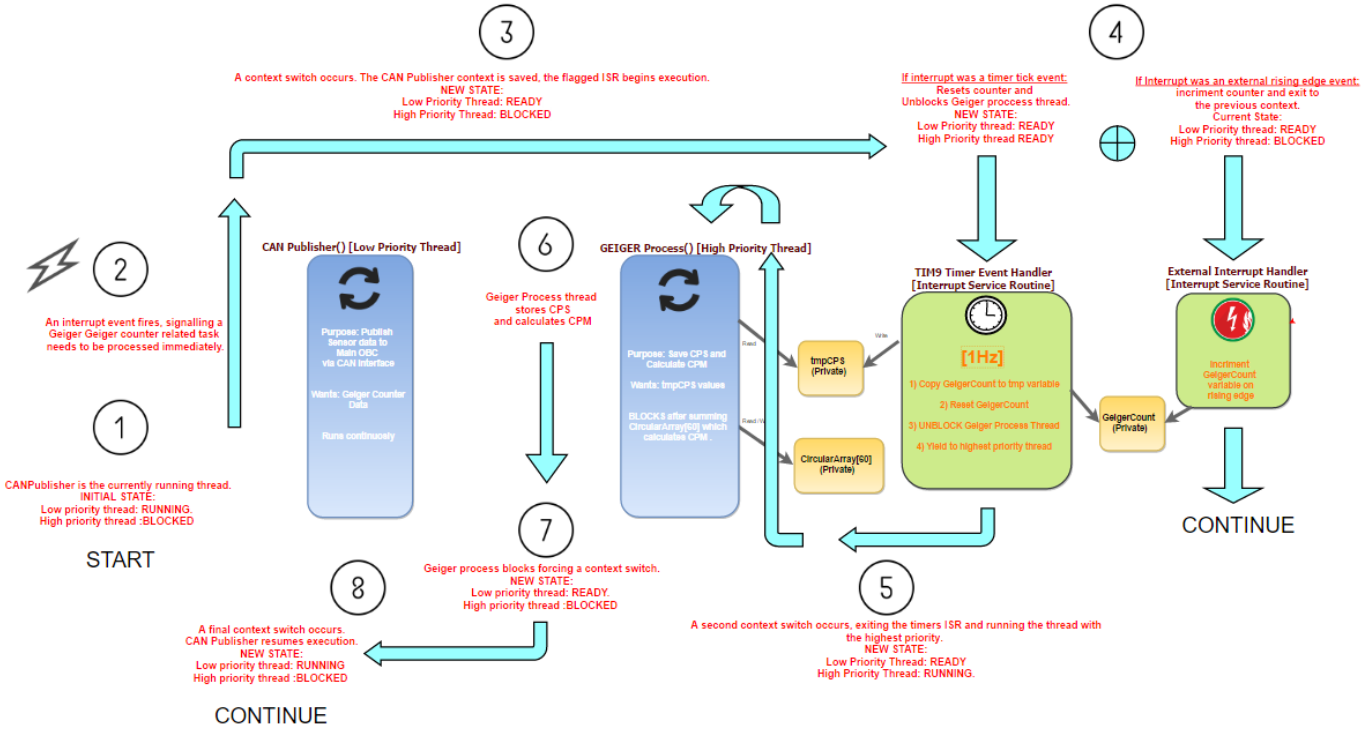


Figure 25. :Geiger Counter Thread Synchronization and Interrupt Handling Routine

Because the Geiger counter operates at 5V (boosting to above 450V) [4], and the spacecraft operates at 3.3V, it was decided it would be best to simulate the circuit in LTSpice before deciding on a permanent implimentation for inclusion in the schematic design. The simulation and resulting waveform outputs tested can be seen in the figures below.

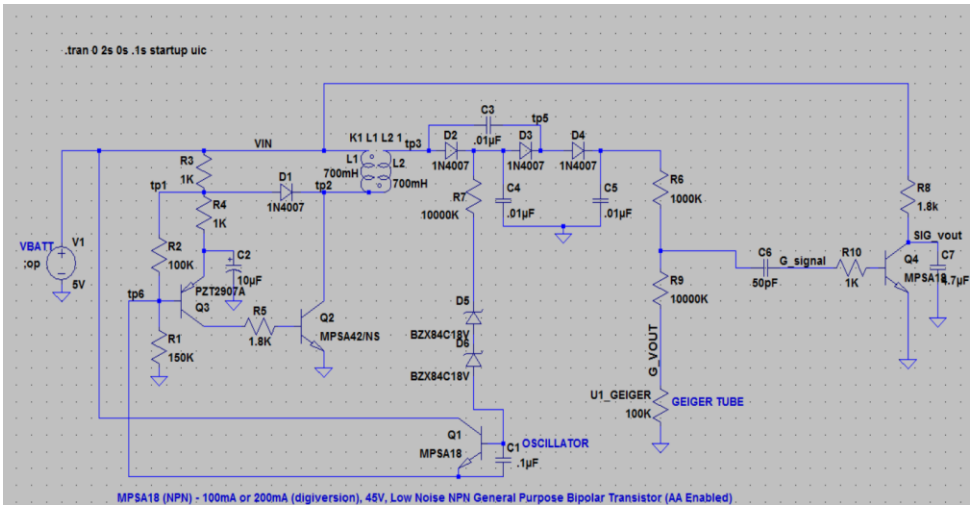


Figure 26. : LTSpice Circuit used for simulation

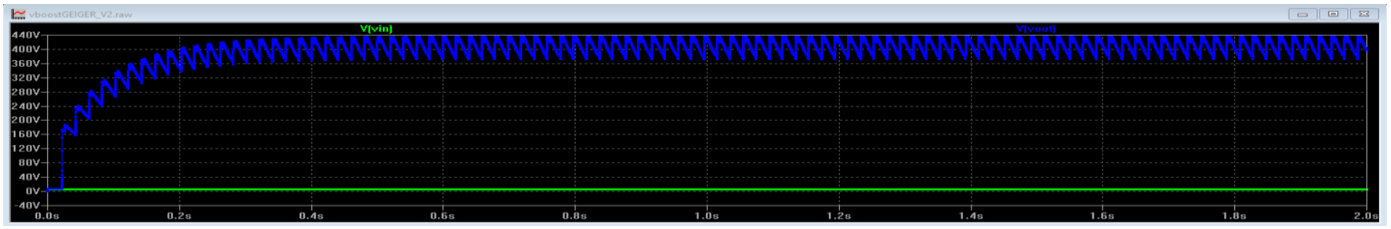


Figure 27. : Voltage output for VIN@5V

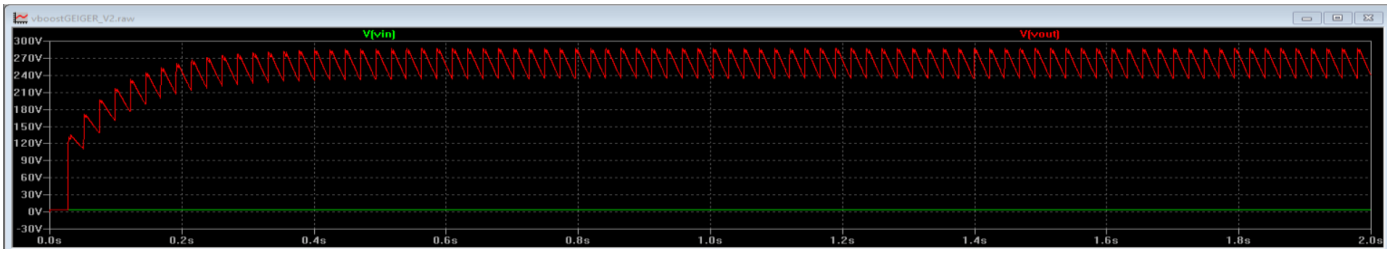


Figure 28. : Voltage output for VIN@3.3V

As can be seen from the figures above, the voltage output for the Geiger Counter Voltage Boost circuit at 3.3V would not be enough to drive the Geiger Tube without significant modifications to the circuit. The large oscillation on the signals seen in both figures was determined to not be due to a flaw in the circuit design but rather due to imperfect models used in the simulation software package.

In order to get around the 3.3V issue, a 5V rail was created on header four, and routed from a separate module on the spacecraft. Unfortunately, this result caused a delay in the development of module B.

8. Conclusion and Future Development

Work is still to be done on both modules before spaceflight can happen. This includes the Pressure Sensor and IMU selected for this design as well as the power sequencing code and CAN transceiver software. The PCB for module A is currently in the layout phase and the PCB for module B should begin work soon. Much of the other systems and modules on the spacecraft are already produced and ready for flight testing and the MS Payload is not far away from a future flight test.

9. Timeline

September – December 2016	Research and acquisition of Development enviroment
February 16th 2017	Implimentation begins. Project proposal submitted.
February 23th 2017	High Level power and communication diagram created.
February 28th 2017	UART module code written. Work on GPS begins.
March 7th 2017	GPS messages aquired.
Mach 22nd 2017	Switch to CubeMX peripheral enviroment. FreeRTOS installed.
March 28th 2017	GPS thread synchronized. GPS Live Demo in class.
April 11th 2017	I2C module implimented. SI7021 code written.
April 18th 2017	Gieger Counter code written. Schematic capture begins on module A.
April 25th 2017	Schematic Capture begins on module B. LTSpice testing on Geiger Counter boost circuit begins.
May 2nd 2017	Geiger counter modeling results presented. Schematic capture of module A complete.

10. References

1. **NASA CubeSAT Launch Initiative** https://www.nasa.gov/directorates/heo/home/CubeSats_initiative
2. **CubeSAT.org – About** http://www.cubesat.org/about/
3. **CubeSAT design Spec. REV. 13** https://static1.squarespace.com/static/5418c831e4b0fa4ecac1bacd/t/56e9b62337013b6c063a655a/1458157095454/cds_rev13_final2.pdf
4. **LN712 Datasheet** https://www.sparkfun.com/datasheets/Components/General/LND-712-Geiger-Tube.pdf
5. **Segger SystemView utility** https://www.segger.com/systemview.html

11. Links

Old Project Software Github: https://github.com/EasonNYC/NYUSat-old

Current Project Github: https://github.com/EasonNYC/NYUSat

Segger Jlink and Software: http://Segger.com

Sparkfun SI7021, Venus GPS and Geiger Counter: http://Sparkfun.com

STCubeMX Software: http://www.st.com/en/development-tools/stm32cubemx.html

FreeRTOS Software: http://www.freertos.org