Дипломна работа

Катедра Системи и Управление

на тема:

Платформа за безпилотен летателен апарат с четири ротора

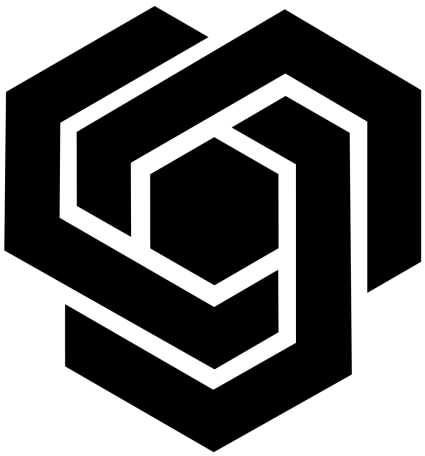
Автор:
Рафаел Калъчев,
IV курс, № 011217071

Ръководител:
гл. ас. д-р Александър Хотмар

Ръководител на кат. СУ:
доц. д-р Теофана Пулева

Технически Университет
София

Факултет Автоматика
АИУТ



Април, 2021

София

Съдържание

Списък на фигурите

Списък на таблиците

1 Въведение

rework intro

Този труд се концентрира върху цялостното изграждане на система за управление като предоставя за пример изграждане на "Безпилотна платформа за летателен апарат с четири ротора". Основните неща на които ще бъде наблеганто е изграшдането на софтуера от основи за безилотният летателен апарат. По този начин ще бъде демонстрирано как може да се изгради основа за софтуер за управление на непознат,иновативен контролер, за който не съществуват библиотеки. Целта е да се елиминира интеграцията с матлаб и да се постигне по-добро разбиране на софтуера на системата за управление и това как той работи и работата само с специфични много популярни модули, за които има безброй библиотеки. Ще бъде разгледан начин за инициализиране и управление на периферията, както методи за обработка на данните поступващи от периферията за сформиране на управляващи въздействия.

Избраната система е многомерна и има състояния, които не могат да бъдат измервани директно. Този труд ще демонстрира изграждането на наблюдателя на състоянията на системата, както и неговата имплементация като част от алгоритъма за управление.

Ще разгледаме и т.нар композитна архитектура за да си позволим лесна промяна и конфигурация на софтуера за управление.

Този труд няма да разглежда изграждането на система за управление с помощта на Операционна система за реално време. Изграждането на ОС за реално време ще бъде плот на отделен бъдещ труд. Работата в реално време ще бъде осигурена от софтуера на конторлера, но тя няма да бъде разпреселена на отделни "задачи" а ще се управлява от регулярните прекъсвания на раймера съпътствани от проста логика и функции имплементирани по начин, който ще гарантира изпълнение за определеното време.

Ще разгледаме отделните инженерни решения взети по време на работата по проекта и причините довели до тях.

2 Използвана среда и инструменти за разработка на платформата

2.1 Среда за разработка на софтуера

Изградената среда за разработка на софтуера е конфигурируема и поддържа базата микроконтролери от семейство *STM32M4xxx*. Като основа е използвана автоматичната система за изгращане *GNU make*, която позволява насочена обработка на файловете, изграждащи софтуера и документацията, с цел намаляване времето за обработка. *GNU make* свързва всички елементи от средата и последователно изпълнява само нужните команди с цел намаляване на използваните ресурси. Интерфейсът е команден, което позволява допълнителни нива на автоматизиране на процесите по разработка.

За компилация е използван свободният комилатор на *GNU GCC (GNU Comiler Collection) ARM NON-EABI (No Embedded-Application Binary Interface)*. Тази разновидност на компилатора е неспецифична към целева операционна система, което е нужно, тъй като разработваният софтуер няма да работи под операционна система. Тази разновидност на комилатора също е неспецифична към производителя на процесора. *GCC ARM NON-EABI* е колекция от свързани инстументи, за разработка на софтуер за системи с ядро *ARM*. Състои се от комилатор на езика C, асемблатор, линкер, инстументи за преглед и конверсия между стандартни формати двоични файлове.

За връзка с контролера е изполван командният пакет за *ST-LINK* на *STMicroelectronics*. Пакетът предоставя команди за връзка с програматора *ST-LINK V2*, чрез който се програмира микроконтролерът. Пакетът се използва също за управление на порта за дебъг, през който се осъществява дебъг комуникацията.

Като дебъгер е използван свободният дебъгер на *GNU GDB*. Той може да се използва както за локално така и за отдалечено дебъгване. Тъй като микроконтролерът не е локален, използваме командния пакет за *ST-LINK* да конфигурираме порт за дебъг, към който се свързваме чрез *GDB*.

За писането на софтуера е изполван терминалният текстов редактор *VIM* поради удобния си команден интерфейс. За генериране на всички софтуерни тагове, които *VIM* ще използва за подпомагане на процеса за разработка, е използван софтуерът *ctags*, който обработва релевантните файлове и поддържа опростена база данни за всички индентификатори, имена и тагове, които *VIM* изпозва спрямо контекста.

Основата за средата за разработка съм публикувал в публично хранилище на GitHub [7].

2.2 Инструменти

За хардуерната част са използвани стандартни инструменти, като отвертки, шестограми, винтоверт, нивелир, макетен нож, рулетка, шублер, трион, шкурка, поялник, пистолет за топъл силикон и т.н.

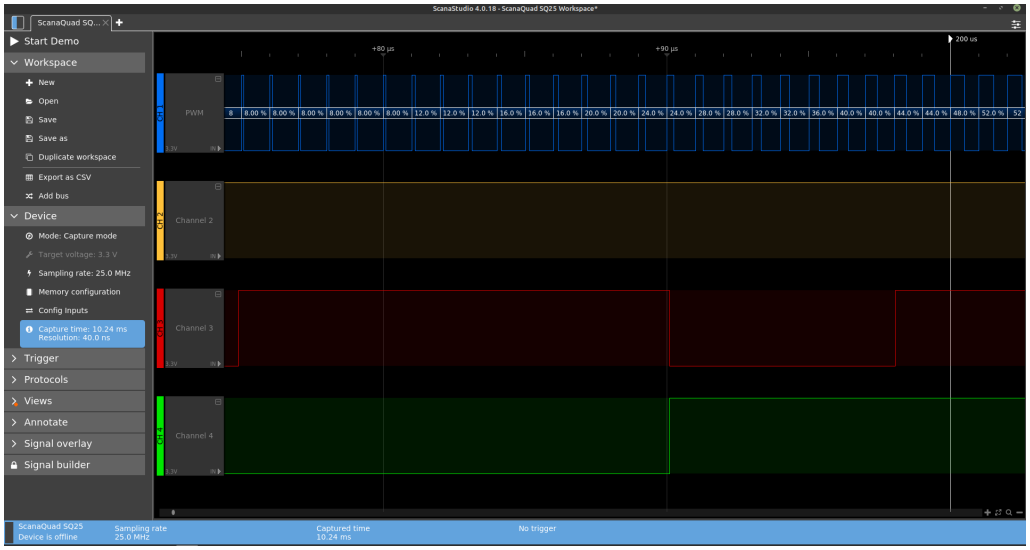
2.2.1 Логически анализатор и генератор на (дигитални) сигнали

Тъй-като този труд има за цел управление и работа с хардуер и създаване на хардуерни драйвери, е нужен начин за анализ и диагностика на крайната комуникация. Затова е избран модулът *SQ50 - logic analyzer* от семейство *SQ series (ScanaQuad)* на *IKALOGIC S.A.S.* (Фигура 1). Модулът разполага с 4 канала с възможност за използване за вход и/или изход, максимална честота на дискретизация $50MHz$, максимална честота на измерим/генерируем дигитален сигнал $12MHz$, входно напрежение $\pm 5V$, входен импеданс $1M\Omega/4pF$, конфигурируем изходен драйвър (Push-Pull/open-drain), и максимален изходен ток $20mA$ [6].

Фигура 1: Логически анализатор SQ50

SQ50 - logic analyzer е използван със софтуерният продукт *ScanaStudio* (Фигура 2) на *IKALOGIC S.A.S.*. *ScanaStudio* е достъпен за платформите *Windows 7, 8 и 10, LINUX UBUNTU 16.04 и по-нови (x64), MACOS 10.9 и по-нови.* Софтуерният продукт позволява възможности за конфигурация на хардуера от семейство *SQ series (ScanaQuad)*, както и четене, експорт и представяне на данните, постъпили от модула. Към продукта *IKALOGIC S.A.S.* предоставят набор от готови разширения, които са свободно достъпни и позволяват: параметрично генериране на основни типове сигнали (ШИМ, честотна модулация, I2C), декодиране

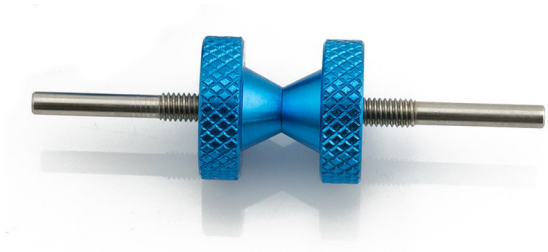
за сигнали (USART, I2C, I2S, SPI, PWM, SWD, 1-Wire и т.н.), конфигурируем източник на задействане (trigger) при определени условия – получена специфична последователност, логическа промяна, специфичен/случаен валиден кадър от комуникационен протокол и т.н.



Фигура 2: Прозорец Scana Studio

2.2.2 Балансьор за витла

Използван е балансьор за витла (Фигура 3) с цел балансиране на витлата. Баланьорът сам по себе си е тънка прецизно изработена права ос с тънка резба, в комбинация с две прецизно изработени конични гайки с успоредна на оста задна част, за да фиксират витлото успоредно спрямо оста, както и да центрират средата на отвора.



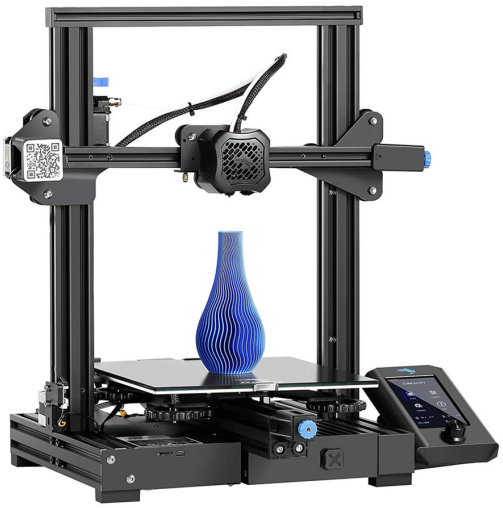
Фигура 3: Балансьор за витла

2.2.3 3D Принтер

За изработване на платформата е използван 3D принтера *Ender 3 V2* (Фигура 4) на *Creality*. Неговата технология на работа е FFF (Fused Fillament Fabrication – производство чрез стопени нишки), която е форма на адитивно производство. Принтерът е оборудван с $0.4mm$

дюза, легло със загряване и стъклена повърхност и има работен обем $220x220x250mm$, и диаметър на нишката $1.75mm$ [9].

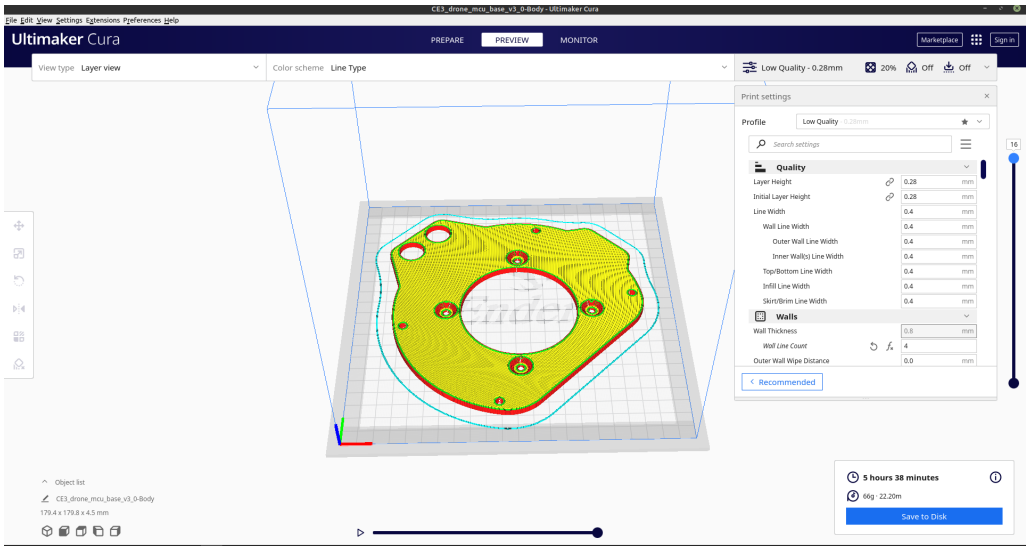
За целите на този труд е използвана нишка от ECO PLA (рециклирана полимлечна киселина), произведена от *3D Jake* с диаметър $1.75mm$, температура на принтиране $195 \rightarrow 215°C$, максимална работна температура $60°C$, якост на опън $70MPa$) и деформация при опън 5% [1].



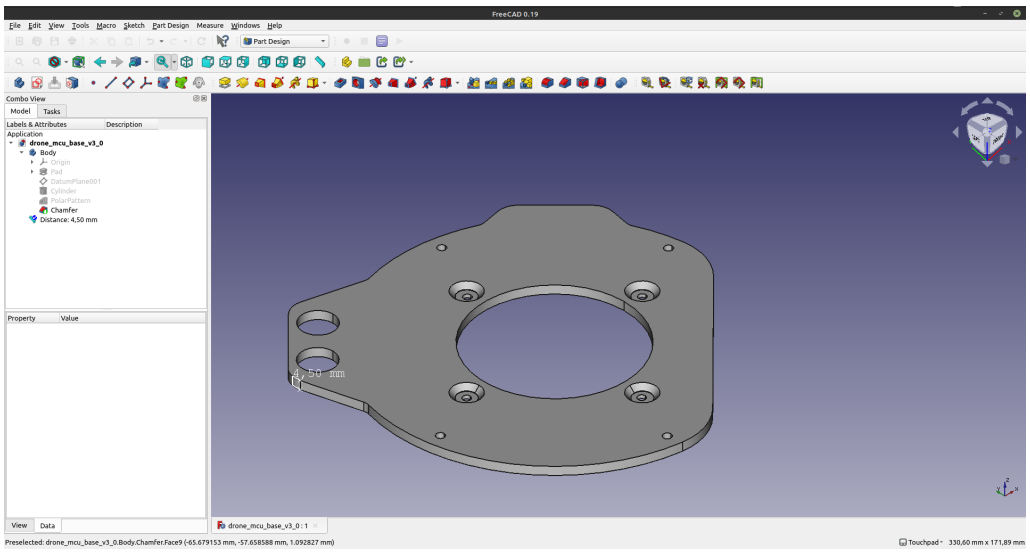
Фигура 4: 3D принтер Creality Ender 3 V2

За генериране на G-code, който ще бъде изпълнен от принтера, е използван софтуерният продукт *Ultimaker Cura* (Фигура 5) на *Ultimaker*. Продуктът е конфигуриран ръчно за устройството *Ender 3 V2*, тъй като не бе налично в базата данни с готови конфигурации. Софтуерът позволява „нарязване“ на триизмерни обекти на слоеве и конвертирането им в G-code програма за изпълнение от устройството, както и конфигуриране на множество параметри относно процеса на принтиране.

За създаване на 3D модел е използван софтуерният продукт *FreeCad* Фигура 6, тъй като работи под *Linux* платформа и е добре документиран, свободен и безплатен софтуер.



Фигура 5: Прозорец на Ultimaker Cura



Фигура 6: Прозорец на FreeCad

3 Използван хардуер

3.1 Микроконтролер

Микроконторлерът, използван за проекта, е с ядро с архитектура *ARM Cortex-M4*, произведен от *STMicroelectronics*. Модел stm32f429ZIT6U.

3.1.1 Характеристики

В следния списък са поместени основните характеристики на микроконтролера [16].

- Ядро: *32b Arm Cortex-M4* с FPU
- Максимална честота на процесора: 180MHz
- Флаш памет: 2048 Kbytes
- SRAM: Системна : 256 (112 + 16 + 64 + 64) Kbytes
- Таймери:
  - General Purpose: 10бр. ()
  - Advanced control: 2бр. ()
  - Basic: 2бр. ()

Add timers

Add timers

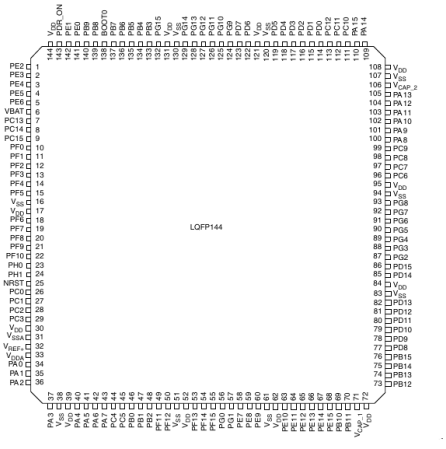
Add timers

- Комуникационни интерфейси:
  - SPI/I2S : 6/2 (пълен дуплекс)
  - I2C: 3
  - USART/UART: 4/4
- GPIO: 114бр.
- Интерфейс за програмиране: *ST-LINK*
- Опаковка: LQFP144 (Фигура 8)

3.1.2 Комплект 32F429IDISCOVERY

Микроконтролерът е част от платка *32F429IDISCOVERY* (Фигура 7) (комплект за оценка на функционалиностите на *STMicroelectronics*). Комплектът предоставя множество

Фигура 7: Комплект 32F429IDISCOVERY



Фигура 8: Наименования на изведените пинове през използвания пакет

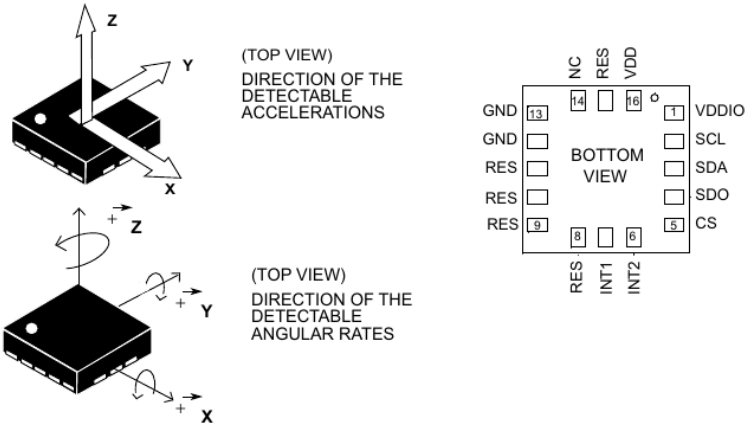
функционалности, вградени в платката като 8MHz външен осцилатор, вграден *ST-LINK* програматор, LCD дисплей, 2 броя LED, както и удобна връзка към пиновете на микроконтролера в THT формат.

Важно е да се отбележи, че в комплекта *32F429IDISCOVERY* има познат проблем с вградения *ST-LINK* програматор. Докато устройството не е свързано чрез USB, програматорът поддържа процесора в *RESET* през дебъгера. Проблемът се решава, чрез ъпдейт на фърмуера на дебъгера или декуплиране чрез физическо разединяване на каналите на дебъгера върху платката (CN4).

3.2 Жироскоп и акселерометър

Изпозван е чип *LSM6DS33* (Фигура 9) [14], който е интегрирано пакетно решение, предоставящо 3D дигитален Жироскоп и 3D дигитален акселерометър. Чипът използва $1.7 \rightarrow 3.6V$ захранване като в нужния ни решим консумира $0.9mA$. Чипът е част от сензорна платка, позволяваща директна връзка чрез протокол *I2C* (адрес: 0xD6).

За целите на настоящия проект е използвана следната конфигурация: директен достъп (без буфериране) до измерените стойности; обхват на акселерометъра $\pm 8g$; обхват на жироскопа $\pm 1000dps$.



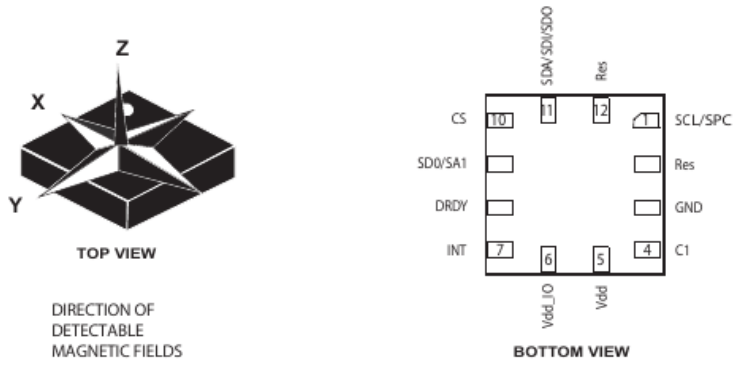
Фигура 9: *LSM6DS33* Ориентация на осите и наименованията на пиновете в пакета

3.3 Магнитометър

Изпозван е чип *LIS3MDL* (Фигура 10) [12], който е интегрирано пакетно решение, предоставящо магнитометър по 3 оси. Чипът използва $1.9 \rightarrow 3.6V$ захранване като в нужния

решим консумира не повече от $270\mu A$. Чипът е част от сензорна платка, позволяваща директна връзка чрез протокол *I2C* (адрес: 0x3C).

За целите на настоящия проект е използвана следната конфигурация: директен достъп (без буфериране) до измерените стойности; обхват на всички оси $\pm 8gaus$.



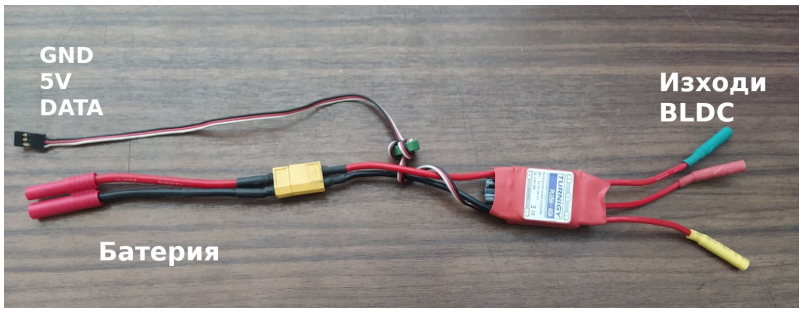
Фигура 10: Магнитометър – оси на измерване и наименования на пиновете

3.4 Електроненн контролер на скоростта за безчеткови постояннотокови мотори (ESC)

Използван е контролер на скоростта за безчеткови постояннотокови мотори *Plush 40amp Speed Controller* (Фигура 11) на *TURNIGY*. Контролерът може да черпи максимум $40A$ ток за продължително време, както и максимален моментен ток $55A$ ($> 10s$) [18].

Използвана е настройка за Li-Po батерия, спирачка-включена, опция за намаляване на мощността, средна граница на намаляване на мощността, нормално стартиране, средно време за реакция. Посоката на въртене не е софтуерно настройваема. За смяна на посоката на въртене се разменят които и да е два от изходите към безчетковия мотор. Това променя последователността на сигналите към отделните намотки и посоката на въртене се обръща.

Опциите за насторйка и процедурата са описани в [18]. Важно е да се отбележи, че моторите се инициализират след около 6 секунди, при възможно най-ниска стойност на управляващия сигнал. Управляващият сигнал е с честота 50Hz и е ШИМ със запълване между $(1000 \pm 50 \rightarrow 2000 \pm 50\mu s)$.

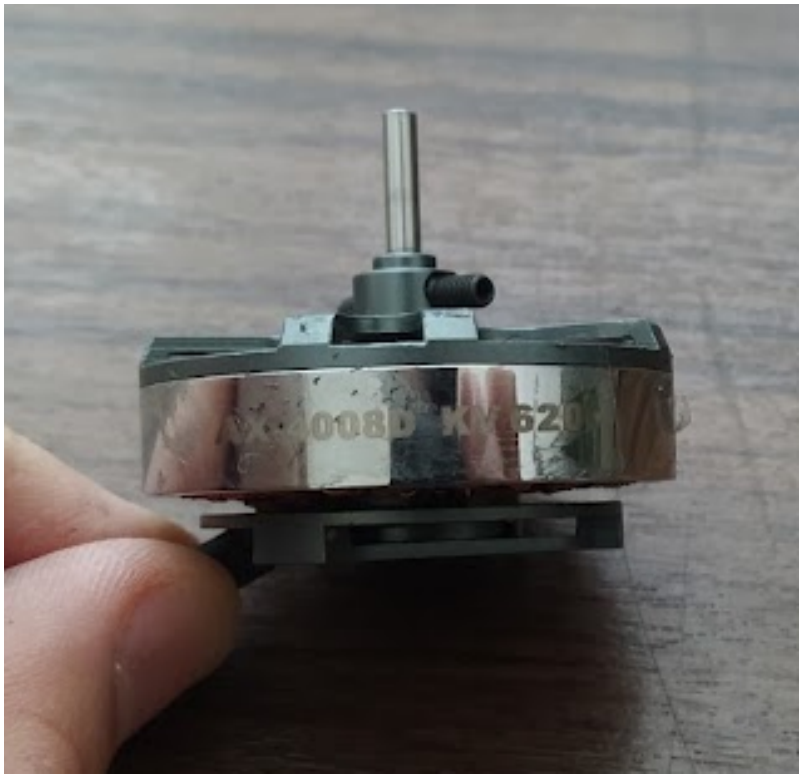


Фигура 11: ESC с анотация на изходите и входовете

3.5 Безчеткови постояннотокови мотори (BLDC)

Използвани са безчетковите постояннотокови мотори *AX-4008D KV620*. С KV константа $620 rpm/V$, тегло $76g$ и максимална мощност $250W$.

Върху оста на мотора е закрепен затягащ конус с резба за монтиране на витлата. За основата им са предварително изработени поставки за монтиране.



Фигура 12: Безчетков постояннотоков мотор AX-4008D KV620

3.6 Li-Po батерия и зарядно

Използвана е Li-Po батерия *TURNIGY NANO 4.5Ah, 14.8V(4S)*, която е четири клетъчна с капацитет 4.5Ah, напрежение 14.8V и максимална скорост на разреждане 50C ($4.5Ah * 50C = 225A$), както и зарядно за батерията *SIGMA 2 EQ* с опции за зареждане и баланс на повечето стандартни типове батерии.

Фигура 13: Li-Po батерия и зарядно

4 Архитектура на системата

Архитектурната част на системата е организирана в три основни части:

Първата част се отнася до дизайна на платформата (физическото оформление), Основните концептуални идеи покрити в физическото оформление, разпределението на хардуера и това как този хардуер ще бъде използван за управлението, както и това каке свързана системата (хардуерно).

Част втора Архитектура на Управлението

Част трета разглежа софтуерната архитектура на управляващото устройство.

FIX THID (write the whole architecture)

[15]

4.1 Конструкция на платформата

4.2 Платформа с четири ротора

Платформата Фигура 14 е конструирана от два метални П-образни профила, сключващи прав ъгъл помежду си и имащи пресечна точка в средата. В края на профилите се намира по един безчетков постояннотоков мотор (без обратна връзка). Перките са свързани директно (без трансмисия) за въртящата ос на моторите. Батерията и контролният модул са позиционирани в средата на платформата. Батерията се намира под пресечната точка на профилите. Управляващото устройство е над пресеччната точка на профилите върху изработена, като част от проекта, платформа.

При този начин на организиране на хардуера центърът на тежеста лежи под пресечната точка на профилите.

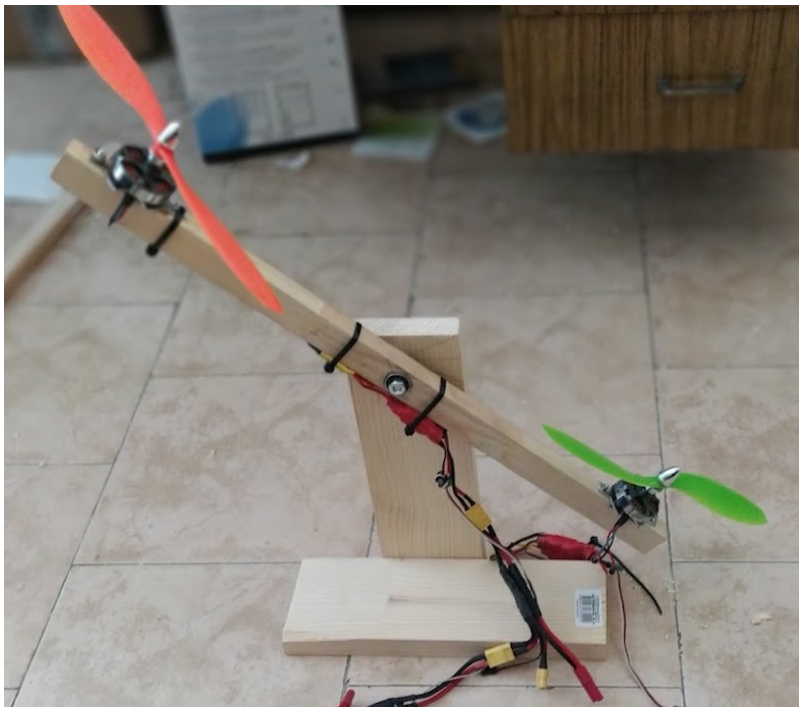
Фигура 14: Конструкция на платформата с четири ротора

4.3 Платформа за управление на ъгъл на завъртане

Платформата за управление на ъгъл на завъртане има за цел да ни позволи лесно и безаварийно да изпробваме алгоритмите за управление на ъгъл на завъртане.

Платфомрата Фигура 15 е изградена от дърво. Състои се от Т-образна основа с ограничители и въртяща част. Основата е висока $30cm$ и е съставена от два правоъгълни дървени профила ($30x10x2.5cm$), съединенин с винтове. Ограничителите ограничават максималния ъгъл, който въртящата част може да сключва с хоризонта в диапазона ($\pm 40°$). Оста на въртене представлява М8 болт. Оста образува болтово съединение с основата, както

и с лагерите на въртяащата част. Въртящата част е съставена от правоъгълен дървен профил ($60x2.5x3cm$) с вложени два лагера (сачмен с дълбок канал $8x22mm$, максимално статично натоварване $138kg$ [8]), като са пробити отвори за болтово съединяване на основите за безчетковите постояннотокови мотори, както и отвори за болтово съединяване на основата на жироскопа и акселерометъра. Останалите нужни компонении като *ESC (Electronic Speed Controller)* за моторите са прикрепени към рамената на въртящата част чрез кабелни превръзки (т.нар свински опашки), като е взето предвид балансиране на платформата чрез разпределение на тежеста на допълнителните елементи.



Фигура 15: Конструкция на платформата за управление на ъгъл

5 Изграждане на системата и решени проблеми

5.1 Изграждане на работната среда

За изграждането на работната среда бяха положени много усилия, вследствие на избора да не се използват готови решения, както и поради нуждата от работа под Linux, тъй като множеството от решения имат целева система Windows.

5.1.1 Система за изграждане (Make)

Работната среда е базирана на *Make*, което ни позволява да изградим собствен инструментариум за целите на проекта. *Make* използва „цели“ (targets) и взаимовръзките „зависимости“ (Dependencies), които изграждат „целите“. За всяка цел се съставя дърво на зависимостите, като се проверява кои зависимости имат нужда от преправяне (rebuild). *Make* е платформно независима и може лесно да бъде пренесена на почти всяка система. *Make* се нуждае единствено от статично или динамично строго описание на взаимовръзките и начините да се премине от зависимостите към целите. Това става чрез дефиниране на командни скриптове „рецепти“ (recipies), което налага ограничението използваният инструментариум да бъде достъпен чрез команден интерфейс.

5.1.2 Компилатор

Подбраният комилатор е *GCC ARM NON-EABI*, който е платформно независим, с целева крайна система ARM. Компилаторът е конфигуриран през командният интерфейс при извикването си от *Make*, като конфигурацията е публикувана Listing 2. Конфигурацията включва използването на хардуерния модул за обработка на числа с плаваща запетая, изключването на всички оптимизации, които комилаторът прави за подобряване на скоростта и използване на паметта с цел гаранция на правилното изпълнение.

5.1.3 Програматор

Подбран е командният пакет за *ST-LINK*, тъй като е платформно независим и позволява командно управление на *ST-LINK V2* устройството, кето е част от комплекта *32F429IDISCOVERY*. Този пакет позволява презаписване на вътрешната флаш памент на конторлера, както и стартиране и управление на порт за дебъг, който да бъде използван за дебъгера.

5.1.4 Дебъгер

Подбран е дебъгер GDB, тъй като е платформно независим и позволява командно базирано дебъгване през дебъг порта отворен с помощта на *ST-LINK*. Дебъгерът позволява условно и безусловно поставяне на точки за прекъсване (breakpoints) на отделни моменти от изпълнението, като се използва генерираният **.elf* обект, който е записан във флаш паметта на процесора.

5.1.5 Получаване и обработка на данни

Данните се изпращат от микроконтролера, през USART порта и се получават на локалната машина (Linux) чрез TTL четец, *picocom*. Данните се записват в **.log* файл, който се обработва автоматично през дефинираната поточна линия от процеси, като създава **.csv* файл, който ще бъде използван от *MATLAB (for Linux)* за обработка на данните и генериране на графики.

5.1.6 Линкерен скрипт

Написан е линкерен скрипт, който дефинира правилното разположение на паметта на контролера и управялва разположението и подравняването на символите и секциите в паметта на микроконтролера. Линкерният скрипт поставя всички константни символи във флаш паметта на контролера, което гарантира, че те няма да бъдат променяни. Линкерният скрипт също гарантира правилното разположение на вектора на прекъсванията в паметта. Линкерният скрипт отстранява всички символи, свързани със стандартните библиотеки, тъй като стандартните библиотеки предполагат наличието на операционна система, която да имплементира подфункциите. По този начин софтуерът, който пишем, е зависим единствено и само от източниците, които ние сме написали или добавили. Крайният резултат е независим статично свързан обект, който ще бъде поставен в микроконтролера.

5.2 Конфигурация на микроконтролера

5.2.1 Стартиране

Написан е прост стартиращ скрипт на ARM assembly за целите на проекта, който инициализира SP на процесора и инициализира PC=Resethandler. Скриптът дефинира символите от таблицата за прекъсванията, като .weak референции, към DefaultHandler, който изпълнява безкраен празен цикъл. Този скрипт има за цел да инициализира .bss секцията на контролера, както и всички статични променливи, като използва позициите, генерирани от линкерния скрипт, както и да извика функцията за системна инициализация, след което се извиква main финкцията.

5.2.2 Системна инициализация

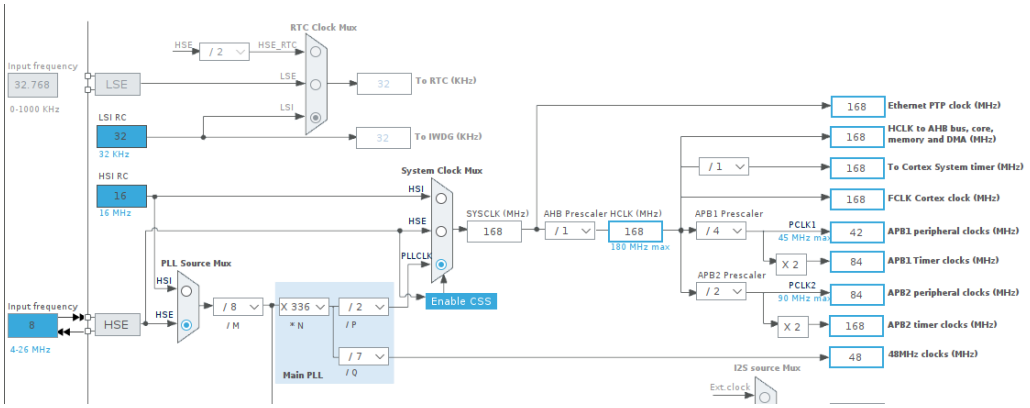
В тази фаза се налага да бъде инициализиран системният часовник. В нашия случай ще инициализираме системния часовник от високочестотният външен осцилатор (HSE). Тъй като при стартиране се използва високочестотният вътрешен осцилатор (HSI), който е неточен и може да доведе до синхроннизационни проблеми при асинхронни комуникации с висок baud-rate като UART. Поради тази причина се налага инициализирането на системния часовник с времева основа високочестотният външен осцилатор (HSE), тъй-като е много по прецизен. Високочестотният външен осцилатор (HSE) на платката е с честота 8MHz, затова използваме хардуерният PLL модул за увеличаване на честотата със следните настройки:

(PLL_M=8, PLL_N=336, PLL_P=2, PLL_Q=7)

, като така получаваме 168MHz честота на системния часовник. След това инициализираме делителите на честота за отделните шини:

(AHB_Prescaler=1, APB1_Prescaler=4, APB2_Prescaler=2).

След настройката на часовниците получаваме следните честоти за отделните шини (Фигура 16)



Фигура 16: Честоти на системните шини и часовници след конфигурация

Дуга основна част на системанта инициализация е инициализирането на хардуерния модул за числа с плаваща запетая (FPU). Тъй като от конфигурацията (Listing 2) настройваме комилатора за работа с модула за числа с плаваща запетая, е важно преди която и да е операция с числа с плаваща запетая този модул да бъде инициализиран. Необходимо е инициализацията на FPU да се случи преди преминването в ограничен режим. Инициализацията се случва, чрез блокът от код (Listing 1).

```
/* Enable The FPU*/
uint32_t* CPACR = (uint32_t*)0xE000ED88;
*CPACR |=  0b00000000111100000000000000000000;
```

Listing 1: Инициализация на модула за числа с плаваща запетая

5.2.3 Инициализация на периферията

WRITE

5.3 Моделиране

5.3.1 Моделиране на едновитлова система.

6 Предложения за надграждане

7 Литература

[1] 3D Jake, *Technical Data Sheet, 3DJAKE ecoPLA*, 3D Jake, 2018. url: https://cdn-3d.niceshops.com/upload/file/Technical_Data_Sheet[0].pdf.

[2] ARM Limited, *ARM v7-M Architecture Reference Manual*, ARM, 2018. url: https://developer.arm.com/documentation/ddi0403/ed.

[3] ARM Limited, *Cortex-M4 Technical Reference Manual (Revision r0p1)*, ARM, 2020. url: https://developer.arm.com/documentation/100166/0001/.

[4] T. Bresciani, „Modelling, identification and control of a quadrotor helicopter“, *MSc theses*, 2008.

[5] GNU, *GCC Manual, Arm Options*, GNU, 2021. url: https://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html.

[6] IKALOGIC S.A.S., *SQ Series User Manual SQ25/SQ50/SQ100/SQ200 4 channels, 200 MSPS logic analyzer and pattern generator*, IKALOGIC S.A.S., -. url: https://cdn.ikalogic.com/docs/ds/sq-series-manual-EN.pdf.

[7] R. Kalachev. (2022). GitHub User: Rafael Kalachev, url: https://github.com/Rafael-Kalachev/thesis_stm32_base (дата на посещ. 12.02.2012).

[8] Mechatronics Bearing Group, *608-2RS BEARING DATASHEET*, Mechatronics Bearing Group, -. url: https://www.mechatronicsbearing.com/data-sheets/608-2RS-Extra-Small-Ball-Bearings.pdf.

[9] Sain samrt, *User Manual: CREALITY ENDER-3 V2 3D PRINTER*, Sain samrt, 2020. url: https://m.media-amazon.com/images/I/B1f9eP6H3OS.pdf.

[10] J. Solà, „Quaternion kinematics for the error-state Kalman filter“, техн. докл., 2017.

[11] ST Microelectronics, *Errata sheet STM32F427/437 and STM32F429/439 line limitations*, ST Microelectronics, 2021. url: https://www.st.com/resource/en/errata_sheet/es0206-stm32f427437-and-stm32f429439-line-limitations-stmicroelectronics.pdf.

[12] ST Microelectronics, *LIS3MDL Digital output magnetic sensor: ultra-low-power, high-performance 3-axis magnetometer*, ST Microelectronics, 2015. url: https://www.st.com/resource/en/datasheet/lis3mdl.pdf.

[13] ST Microelectronics, *LPS25H MEMS pressure sensor: 260-1260 hPa absolute digital output barometer*, ST Microelectronics, 2014. url: https://www.st.com/resource/en/datasheet/lps25h.pdf.

[14] ST Microelectronics, *LSM6DS33 iNEMO inertial module: accelerometer and 3D gyroscope*, ST Microelectronics, 2015. url: https://gcc.gnu.org/onlinedocs/gcc/ARM-Options.html.

[15] ST Microelectronics, *RM0090 Reference manual STM32F405/415, STM32F407/417, STM32F427/437 and STM32F429/439 advanced ARM-based 32-bit MCUs*, ST Microelectronics, 2020. url: https://www.st.com/resource/en/reference_manual/dm00031020-stm32f405-415-stm32f407-417-stm32f427-437-and-stm32f429-439-advanced-arm-based-32-bit-mcus-stmicroelectronics.pdf.

[16] ST Microelectronics, *STM32F427xx STM32F429xx 32b Arm Cortex-M4 MCU+FPU, 225DMIPS, up to 2MB Flash/256+4KB RAM, USB OTG HS/FS, Ethernet, 17 TIMs, 3 ADCs, 20 com. interfaces, camera and LCD-TFT*, ST Microelectronics, 2021.

[17] ST Microelectronics, *UM1061 Description of STM32F2xx Standard Peripheral Library*, ST Microelectronics, 2011. url: https://www.st.com/resource/en/user_manual/um1061-description-of-stm32f2xx-standard-peripheral-library-stmicroelectronics.pdf.

[18] TURNIGY, *TURNIGY Manual for Brushless motor Speed Controller*, TURNIGY, -. url: https://cdn-global-hk.hobbyking.com/media/file/t/u/turnigy_plush_manual.pdf.

[19] M. Watson, „The design and implementation of a robust ahrs for integration into a quadrotor platform“, *Meng electronic engineering, Department of electronic and electrical engineering*, 2013.

8 Приложения

```

##
# LOGGING
##
```

```
5
6  LOGFILE := build/buildlog.log
7
8  ## VERBOSITY
9  # 0 : Do NOT log
10 # 1 : Log only the supplied message
11
12 VERBOSITY := 1
13 VERBOSITY_ECHO  := $(VERBOSITY)
14 VERBOSITY_LOG := $(VERBOSITY)
15
16 LOG_FLAGS :=
17 LOG_FLAGS += --vecho=$(VERBOSITY_ECHO)
18 LOG_FLAGS += --vlog=$(VERBOSITY_LOG)
19 LOG_FLAGS += --logfile=$(LOGFILE)
20
21 ##
22 # CTAGS
23 ##
24
25 CTAGS :=  ctags
26 CTAGS_FLAGS :=  --recurse=yes
27 CTAGS_FLAGS +=  --exclude=.git
28 CTAGS_FLAGS +=  --exclude=BUILD
29 CTAGS_FLAGS +=  --exclude=\*.swp
30 CTAGS_FLAGS +=  --exclude=\*.o
31 CTAGS_FLAGS +=  --exclude=\*.a
32
33
34
35
36 ##
37 # CONTROLLER
38 ##
39
40 MCU_FLASH_MEMORY_START_ADDRESS  := 0x08000000
41 MCU_MODEL := STM32F429_439xx
42
43 ##
44 # COMPILATION
45 ##
46
47
48 C_FLAGS  :=
49
50 # for debugging
51 C_FLAGS += -g
52 # optimization level
53 C_FLAGS += -O0
```

```
54 # configure for the cortex m4
55 C_FLAGS += -mlittle-endian -mthumb -mcpu=cortex-m4 -mthumb-interwork
56 # configure FPU
57 C_FLAGS += -mfloat-abi=hard -mfpu=fpv4-sp-d16
58 # configure Warnings
59 C_FLAGS += -Wall -Wextra
60 # system includes
61 C_FLAGS += -I./startup
62 C_FLAGS += -I./std_perif/inc
63 C_FLAGS += -I./system
64 C_FLAGS += -I./cmsis/inc
65 # home includes
66 C_FLAGS += -I.
67 # controller model
68 C_FLAGS += -D$(MCU_MODEL)
69 C_FLAGS += -DUSE_STDPERIPH_DRIVER
70 C_FLAGS += -DUSE_FULL_ASSERT
71 C_FLAGS += -DARM_MATH_CM4
72
73
74
75 LD_FLAGS :=
76 # add the linker script
77 LD_FLAGS  += -Tlinker/stm32_flash.ld
```

Listing 2: config.mk