Командный практический тур

Общее описание задачи

Легенда

Современная космонавтика демонстрирует все больше примеров применения сверхмалых космических аппаратов для прикладных научных и коммерческих задач, причем пределы возможностей для сокращения их размера и стоимости постоянно расширяются. На данный момент известны эффективные группировки наноспутников с линейными размерами 10-30 см для решения прикладных задач (в т.ч. например, аппараты Planet Labs, Inc с задачами дистанционного зондирования Земли).

Миссии на основе аппаратов меньшего размера (пико- или фемтоспутников) применяются в области поиска новых технологических решений и/или бюджетных научных исследований, таких как проведение многоточечных измерений каких-либо физических величин на орбите Земли.

Рассматриваемый вариант экспериментальной космической миссии заключается в том, что спутник-носитель формата CubeSat после выхода на орбиту совершает один или несколько маневров и отстыковывает несколько пикоспутников, оснащенных простым измерительным оборудованием. Пикоспутники в данном случае берут на себя задачу сбора научных данных, рассеиваясь в пространстве вокруг спутника-носителя, а он, в свою очередь, собирает данные и перенаправляет их на наземную приемную станцию.

Целевая задача

В задачи участников командного тура заключительного дня входит работа над проектом и прототипами аппаратов миссии с выведением пикоспутников аппаратом-носителем, в том числе:

- Сборка и доработка функциональной модели спутника-носителя, включая алгоритмы одноосной стабилизации, ориентации и раскрытия блока солнечных батарей для поддержания энергобаланса;
- Сборка и доработка функциональной модели пикоспутника, включая разработку и пайку его подсистемы питания с применением предоставленных организаторами компонент;
- Отработка взаимодействия пикоспутника со спутником-носителем, включая организацию помехозащищенного радиоканала;
- Решение прикладных задач управления миссией на орбите в системе численного моделирования космических миссий.

Решение задачи предполагает наличие четырех направлений работы и соответствующих им рекомендованных (но не обязательных) ролей:

- Разработка и программирование спутников. В задачи этого направления входит управление спутниковой платформой формата Cubesat, который является носителем пикоспутников. В задачи входит конструирование аппарата и его системы раскрытия солнечных батарей, написание алгоритмов управления бортовыми устройствами стабилизации, ориентации и отработка полной циклограммы работы спутника-носителя.
- Схемотехника. В задачи этого направления входит создание малогабаритной системы автономного питания для пикоспутника, которая позволит работать в условиях переменного доступа к солнечному свету, а также вспомогательные задачи разработки спутника-носителя, связанные со схемотехникой.
- Радиотехника. В задачи этого направления входит обеспечение связи между спутником-носителем и пикоспутником на основе программируемых радиомодулей, передача полезных данных с пикоспутника по запросу и защита канала связи от помех.
- Орбитальная механика. В задачи этого направления входило уточнение орбиты и разработка алгоритмов управления аппаратом на орбите, включая ориентирование и применение маневрового двигателя.

Участники одной команды с разными ролями имеют право помогать и замещать друг друга в командной групповой работе.

Предоставляемое оборудование

Командам участников предоставлено в равном для всех команд количестве следующее оборудование:

- Персональные компьютеры;
- Конструктор спутника IntroSat, соответствующий форм-фактору Cubesat 2U (1 шт на команду);
- Дополнительный набор IntroSat Pico (1 шт на команду);
- Дополнительная плата семейства STM32 (1 шт на команду);
- Дополнительный приемопередатчик (1 шт на команду);
- SDR-приемник (1 шт на команду);
- Набор компонентов для разработки схем (1 шт на команду);
- Паяльные станции, ручные инструменты, станок лазерной резки и 3д-принтеры — в режиме совместного использования в предварительно согласованные окна времени в рабочее время в производственных областях;

Используемое программное обеспечение

Для моделирования миссии на орбите и для учета достижений участников использовался сервис численного моделирования космических миссий с режимом проведения соревнований Орбита.Челлендж, доступ к которому на время проведения деньов профиля был предоставлен организаторами (https://nti.orbitagame.ru).

Программы из списка ниже являлись рекомендованными:

- Основная среда разработки для микроконтроллеров — STM32CubeIDE, может быть скачана с сайта производителя после бесплатной регистрации по адресу

https://www.st.com/en/development-tools/stm32cubeide.html

- Монитор порта для работы с функциональными моделями спутников и их компонентами — PuTTY (https://www.putty.org/) или аналог в т.ч.:
- SmartRF Studio 7 — необходимое ПО для конфигурации приемопередатчика, доступно для загрузки с сайта производителя после бесплатной регистрации: https://www.ti.com/tool/SMARTRFTM-STUDIO
- SDR# — ПО для работы с SDR-приемником: https://airspy.com/download/
- GMAT (General Mission Analysis Tool) — ПО для моделирования орбитального движения космических аппаратов: https://sourceforge.net/projects/gmat/

Не допускалось программирование микроконтроллеров с применением Arduino IDE и его прямых аналогов (т. е. сред разработки, предназначенных для работы с Ардуино-совместимыми микроконтроллерами, но не с микроконтроллерами семейства STM32). Вся разработка и загрузка кода должна осуществляться только через STM32CubeIDE и его аналоги. При отсутствии альтернатив допускается использование Arduino IDE для монитора порта. В остальном, допустимо использование бесплатных аналогов рекомендованного ПО.

Порядок открытия и сдачи заданий заключительного дня

Задания заключительного дня делились на промежуточные задания по направлениям работы и итоговую защиту проектов. Задания могли подразумевать наличие знаний и навыков, полученных в рамках предварительных практикумов, которые проводились для участников заключительного дня после окончания 2 отборочного тура. Материалы практикумов были также доступны командам и на заключительном денье.

Промежуточные задания предназначались для последовательной подготовки команд к итоговой защите проекта и демонстрации владения соответствующими знаниями и навыками по каждому направлению. Промежуточные задания сдавались командами в течении рабочего времени до начала итоговой защиты.

Итоговая защита проходила в последний рабочий день и подразумевала демонстрацию командой всех разработанных образцов в действии, в соответствии с общим обликом задачи заключительного дня.

Во всех типах заданий контролировалась фактическая реализация требуемых функций, конкретный способ реализации в некоторых случаях мог рекомендоваться, но не был обязательным.

Тип заданий	Назначение и комментарии	Режим открытия
Промежуточные задания 1 дня	Организация командной работы, освоение предоставленного оборудования и программного обеспечения.	Доступ для участников с 1 дня, начиная с 12:00 второго дня сдача таких заданий допускалась со штрафом 50%
Промежуточные задания 2 дня	Первичная сборка прототипов, разработка базовой функциональности по каждому направлению работы.	Доступ для участников с 2 дня, начиная с 12:00 второго дня сдача таких заданий допускалась со штрафом 50%

Промежуточные задания 3 дня	Разработка расширенной функциональности по каждому направлению работы, востребованной на итоговой защите.	Доступ для участников с 3 дня, прием до начала итоговой защиты
Итоговые защиты	Сборка результатов работы по всем инженерным направлениям в единый действующий программно-аппаратный комплекс, соответствующий целевой задаче.	Доступ для участников с 3 дня, прием на итоговой защите

Промежуточные задачи участников, 1 день

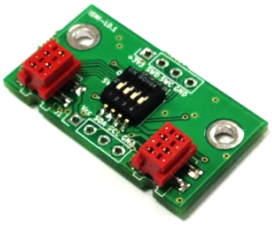
Разработка и программирование спутника

Материалы соответствующего предварительного практикума можно найти здесь: https://drive.google.com/file/d/1xxFppA0v38G2lbg0mR5UY5HsJqymjU8f

В рамках заданий 1 дня было необходимо:

- настроить беспроводную передачу информации по UART;
- считать данные с датчиков позиционирования и освещенности по I2C;
- задать управление двигателем, продемонстрировать изменение направления и скорости вращения;
- отработать сигнал на раскрытие СБ (схема сборки с транзистором приведена в задаче по схемотехнике).

Как и датчик позиционирования на практикуме, датчик освещенности подключается к шине I2C, правда в отличие от него датчик освещенности может иметь до четырех адресов, что позволяет подключить к шине до четырех таких устройств. За адрес устройства отвечают переключатели на обратной стороне датчика.



Для датчиков определены адреса 0x50, 0x51, 0x52 и 0x53. Если установить на переключателе джампер под номером 1 в верхнее положение, то данному датчику присвоится номер на шине I2C 0x50. Если установить в верхнее положение джампер под номером два, то адрес датчика будет 0x51. Более подробно соотношение адресов и положений джампера указано в таблице.

Положение джампера на переключателе датчика	Адрес, соответствующий положению джампера	Положение джампера на переключателе датчика	Адрес, соответствующий положению джампера
	0x50		0x52
	0x51		0x53

По I2C датчик передает значение в 2 байта. Преобразовать данные можно следующим образом:

```
while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY);
HAL_StatusTypeDef d = HAL_I2C_Master_Receive(&hi2c1, 0x51 << 1, buf_in, 2, 1000);

level = (buf_in[1] << 8) + buf_in[0];

snprintf(buff, sizeof(buff), "%d", level);
HAL_UART_Transmit(&huart1, (uint8_t*)buff, strlen(buff), 100);
HAL_Delay(1000)
```

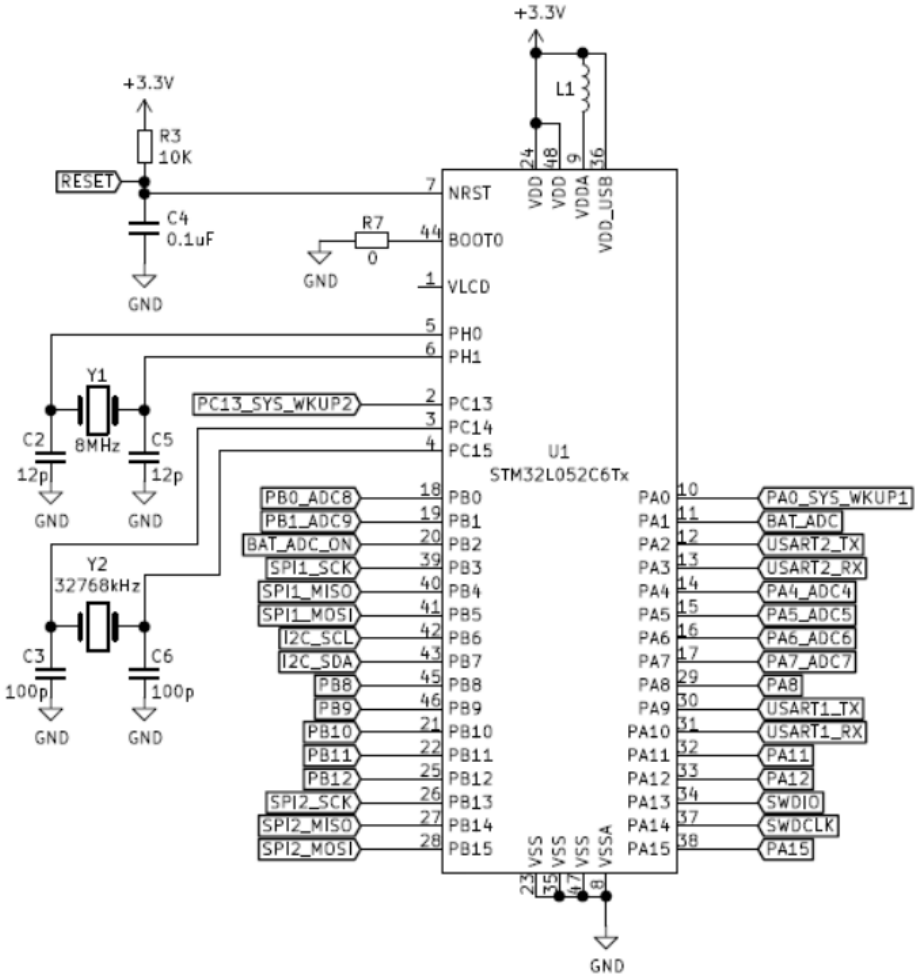
Радиотехника

Материалы предварительного практикума можно найти здесь: https://drive.google.com/file/d/1y6SoB-LNaWEOZmrtYQQZnCVxt90urXWt/view?usp=sharing

В течение дня было необходимо:

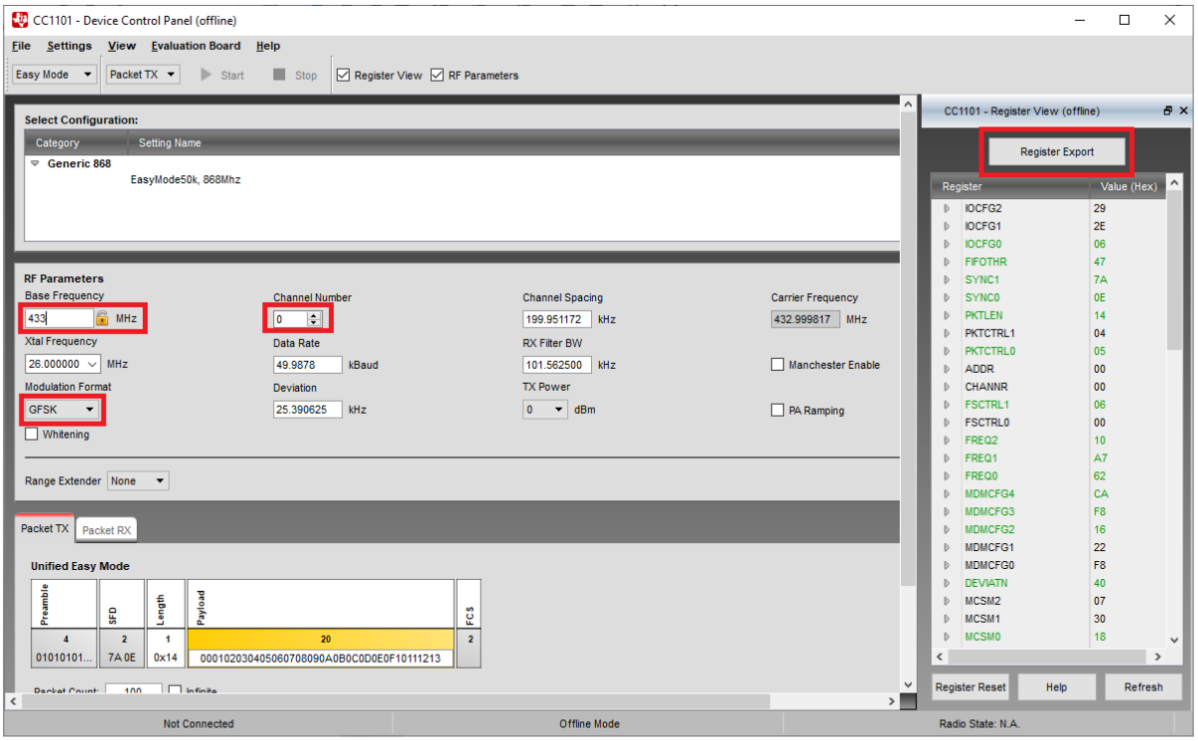
- настроить тестовую передачу с передатчика, расположенного на пикоспутнике;
- настроить тестовый прием с приемника (для этого использовать дополнительный контроллер STM32);
- вывести передаваемые данные в UART.

Для проверки трансляции сигнала можно использовать SDR# или любой его аналог и RTL-SDR-приемник. Микроконтроллер, который используется на плате пикоспутника — STM32L052C6 поэтому при распиновке в STM32CubeIDE необходимо выбирать именно его (LQFP). Распиновку микроконтроллера, соответствующую разводке платы можно видеть ниже.



CC1101, расположенный на плате, подключен к интерфейсу SPI2, а датчик MPU-6250 — к шине I2C1. Пины PB8 и PB9 необходимо задать как выходные и поставить низкий уровень напряжения: они отвечают за питание MPU-6250 и CC1101. Необходимо учитывать эту информацию при определении распиновки микроконтроллера.

Также с помощью Smart RF необходимо сменить канал передачи и изменить соответствующий регистр (CHANNR) в конфигурации приемника и передатчика.



Параметры для радиоканала:

- частота передачи: 433 МГц
- канал передачи: см. таблицу ниже
- модуляция: на выбор команды

Для команд на все время финала были заданы следующие номера каналов:

Satelite STR	1
Суета	3
Инженеры будущего в космосе	5
GDZZ 2.1	7
Реверсивное движение vc next to space	9
Baystation12	11
IncredibleSats	17
зеленая мыша	19

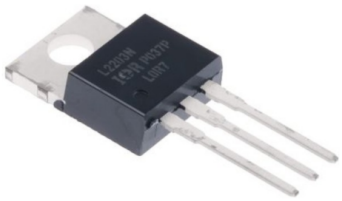
В примере конфигурации передатчика в описании практикума используется модуляция 2-FSK, но команды могли выбрать любой доступный способ. Также в приводимой конфигурации используется передача с нефиксированной длиной пакета данных, однако команды могли зафиксировать в регистре PKTLEN длину передаваемого пакета.Отдельно обращалось внимание на структуру пакета, которая приводится в SmartRF.

В datasheet дано описание каждого регистра в конфигурации. Ссылка на datasheet CC1101:

https://www.ti.com/lit/ds/symlink/cc1101.pdf

Схемотехника

Первая задача касалась подготовки к системе раскрытия солнечных панелей. В ней необходимо разместить на плате прототипирования транзистор, который при подаче на соответствующий контакт управления будет нагревать вольфрамовую нить, к которой будет подсоединена леска. При нагреве леска будет перегорать, а солнечные панели раскрываться.



Ссылка на даташит транзистора:

https://static.chipdip.ru/lib/197/DOC000197034.pdf

Затвор должен будет подключаться к пину GPIO управляющего микроконтроллера. На него будет подаваться сигнал управления. Между стоком и плюсом защищенного выхода с батареи, расположенным на плате питания и подписанным как Battery, необходимо закрепить нихромовую проволоку. Нихромовая проволока будет использоваться в качестве нагревателя, переправляющего леску. Длина (сопротивление) закрепляемой проволоки должна быть рассчитана в соответствии с максимально доступным для снятия с разъема Battery током (1А) при напряжении 3.7в. Исток нужно подсоединить к земле.

Допускалось предварительно отработать схему на макетной плате, а затем перенести на плату прототипирования.

Помимо сборки транзистора для раскрытия солнечных панелей необходимо начать работу с платой питания для пикоспутника — ключевой для этого направления на итоговой защите.

Общая логика схемы питания выглядит следующим образом:

Для начала необходимо подсоединить солнечные батареи к стабилизатору, чтобы ограничить напряжение (до 5.5В) для зарядки ионисторов. На выбор предоставляется два стабилизатора: LM317 и LP2951. Плата с солнечными батареями содержит 4 солнечных элемента, соединенных последовательно.

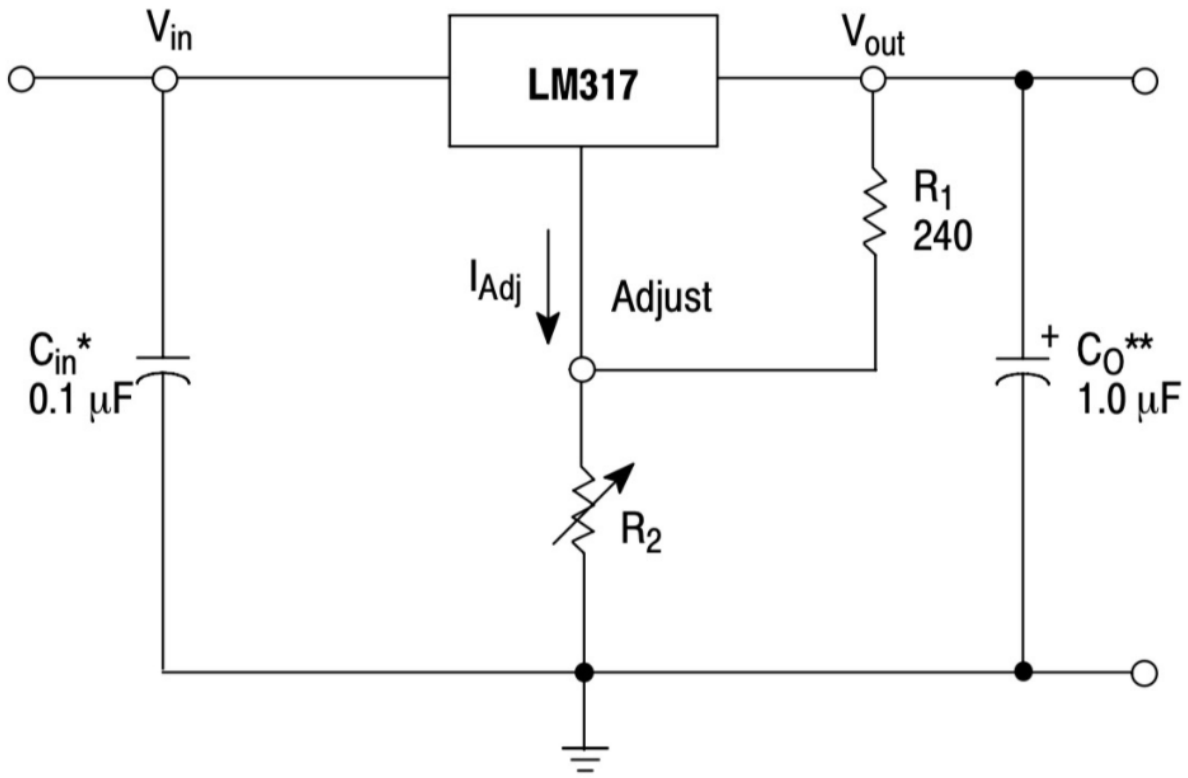
LM317 — это трехвыводной регулируемый стабилизатор. Его особенность состоит в том, что можно задавать различные значения напряжения на его выходе. Он

имеет три вывода: вход, выход и регулировочный вывод, причем между регулировочным выводом и выходом всегда поддерживается постоянное напряжение UREF, обусловленное внутренней схемой стабилизатора. Значение этого напряжения можно посмотреть в спецификации на конкретный стабилизатор; для LM317 оно равно 1.25 В.

Пример с трехвыводным регулируемым стабилизатором рассматривался на вебинаре по схемотехнике:

- презентация с вебинара:
  `https://docs.google.com/presentation/d/16wOmN6W7xH6pQp2J2ddWcqfpzwHO7uvxj549TP4O2lg/edit#slide=id.gc095841742_0_28`
- видеозапись с вебинара:
  https://www.youtube.com/watch?v=_4OwfkkexEU&t=1s

Схема подключения LM317:



Регулировочный вывод и выход соединяют через делитель напряжения (резисторы R1 и R2), чтобы можно было настраивать значение напряжения на выходе. Рекомендации по значению сопротивления резистора R1 обычно даются изготовителем, например, для LM317 подойдет резистор номиналом 240 Ом. Номинал резистора R2 можно рассчитать или, используя подстроечный резистор, подобрать.

Cin и Co являются блокировочными конденсаторами. Они уменьшают уровень помехи и не дают схеме перейти в режим генерации.

Datasheet на стабилизатор LM317:

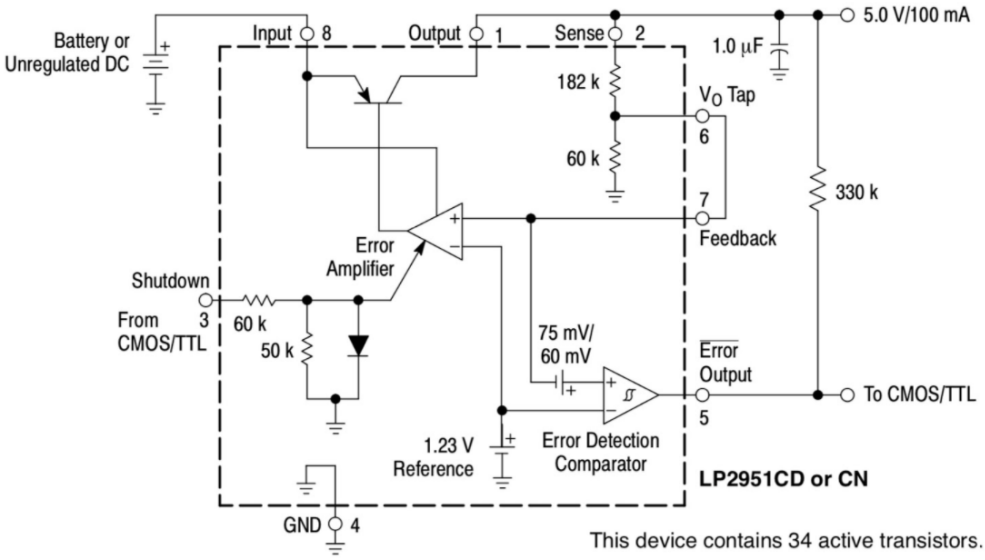
https://static.chipdip.ru/lib/159/DOC000159839.pdf

Стабилизатор LP2951 также является регулируемым, но он имеет встроенный

делитель напряжения, позволяющий получить на выходе 5 В. Данный стабилизатор примечателен тем, что имеет малое падение напряжения между входом и выходом (low-dropout). Для подключения стабилизатора также потребуются блокировочные конденсаторы.

Имеющийся стабилизатор имеет большее количество выводов, которые предоставляют дополнительные возможности для работы с ним. Подробнее о назначении и подключении этих выводов можно прочитать в datasheet: http://www.farnell.com/datasheets/2255355.pdf

Упрощенная схема и выводы LP2951:



Чтобы использовать встроенный делитель напряжения, необходимо соединить выводы Feedback и выход встроенного делителя Vo Tap, а также вход встроенного делителя Sense с выходом стабилизатора Output. Выход Error рекомендуется подтягивать к выходу стабилизатора через резистор большого номинала, например, 300 кОм.

Перед подключением к СБ схемы не лишним также будет добавить конденсатор, сглаживающий пульсации. Помимо этого, возможно нужно будет скорректировать силу тока от СБ.

В течение дня необходимо продемонстрировать в сборке на макетной плате схему с выбранным стабилизатором и СБ от пикоспутника с необходимой обвязкой, которая на выходе стабильно выдает 5В.

Орбитальная механика

В течение дня необходимо было решить первую доступную в симуляторе задачу на платформе Орбита.Челлендж. Задача было доступно в событии финала под названием «Задание на 1 день. Орбитальная механика.».

Для решения задачи можно использовать аналитические методы и/или ПО GMAT для исследования и подбора орбит.

Условие задачи:

Аппарат вышел на орбиту, однако по расчетному времени пролетов над ЦУП не выходит на связь. Через некоторое время удалось поймать сигнал от аппарата, однако не в то время и не в том месте, что может свидетельствовать о том, что аппарат находится не на той орбите.

Необходимо уточнить параметры текущей орбиты аппарата. Известно, что:

- проход над Сан-Франциско (37.7749; 237.581) был в момент времени с 22:07 по 22:12;
- проход над Портсмутом (50.799; 358.909) был в момент времени с 13:54 по 13:59;
- проходы над Йоханненсбургом (-26.2023; 28.0436) были в моменты времени с 12:39 по 12:44 и с 00:41 по 00:43.

Параметры планируемой орбиты миссии:

- большая полуось: 6870 км
- наклонение: 80°
- эксцентриситет: 0
- аргумент перицентра: 0
- долгота восходящего узла: 236°
- истинная аномалия: 45°

Время вывода аппарата на орбиту: 04 апреля 2021 года 12:00 по UTC. Время моделирования — 24 часа.

Известно, что наклонение и долгота восходящего узла остались неизменными.

Радиус Земли, м: 6371008.8

Угол над горизонтом, при котором возможен прием данных: 30°

За каждый проход над станцией в установленный промежуток времени начисляется 2.5 балла. Максимальный балл за задачу — 10 баллов. Начиная с 5-ой попытки начинается дисконт за каждую попытку, равный 10% максимального балла. Балл уменьшается вплоть до 1 балла, и далее не уменьшается с числом попыток.

Исходные данные и промежуточные задачи участников, 2 день

Разработка и программирование спутника

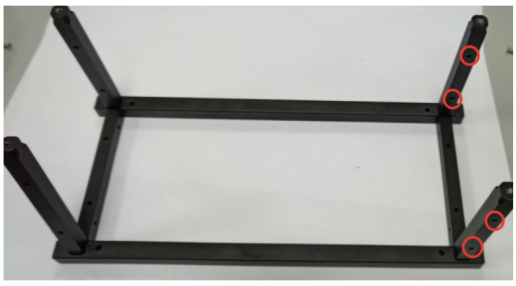
В рамках направления было необходимо осуществить сборку прототипа аппарата, реализовать раскрытие солнечных батарей и простую циклограмму полета (автоматическую последовательность действий).

Сборка:

Корпус собирается из трех элементов: рельса и двух типов перекладин.

Вставьте в пазы рельсов перекладины типа 1

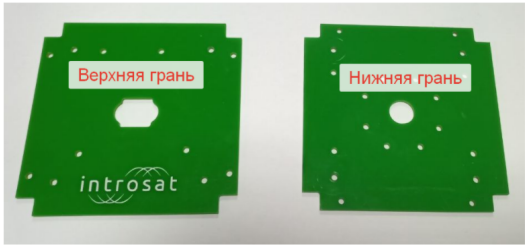
Возьмите перекладины типа 2 и вставьте в соединения рельсов и перекладин типа 1 так, чтобы отверстия на перекладине были соосны перекладине (расположение отверстий выделено на фото красным цветом).



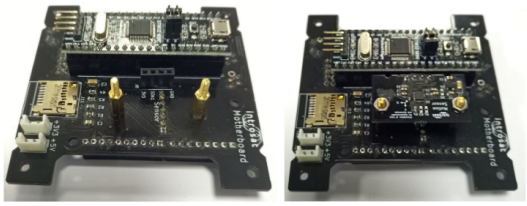
Закрепите сборку винтами.

Соедините оставшиеся перекладины и рельсы аналогичным способом и закрепите винтами.

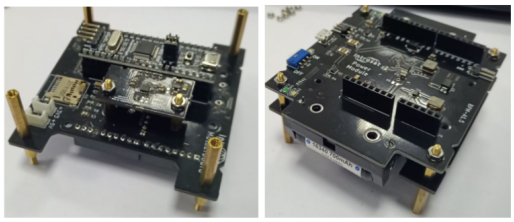
Рама собрана. Теперь необходимо закрепить электронику. Сборка «материнская плата + плата питания + плата прототипирования» крепится к верхней грани, а плата маховика — к нижней.



Установите на материнской плате стойки, как показано на фото. Закрепите стойки винтами. Установите на стойки и соответствующий разъем датчик позиционирования и закрепите его положение гайками.

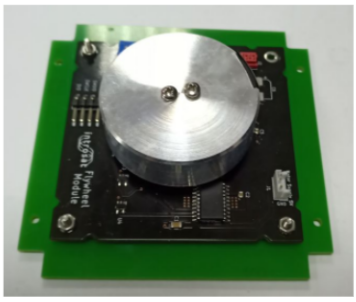


В углах материнской платы установите стойки как показано на фото. Далее как «шилд» установите плату питания и закрепите гайками.



При необходимости подсоедините плату прототипирования соответствующим образом поверх платы питания. Затем вставьте стек плат в раму и прикрепите ее к верхней плате. Обязательно сначала вставьте платы в раму и только потом прикрепите ее к грани.

К нижней грани прикрепите плату маховика. Закрепите ее винтами.



Раскрытие СБ:

Раскрытие солнечных батарей на материнском спутнике осуществляется с помощью освобождения энергии сжатого ленточно-пружинного шарнира. Такой тип шарниров часто используется на космических аппаратах.

На борту спутника имеются две секции раскрывающихся панелей СБ (объединенные по две панели СБ в одной секции). Две панели СБ закрепляются на ленте с помощью винтов и гаек.

Собранная секция крепится к одной из рельс и складывается таким образом, чтобы соблюдался минимальный выступ над рельсами над внешней плоскостью рельс.

После установки и закрепления двух секций СБ необходимо произвести складывание панелей и т. н. зачековку (фиксация с сложенном состоянии, рассчитанное на последующее автоматическое раскрытие). Зачековка производится с помощью лески, один конец которой привязывается к нихромовой нити, другой конец пропускается через отверстия 1 мм на панелях, обеспечивая плотное прижатие панелей и их свободное раскрытие. Имеется несколько способов фиксации свободного конца лески: обратно к нихромовой нити на плате, обвязать винт и прижать леску гайкой, привязать к латунной стойке.

Циклограмма полета:

После финальной сборки необходимо отработать первичную циклограмму работы спутника, которая состоит из двух последовательных:

- раскрытие СБ;
- одноосная стабилизация.

Раскрытие СБ осуществляется простой командой подачи высокого уровня на

GPIO-пин, а чтобы осуществить стабилизацию можно воспользоваться материалом практикума, а также следующими материалами:

- пример одноосной стабилизации на примере конструктора Орбикрафт (используется язык Python и C, а также другой API, так что прежде всего следует обращать внимание на логику алгоритма): http://www.orbicraft.sputnix.ru/doku.php?id=stabilization
- презентация о ПИД-регулировании: https://docs.google.com/presentation/d/1sryv9FPD7bgnth8GfgXVUfTG1C4VnbKOAKYBJpgS7FQ/edit#slide=id.g61b1d0c8d6_0_226

Радиотехника

Помимо CC1101 на плате пикоспутника расположен датчик MPU-9250. Необходимо сделать так, чтобы с датчика по I2C считывались данные с датчика. Пример чтения данных с датчика приведен в практикуме:

https://drive.google.com/file/d/1xxFppA0v38G2lbg0mR5UY5HsJqymjU8f/view?usp=sharing

Далее данные должны будут передаваться в радиоканал по запросу. Циклограмма работы радиоканала:

- запрос с материнского спутника на передачу данных;
- передача пакета данных с пикоспутника.

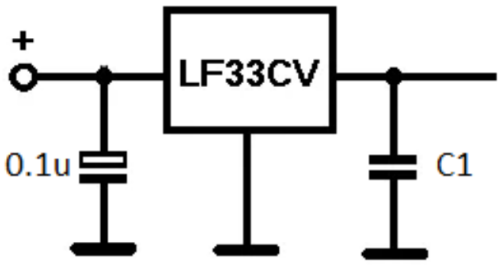
После реализации данной задачи можно будет приступить к помехоустойчивому кодированию.

Схемотехника

К собранному стабилизатору можно подключить от одного до четырех ионисторов. Вашей задачей является определить сколько необходимо ионисторов для работы схемы и собрать схему с ними. Перед подключением ионисторов необходимо ограничить протекающий через них ток и предотвратить разряд ионисторов при затемнении СБ.

После этого необходимо, используя схему на LF33, собрать стабилизатор для выходного значения напряжения с ионисторов. На выходе со стабилизатора должно быть 3.3В.

Схема подключения LF33:



Необходимо подобрать номинал конденсатора C1 так, чтобы на выходе устойчиво было 3.3В.

Ваша собранная схема питания при заряженных ионисторах должна выдерживать 10 минут без освещения, падающего на СБ, а также обеспечивать достаточно быструю зарядку ионисторов.

После сборки схемы на макетной плате и проверки ее на режим освещения можно переходить к пайке. Орбитальная механика В течение дня необходимо решить вторую доступную в симуляторе задачу на платформе Орбита.Челлендж.

Условие задачи:

Аппарат был выведен на полярную орбиту высотой 420 км. После вывода на орбиту аппарат остался ориентирован в плоскости своей орбиты; после завершения испытаний маховики аппарата были отключены, так что аппарат может быть подвержен вращению в этой плоскости.

Перед выводом пикоспутников необходимо перевести его на орбиту 320 км (ниже большинства орбит других космических аппаратов, и для сокращения срока пребывания пикоспутников на орбите), задав программу управления аппаратом. На аппарате находится двигатель с удельным импульсом 1079 м/с и максимальной тягой 0.98 Н. Сухая масса аппарата — 10 кг. Количество топлива — 0.572 кг.

Симуляция идет 7 часов. Баллы за нахождение на нужной высоте начинают начисляться через 2 часа после начала симуляции.

Начальная ориентация аппарата — по ходу движения, над экватором. Двигатель направлен при этом против движения.

Для решения задачи рекомендуется воспользоваться решением задачи ориентации прошлого года:

https://drive.google.com/file/d/1d0QSN4r0BFuCyu8VUyoRolTturQMdAcr/view?usp=sharing

Радиус Земли, м: 6371008.8

Гравитационный параметр, $\mu = G \cdot M$, м^3/с^2: $3.986004418 \cdot 10^{14}$

API для аппарата:

`https://docs.google.com/document/d/1qLeV2Prk4sPQuiS1D1GjCCjJSS43NVlX6_dKb-vRTmg/edit?usp=sharing`

За каждую 0.01 сек начисляется 0.0008333 балла, если аппарат находится на требуемой высоте с погрешностью 5 км или 0.0001666 балла, если с погрешностью 10 км.

Максимальный балл за задачу — 15 баллов. Начиная с 5-ой попытки начинается дисконт за каждую попытку, равный 10% максимального балла. Балл уменьшается вплоть до 1.5 балла, и далее не уменьшается с числом попыток.

Исходные данные и промежуточные задачи участников, 3 день

Разработка и программирование спутника

Алгоритм стабилизации необходимо доработать до ориентации на свет. В таком случае, циклограмма спутника будет выглядеть следующим образом:

- раскрыв СБ;
- ориентация СБ на свет;
- переход в режим стабилизации.

Далее необходимо в спутник интегрировать передатчик CC1101. Сеанс связи должен начинаться после перехода в режим стабилизации.

Радиотехника

В течение для необходимо окончательно реализовать помехоустойчивое кодирование, а также интегрировать работу радиоканала в циклограмму спутника.

Схемотехника

В течение дня необходимо окончательно спаять плату питания и соединить ее с материнской платой пикоспутника. Пикоспутник должен работать по следующей циклограмме:

- зарядка ионисторов от освещения;
- осуществление сеанса связи с материнским спутником по запросу.

Орбитальная механика

В течение дня необходимо решить третью доступную в симуляторе задачу на платформе Орбита.Челлендж.

Условие задачи:

Аппарат вышел на орбиту, однако по расчетному времени пролетов над ЦУП не выходит на связь. Через некоторое время удалось поймать сигнал от аппарата, однако не в то время и не в том месте, что может свидетельствовать о том, что аппарат находится не на той орбите. Необходимо уточнить параметры текущей орбиты аппарата и произвести корректировку высоты и эксцентриситета орбиты до желаемых высоты и эксцентриситета. Известно, что:

- проход над Сан-Франциско (37.7749; 237.581) был в момент времени с 16:52 по 16:56;
- проходы над Портсмутом (50.799; 358.909) были в моменты времени с 21:25 по 21:27 и с 09:13 по 09:16;
- проходы над Йоханненсбургом (-26.2023; 28.0436) были в моменты времени с 19:26 по 19:30 и с 06:18 по 06:20.

Время вывода аппарата на орбиту: 06 апреля 2021 года 12:00 по UTC. Время моделирования — 29 часа. После вывода на орбиту аппарат остался ориентирован в плоскости своей орбиты.

С 13:00 7 апреля надо быть на скорректированной орбите. До этого аппарат будет двигаться по введенной орбите, при прохождении над станциями в указанный промежуток времени начисляется балл.

На аппарате находится двигатель с удельным импульсом 1079 м/с и максимальной тягой 0.98 Н. Сухая масса аппарата — 10 кг. Количество топлива — 2 кг. Минимальная тяга двигателя — 0.0098 Н.

Параметры планируемой орбиты:

- большая полуось: 6840 км
- наклонение: 90°
- эксцентриситет: 0

Наклонение осталось неизменным.

Для решения задачи рекомендуется воспользоваться решением задачи ориентации прошлого года:

https://drive.google.com/file/d/1CfsIpF-1HHEsRLwjwaPo7K4a6o9GOpBd/view?usp=sharing

Радиус Земли, м: 6371008.8

Гравитационный параметр, $\mu = G \cdot M$, м^3/с^2: $3.986004418 \cdot 10^{14}$

API для аппарата актуально из предыдущей задачи: https://docs.google.com/document/d/1qLeV2Prk4sPQuiS1D1GjCCjJSS43NVlX6_dKb-vRTmg/edit?usp=sharing

Угол над горизонтом, при котором возможен прием данных: 30°

За каждый проход над станцией в установленный промежуток времени начисляется 2 балла.

За каждую 0.01 сек начисляется 0.0000104166 балла, если аппарат находится на требуемой высоте с погрешностью 10 км или 0.0000052083 балла, если с погрешностью 15 км.

Максимальный балл за задачу — 25 баллов. Начиная с 5-ой попытки начинается дисконт за каждую попытку, равный 10% максимального балла. Балл уменьшается вплоть до 2.5 балла, и далее не уменьшается с числом попыток.

Критерии оценки

Критерии оценки промежуточных задач

Задача		Макс. балл
	Разработка и программирование спутников	
1 день	Данные с датчиков выводятся по UART	4

1 день	Реализовано управление двигателем	4
1 день	Отработано управление транзистором	2
2 день	Реализовано раскрытие СБ	3
2 день	Реализована стабилизация КА	7
2 день	Реализована циклограмма спутника, соответствующая заданному порядку действий	5
3 день	Реализован алгоритм ориентации	25
Всего по направлению «Разработка и программирование спутников»		50
	Радиотехника	
1 день	Реализована передача сигнала	5
1 день	Реализован прием сигнала	5
2 день	Осуществлена передача данных с датчика	3
2 день	Полностью осуществлена циклограмма работы передатчика	12
3 день	Реализован помехоустойчивый алгоритм	25
Всего по направлению «Радиотехника»		50
	Схемотехника	
1 день	Собрана схема с транзистором	4
1 день	Схема со стабилизатором на LM317 выдает стабильное напряжение	6
2 день	На выходе LF33 получено устойчиво напряжение 3.3В	5
2 день	Собранная схема питания работает в требуемом режиме (обеспечивает питание в течение 10 минут без освещения)	10
3 день	Плата питания работает в штатном режиме	25
Всего по направлению «Схемотехника»		50
	Орбитальная механика	
1 день	Первая задача в симуляторе	10
2 день	Вторая задача в симуляторе	15
3 день	Третья задача в симуляторе	25
Всего по направлению «Орбитальная механика»		50
ИТОГО за промежуточные задания		200

Критерии оценки финальных защит

Критерий		Макс. балл
Спутник-носитель собран, все компоненты надежно закреплены, не используются кустарные методы крепления (клей, изолента и т.п.) форм-фактор кубсат не нарушен (с допущением на складные панели), телеметрия получается на ПК через модуль связи		10
Пикоспутник собран в единый аппарат, система электропитания надежно спаяна на бортовой плате расширения, не используются дополнительные платы расширения, работает в автономном режиме, осуществляет связь со спутником-носителем, включая передачи данных датчика положения		10
Функций должны быть выполнены последовательно, начиная с получения сигнала с ПК. Начисляется до 3 баллов за функцию и еще до 3 баллов за каждый не пропущенный, в правильной последовательности выполненный пункт	Раскрытие солнечных батарей в плоскость	3+3
	Переход в режим стабилизации	3+3
	Запрос на получение данных	3+3
	Прием данных с пикосата во время циклограммы.	3+3
Осуществляется передача сигнала при включенном постановщике помех. Команда с наименьшим числом повреждений переданного сигнала получает максимум баллов, каждая следующая по возрастанию количества ошибок получает на 3 балла меньше. Отсутствие передачи тестового сигнала оценивается как 0 баллов.		25
Сравнение эффективности системы электропитания пикоспутника (по протоколу). При отсутствии автономности или использовании готовой платы от организаторов — 0 баллов за весь раздел.	Пикоспутник заряжается менее, чем за 5 минут	5
	Используется наименьшее число ионисторов (среди образцов, представленных всемикомандами) для действующей автономной системы	5
	Сравниваются пикоспутники по продолжительности автономной передачи сигнала при идущей радиопередаче. Лучшая команда получает 15 баллов, далее каждая следующая команда получает на 2 балла меньше.	15
ИТОГО:		100

Решение

Разработка и программирование спутника

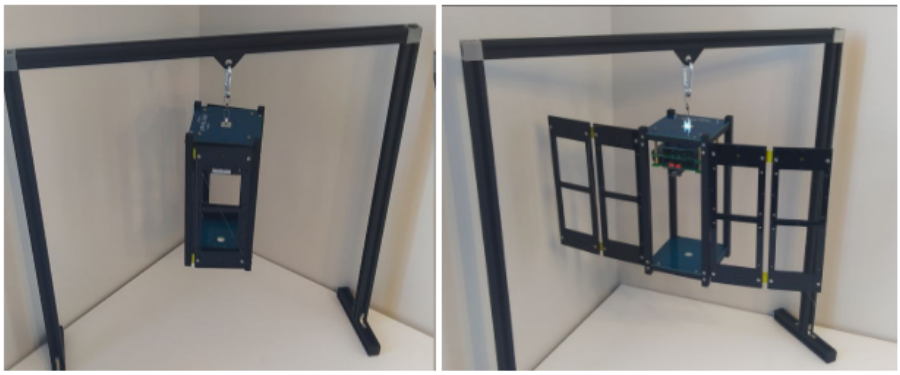
Код чтения данных с датчиков и управления двигателем и транзистором:

```
/* Infinite loop */
  /* USER CODE BEGIN WHILE */
  while (1)
  {
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */

        /*ЧТЕНИЕ ДАННЫХ С ДАТЧИКА УГЛОВОЙ СКОРОСТИ*/
```

```
10          char buff[8]; // переменная для вывода значений в UART
11          uint8_t gyro[6]; // переменная, в которую записываются данные с гироскопа
12          int16_t gx; // переменная со значением угловой скорости по оси X
13          int16_t gy; // переменная со значением угловой скорости по оси Y
14          int16_t gz; // переменная со значением угловой скорости по оси Z
15
16          //устанавливаем регистр, с которого необходимо считать данные
17          while (HAL_I2C_GetState((&hi2c1) != HAL_I2C_STATE_READY);
18          HAL_I2C_Master_Transmit((&hi2c1, 0x68 << 1, 0x43, 1, 100);
19
20          //считываем данные с датчика
21          while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY);
22          HAL_I2C_Master_Receive(&hi2c1, 0x68 << 1,  gyro, 6, 100);
23
24          //преобразуем считанные данные
25          gx = gyro[0] << 8 | gyro[1];
26          gy = gyro[2] << 8 | gyro[3];
27          gz = gyro[4] << 8 | gyro[5];
28
29          //выводим данные в UART
30          HAL_UART_Transmit(&huart1, (uint8_t*)"X Gyro:", strlen("X Gyro:"), 10);
31          snprintf(buff, sizeof(buff), "%d", gx);
32          HAL_UART_Transmit(&huart1, (uint8_t*)buff, strlen(buff), 10);
33          HAL_UART_Transmit(&huart1, (uint8_t*)"\r\n", 2, 10);
34          HAL_Delay(10);
35
36          HAL_UART_Transmit(&huart1, (uint8_t*)"Y Gyro:", strlen("Y Gyro:"), 10);
37          snprintf(buff, sizeof(buff), "%d", gy);
38          HAL_UART_Transmit(&huart1, (uint8_t*)buff, strlen(buff), 10);
39          HAL_UART_Transmit(&huart1, (uint8_t*)"\r\n", 2, 10);
40          HAL_Delay(10);
41
42          HAL_UART_Transmit(&huart1, (uint8_t*)"Z Gyro:", strlen("Z Gyro:"), 10);
43          snprintf(buff, sizeof(buff), "%d", gz);
44          HAL_UART_Transmit(&huart1, (uint8_t*)buff, strlen(buff), 10);
45          HAL_UART_Transmit(&huart1, (uint8_t*)"\r\n", 2, 10);
46          HAL_Delay(10);
47
48
49          /*ЧТЕНИЕ ДАННЫХ С ДАТЧИКОВ ОСВЕЩЁННОСТИ*/
50          uint8_t ISMI_ADDR;
51          int16_t val;
52          uint8_t buf[32];
53
54          for(int i = 0; i<4; i++){
55
56              ISMI_ADDR = 0x50+i;
57              ISMI_ADDR = ISMI_ADDR << 1;
58
59              // устанавливаем регистр, с которого необходимо считать данные
60              HAL_StatusTypeDef ret = HAL_I2C_Master_Transmit(&hi2c1, ISMI_ADDR, buf, 1,
                ↪  HAL_MAX_DELAY); //1000
61                  if ( ret != HAL_OK )
62                  {
63                  }
64                  else {
65                          ret = HAL_I2C_Master_Receive(&hi2c1, ISMI_ADDR, buf, 2,
                            ↪  HAL_MAX_DELAY);
66                          if ( ret != HAL_OK )
67                          {
```

```
                        } else
                        {
                                val = (buf[1] << 8) + buf[0];
                                snprintf(buf, sizeof(buf), "%d", val);
                                HAL_UART_Transmit(&huart1, buf, strlen((char*)buf),
                                ↪  1000);
                        }
                }
        }

        /*УПРАВЛЕНИЕ ДВИГАТЕЛЕМ*/
        uint8_t speed[3];
    if (gz > 0)
    {
      speed[0] = 0;
    }
    else
    {
      speed[0] = 1;
    }

    uint16_t mot_speed = 1000;
    speed[1] = mot_speed ^ 0x0F;
    speed[2] = mot_speed  >> 8;

    HAL_I2C_Master_Transmit(&hi2c1, 0x33 << 1, motor, 3, 100);

    /*УПРАВЛЕНИЕ ТРАНЗИСТОРОМ*/
    HAL_GPIO_WritePin(DEPLOY_PIN_GPIO_Port, DEPLOY_PIN_Pin, GPIO_PIN_SET);
    HAL_Delay(1000);
    HAL_GPIO_WritePin(DEPLOY_PIN_GPIO_Port, DEPLOY_PIN_Pin, GPIO_PIN_RESET);
    HAL_Delay(1000);

}
```

Сборка граней с раскрытием СБ:



Код для стабилизации КА:

```
/* USER CODE BEGIN 1 */
        uint8_t buf[32];

        char buff[8]; // переменная для вывода значений в UART
        uint8_t gyro[6]; // переменная, в которую записываются данные с гироскопа
```

```
6           int16_t gx; // переменная со значением угловой скорости по оси X
7           int16_t gy; // переменная со значением угловой скорости по оси Y
8           int16_t gz; // переменная со значением угловой скорости по оси Z
9
10          int kp = 1;// пропорциональная составляющая регулятора
11          int ki = 0.01;// интегральная составляющая регулятора
12          int kd = 1.2;// дифференциальная составляющая регулятора
13
14          int16_t gz_old = 0;
15               uint16_t mot_speed;
16          uint8_t speed[3];
17
18    /* USER CODE END 1 */
19
20  /* ЧАСТЬ АВТОМАТИЧЕСКИ ГЕНЕРИРУЕМОГО КОДА ЗДЕСЬ НЕ ПРИВОДИТСЯ*/
21
22  while (1)
23    {
24      /* USER CODE END WHILE */
25
26      /* USER CODE BEGIN 3 */
27
28          /*ЧТЕНИЕ ДАННЫХ С ДАТЧИКА УГЛОВОЙ СКОРОСТИ*/
29
30          //устанавливаем регистр, с которого необходимо считать данные
31          while (HAL_I2C_GetState((&hi2c1) != HAL_I2C_STATE_READY);
32          HAL_I2C_Master_Transmit((&hi2c1, 0x68 << 1, 0x43, 1, 100);
33
34          //считываем данные с датчика
35          while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY);
36          HAL_I2C_Master_Receive(&hi2c1, 0x68 << 1,  gyro, 6, 100);
37
38          //преобразуем считанные данные
39          gx = gyro[0] << 8 | gyro[1];
40          gy = gyro[2] << 8 | gyro[3];
41          gz = gyro[4] << 8 | gyro[5];
42
43          /*РАСЧЁТ СКОРОСТИ СТАБИЛИЗАЦИИ*/
44          mot_speed = kp*gz + ki*(gz + gz_old)/2 + kd*(gz - gz_old);
45
46          if (mot_speed > 3000)
47          {
48                  mot_speed = 3000;
49          }else if(mot_speed < -3000){
50                  mot_speed = -3000;
51          }
52
53          if (mot_speed > 0)
54      {
55        speed[0] = 0;
56      }
57      else
58      {
59        speed[0] = 1;
60      }
61
62      speed[1] = mot_speed ^ 0x0F;
63      speed[2] = mot_speed  >> 8;
64
65      HAL_I2C_Master_Transmit(&hi2c1, 0x33 << 1, motor, 3, 100);
```

```
66
67          gz_old = gz;
68  }
```

Код для ориентации КА

```
1   while (1)
2     {
3       /* USER CODE END WHILE */
4
5       /* USER CODE BEGIN 3 */
6
7           /*ЧТЕНИЕ ДАННЫХ С ДАТЧИКОВ ОСВЕЩЁННОСТИ*/
8           uint8_t ISMI_ADDR;
9           int16_t val[4];
10          uint8_t buf[32];
11
12          for(int i = 0; i<4; i++){
13
14                  ISMI_ADDR = 0x50+i;
15                  ISMI_ADDR = ISMI_ADDR << 1;
16
17                  // устанавливаем регистр, с которого необходимо считать данные
18                  HAL_StatusTypeDef ret = HAL_I2C_Master_Transmit(&hi2c1, ISMI_ADDR,
                    ↪  buf, 1, HAL_MAX_DELAY); //1000
19                  if ( ret != HAL_OK )
20                  {
21                  }
22                  else {
23                          ret = HAL_I2C_Master_Receive(&hi2c1, ISMI_ADDR, buf, 2,
                            ↪  HAL_MAX_DELAY);
24                          if ( ret != HAL_OK )
25                          {
26                          } else
27                          {
28                                  val[i] = (buf[1] << 8) + buf[0];
29                                  snprintf(buf, sizeof(buf), "%d", val[i]);
30                                  HAL_UART_Transmit(&huart1, buf, strlen((char*)buf),
                                    ↪  1000);
31                          }
32                  }
33          }
34
35          int min_light = val[0];
36          int k = 1;
37      for(int j = 1; j < 4; j++)
38      {
39          if(val[j] < min_light)
40          {
41              min_light = val[j];
42              k = j + 1;
43          }
44      }
45
46          /*УПРАВЛЕНИЕ ДВИГАТЕЛЕМ*/
47          uint8_t speed[3];
48
49          if (k == 1 | | k == 2)
50          {
```

```
                speed[0] = 0;
        }

        if (k == 3)
        {
                speed[0] = 1;
        }


        uint16_t mot_speed = 1000;

    speed[1] = mot_speed ^ 0x0F;
    speed[2] = mot_speed  >> 8;

    HAL_I2C_Master_Transmit(&hi2c1, 0x33 << 1, motor, 3, 100);
}
```

Радиотехника

од для приемопередатчика спутника-носителя:

```
/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
#define WRITE_BURST              0x40
#define READ_SINGLE              0x80
#define READ_BURST               0xC0

/*
 ===========CC10xx registers=================
 */

/**
 * PATABLE & FIFO's
 */
#define CC1101_PATABLE           0x3E        // PATABLE address
#define CC1101_TXFIFO            0x3F        // TX FIFO address
#define CC1101_RXFIFO            0x3F        // RX FIFO address

/**
 * Command strobes
 */
#define CC1101_SRES              0x30        // Reset CC1101 chip
#define CC1101_SFSTXON           0x31        // Enable and calibrate frequency
↪  synthesizer (if MCSM0.FS_AUTOCAL=1). If in RX (with CCA):
                                             // Go to a wait state where only the
                                             ↪  synthesizer is running (for quick RX
                                             ↪  / TX turnaround).
#define CC1101_SXOFF             0x32        // Turn off crystal oscillator
#define CC1101_SCAL              0x33        // Calibrate frequency synthesizer and
↪  turn it off. SCAL can be strobed from IDLE mode without
                                             // setting manual calibration mode
                                             ↪  (MCSM0.FS_AUTOCAL=0)
#define CC1101_SRX               0x34        // Enable RX. Perform calibration first
↪  if coming from IDLE and MCSM0.FS_AUTOCAL=1
#define CC1101_STX               0x35        // In IDLE state: Enable TX. Perform
↪  calibration first if MCSM0.FS_AUTOCAL=1.
```

```
29                                                  // If in RX state and CCA is enabled:
                                                    ↪  Only go to TX if channel is clear
30  #define CC1101_SIDLE              0x36          // Exit RX / TX, turn off frequency
    ↪  synthesizer and exit Wake-On-Radio mode if applicable
31  #define CC1101_SWOR               0x38          // Start automatic RX polling sequence
    ↪  (Wake-on-Radio) as described in Section 19.5 if
32                                                  // WORCTRL.RC_PD=0
33  #define CC1101_SPWD               0x39          // Enter power down mode when CSn goes
    ↪  high
34  #define CC1101_SFRX               0x3A          // Flush the RX FIFO buffer. Only issue
    ↪  SFRX in IDLE or RXFIFO_OVERFLOW states
35  #define CC1101_SFTX               0x3B          // Flush the TX FIFO buffer. Only issue
    ↪  SFTX in IDLE or TXFIFO_UNDERFLOW states
36  #define CC1101_SWORRST            0x3C          // Reset real time clock to Event1 value
37  #define CC1101_SNOP               0x3D          // No operation. May be used to get
    ↪  access to the chip status byte
38
39  /**
40   * Status registers
41   */
42  #define CC1101_PARTNUM            0x30          // Chip ID
43  #define CC1101_VERSION            0x31          // Chip ID
44  #define CC1101_FREQEST            0x32          // Frequency Offset Estimate from
    ↪  Demodulator
45  #define CC1101_LQI                0x33          // Demodulator Estimate for Link Quality
46  #define CC1101_RSSI               0x34          // Received Signal Strength Indication
47  #define CC1101_MARCSTATE          0x35          // Main Radio Control State Machine State
48  #define CC1101_WORTIME1           0x36          // High Byte of WOR Time
49  #define CC1101_WORTIME0           0x37          // Low Byte of WOR Time
50  #define CC1101_PKTSTATUS          0x38          // Current GDOx Status and Packet Status
51  #define CC1101_VCO_VC_DAC         0x39          // Current Setting from PLL Calibration
    ↪  Module
52  #define CC1101_TXBYTES            0x3A          // Underflow and Number of Bytes
53  #define CC1101_RXBYTES            0x3B          // Overflow and Number of Bytes
54  #define CC1101_RCCTRL1_STATUS     0x3C          // Last RC Oscillator Calibration Result
55  #define CC1101_RCCTRL0_STATUS     0x3D          // Last RC Oscillator Calibration Result
56
57  /**
58   *  Configuration Register number defines
59   */
60  #define CC1101_IOCFG2           0x00
61  #define CC1101_IOCFG1           0x01
62  #define CC1101_IOCFG0           0x02
63  #define CC1101_FIFOTHR          0x03
64  #define CC1101_SYNC1            0x04
65  #define CC1101_SYNC0            0x05
66  #define CC1101_PKTLEN           0x06
67  #define CC1101_PKTCTRL1         0x07
68  #define CC1101_PKTCTRL0         0x08
69  #define CC1101_ADDR             0x09
70  #define CC1101_CHANNR           0x0A
71  #define CC1101_FSCTRL1          0x0B
72  #define CC1101_FSCTRL0          0x0C
73  #define CC1101_FREQ2            0x0D
74  #define CC1101_FREQ1            0x0E
75  #define CC1101_FREQ0            0x0F
76  #define CC1101_MDMCFG4          0x10
77  #define CC1101_MDMCFG3          0x11
78  #define CC1101_MDMCFG2          0x12
79  #define CC1101_MDMCFG1          0x13
```

```
80  #define CC1101_MDMCFG0          0x14
81  #define CC1101_DEVIATN          0x15
82  #define CC1101_MCSM2            0x16
83  #define CC1101_MCSM1            0x17
84  #define CC1101_MCSM0            0x18
85  #define CC1101_FOCCFG           0x19
86  #define CC1101_BSCFG            0x1A
87  #define CC1101_AGCCTRL2         0x1B
88  #define CC1101_AGCCTRL1         0x1C
89  #define CC1101_AGCCTRL0         0x1D
90  #define CC1101_WOREVT1          0x1E
91  #define CC1101_WOREVT0          0x1F
92  #define CC1101_WORCTRL          0x20
93  #define CC1101_FREND1           0x21
94  #define CC1101_FREND0           0x22
95  #define CC1101_FSCAL3           0x23
96  #define CC1101_FSCAL2           0x24
97  #define CC1101_FSCAL1           0x25
98  #define CC1101_FSCAL0           0x26
99  #define CC1101_RCCTRL1          0x27
100 #define CC1101_RCCTRL0          0x28
101 #define CC1101_FSTEST           0x29
102 #define CC1101_PTEST            0x2A
103 #define CC1101_AGCTEST          0x2B
104 #define CC1101_TEST2            0x2C
105 #define CC1101_TEST1            0x2D
106 #define CC1101_TEST0            0x2E
107 #define CC1101_PARTNUM          0x30
108 #define CC1101_VERSION          0x31
109 #define CC1101_FREQEST          0x32
110 #define CC1101_LQI              0x33
111 #define CC1101_RSSI             0x34
112 #define CC1101_MARCSTATE        0x35
113 #define CC1101_WORTIME1         0x36
114 #define CC1101_WORTIME0         0x37
115 #define CC1101_PKTSTATUS        0x38
116 #define CC1101_VCO_VC_DAC       0x39
117 #define CC1101_TXBYTES          0x3A
118 #define CC1101_RXBYTES          0x3B
119 #define CC1101_RCCTRL1_STATUS   0x3C
120 #define CC1101_RCCTRL0_STATUS   0x3D
121
122
123 /* USER CODE END PD */
124
125 /* Private macro -------------------------------------------------------------*/
126 /* USER CODE BEGIN PM */
127 // Select (SPI) CC1101
128 #define cc1101_Select()    HAL_GPIO_WritePin(SPI2_CS_GPIO_Port, SPI2_CS_Pin,
    ↪  GPIO_PIN_RESET);
129 // Deselect (SPI) CC1101
130 #define cc1101_Deselect()    HAL_GPIO_WritePin(SPI2_CS_GPIO_Port, SPI2_CS_Pin,
    ↪  GPIO_PIN_SET);
131
132 /* USER CODE END PM */
133
134 /* ЧАСТЬ АВТОМАТИЧЕСКИ ГЕНЕРИРУЕМОГО КОДА ОПУЩЕНА */
135
136 /* Private user code ---------------------------------------------------------*/
137 /* USER CODE BEGIN 0 */
```

```
138
139 typedef struct registerSetting
140 {
141                 uint8_t reg;
142                 uint8_t value;
143 } registerSetting_t;
144
145 static const registerSetting_t config_MODE1[]=
146  {
147  {CC1101_FSCTRL1,   0x0A},   //      Frequency synthesizer control.
148  {CC1101_FSCTRL0,   0x00},   //      Frequency synthesizer control.
149  {CC1101_FREQ2 ,  0x10},   //       Frequency control word, high byte.
150  {CC1101_FREQ1,   0xA7},   //       Frequency control word, middle byte.
151  {CC1101_FREQ0,   0x62},   //       Frequency control word, low byte.
152  {CC1101_MDMCFG4,     0xC7}, //0010, Modem Configuration
153  {CC1101_MDMCFG3,     0x83}, //0011, Modem Configuration
154  {CC1101_MDMCFG2,     0x03}, //0012, Modem Configuration = 2-FSK
155  {CC1101_MDMCFG1,   0x22},   //      Modem configuration.
156  {CC1101_MDMCFG0 ,  0xF8},   //      Modem configuration.
157  {CC1101_CHANNR,   0x00},   //       Channel number.
158  {CC1101_DEVIATN,   0x34},   //      Modem deviation setting (when FSK modulation is
     ↪  enabled).
159  {CC1101_FREND1,   0xB6},   //      Front end RX configuration.
160  {CC1101_FREND0,   0x10},   //      Front end TX configuration.
161  {CC1101_MCSM0,   0x18},   //       Main Radio Control State Machine configuration.
162  {CC1101_FOCCFG ,  0x1D},   //      Frequency Offset Compensation Configuration.
163  {CC1101_BSCFG,   0x1C},   //       Bit synchronization Configuration.
164  {CC1101_AGCCTRL2 ,  0xC7},   //    AGC control.
165  {CC1101_AGCCTRL1,   0x00},   //    AGC control.
166  {CC1101_AGCCTRL0,   0xB2},   //    AGC control.
167  {CC1101_FSCAL3 ,  0xE9},   //      Frequency synthesizer calibration.
168  {CC1101_FSCAL2 ,  0x2A},   //      Frequency synthesizer calibration.
169  {CC1101_FSCAL1 ,  0x00},   //      Frequency synthesizer calibration.
170  {CC1101_FSCAL0,   0x1F},   //      Frequency synthesizer calibration.
171  {CC1101_FSTEST,   0x59},   //      Frequency synthesizer calibration.
172  {CC1101_TEST2,   0x81},   //       Various test settings.
173  {CC1101_TEST1,   0x35},   //       Various test settings.
174  {CC1101_TEST0,   0x09},   //       Various test settings.
175  {CC1101_FIFOTHR,   0x07},   //     RXFIFO and TXFIFO thresholds.
176  {CC1101_IOCFG2,   0x0B},   //      GDO2 output pin configuration.
177  {CC1101_IOCFG0,   0x06},   //     GDO0 output pin configuration.
178  {CC1101_PKTCTRL1,   0x04},   //    Packet automation control.
179  {CC1101_PKTCTRL0,   0x05},   //    Packet automation control.
180  {CC1101_ADDR,   0x00},   //       Device address.
181  {CC1101_PKTLEN,   0x00 } ,  //      Packet length.
182  {CC1101_PATABLE,     0x0E},
183  {CC1101_SYNC1,       0xD3}, //0004, Sync Word, High Byte  (1101001110010001 = 01 01
     ↪  00 01 00 00 01 01 01 00 00 01 00 00 00 01)
184  {CC1101_SYNC0,       0x91}, //0005, Sync Word, Low Byte
185 };
186
187
188 // Инициализация датчика. В соответствующий регистр пишется указанное значение
189 int CC_init(const registerSetting_t* config_array,int num_registers)
190 {
191     uint8_t data = CC1101_SRES; // Reset CC1101 chip
192     uint8_t tx[2];
193     uint8_t rx[2]; rx[0]='\0';rx[1]='\0';
194
195     HAL_SPI_Transmit(&hspi2, &data, 1, HAL_MAX_DELAY);
```

```
196
197         for (int regi=0;regi<num_registers;regi++)
198         {
199             tx[0] = config_array[regi].reg;
200             tx[1] = config_array[regi].value;
201             HAL_SPI_TransmitReceive(&hspi2, tx, rx, 2, HAL_MAX_DELAY);
202         }
203         return 0;
204 }
205
206 /* USER CODE END 0 */
207
208 /**
209   * @brief  The application entry point.
210   * @retval int
211   */
212 int main(void)
213 {
214   /* USER CODE BEGIN 1 */
215         char* buffer = "Check";
216         int num_registers = sizeof(config_MODE1)/sizeof(registerSetting_t);
217         uint8_t tx[2];
218         uint8_t rx[2]; rx[0]='\0';rx[1]='\0';
219         int flag = 0;
220
221   /* USER CODE END 1 */
222
223   /* MCU Configuration--------------------------------------------------------*/
224
225   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
226   HAL_Init();
227
228   /* USER CODE BEGIN Init */
229
230   /* USER CODE END Init */
231
232   /* Configure the system clock */
233   SystemClock_Config();
234
235   /* USER CODE BEGIN SysInit */
236
237   /* USER CODE END SysInit */
238
239   /* Initialize all configured peripherals */
240   MX_GPIO_Init();
241   MX_SPI2_Init();
242   MX_USART1_UART_Init();
243   /* USER CODE BEGIN 2 */
244   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET); //SPI2_SCK --> to 1
245   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET); //SPI2_MOSI--> to 0
246   cc1101_Select();
247   HAL_Delay(10);
248   CC_init(config_MODE1, num_registers);
249   cc1101_Deselect();
250
251   /* USER CODE END 2 */
252
253   /* Infinite loop */
254   /* USER CODE BEGIN WHILE */
255   while (1)
```

```
256   {
257     /* USER CODE END WHILE */
258
259     /* USER CODE BEGIN 3 */
260
261                 if (!flag)
262                 {
263                     // Set data length at the first position of the TX FIFO
264                     tx[0] = (uint8_t) CC1101_TXFIFO;
265                     tx[1] = (uint8_t) strlen(buffer);
266                     cc1101_Select();
267                     // отправляем на передатчик данные о длине передаваемого сообщения
268                     HAL_SPI_TransmitReceive(&hspi2, tx, rx, 2, HAL_MAX_DELAY);
269                     cc1101_Deselect();
270
271                     // Write data into the TX FIFO
272                     cc1101_Select();
273                     tx[0] = CC1101_TXFIFO | WRITE_BURST;
274                     // готовим передатчик к записи данных
275                     HAL_SPI_Transmit(&hspi2, tx, 1, HAL_MAX_DELAY);
276                     // записываем данные для передачи
277                     HAL_SPI_Transmit(&hspi2, (unsigned char*)buffer, strlen(buffer),
                        ↪  HAL_MAX_DELAY);
278                     cc1101_Deselect();
279
280                     // CCA enabled: will enter TX state only if the channel is clear
281                       cc1101_Select();
282                     tx[0] = CC1101_STX; //setTxState();
283                     rx[0]='\0';
284                     // задаем режим передачи данных для передатчика
285                     HAL_SPI_TransmitReceive(&hspi2, tx, rx, 1, HAL_MAX_DELAY);
286                     cc1101_Deselect();
287                 }
288
289             HAL_Delay(100);
290
291                 // Очищаем буфер
292                   cc1101_Select();
293                 tx[0] = CC1101_SFRX;
294                 rx[0]='\0';
295                 HAL_SPI_TransmitReceive(&hspi2, tx, rx, 1, HAL_MAX_DELAY); //single
                    ↪  byte strobe(reg no) out, status byte in
296                 cc1101_Deselect();
297
298                 // Устанавливаем режим приема данных
299                 cc1101_Select();
300                 tx[0] = CC1101_SRX;
301                 rx[0]='\0';
302                 HAL_SPI_TransmitReceive(&hspi2, tx, rx, 1, HAL_MAX_DELAY); //single
                    ↪  byte strobe(reg no) out, status byte in
303                 cc1101_Deselect();
304
305             // Считываем статус регистра
306                 uint8_t size;
307                 tx[0] = (uint8_t) CC1101_RXBYTES | READ_BURST;
308                 tx[1] = (uint8_t) 0;
309                 rx[0]='\0';
310                 rx[1]='\0';
311                 cc1101_Select();
312             HAL_SPI_TransmitReceive(&hspi2, &tx[0], &rx[0], 1, HAL_MAX_DELAY);
```

```
313             HAL_SPI_TransmitReceive(&hspi2, &tx[1], &rx[1], 1, HAL_MAX_DELAY);
314             cc1101_Deselect();
315 
316                 // Выделяем количество принятых байт
317                 size = rx[1] & 0x7F;
318                 if (size > 0)
319                 {
320                         flag = 1;
321 
322                         // Считываем значение регистра (сколько байт принято)
323                     tx[0] = (uint8_t) CC1101_RXFIFO | READ_SINGLE;
324                     tx[1] = (uint8_t) 0;
325                      rx[0]='\0';
326                        rx[1]='\0';
327                        cc1101_Select();
328                        HAL_SPI_TransmitReceive(&hspi2, tx, rx, 1, HAL_MAX_DELAY);
329                        HAL_SPI_TransmitReceive(&hspi2, &tx[1], &rx[1], 1,
                           ↪  HAL_MAX_DELAY);
330 
331                     cc1101_Deselect();
332                            // Выделяем количество принятых байт
333                        size = rx[1];// & 0x7F;
334 
335                        if (size > 0)
336                        {
337                                char buffer_rec[size];
338 
339                         // Считываем данные из буфера
340                         tx[0] = CC1101_RXFIFO | READ_BURST;
341                         tx[1] = (uint8_t) 0;
342                         rx[0]='\0';
343                         rx[1]='\0';
344                         cc1101_Select();
345                         HAL_SPI_Transmit(&hspi2, tx, 1, HAL_MAX_DELAY);
346                                HAL_SPI_Receive(&hspi2, (uint8_t*)buffer_rec, size,
                                   ↪  HAL_MAX_DELAY);
347                                HAL_UART_Transmit(&huart1, (uint8_t*)buffer_rec,
                                   ↪  strlen(buffer_rec), HAL_MAX_DELAY);
348                         cc1101_Deselect();
349                     }
350 
351                     HAL_Delay(100);
352   }
353 
354   /* USER CODE END 3 */
355 }
```

Код для приемопередатчика пикоспутника:

```
1  /* Private includes ----------------------------------------------------------*/
2  /* USER CODE BEGIN Includes */
3  #include <string.h>
4 
5  /* USER CODE END Includes */
6 
7  /* Private typedef -----------------------------------------------------------*/
8  /* USER CODE BEGIN PTD */
9 
10 /* USER CODE END PTD */
```

```
11
12  /* Private define ------------------------------------------------------------*/
13  /* USER CODE BEGIN PD */
14
15  #define WRITE_BURST             0x40
16  #define READ_SINGLE             0x80
17  #define READ_BURST              0xC0
18
19  /*
20   ==========CC10xx registers=================
21   */
22
23  /**
24   * PATABLE & FIFO's
25   */
26  #define CC1101_PATABLE          0x3E        // PATABLE address
27  #define CC1101_TXFIFO           0x3F        // TX FIFO address
28  #define CC1101_RXFIFO           0x3F        // RX FIFO address
29
30  /**
31   * Command strobes
32   */
33  #define CC1101_SRES             0x30        // Reset CC1101 chip
34  #define CC1101_SFSTXON          0x31        // Enable and calibrate frequency
    ↪  synthesizer (if MCSM0.FS_AUTOCAL=1). If in RX (with CCA):
35                                              // Go to a wait state where only the
                                                ↪  synthesizer is running (for quick RX
                                                ↪  / TX turnaround).
36  #define CC1101_SXOFF            0x32        // Turn off crystal oscillator
37  #define CC1101_SCAL             0x33        // Calibrate frequency synthesizer and
    ↪  turn it off. SCAL can be strobed from IDLE mode without
38                                              // setting manual calibration mode
                                                ↪  (MCSM0.FS_AUTOCAL=0)
39  #define CC1101_SRX              0x34        // Enable RX. Perform calibration first
    ↪  if coming from IDLE and MCSM0.FS_AUTOCAL=1
40  #define CC1101_STX              0x35        // In IDLE state: Enable TX. Perform
    ↪  calibration first if MCSM0.FS_AUTOCAL=1.
41                                              // If in RX state and CCA is enabled:
                                                ↪  Only go to TX if channel is clear
42  #define CC1101_SIDLE            0x36        // Exit RX / TX, turn off frequency
    ↪  synthesizer and exit Wake-On-Radio mode if applicable
43  #define CC1101_SWOR             0x38        // Start automatic RX polling sequence
    ↪  (Wake-on-Radio) as described in Section 19.5 if
44                                              // WORCTRL.RC_PD=0
45  #define CC1101_SPWD             0x39        // Enter power down mode when CSn goes
    ↪  high
46  #define CC1101_SFRX             0x3A        // Flush the RX FIFO buffer. Only issue
    ↪  SFRX in IDLE or RXFIFO_OVERFLOW states
47  #define CC1101_SFTX             0x3B        // Flush the TX FIFO buffer. Only issue
    ↪  SFTX in IDLE or TXFIFO_UNDERFLOW states
48  #define CC1101_SWORRST          0x3C        // Reset real time clock to Event1 value
49  #define CC1101_SNOP             0x3D        // No operation. May be used to get
    ↪  access to the chip status byte
50
51  /**
52   * Status registers
53   */
54  #define CC1101_PARTNUM          0x30        // Chip ID
55  #define CC1101_VERSION          0x31        // Chip ID
56  #define CC1101_FREQEST          0x32        // Frequency Offset Estimate from
    ↪  Demodulator
```

```
57  #define CC1101_LQI                0x33        // Demodulator Estimate for Link Quality
58  #define CC1101_RSSI               0x34        // Received Signal Strength Indication
59  #define CC1101_MARCSTATE          0x35        // Main Radio Control State Machine State
60  #define CC1101_WORTIME1           0x36        // High Byte of WOR Time
61  #define CC1101_WORTIME0           0x37        // Low Byte of WOR Time
62  #define CC1101_PKTSTATUS          0x38        // Current GDOx Status and Packet Status
63  #define CC1101_VCO_VC_DAC         0x39        // Current Setting from PLL Calibration
    ↪  Module
64  #define CC1101_TXBYTES            0x3A        // Underflow and Number of Bytes
65  #define CC1101_RXBYTES            0x3B        // Overflow and Number of Bytes
66  #define CC1101_RCCTRL1_STATUS     0x3C        // Last RC Oscillator Calibration Result
67  #define CC1101_RCCTRL0_STATUS     0x3D        // Last RC Oscillator Calibration Result
68
69  /**
70   *  Configuration Register number defines
71   */
72  #define CC1101_IOCFG2           0x00
73  #define CC1101_IOCFG1           0x01
74  #define CC1101_IOCFG0           0x02
75  #define CC1101_FIFOTHR          0x03
76  #define CC1101_SYNC1            0x04
77  #define CC1101_SYNC0            0x05
78  #define CC1101_PKTLEN           0x06
79  #define CC1101_PKTCTRL1         0x07
80  #define CC1101_PKTCTRL0         0x08
81  #define CC1101_ADDR             0x09
82  #define CC1101_CHANNR           0x0A
83  #define CC1101_FSCTRL1          0x0B
84  #define CC1101_FSCTRL0          0x0C
85  #define CC1101_FREQ2            0x0D
86  #define CC1101_FREQ1            0x0E
87  #define CC1101_FREQ0            0x0F
88  #define CC1101_MDMCFG4          0x10
89  #define CC1101_MDMCFG3          0x11
90  #define CC1101_MDMCFG2          0x12
91  #define CC1101_MDMCFG1          0x13
92  #define CC1101_MDMCFG0          0x14
93  #define CC1101_DEVIATN          0x15
94  #define CC1101_MCSM2            0x16
95  #define CC1101_MCSM1            0x17
96  #define CC1101_MCSM0            0x18
97  #define CC1101_FOCCFG           0x19
98  #define CC1101_BSCFG            0x1A
99  #define CC1101_AGCCTRL2         0x1B
100 #define CC1101_AGCCTRL1         0x1C
101 #define CC1101_AGCCTRL0         0x1D
102 #define CC1101_WOREVT1          0x1E
103 #define CC1101_WOREVT0          0x1F
104 #define CC1101_WORCTRL          0x20
105 #define CC1101_FREND1           0x21
106 #define CC1101_FREND0           0x22
107 #define CC1101_FSCAL3           0x23
108 #define CC1101_FSCAL2           0x24
109 #define CC1101_FSCAL1           0x25
110 #define CC1101_FSCAL0           0x26
111 #define CC1101_RCCTRL1          0x27
112 #define CC1101_RCCTRL0          0x28
113 #define CC1101_FSTEST           0x29
114 #define CC1101_PTEST            0x2A
115 #define CC1101_AGCTEST          0x2B
```

```
116 #define CC1101_TEST2           0x2C
117 #define CC1101_TEST1           0x2D
118 #define CC1101_TEST0           0x2E
119 #define CC1101_PARTNUM         0x30
120 #define CC1101_VERSION         0x31
121 #define CC1101_FREQEST         0x32
122 #define CC1101_LQI             0x33
123 #define CC1101_RSSI            0x34
124 #define CC1101_MARCSTATE       0x35
125 #define CC1101_WORTIME1        0x36
126 #define CC1101_WORTIME0        0x37
127 #define CC1101_PKTSTATUS       0x38
128 #define CC1101_VCO_VC_DAC      0x39
129 #define CC1101_TXBYTES         0x3A
130 #define CC1101_RXBYTES         0x3B
131 #define CC1101_RCCTRL1_STATUS  0x3C
132 #define CC1101_RCCTRL0_STATUS  0x3D
133
134
135
136 #define MPU9250_ADDRESS 0x69 // I2C-адрес датчика позиционирования
137 #define GYRO_FULL_SCALE_250_DPS 0x00 // значение для записи в регистр для установки
    ↪  шкалы измерения датчика позиционирования
138 #define GYRO_FULL_SCALE_500_DPS 0x08
139 #define GYRO_FULL_SCALE_1000_DPS 0x10
140 #define GYRO_FULL_SCALE_2000_DPS 0x18
141
142
143 /* USER CODE END PD */
144
145 /* Private macro -------------------------------------------------------------*/
146 /* USER CODE BEGIN PM */
147 // Select (SPI) CC1101
148 #define cc1101_Select()    HAL_GPIO_WritePin(SPI2_CSN_GPIO_Port, SPI2_CSN_Pin,
    ↪  GPIO_PIN_RESET);
149 // Deselect (SPI) CC1101
150 #define cc1101_Deselect()    HAL_GPIO_WritePin(SPI2_CSN_GPIO_Port, SPI2_CSN_Pin,
    ↪  GPIO_PIN_SET);
151 /* USER CODE END PM */
152
153 /* Private variables ---------------------------------------------------------*/
154 I2C_HandleTypeDef hi2c1;
155
156 SPI_HandleTypeDef hspi2;
157
158 /* USER CODE BEGIN PV */
159
160 /* USER CODE END PV */
161
162 /* Private function prototypes -----------------------------------------------*/
163 void SystemClock_Config(void);
164 static void MX_GPIO_Init(void);
165 static void MX_I2C1_Init(void);
166 static void MX_SPI2_Init(void);
167 /* USER CODE BEGIN PFP */
168
169 /* USER CODE END PFP */
170
171 /* Private user code ---------------------------------------------------------*/
172 /* USER CODE BEGIN 0 */
```

```
173 uint8_t I2C_Read_Buff(uint8_t* buff, int size)
174 {
175         uint8_t sensreg = 0x43;//
176         uint8_t d;
177 
178         while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY);
179         HAL_I2C_Master_Transmit(&hi2c1, MPU9250_ADDRESS << 1 , &sensreg, 1, 100);//
            ↪  установка регистра, с которого будет идти чтение данных
180 
181         while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY);
182         d = HAL_I2C_Master_Receive(&hi2c1, MPU9250_ADDRESS << 1 ,  &buff[0], size,
            ↪  100);// чтение данных в переменную In_Buff (6 байт)
183 
184         return d;
185 }
186 
187 
188 uint8_t I2C_Write_Byte(uint8_t addr_out, uint8_t data_out)
189 // запись в регистр данных для датчика позиционирования
190 {
191 
192 uint8_t buf_out[] = {addr_out, data_out};// массив с регистром и значением
193 uint8_t d;
194 
195 while (HAL_I2C_GetState(&hi2c1) != HAL_I2C_STATE_READY);//
196 
197 d = HAL_I2C_Master_Transmit(&hi2c1, MPU9250_ADDRESS << 1 , &buf_out[0], 2, 100);//
    ↪  запись в указанный регистр указанного значения для датчика позиционирования
198 
199 return d;
200 }
201 
202 typedef struct registerSetting
203 {
204                 uint8_t reg;
205                 uint8_t value;
206 } registerSetting_t;
207 
208 
209 static const registerSetting_t config_MODE1[]=
210  {
211  {CC1101_FSCTRL1,   0x0A},   //     Frequency synthesizer control.
212  {CC1101_FSCTRL0,   0x00},   //     Frequency synthesizer control.
213  {CC1101_FREQ2 ,  0x10},   //       Frequency control word, high byte.
214  {CC1101_FREQ1,   0xA7},   //       Frequency control word, middle byte.
215  {CC1101_FREQ0,   0x62},   //       Frequency control word, low byte.
216  {CC1101_MDMCFG4,     0xC7}, //0010, Modem Configuration
217  {CC1101_MDMCFG3,     0x83}, //0011, Modem Configuration
218  {CC1101_MDMCFG2,     0x05}, //0012, Modem Configuration = 2-FSK
219  {CC1101_MDMCFG1,   0x22},   //     Modem configuration.
220  {CC1101_MDMCFG0 ,  0xF8},   //     Modem configuration.
221  {CC1101_CHANNR,   0x04},   //      Channel number.
222  {CC1101_DEVIATN,   0x34},   //     Modem deviation setting (when FSK modulation is
     ↪  enabled).
223  {CC1101_FREND1,   0xB6},   //      Front end RX configuration.
224  {CC1101_FREND0,   0x10},   //      Front end TX configuration.
225  {CC1101_MCSM0,   0x18},   //       Main Radio Control State Machine configuration.
226  {CC1101_FOCCFG ,  0x1D},   //      Frequency Offset Compensation Configuration.
227  {CC1101_BSCFG,   0x1C},   //       Bit synchronization Configuration.
228  {CC1101_AGCCTRL2 ,  0xC7},   //    AGC control.
```

```
229  {CC1101_AGCCTRL1,   0x00},   //    AGC control.
230  {CC1101_AGCCTRL0,   0xB2},   //    AGC control.
231  {CC1101_FSCAL3 ,  0xE9},   //      Frequency synthesizer calibration.
232  {CC1101_FSCAL2 ,  0x2A},   //      Frequency synthesizer calibration.
233  {CC1101_FSCAL1 ,  0x00},   //      Frequency synthesizer calibration.
234  {CC1101_FSCAL0,   0x1F},   //      Frequency synthesizer calibration.
235  {CC1101_FSTEST,   0x59},   //      Frequency synthesizer calibration.
236  {CC1101_TEST2,   0x81},   //       Various test settings.
237  {CC1101_TEST1,   0x35},   //       Various test settings.
238  {CC1101_TEST0,   0x09},   //       Various test settings.
239  {CC1101_FIFOTHR,   0x07},   //     RXFIFO and TXFIFO thresholds.
240  {CC1101_IOCFG2,   0x0B},   //      GDO2 output pin configuration.
241  {CC1101_IOCFG0,   0x06},   //     GDO0 output pin configuration.
242  {CC1101_PKTCTRL1,   0x04},   //    Packet automation control.
243  {CC1101_PKTCTRL0,   0x05},   //    Packet automation control.
244  {CC1101_ADDR,   0x00},   //        Device address.
245  {CC1101_PKTLEN,   0x00 } ,  //      Packet length.
246  {CC1101_PATABLE,     0x84},
247  {CC1101_SYNC1,       0xD3}, //0004, Sync Word, High Byte  (1101001110010001 = 01 01
     ↪  00 01 00 00 01 01 01 00 00 01 00 00 00 01)
248  {CC1101_SYNC0,       0x91}, //0005, Sync Word, Low Byte
249 };
250
251
252 int CC_init(const registerSetting_t* config_array,int num_registers)
253 {
254     uint8_t data = CC1101_SRES; // Reset CC1101 chip
255     uint8_t tx[2];
256     uint8_t rx[2]; rx[0]='\0';rx[1]='\0';
257
258     HAL_SPI_Transmit(&hspi2, &data, 1, HAL_MAX_DELAY);
259
260         for (int regi=0;regi<num_registers;regi++)
261         {
262             tx[0] = config_array[regi].reg;
263             tx[1] = config_array[regi].value;
264             HAL_SPI_TransmitReceive(&hspi2, tx, rx, 2, HAL_MAX_DELAY);
265         }
266         return 0;
267 }
268 /* USER CODE END 0 */
269
270 /**
271   * @brief  The application entry point.
272   * @retval int
273   */
274 int main(void)
275 {
276   /* USER CODE BEGIN 1 */
277           uint8_t In_Buff[6]; // переменная для чтения данных
278         int16_t gx; // показания угловой скорости по оси X
279         int16_t gy; // показания угловой скорости по оси Y
280         int16_t gz; // показания угловой скорости по оси Z
281         char buff[20];
282         int flag = 0;
283
284         uint8_t tx[2];
285         uint8_t rx[2]; rx[0]='\0';rx[1]='\0';
286
287         int num_registers = sizeof(config_MODE1)/sizeof(registerSetting_t);
```

```
288   /* USER CODE END 1 */
289
290   /* MCU Configuration--------------------------------------------------------*/
291
292   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
293   HAL_Init();
294
295   /* USER CODE BEGIN Init */
296
297   /* USER CODE END Init */
298
299   /* Configure the system clock */
300   SystemClock_Config();
301
302   /* USER CODE BEGIN SysInit */
303
304   /* USER CODE END SysInit */
305
306   /* Initialize all configured peripherals */
307   MX_GPIO_Init();
308   MX_I2C1_Init();
309   MX_SPI2_Init();
310   /* USER CODE BEGIN 2 */
311   I2C_Write_Byte(26, 0x06);
312   I2C_Write_Byte(27, GYRO_FULL_SCALE_1000_DPS);
313
314   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_13, GPIO_PIN_SET); //SPI2_SCK --> to 1
315   HAL_GPIO_WritePin(GPIOB, GPIO_PIN_15, GPIO_PIN_RESET); //SPI2_MOSI--> to 0
316   cc1101_Select();
317   HAL_Delay(10);
318   CC_init(config_MODE1, num_registers);
319   cc1101_Deselect();
320   /* USER CODE END 2 */
321
322   /* Infinite loop */
323   /* USER CODE BEGIN WHILE */
324   while (1)
325   {
326     /* USER CODE END WHILE */
327
328     /* USER CODE BEGIN 3 */
329
330         if (!flag)
331         {
332                 // Устанавливаем режим приема данных
333                 cc1101_Select();
334                 tx[0] = CC1101_SRX;
335                 rx[0]='\0';
336                 HAL_SPI_TransmitReceive(&hspi2, tx, rx, 1, HAL_MAX_DELAY); //single
                    ↪  byte strobe(reg no) out, status byte in
337                 cc1101_Deselect();
338
339                 // Считываем статус регистра
340                 uint8_t size;
341                 tx[0] = (uint8_t) CC1101_RXBYTES | READ_BURST;
342                 tx[1] = (uint8_t) 0;
343                 rx[0]='\0';
344                 rx[1]='\0';
345                 cc1101_Select();
346                 HAL_SPI_TransmitReceive(&hspi2, &tx[0], &rx[0], 1, HAL_MAX_DELAY);
```

```
347                 HAL_SPI_TransmitReceive(&hspi2, &tx[1], &rx[1], 1, HAL_MAX_DELAY);
348                 cc1101_Deselect();
349 
350                 // Выделяем количество принятых байт
351                 size = rx[1] & 0x7F;
352                 if (size > 0)
353                 {
354                         flag = 1;
355                 }
356         }
357 
358         if (flag)
359         {
360                   I2C_Read_Buff(In_Buff, 6);
361                 gx = In_Buff[0] << 8 | In_Buff[1];// запись значения по оси X
362                 gy = In_Buff[2] << 8 | In_Buff[3];// запись значения по оси Y
363                 gz = In_Buff[4] << 8 | In_Buff[5];// запись значения по оси Z
364 
365             HAL_Delay(100);
366 
367 
368                 snprintf(buff, sizeof(buff), "%d, %d, %d", gx, gy, gz);
369 
370             // Set data length at the first position of the TX FIFO
371             tx[0] = (uint8_t) CC1101_TXFIFO;
372             tx[1] = (uint8_t) strlen(buff);
373             cc1101_Select();
374             // отправляем на передатчик данные о длине передаваемого сообщения
375             HAL_SPI_TransmitReceive(&hspi2, tx, rx, 2, HAL_MAX_DELAY);
376             cc1101_Deselect();
377 
378             // Write data into the TX FIFO
379             cc1101_Select();
380             tx[0] = CC1101_TXFIFO | WRITE_BURST;
381             // готовим передатчик к записи данных
382             HAL_SPI_Transmit(&hspi2, tx, 1, HAL_MAX_DELAY);
383             // записываем данные для передачи
384             HAL_SPI_Transmit(&hspi2, (unsigned char*)buff, strlen(buff),
                ↪  HAL_MAX_DELAY);
385             cc1101_Deselect();
386 
387             // CCA enabled: will enter TX state only if the channel is clear
388             cc1101_Select();
389             tx[0] = CC1101_STX; //setTxState();
390             rx[0]='\0';
391             // задаем режим передачи данных для передатчика
392             HAL_SPI_TransmitReceive(&hspi2, tx, rx, 1, HAL_MAX_DELAY);
393             cc1101_Deselect();
394         }
395 
396         HAL_Delay(100);
397 
398   }
399   /* USER CODE END 3 */
400 }
```

Помехоустойчивое кодирование

Ниже приводится пример реализации кода Хэмминга, являющегося одним из простейших помехоустойчивых кодов. Выбор метода кодирования остается на усмотрение участников. Непосредственно на финале команда-победитель использовала более эффективный код Рида-Соломона.

```
byte * hamming_encoder(char input_symbol) {
  bool symbol_bin_array[8], data[12];
  static byte encoded_data[2];
  int N;
  for (int i = 0, j = 7; i <= 7; i++, j--)
    symbol_bin_array[j] = input_symbol & (1 << i);
  for (int i = 0, j = 0; i <= 11; i++) {
     N = i + 1;
    if ((int) log2(N) == log2(N)) {
      data[i] = 0;
    } else {
      data[i] = symbol_bin_array[j];
      j++;
    }
  }
   for (int i = 0, sum; i <= 11; i++) {
    N = i + 1;
      if ((int) log2(N) == log2(N)) {
      sum = 0;
      for (int j = N - 1; j <= 11; j += N) {
        for (int k = 0; k < N; k++) {
          if (j > 11) {
            break;
          }
          sum = sum + data[j];
          j++;
        }
      }
      if (sum % 2 != 0) {
        data[i] = 1;
      }
    }
  }
  encoded_data[0] = 0;
   for (int i = 0; i <= 3; i++) {
    encoded_data[0] = encoded_data[0] | (data[i] << (3 - i));
  }
  for (int i = 4; i <= 11; i++) {
    encoded_data[1] = encoded_data[1] | (data[i] << (11 - i));
  }
  return encoded_data;
}
```

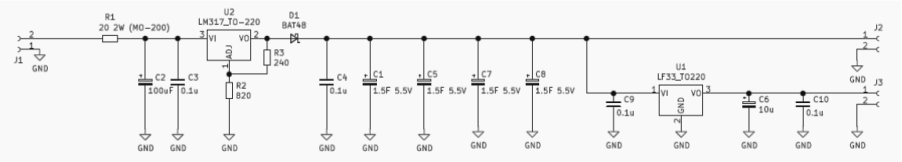
Пример программы для декодировщика:

```
#include <math.h>

int hamming_decoder(char bits_to_encode){

    bool symbol_bin_array[8];
    for (int i = 0, j = 11; i <= 11; i++, j--)
```

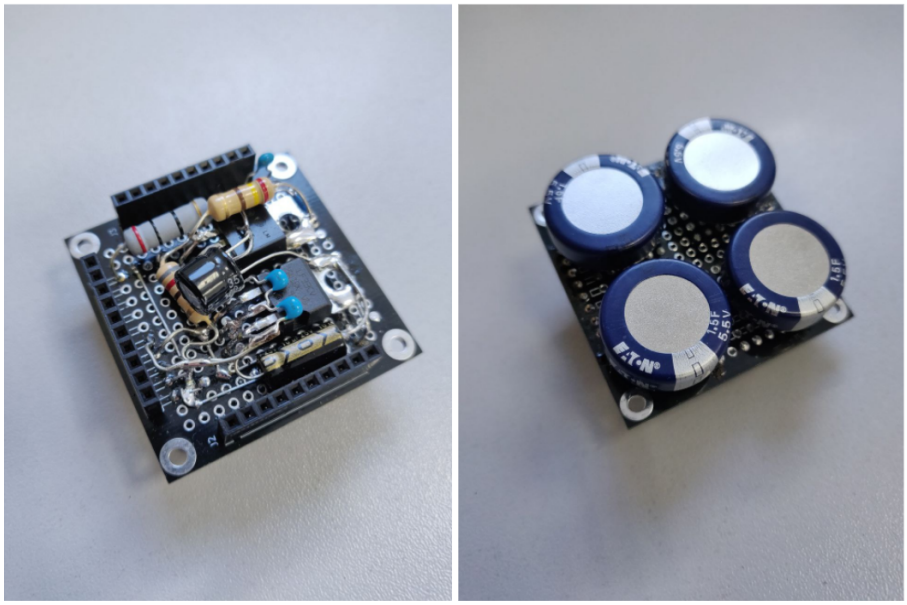
```
7           symbol_bin_array[j] = bits_to_encode & (1 << i);
8
9       int sum = 0;
10      int recieved_control_bits[4];
11      int calculated_control_bits[4];
12      int correct_symbol[8];
13      bool incorrect_message = false;
14      int wrong_bit_index = 0;
15
16      int k = 0;
17      for (int i = 0, sum; i <= 11; i++) {
18          if ((int) log2(i + 1) == log2(i + 1)) {
19              recieved_control_bits[k] = symbol_bin_array[i];
20              symbol_bin_array[i] = 0;
21              k += 1;
22          }
23      }
24
25    for (int i = 0, sum; i <= 11; i++) {
26      if ((int) log2(i + 1) == log2(i + 1)) {
27        sum = 0;
28        for (int j = i; j <= 11; j += i + 1) {
29          for (int k = 0; k < i + 1; k++) {
30            if (j > 11) {
31              break;
32            }
33            sum = sum + symbol_bin_array[j];
34            j++;
35          }
36        }
37          calculated_control_bits[i] = sum % 2;
38      }
39    }
40      for (int i = 0, sum; i <= 3; i++) {
41          if (recieved_control_bits[i] != calculated_control_bits[i]){
42              incorrect_message = true;
43              wrong_bit_index = wrong_bit_index + pow(2, i);
44          }
45      }
46      wrong_bit_index = wrong_bit_index - 1;
47      if (incorrect_message){
48          symbol_bin_array[wrong_bit_index] = 1 - symbol_bin_array[wrong_bit_index];
49      }
50      for (int i = 0, sum; i <= 11; i++) {
51          if ((int) log2(i + 1) != log2(i + 1)) {
52              correct_symbol[i] = symbol_bin_array[i];
53          }
54      }
55      for (int i = 0, sum; i <= 8; i++) {
56          if ((int) log2(i + 1) != log2(i + 1)) {
57              correct_symbol[1] = correct_symbol[1] | (symbol_bin_array[i] << (8 - i));
58          }
59      }
60      return correct_symbol[1];
61  }
```

Схемотехника

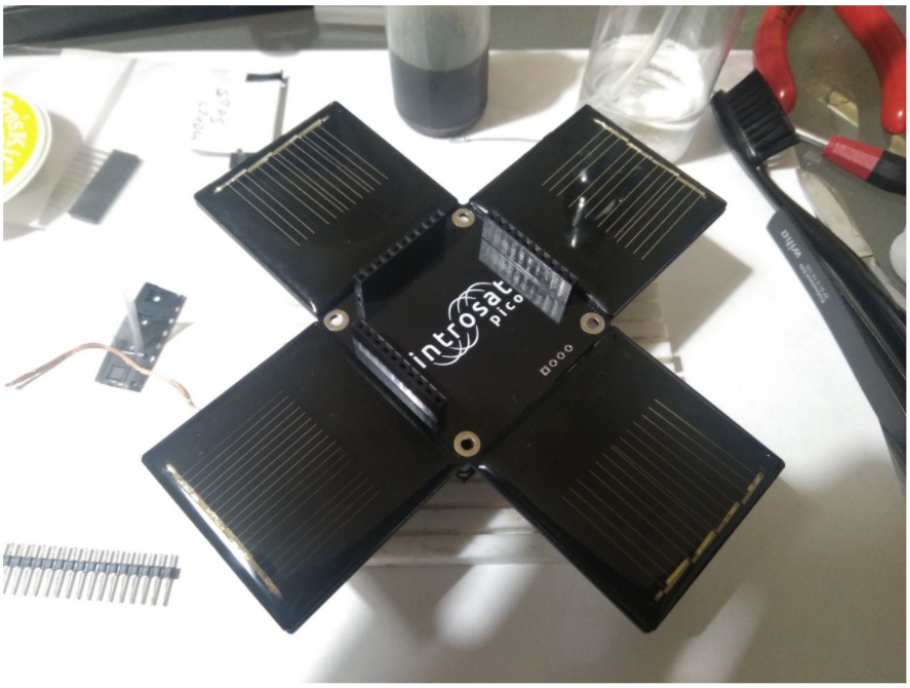
Принципиальная схема платы питания пикоспутника:



Плата питания пикоспутника в сборке:



Пикоспутник в сборе:



Орбитальная механика

Орбитальная механика

Первая задача

Условие:

Аппарат вышел на орбиту, однако по расчетному времени пролетов над ЦУП не выходит на связь. Через некоторое время удалось поймать сигнал от аппарата, однако не в то время и не в том месте, что может свидетельствовать о том, что аппарат находится не на той орбите.

Необходимо уточнить параметры текущей орбиты аппарата. Известно, что:

- проход над Сан-Франциско (37.7749; 237.581) был в момент времени с 22:07 по 22:12;
- проход над Портсмутом (50.799; 358.909) был в момент времени с 13:54 по 13:59;
- проходы над Йоханненсбургом (-26.2023; 28.0436) были в моменты времени с 12:39 по 12:44 и с 00:41 по 00:43.

Параметры планируемой орбиты миссии:

- большая полуось: 6870 км
- наклонение: 80°
- эксцентриситет: 0
- аргумент перицентра: 0
- долгота восходящего узла: 236°
- истинная аномалия: 45°

Время вывода аппарата на орбиту: 04 апреля 2021 года 12:00 по UTC. Время моделирования — 24 часа.

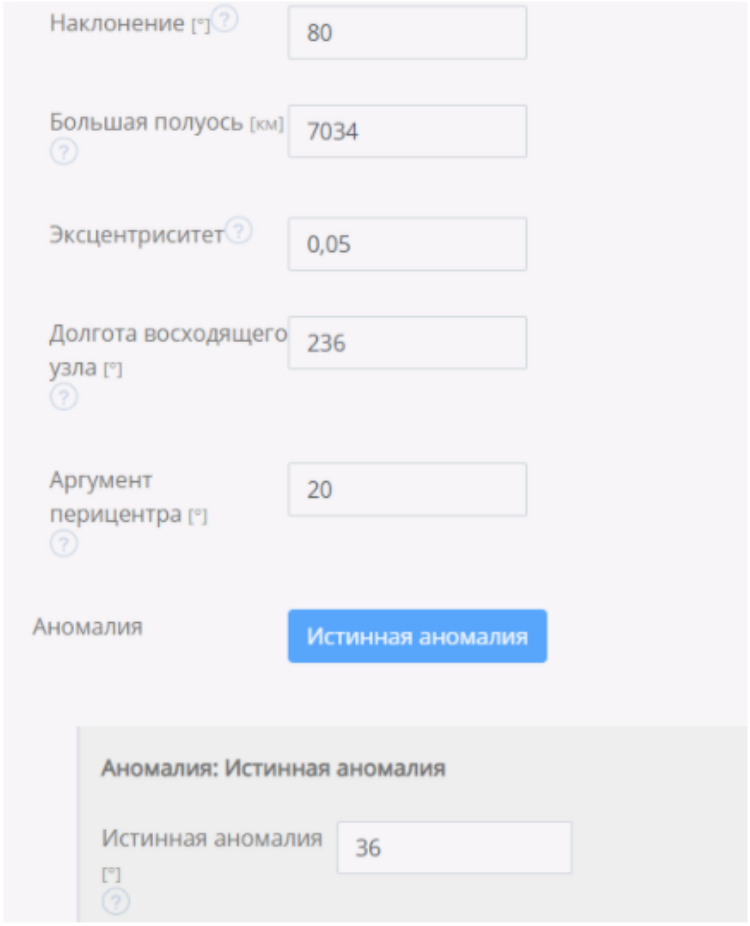
Известно, что наклонение и долгота восходящего узла остались неизменными.

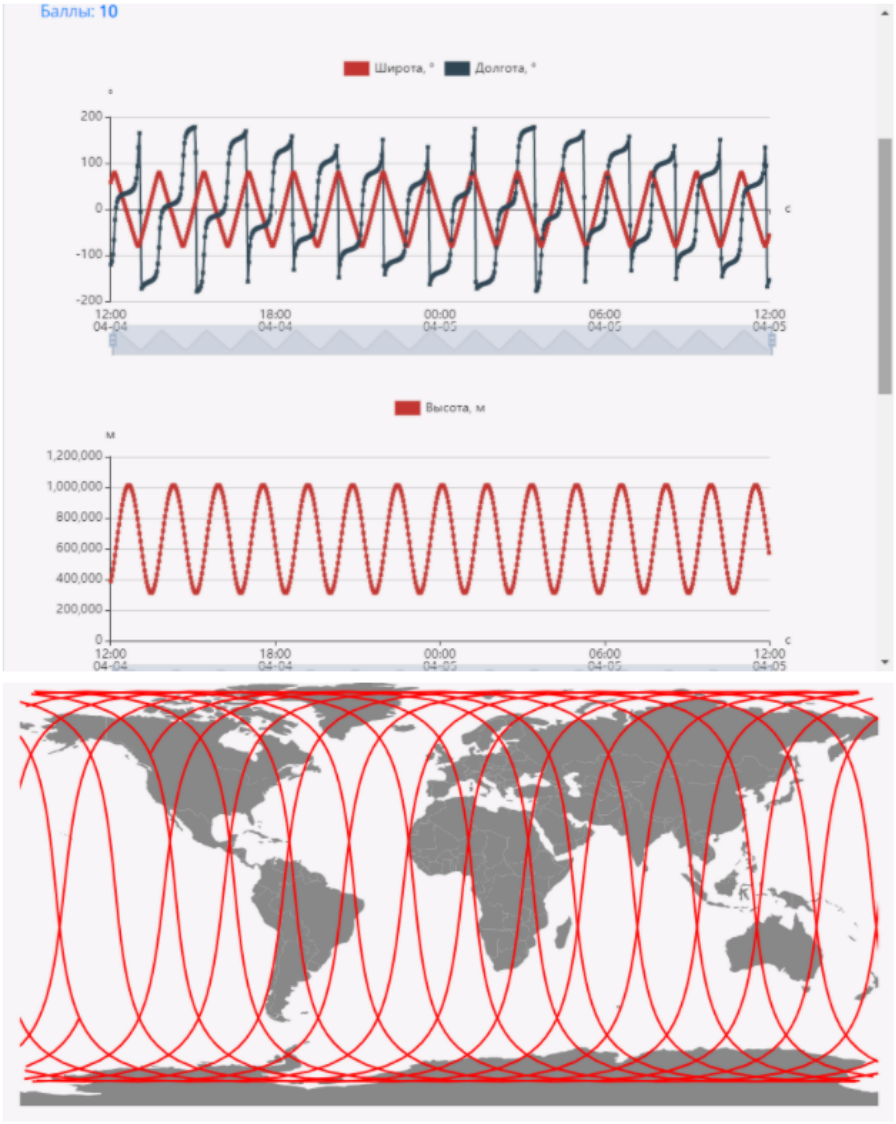
Радиус Земли, м: 6371008.8

Угол над горизонтом, при котором возможен прием данных: 30°

За каждый проход над станцией в установленный промежуток времени начисляется 2.5 балла. Максимальный балл за задачу — 10 баллов. Начиная с 5-ой попытки начинается дисконт за каждую попытку, равный 10% максимального балла. Балл уменьшается вплоть до 1 балла, и далее не уменьшается с числом попыток.

Решение:





Вторая задача

Условие:

Аппарат был выведен на полярную орбиту высотой 420 км. После вывода на орбиту аппарат остался ориентирован в плоскости своей орбиты; после завершения испытаний маховики аппарата были отключены, так что аппарат может быть подвержен вращению в этой плоскости.

Перед выводом пикоспутников необходимо перевести его на орбиту 320 км (ниже большинства орбит других космических аппаратов, и для сокращения срока пребывания пикоспутников на орбите), задав программу управления аппаратом. На аппарате находится двигатель с удельным импульсом 1079 м/с и максимальной тягой 0.98 Н. Сухая масса аппарата — 10 кг. Количество топлива — 0.572 кг.

Симуляция идет 7 часов. Баллы за нахождение на нужной высоте начинают начисляться через 2 часа после начала симуляции.

Начальная ориентация аппарата — по ходу движения, над экватором. Двигатель направлен при этом против движения.

Для решения задачи рекомендуется воспользоваться решением задачи ориентации прошлого года:

https://drive.google.com/file/d/1d0QSN4r0BFuCyu8VUyoRolTturQMdAcr/view?usp=sharing

Радиус Земли, м: 6371008.8

Гравитационный параметр, $\mu = G \cdot M$, м^3/с^2: $3.986004418 \cdot 10^{14}$

API для аппарата:

`https://docs.google.com/document/d/1qLeV2Prk4sPQuiS1D1GjCCjJSS43NVlX6_dKb-vRTmg/edit?usp=sharing`

За каждую 0.01 сек начисляется 0.0008333 балла, если аппарат находится на требуемой высоте с погрешностью 5 км или 0.0001666 балла, если с погрешностью 10 км.

Максимальный балл за задачу — 15 баллов. Начиная с 5-ой попытки начинается дисконт за каждую попытку, равный 10% максимального балла. Балл уменьшается вплоть до 1.5 балла, и далее не уменьшается с числом попыток.

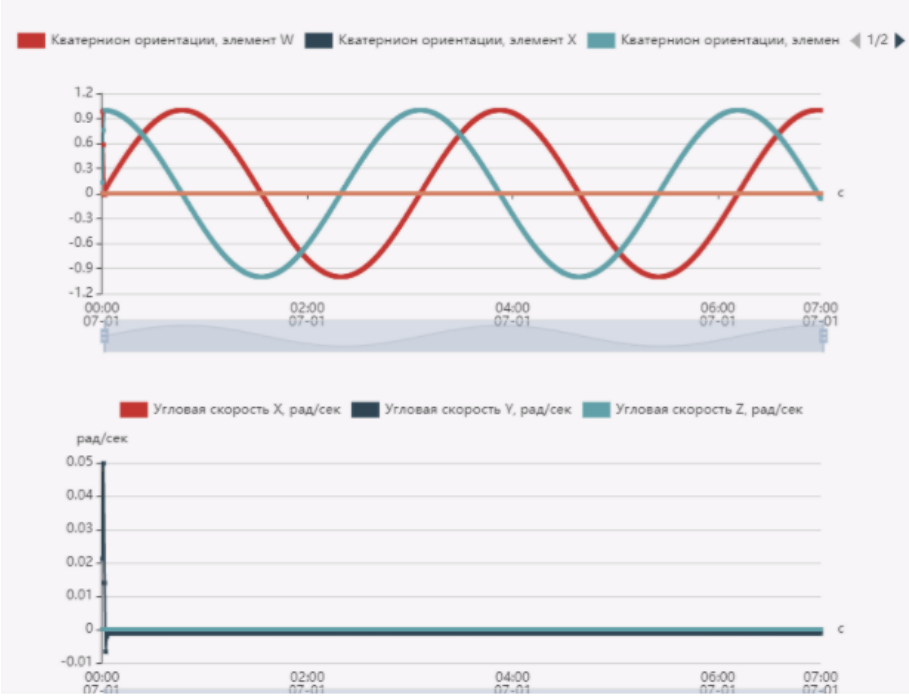
С 1 апреля максимальный балл сокращается вдвое.

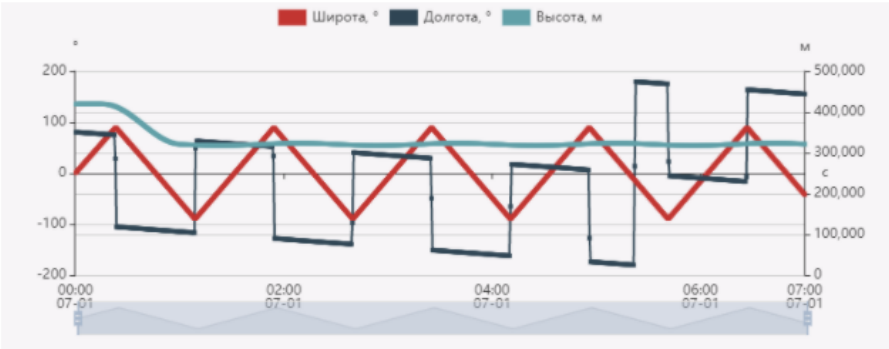
Решение:

Пример кода управления аппаратом:

```
'use strict';

var wheels;
var gyros;
let thruster;
function setup() {
    wheels = spacecraft.devices[0+1];
    gyros = spacecraft.devices[1+1];
    thruster = spacecraft.devices[0].functions[0];

    runtime.debug(String(spacecraft.devices[3+1].functions[0].orientation));
}
var old_a = 0;
var angle = 180;
var t1 = 289;
var t2 = 290;
var treshtime = 1*60*60 +1*60 -t2/2;

function loop() {
    wheels.functions[1].motor_torque = 0;
    if (spacecraft.flight_time < 400) {
        var a = (orientation_angle() + 360)%360;
        var m = angle - spacecraft.devices[2+1].functions[0].location[0];
        var g = Math.abs(gyros.functions[1].angular_velocity);
        var av = wheels.functions[1].angular_velocity;
        var s = 0.0001*(a-m) - av*0.00015;
        wheels.functions[1].motor_torque = s;
        old_a = a;
    }

    if (spacecraft.flight_time > 800 && spacecraft.flight_time < 800+t1) {
        thruster.thrust = thruster.maximum_thrust;
    }
```

```
    else if (spacecraft.flight_time > treshtime && spacecraft.flight_time <
    ↪  treshtime+t2) {
        thruster.thrust = thruster.maximum_thrust;
    } else {
        thruster.thrust = 0;
    }
}

function orientation_angle() {
    let star_tracker = spacecraft.devices[3+1].functions[0];
    let orientation = star_tracker.orientation;

    let qx = orientation[0];
    let qy = orientation[1];
    let qz = orientation[2];
    let qw = orientation[3];

    let x = 2 * qw * qy + 2 * qx * qz;
    let z = qw * qw - qx * qx - qy * qy + qz * qz;

    return Math.atan2(x, z) * 180.0 / Math.PI;
}
```





Третья задача

Условие:

Аппарат вышел на орбиту, однако по расчетному времени пролетов над ЦУП не выходит на связь. Через некоторое время удалось поймать сигнал от аппарата, однако не в то время и не в том месте, что может свидетельствовать о том, что аппарат находится не на той орбите. Необходимо уточнить параметры текущей орбиты аппарата и произвести корректировку высоты и эксцентриситета орбиты до желаемых высоты и эксцентриситета. Известно, что:

- проход над Сан-Франциско (37.7749; 237.581) был в момент времени с 16:52 по 16:56;
- проходы над Портсмутом (50.799; 358.909) были в моменты времени с 21:25 по 21:27 и с 09:13 по 09:16;
- проходы над Йоханненсбургом (-26.2023; 28.0436) были в моменты времени с 19:26 по 19:30 и с 06:18 по 06:20.

Время вывода аппарата на орбиту: 06 апреля 2021 года 12:00 по UTC. Время моделирования — 29 часа. После вывода на орбиту аппарат остался ориентирован в плоскости своей орбиты.

С 13:00 7 апреля надо быть на скорректированной орбите. До этого аппарат будет двигаться по введенной орбите, при прохождении над станциями в указанный промежуток времени начисляется балл.

На аппарате находится двигатель с удельным импульсом 1079 м/с и максимальной тягой 0.98 Н. Сухая масса аппарата — 10 кг. Количество топлива — 2 кг. Минимальная тяга двигателя — 0.0098 Н.

Параметры планируемой орбиты:

- большая полуось: 6840 км
- наклонение: 90°
- эксцентриситет: 0

Наклонение осталось неизменным.

Для решения задачи рекомендуется воспользоваться решением задачи ориентации прошлого года:

https://drive.google.com/file/d/1CfsIpF-1HHEsRLwjwaPo7K4a6o9GOpBd/view?usp=sharing

Радиус Земли, м: 6371008.8

Гравитационный параметр, $\mu = G \cdot M$, м^3/с^2: $3.986004418 \cdot 10^{14}$

API для аппарата актуально из предыдущей задачи: https://docs.google.com/document/d/1qLeV2Prk4sPQuiS1D1GjCCjJSS43NVlX6_dKb-vRTmg/edit?usp=sharing

Угол над горизонтом, при котором возможен прием данных: 30°

За каждый проход над станцией в установленный промежуток времени начисляется 2 балла.

За каждую 0.01 сек начисляется 0.0000104166 балла, если аппарат находится на требуемой высоте с погрешностью 10 км или 0.0000052083 балла, если с погрешностью 15 км.

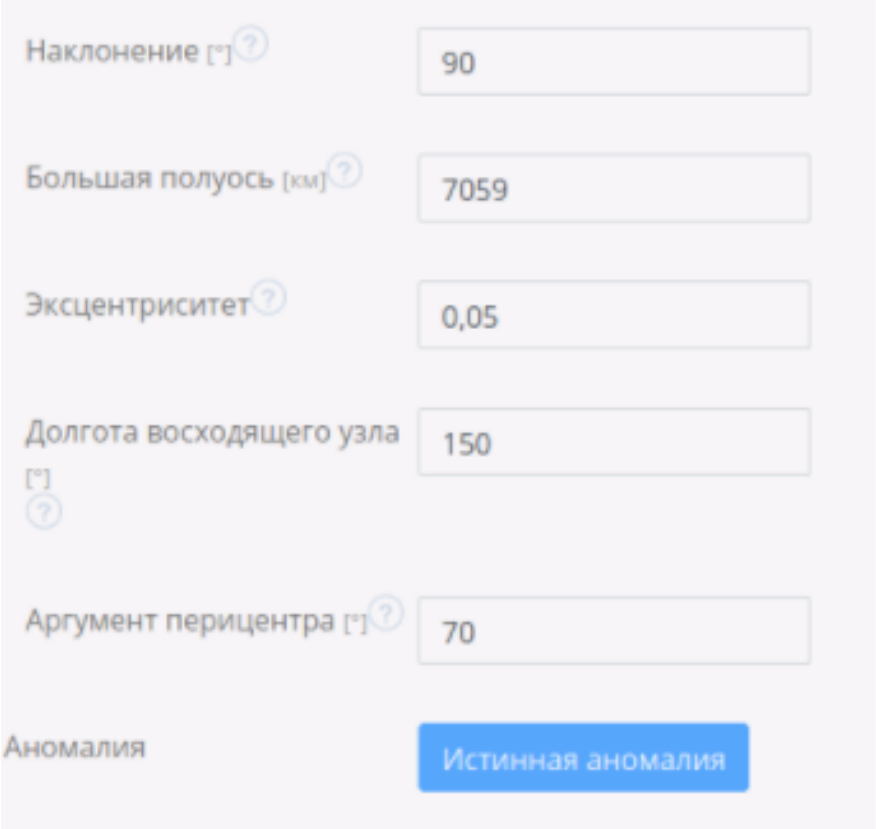
Максимальный балл за задачу — 25 баллов. Начиная с 5-ой попытки начинается дисконт за каждую попытку, равный 10% максимального балла. Балл уменьшается вплоть до 2.5 балла, и далее не уменьшается с числом попыток.

Решение:

В этой задаче есть два компонента решения — подбор орбиты и программа управления. Для расчета орбиты, а также моментов времени и коэффициентов для программы управления на практике целесообразно использовать последовательную оптимизацию с применением численного моделирования миссии (орбиты — в ПО Орбита.Челлендж или GMAT, программы — в ПО Орбита.Челлендж).

Ниже приведен вариант решения, умеренно оптимизирующий оба компонента задачи:

Орбита:



Программа управления:

```
'use strict';

var wheels;
var gyros;
let thruster;
function setup() {
        wheels = spacecraft.devices[0+1];
        gyros = spacecraft.devices[1+1];
        thruster = spacecraft.devices[0].functions[0];

        runtime.debug(String(spacecraft.devices[3+1].functions[0].orientation));
}
```

```
13 var old_a = 0;
14 var angle = 180;
15 var t1 = 289;
16 var t2 = 290;
17 var treshtime = 1*60*60 +1*60 -t2/2;
18
19 function loop() {
20         wheels.functions[1].motor_torque = 0;
21         if (spacecraft.flight_time < 400) {
22             var a = (orientation_angle() + 360)%360;
23             var m = angle - spacecraft.devices[2+1].functions[0].location[0];
24             var g = Math.abs(gyros.functions[1].angular_velocity);
25             var av = wheels.functions[1].angular_velocity;
26             var s = 0.0001*(a-m) - av*0.00015;
27             // var s = av*0.00012;
28             wheels.functions[1].motor_torque = s;
29             old_a = a;
30         }
31
32         if (spacecraft.flight_time > 800 && spacecraft.flight_time < 800+t1) {
33             thruster.thrust = thruster.maximum_thrust;
34         }
35         else if (spacecraft.flight_time > treshtime && spacecraft.flight_time <
           ↪  treshtime+t2) {
36             thruster.thrust = thruster.maximum_thrust;
37         } else {
38             thruster.thrust = 0;
39         }
40
41
42 }
43
44 function orientation_angle() {
45         let star_tracker = spacecraft.devices[3+1].functions[0];
46         let orientation = star_tracker.orientation;
47
48         let qx = orientation[0];
49         let qy = orientation[1];
50         let qz = orientation[2];
51         let qw = orientation[3];
52
53         let x = 2 * qw * qy + 2 * qx * qz;
54         let z = qw * qw - qx * qx - qy * qy + qz * qz;
55
56         return Math.atan2(x, z) * 180.0 / Math.PI;
57 }
```