МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КІБЕРБЕЗПЕКИ, КОМП'ЮТЕРНОЇ ТА ПРОГРАМНОЇ ІНЖЕНЕРІЇ
КАФЕДРА КОМП'ЮТЕРИЗОВАНИХ СИСТЕМ УПРАВЛІННЯ

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач кафедри

_____ Литвиненко О.Є.
«_____»_____2021 р.

ДИПЛОМНИЙ ПРОЄКТ
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ЗДОБУВАЧА ОСВІТНЬОГО СТУПЕНЯ
«БАКАЛАВР»

Тема: «Система відтворення креслень. Підсистема керування виконуючими пристроями»

Виконавець: Джураєв О.В.

Керівник: Масловський Б.Г.

Нормоконтролер: Тупота Є.В.

Київ 2021

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет Кібербезпеки, комп'ютерної та програмної інженерії
Кафедра комп'ютеризованих систем управління
Спеціальність 123 «Комп'ютерна інженерія»
(шифр, найменування)

Освітньо-професійна програма «Системне програмування»
Форма навчання денна

ЗАТВЕРДЖУЮ
Завідувач кафедри

Литвиненко О.Є.
«_____»_____2021 р.

ЗАВДАННЯ
на виконання дипломної роботи (проєкту)

Джураєву Олегу Володимировичу
(прізвище, ім'я, по батькові випускника в родовому відмінку)

1. Тема дипломної роботи (проекту) «Система відтворення креслень. Підсистема керування виконуючими пристроями»

затверджена наказом ректора від « 04 » лютого 2021 р. № 135/ст.

2. Термін виконання роботи (проєкту): з 17 травня 2021 р. по 20 червня 2021 р.

3. Вихідні дані до роботи (проєкту): програмна та технічна документація до *STM32*, редактор вихідного коду *Visual Studio Code*, середовище конфігурації *STM32CubeMX*, вимоги до оформлення дипломних проектів та ДСТУ 3008-95

4. Зміст пояснювальної записки:

Аналіз принципів побудови сучасних систем числового програмного керування.
Підсистема керування виконуючими пристроями.
Друкована плата виконуючого пристрою.
Програма підсистеми керування виконуючими пристроями

5. Перелік обов'язкового графічного (ілюстративного) матеріалу:

1) Система відтворення креслень. Схема структурна;
2) Підсистема керування виконуючими пристроями. Схема алгоритму;
3) Підсистема керування виконуючими пристроями. Діаграма послідовності;
4) Плата друкована.

6. Календарний план-графік

№ пор.	Завдання	Термін виконання	Відмітка про виконання
1.	Провести огляд літератури за темою дипломного проекту та аналіз існуючих систем.	17.05.2021-18.05.2021	
2.	Зробити вибір середовища програмування і компонентів програмного комплексу	18.05.2021-20.05.2021	
3.	Розробити структуру програмних засобів системи	21.05.2021-22.05.2021	
4.	Розробити програмні засоби. Провести відладку програмних засобів	23.05.2021-05.06.2021	
5.	Написати пояснювальну записку	06.06.2021-10.06.2021	
6.	Підготувати графічний та ілюстративний матеріал	11.06.2021-12.06.2021	
7.	Підготувати доповідь та презентацію	12.06.2021-13.06.2021	

7. Дата видачі завдання: «_17_»___травня____ 2021 р.

Керівник дипломної роботи (проєкту) _____ __Масловський Б.Г.
(підпис керівника) (П.І.Б.)

Завдання прийняв до виконання _____ _____Джураєв О.В.

РЕФЕРАТ

Пояснювальна записка до дипломного проекту «Система відтворення креслень. Підсистема керування виконуючими пристроями.»: 58 сторінок, 22 рисунки, 15 літературних джерел, 5 таблиць, 1 додаток.

ЧИСЛОВЕ ПРОГРАМНЕ КЕРУВАННЯ, КЕРУЮЧА ПРОГРАМА, МІКРОКОНТРОЛЕР *STM32*, *G*-КОД, ПРОГРАМА *ALTIUM DESIGNER*.

Об’єкт – керування виконуючими пристроями за допомогою *G*-кодів.

Предмет – програмна підсистема керування виконуючими пристроями.

Мета дипломного проекту – програма підсистеми керування виконуючими пристроями.

Методи проектування – мова програмування *C*, редактор вихідного коду *Visual Studio Code*, середовище конфігурації *STM32CubeMX*.

Результати дипломного проектування рекомендується використовувати для автоматизації графобудування, гравірування та різання матеріалів без участі людини, для забезпечення якості та швидкості виробництва.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ, ТЕРМІНІВ

МК – мікроконтролер

АСТПВ – автоматизована система технологічної підготовки виробництва

САПР ТП – система автоматизованого проектування і розрахунку технологічних процесів

ПГКП – підсистема генерації керуючих програм

CNC – computer numerical control

CAD – computer-aided design

CAM – computer-aided manufacturing

CAPP – computer-aided process planning

ЧПК – числове програмне керування

ПЗП – постійний запам’ятовуючий пристрій

ОЗП – оперативний запам’ятовуючий пристрій

ВСТУП

Системи комп'ютерного числового керування (ЧПК) – це розумні, багатовісні, багатофункціональні, скоординовані системи управління рухом, що використовуються для точного контурування. ЧПК можуть працювати з різними матеріалами в широкому діапазоні галузей. Незважаючи на те, що ЧПК та цифрові елементи керування є синонімами, ЧПК відрізняються від контролерів руху загального призначення. ЧПК відповідають спеціальним вимогам верстатобудівної промисловості, тоді як контролери руху забезпечують загальноприйняте координоване управління рухом.

Ринок комп'ютерних числових систем управління, включаючи обладнання, програмне забезпечення та послуги, динамічний, і основні сегменти зазнають швидких інновацій у галузі. Цей ринок зумовлений інноваціями у виробничих технологіях, необхідними переважно для автомобілебудування, машинобудування та аерокосмічної та оборонної промисловості. В останні роки інші сегменти, такі як медичні вироби, електроніка, меблі та вироби з дерева, стають все більш важливими ринками для кінцевих споживачів.

Ринок ЧПК є дуже консолідованим, як і протягом багатьох років, серед трьох найкращих світових постачальників: *Siemens, FANUC* та *Mitsubishi Electric*. Ці три постачальники мають переважну більшість від загальної частки ринку, а *Siemens* і *FANUC* перевершують *Mitsubishi Electric* за часткою ринку.

Рішення з ЧПК все частіше застосовуються на заводах масового виробництва завдяки їх здатності виготовляти високоточні деталі та компоненти. Більше того, різноманітний спектр таких технологій, як САПР і САМ, що використовуються в ЧПК для виготовлення стандартизованих деталей за допомогою верстатів, є повністю взаємозамінними. Крім того, застосування ЧПК на заводах-виробниках має прямий вплив на зниження собівартості разом із суттєвим підвищенням продуктивності та якості кінцевого продукту. Таким чином, усі вищезазначені можливості ЧПК впливають на зростання вертикальних

галузей кінцевих споживачів, таких як автомобільна та аерокосмічна та оборонна, що, у свою чергу, збільшує попит на верстати з ЧПК на заводах масового виробництва.

Збільшення потреби в складних деталях, виготовлених із точною обробкою, поряд з експлуатаційною ефективністю, головним чином сприяє зростанню ринку комп'ютерного цифрового управління. Верстати з ЧПК в основному використовуються для гравірування на поверхні дерева та металу завдяки їх високоточним і точним операціям. ЧПК має мікро- та нано-обробні можливості, які допомагають йому обробляти нові матеріали, такі як полімери, і забезпечувати результати точно відповідно до вимог замовника. Застосування ЧПК в технологічних установках скорочує час, необхідний для ефективного проектування та виробництва. Ця особливість сприяє компаніям застосовувати рішення з ЧПК, тим самим стимулюючи ріст ринку ЧПК.

Ринок ЧПК зазнає технологічних змін. Технологія розвинулася настільки, що швидкості обробки зросли на багато порядків. Типові швидкості контурування 600 *CPM* тепер зростають до 3000 *CPM* на багатьох верстатах. Майстерні, що тримаються за старі верстати, швидко знайдуть свій бізнес у конкурентно невигідному положенні. Потреба в покращеній обробці поверхні, підвищеній точності та скороченні часу налаштування робить ручний верстат практично застарілим.

Системи з ЧПК все частіше застосовуються в широкому спектрі задач не тільки обробки металевих виробів. Електронна промисловість все частіше використовує стандартні системи з ЧПК для обробки матеріалів та високошвидкісного свердління. Різання дерева, різання скла, контурування металевих пластин та ринок електронних плат для ПК розширюють область застосування систем з ЧПК.

Мікроконтролер для пристроїв з числовим програмним керуванням відіграє важливу роль, від вибору мікроконтролеру залежать такі фактори як швидкість виконання керуючих програм та точність обробки деталі.

На ринку доступний широкий асортимент мікроконтролерів різних виробників. Всі ці мікроконтролери мають унікальні функції та постачаються з різною ємністю оперативної пам’яті та ПЗП, різним набором інструкцій, різною архітектурою, регістрами тощо. Всі ці мікроконтролери відрізняються один від одного. Тож вибір відповідного мікроконтролера – це завжди непросте завдання, оскільки слід врахувати ряд технічних особливостей.

Об’єкт проектування – керування виконуючими пристроями за допомогою *G*-кодів.

Предмет проектування – програмна підсистема керування виконуючими пристроями.

Мета дипломного проекту – програмна підсистема керування виконуючими пристроями.

Методи проектування – мова програмування *C*, редактор вихідного коду *Visual Studio Code*, середовище конфігурації *STM32CubeMX*.

Практичне значення отриманих результатів. Розроблена в даному дипломному проекті підсистема керування виконуючими пристроями дозволяє створити верстат з числовим програмним керуванням з високою точністю і швидкістю для обробки деталей.

РОЗДІЛ 1
АНАЛІЗ ПРИНЦИПІВ ПОБУДОВИ СИСТЕМ ЧИСЛОВОГО ПРОГРАМНОГО КЕРУВАННЯ

1.1. Основні поняття в системах числового програмного керування

На сьогодні верстати з ЧПК знайшли широке розповсюдження як на великих, середніх і малих підприємствах так і в побуті. Розвиток напрямку потребує від операторів верстатів і розробників ПЗ розуміння структури, можливостей пристроїв з ЧПК та вмінь для ефективного застосування цього обладнання. Для повноцінного сприйняття згаданих вище пунктів важливо знати та орієнтуватись у галузевій термінології.

Числове програмне керування – це автоматичне управління шляхом передачі інформації у формі чисел від програмоносія до виконавчого органу, яке визначає його рух або виконання ним інших функцій.

Верстат з ЧПК – це моторизований маневрений інструмент та платформа якими за допомогою керуючих програм керує комп'ютер. Інструкції доставляються до верстату у вигляді послідовних програм складених зазвичай за допомогою *G*-кодів та *M*-кодів. Програми можуть створюватись людьми, але частіше всього створюються за допомогою програмного забезпечення (САПР).

Система ЧПК (СЧПК) – це поняття під яким розуміють сукупність функціонально взаємозалежних і взаємодіючих технічних і програмних засобів, що забезпечують числове програмне управління устаткуванням. Безпосередньо пристрій числового програмного керування (ПЧПК) складає частину системи і конструктивно виконується у вигляді окремого блока.

Керуюча програма – це системна програма, що реалізує набір функцій керування, який включає у себе керування ресурсами та взаємодію з оточуючим середовищем системи обробки інформації, відновлення роботи системи після виявлення несправностей у технічних засобах. Керуюча програма включає в себе закодовану інформацію про траєкторії, швидкості та іншу інформацію що необхідна для виконання обробки деталі. Під час роботи керуюча програма кадр за кадром поступає на виконання. Відповідно до команд керуючої програми контролер викликає із постійної пам'яті відповідні системні підпрограми, які і змушують працювати підключене до ЧПК обладнання в заданому режимі. [3]

G-код – загальноприйнята назва однієї з найбільш широковживаних мов програмування числовим програмним керуванням. Здебільшого застосовується в галузі автоматизації. Структура мови стандартизована документом *ISO6983* міжнародного комітету стандартів. Попри назву включає в себе програмування за допомогою *G*-кодів та *M*-кодів. Офіційно мова *G*-кодів визнана стандартом для європейських та американських виробників пристроїв ЧПК також має назву «*ISO* 7 біт». [2]

Графопобудовник – пристрій, призначений для виведення даних в графічній формі на папір.

Система автоматизованого проектування і розрахунку (САПР, *CAD*) – інформаційний комплекс призначений для автоматизації проектування, що складається із апаратного забезпечення (ЕОМ), програмного забезпечення, описів способі і методів роботи зі системою, протоколом правил зберігання. Однак з поширенням в середовищі спеціалістів термін дещо втратив свій початковий сенс, і тепер став позначати тільки програмну частину від початкової системи автоматизованого програмування. Зараз здебільшого вживається у значенні деякої програми призначеної для автоматизації технологічного процесу проєктування виробу, результатом якого є набір проєктно-конструкторської документації, достатньої для виготовлення та подальшої експлуатації деталі

На сучасному ринку існує велика кількість САПР, які дозволяють вирішувати різні задачі. Умовно групу програм можна розділити на два типи:

– базові або ж легкі САПР призначені для двовимірного проектування та креслення, а також для створення деяких окремих тривимірних моделей без можливості роботи зі складальними одиницями;

– складні або ж важкі САПР призначені для роботи із складними виробами, наприклад складними конструкціями авіабудування, суднобудування та інше. Функціонально мають розширений набір вирішуваних задач, іншу архітектуру та алгоритми роботи у порівнянні із легкими.

Система автоматизованого проєктування і розрахунків технологічних процесів (САПР ТП, *CAPP*) – допоміжні програмні продукти для автоматизації процесу технологічної підготовки виробництва, а саме: планування технологічних процесів та оформлення відповідної технологічної документації. Основним завданням САПР ТП є складання плану виробництва, маршруту виготовлення виробу по його електронній моделі. У цей маршрут мають включатись відомості про послідовність технологічних операцій виготовлення деталі, режими здійснення технологічних операцій та інше. Система САПР ТП є елементом, що сполучає САПР і АСТПВ. [4]

Автоматизована система технологічної підготовки виробництва (АСТПВ, *CAM*) – система для підготовки технологічного процесу виробництва виробів, орієнтована на використання ЕОМ. Під терміном розуміють як сам процес комп'ютеризованої підготовки виробництва, так і програмно обчислювальні комплекси. Фактично етап технологічної підготовки зводиться до автоматизації програмування устаткування з ЧПК, генерації керуючих програм. [5]

Підсистема керування виконуючими пристроями – підсистема є основною частиною загальної системи числового програмного керування. З одного боку вона зчитує керуючу програму та інтерпретує їх в аналогові сигнали та віддає різним агрегатам верстата. З іншого – дозволяє верстату часткового взаємодіяти із людиною оператором, дозволяючи контролювати процес обробки. Існують закриті та відкриті (ПК-сумісні) системи керування.

Закриті системи керування мають власні алгоритми обробки, власні методи. Виробники закритих систем, в більшості випадків не надають інформацію про їх

структуру. В таких випадках практично неможливо оновлювати та редагувати програмне забезпечення. Водночас закриті системи мають суттєву перевагу – високу ступінь надійності, оскільки усі компоненти системи тестуються на предмет сумісності.

ПК-сумісні системи мають вищий ступінь відкритості. Їх апаратні складові такі ж як і у звичайного побутового комп'ютера. Перевагою такого типу систем є доступність та відносно невелика вартість електронних компонентів, проте прийнято вважати, що надійність таких систем поступається надійності закритих систем. [1]

1.2. Принципи побудови систем числового програмного керування

Головним елементом підсистеми керування є контролер або процесор, який найчастіше розташовується в корпусі стійки з ЧПК, що має клавіатуру та монітор для вводу і виводу необхідної інформації.

Контролер – комп'ютеризований пристрій, що вирішує наступні задачі:

– формування траєкторії руху інструменту;

– реалізацію технологічних команд керування пристроями автоматики верстата;

– загальне керування;

– редагування керуючих програм;

– діагностику та допоміжні розрахунки, наприклад, режимів різання і інше.

Конструктивні особливості та основи програмування верстатів з числовим програмним керуванням.

В якості контролера використовують мікропроцесор, на якому побудована система або логічний контролер, що програмується, чи більш складну систему – промисловий комп'ютер.

Структурно до складу ЧПК також входять постійна пам'ять (ПЗП) у вигляді карти пам'яті або жорсткого диску для довготривалого зберігання інформації та

оперативна пам’ять (ОЗП) для тимчасового зберігання керуючих та системних програм, що використовуються в даний момент.

Розробка керуючих програм на сьогоднішній день виконується з використанням спеціальних модулів для систем автоматичного проектування (САПР) або окремих систем автоматизованого програмування (*CAM*), які у відповідності до електронної моделі генерують програму обробки.

В процесі створення або після введення керуючої програми оператор може відредагувати її, включивши в роботу системну програму редактора та виводячи на дисплей всю або потрібні частини керуючої програми і вносячи потрібні зміни.

Під час роботи в режимі виготовлення деталі керуюча програма кадр за кадром поступає на виконання.

У відповідності до команд керуючої програми контролер викликає із постійної пам'яті відповідні системні підпрограми, які і заставляють працювати підключене до ЧПК обладнання в заданому режимі.

Результат роботи контролера у вигляді електричних сигналів поступає на виконавчий орган – привод подач, головний привод чи пристрій керування автоматикою верстата.

Для визначення потрібної траєкторії переміщення робочого органу у відповідності до керуючої програми використовується інтерполятор, що розраховує положення проміжних точок траєкторії за заданими в програмі кінцевими точками.

Завдяки системі керування верстати з ЧПК мають розширені технологічні можливості при збереженні високої надійності.

За характером руху виконавчих органів системи ЧПК розділяють на:

– позиційні;

– контурні;

– синхронні;

– універсальні.

При позиційному керуванні засоби ЧПК забезпечують переміщення виконавчого органу в задану координату. Привод подач повинен забезпечувати

високу швидкість переміщення для зменшення часу холостих ходів. Оброблювання при цьому не виконується і вид траєкторії переміщення не задається. Точність вимагається лише при зупинці в заданій координаті.

При контурному керуванні переміщення виконавчих органів виконується по заданій траєкторії з установленою швидкістю для отримання потрібної форми оброблюваної поверхні.

Розрізняють контурні прямокутні системи, контурні криволінійні та синхронні системи ЧПК.

Контурні прямокутні системи забезпечують рух лише по одній координаті для оброблювання поверхні, що паралельна даній вісі. Програмуються кінцеві координати точок переміщення як і в позиційній системі, але вказується швидкість переміщення відповідно до заданих режимів різання. Переміщення виконуються послідовно по кожній із координатних вісей. Прямокутні системи використовують в токарних, фрезерних та шліфувальних верстатах.

Контурні криволінійні системи забезпечують формоутворення при оброблюванні заготовки з одночасним узгодженим рухом виконавчого органу по декількох координатах. Програму переміщення виконавчих органів розробляють відповідно до заданої форми обробленої поверхні та результуючої швидкості, що розрахована для заданих режимів різання. Ці системи найбільш складні як по створенню програм так і по вимогах до приводу подач.

Синхронні системи використовуються в основному в зубооброблювальних верстатах, де необхідно витримувати постійне співвідношення швидкостей не менше ніж по двох координатних вісях. Формоутворення реалізується завдяки конфігурації інструменту.

Універсальне керування об'єднує принципи позиційного та контурного криволінійного, що дозволяє виконувати позиціювання в задану координату та рух виконавчих органів по відповідній траєкторії. Воно найбільш ефективне для багатоцільових верстатів.

1.3. Аналіз сучасних систем числового програмного керування

Наразі існує доволі невелика кількість безкоштовних прошивок для систем з числовим програмним керуванням, так як більшість прошивок є пропрієтарними і постачаються тільки в комплекті з верстатами.

GRBL – чи не найпростіша прошивка в цьому списку. *GRBL* розпочався як скромний проект, який прагнув інтерпретувати *G*-код на *Arduino Uno*. Зараз він переріс у дві різні версії, звичайну *GRBL* та *GRBL Mega*. Різниця між ними – плати, які вони підтримують. Перший підтримує *Arduino Uno* та інші *328p* плати *Arduino*. *GRBL Mega* підтримує *Arduino Mega*, який має більше пам'яті та контактів для майбутніх удосконалень.

GRBL отримує свою цінність завдяки простоті; він інтерпретує *G*-код без жодних наворотів. Незважаючи на те, що він має декілька чудових функцій, таких як прискорення та режим «*Jog*», він виключає деякі більш розширені функції, такі як макроси та цикли свердління. За словами розробників, це тому, що ці функції краще робити з боку програмного забезпечення.

На даний момент *GRBL* підтримує лише 3-осьові верстати. Наприклад, є відгалуження проекту *GRBL Mega*, яке підтримує більше вісей.

Встановлення та налаштування *GRBL* досить просте. Прошивка *GRBL* так само проста, як додавання бібліотеки та завантаження ескізу в *IDE Arduino*. Конфігурація *GRBL* має послідовні команди для налаштування параметрів після прошивки. Після налаштування *GRBL* можливо вибрати одного з багатьох відправників *G*-коду, створених для використання з *GRBL*.

Klipper – це прошивка, яка використовує декілька плат *Arduino* та поєднує їх із комп’ютером для управління ЧПК. *Klipper* застосовує інший підхід до управління ЧПК, ніж більшість. За допомогою *Klipper* комп’ютер дбає про синхронізацію, фізику та інтерпретацію *G*-коду, тоді як мікроконтролер вчасно виконує інструкції.

Є можливість використання декількох плат *Arduino*, що керують різними частинами ЧПК. Це вирішує проблему з недостатньою кількістю драйверів або

недостатньою потужністю. Вся конфігурація робиться швидко і легко на *Raspberry Pi*.

Klipper в даний час працює з точністю синхронізації до 25 *мс* і приблизно від 100000 до 700000 кроків в секунду, залежно від мікроконтролера. [10]

Недолік *Klipper* полягає в тому, що він підтримує лише *3D*-принтери.

Marlin – чи не найдосконаліша прошивка у цьому списку. Це прошивка для будь-якого *3D*-принтера і навіть підтримує конфігурації лазера та шпинделя з ЧПК. *Marlin* можна налаштувати для запуску різних стилів та налаштувань принтера. Якщо є функція, яку користувач хоче додати до свого принтера, *Marlin* скоріше за все зможе її підтримати

Marlin має чудові функції, такі як автоматичне вирівнювання станини за допомогою щупа та файлу автозапуску. *Marlin* виготовляється з урахуванням усіх принтерів, тому постачається з великим переліком налаштувань. Вони дозволяють увімкнути будь-що – від датчика ширини волокна до підсвічування корпусу.

Завдяки цій гнучкості, налаштування *Marlin* може бути тривалим залежно від конфігурації.

Repetier-Firmware – це універсальне програмне забезпечення від *Repetier*. Воно поставляється з багатьма конфігураціями для роботи на різних принтерах і має режим ЧПК для використання з іншими інструментами.

Repetier має клієнтське програмне забезпечення, та серверне, яке встановлено безпосередньо в міроконтролер. Це є перевагою в тих випадках, коли потрібно турбуватись про сумісність.

Найбільш привабливим і корисним аспектом цієї прошивки є її зручність для користувачів. *Repetier* має поглиблений покроковий посібник для налаштування всіх параметрів принтера. Розробники надають інструмент конфігурації для тих, кого залякує безліч варіантів налаштування.

Недоліком для всіх цих прошивок є використання і сумісність тільки з платами *Arduino*. Недоліками *Arduino* в порівнянні з мікроконтролерами та платами побудованими на основі мікроконтролерів *STM32*, для порівняння

будемо використовувати мікроконтролер *ATmega328*, на основі якого побудована плата *Arduino UNO* та *STM32F103C8* (табл. 1.1).

Таблиця 1.1

Порівняння мікроконтролерів

Мікроконтролер	*STM32F103C8*	*ATmega328*
Частота	До 72 *МГц*	До 20 *МГц*
Розрядність	32 *біт*	8 *біт*
ПЗП	64 *КБ*	32 *КБ*
ОЗП	20 *КБ*	2 *КБ*
Таймери	7	3
GPIO	37	До 28
USB	*USB 2.0 FS*	–
UART	3	1
SPI	2	2
CAN	1	–
I^2C	2	1

1.4. Висновки до розділу

Розглянуто основні поняття та терміни, які використовуються в системах з числовим програмним керуванням.

Проаналізовано сучасні програмно-апаратні рішення в області проектування систем з числовим програмним керуванням та виявлено основні особливості до структури побудови систем числового програмного керування.

Розглянуто особливості структури систем числового програмного керування, розглянуті задачі виконувані контролером, розглянуто процес розробки керуючих програм, класифіковано типи виконавчих органів системи числового програмного керування та проведено їх огляд.

Розглянуто та порівняно мікроконтролери *ATmega328* та *STM32F103C8* для побудування системи числового програмного керування, порівняні між собою наступні характеристики:

– частота;

– розрядність;

– кількість пам’яті;

– кількість інтерфейсів вводу-виводу.

Проведений аналіз показує, що для розкриття теми дипломного проекту потрібно виконати наступне:

1) розробити структурну схему системи відтворення креслень;

2) розробити схему алгоритму підсистеми керування виконуючими пристроями;

3) провести програмну реалізацію розробленого алгоритму;

4) розробити інструкцію користувача;

5) провести тестування підсистеми керування виконуючими пристроями.

РОЗДІЛ 2
ПІДСИСТЕМА КЕРУВАННЯ ВИКОНУЮЧИМИ ПРИСТРОЯМИ

2.1. Функціональні вимоги

Функціональні вимоги підсистеми:

- з’єднання з *CAM* системою через *USB*;
- дешифрування керуючих команд та перетворення їх в аналогові сигнали;
- керування кроковими двигунами в вісях *X*, *Y*, *Z*;
- зворотній зв’язок з системою;
- опитування датчиків.

Вимоги до структури підсистеми:

- модуль з’єднання та взаємодії з керуючим пристроєм;
- модуль аналізу керуючих програм;
- модуль керування кроковими двигунами;
- модуль опитування датчиків;

2.2. Структура програмного модуля

Комплекс для відтворення креслень за допомогою числового програмного керування складається з двох підсистем:

- підсистема генерації керуючих програм;
- підсистема керування виконуючими пристроями.

Підсистема генерації керуючих програм представляє собою комп’ютерну програму, яка складається з модулів САПР та *CAM*, програма комунікує з користувачем за допомогою інтерфейсу користувача, та після генерації керуючої програми передає керуючу програму за допомогою інтерфейсу зв’язку з виконуючим пристроєм та через універсальну послідовну шину (*USB*).

Підсистема керування виконуючим пристроєм отримує данні за допомогою інтерфейсу зв’язку з підсистемою генерації виконуючих програм за допомогою модуля розшифровки перетворює керуючі програми в зрозумілі для контролера команди і передає їх в модуль перетворення команд в аналогові сигнали, які в свою виконуються в модулі керування кроковими двигунами, зворотній зв’язок між підсистемами виконується за допомогою модуля моніторингу стану системи. Структурна схема системи відтворення креслень зображена на рис. 2.1.

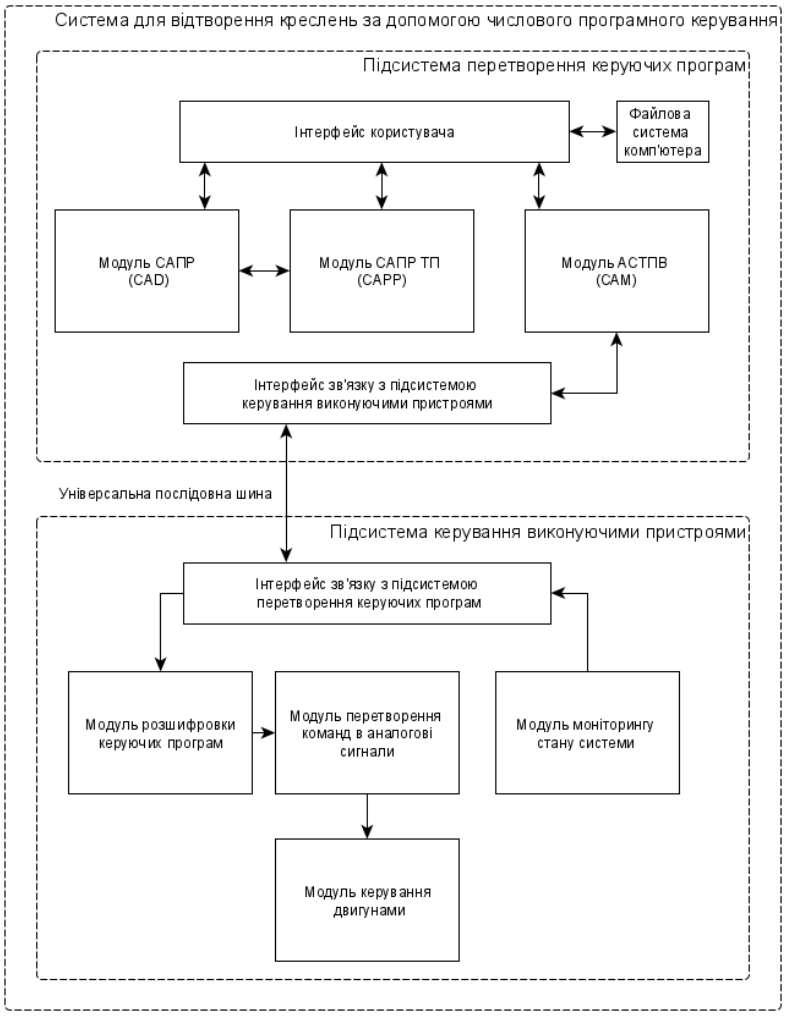

Рис. 2.1. Структурна схема системи

На діаграмі послідовності (рис. 2.2) підсистеми керування виконуючими пристроями відображена взаємодія між елементами програми та повідомлення між елементами впорядковані за часом.

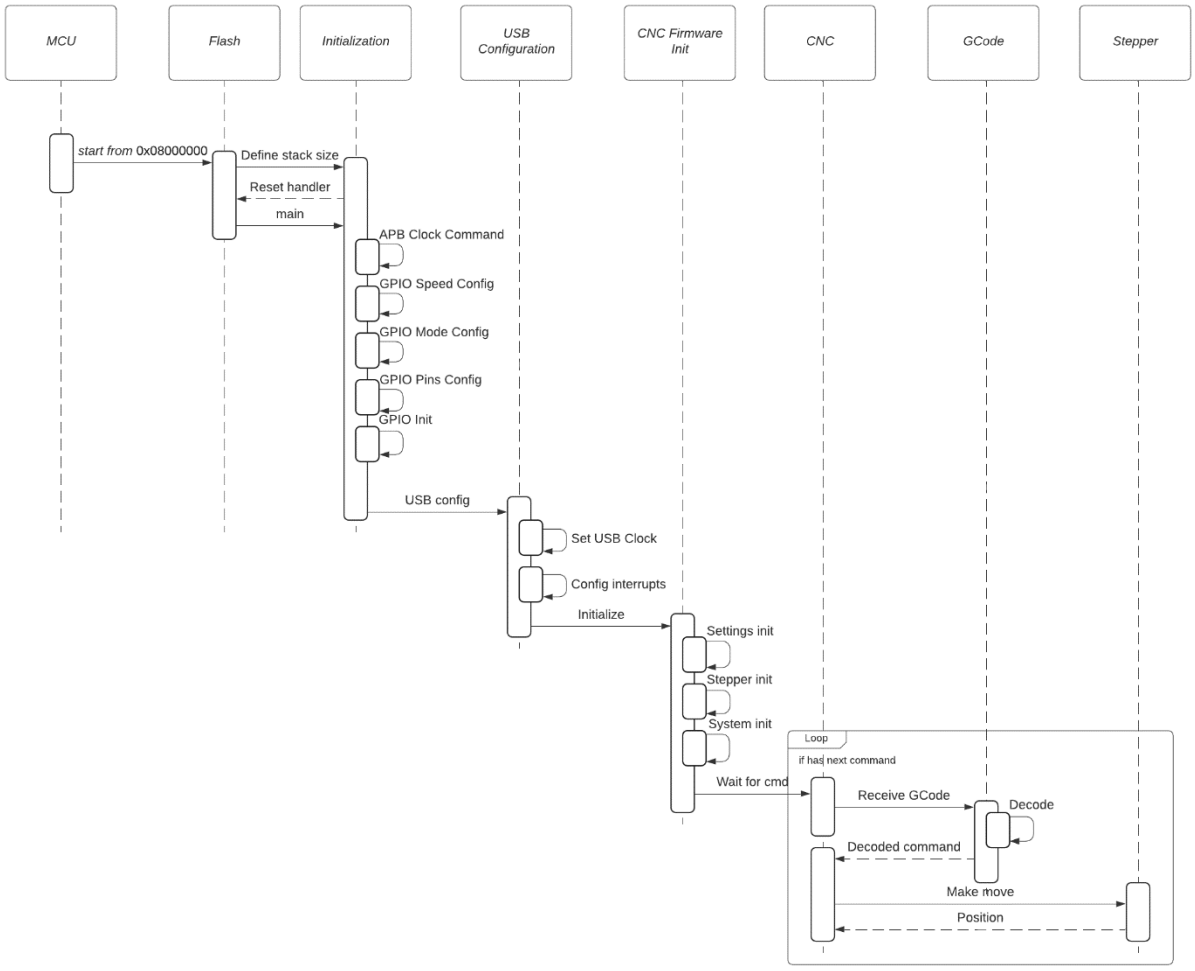

Рис. 2.2. Діаграма послідовності підсистеми

Алгоритм програми керування виконуючими пристроями виконує ініціалізацію тактових генераторів системи та периферії, після чого конфігуруються вводи-виводи, їх напрям та режим, далі виконується ініціалізація самої підсистем керування виконуючими пристроями, після чого система входить в нескінченний цикл, цикл починається з перевірки датчиків кінця руху, при спрацьовуванні датчика відправляється відповідне повідомлення в підсистему генерації керуючих програм, яка має відпрацювати повідомлення, далі програма перевіряє на наявність даних на універсальній шині і в разі їх відсутності

підсистема переходить в режим очікування, після отримання даних мікроконтролер генерує переривання і виводить програму з стану очікування, починається обробка та виконання керуючої програми. Схема алгоритму зображена на рисунку 2.3.

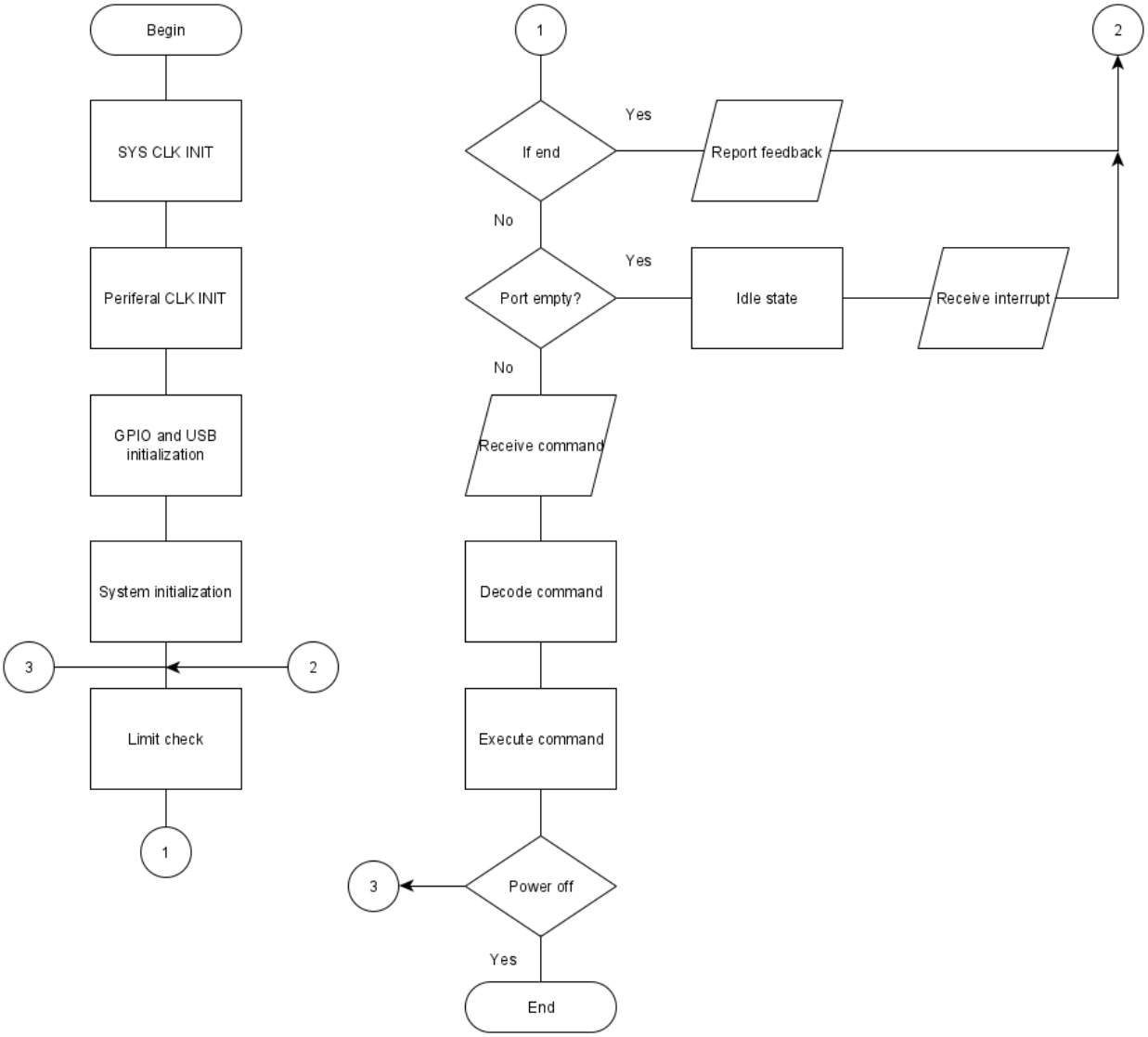

Рис. 2.3. Схема алгоритму програми

2.3. Інструментальні засоби розробки

Visual Studio Code – це легкий, але потужний редактор вихідного коду, доступний для *Windows*, *macOS* та *Linux*. Він постачається із вбудованою підтримкою *JavaScript*, *TypeScript* та *Node.js* і має багату екосистему розширень

для інших мов (таких як *C ++*, *C#*, *Java*, *Python*, *PHP*, *Go*) та середовищ виконання (таких як *.NET* та *Unity*) .

Завдяки підтримці сотень мов, *VS Code* допомагає миттєво підвищити продуктивність за допомогою підсвічування синтаксису, відповідності дужок, автоматичного відступу, виділення вікон, фрагментів тощо. Інтуїтивно зрозумілі комбінації клавіш, просте налаштування та відображення комбінацій клавіш, внесених спільнотою, дозволяють легко орієнтуватися в коді.

Visual Studio Code включає вбудовану підтримку заповнення коду *IntelliSense*, розширене розуміння семантичного коду та навігацію, а також рефакторинг коду.

Visual Studio Code включає інтерактивний налагоджувач, налагоджувач дозволяє перевіряти змінні, переглядати стеки викликів та виконувати команди на консолі.

VS Code також інтегрується з інструментами побудови та сценарію для виконання загальних завдань, що пришвидшує щоденні робочі процеси. *VS Code* має підтримку *Git*, тому є можливість працювати з системами контроля версіями, не виходячи з редактора, включаючи перегляд змін, що очікуються.

STM32CubeMX – це графічний інструмент, який дозволяє дуже легко конфігурувати мікроконтролери та мікропроцесори *STM32*, а також генерувати відповідний код ініціалізації *C* для ядра *Arm Cortex-M* або часткове дерево пристроїв *Linux* для *Arm Cortex-A*, через покроковий процес. Перший крок полягає у виборі: мікроконтролера *STMicrolectronics STM32*, мікропроцесора або платформи розробки, яка відповідає необхідному набору периферійних пристроїв.

Для мікропроцесорів другий крок дозволяє налаштувати *GPIO* і налаштування тактування для всієї системи, а також інтерактивно призначити периферію для *Arm Cortex-M*, або *Cortex A*. Спеціальні утиліти, такі як конфігурація та налаштування *DDR*, полегшують початок роботи з мікропроцесорами *STM32*. Для ядра *Cortex-M* конфігурація включає додаткові кроки, які точно відповідають описаним для мікроконтролерів.

Для мікроконтролерів та мікропроцесора *Arm Cortex-M* другий крок полягає у налаштуванні кожного необхідного вбудованого програмного забезпечення завдяки вирішувачу конфлікту між контактами, помічнику налаштування тактування, калькулятору енергоспоживання та утиліті, яка налаштовує периферію такі як *GPIO* або *USART* та стеки проміжного програмного забезпечення (наприклад, *USB* або *TCP/IP*).

Продуктивне сімейство середньої щільності *STM32F103xx* включає високопродуктивне 32-розрядне ядро *RISC ARM Cortex-M3*, що працює на частоті 72 *МГц*, високошвидкісні вбудовані пам'яті (флеш-пам'ять до 128 *Кбайт* і *SRAM* до 20 Кбайт) , а також широкий спектр вдосконалених входів/виходів та периферійних пристроїв (рис 2.4), підключених до двох шин *APB*. Усі пристрої пропонують два 12-*бітові* АЦП, три 16-*бітові* таймери загального призначення плюс один ШІМ-таймер, а також стандартні та вдосконалені інтерфейси зв'язку: до двох *I2C* і *SPI*, три *USART*, *USB* і *CAN*. Пристрої працюють від джерела живлення від 2,0 до 3,6*В*. Вони доступні як в діапазоні температур від –40 до +85°*С*, так і в розширеному діапазоні температур від –40 до +105°*С*. Широкий набір режиму енергозбереження дозволяє розробляти програми з низьким енергоспоживанням.

Продуктивне сімейство середньої щільності *STM32F103xx* включає пристрої у шести різних типах упаковок: від 36 контактів до 100 контактів. Залежно від обраного пристрою включені різні набори периферійних пристроїв.

Ці особливості роблять лінійку мікроконтролерів продуктивності середньої щільності *STM32F103xx* придатною для широкого спектра застосувань, таких як мотор-приводи, управління додатками, медичне та кишенькове обладнання, периферія для ПК та ігор, GPS-платформи, промислові програми, ПЛК, інвертори, принтери, сканери , сигналізація, відеодомофони та ОВК.

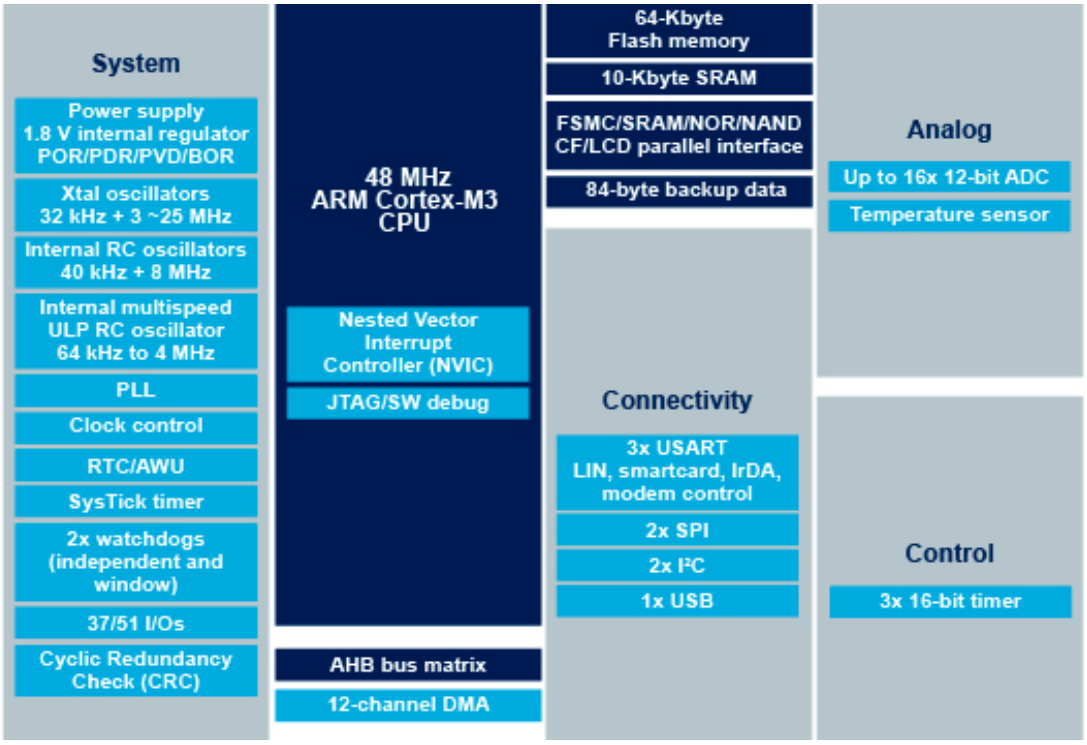

Рис. 2.4. Діаграма інтерфейсів та компонентів *STM32F103xx*

2.4. Розробка друкованої плати

В основі друкованої плати для підсистеми керування виконуючим пристроєм буде використовуватись мікроконтролер *STM32F103C8T6* в конфігурації з 64 *Кбайт* флеш-пам'яті, 20 *Кбайт* статичної оперативної пам'яті з довільним доступом, 2 інтерфейси *SPI* та I^2C, 3 інтерфейси *USART* та 37 вводів\виводів загального призначення, пакування *LQPF48*. [6]

Керування кроковими двигунами буде виконуватись за допомогою трьох драйверів *A4988*, для керування кроковими двигунами на вісях *X*, *Y* та *Z*.

A4988 – це повний мікрокроковий драйвер двигуна з вбудованим транслятором для зручності роботи. Він призначений для роботи з біполярними кроковими двигунами в повній, напів-, чверть-, 1/8 та 1/16 режимах з вихідною потужністю приводу до 35*B* і ± 2*A*. *A4988* включає регулятор відключення постійного струму, який має можливість працювати повільно або змішано в режимі спаду. [7] Схему підключення драйверу зображено на рис. 2.5.

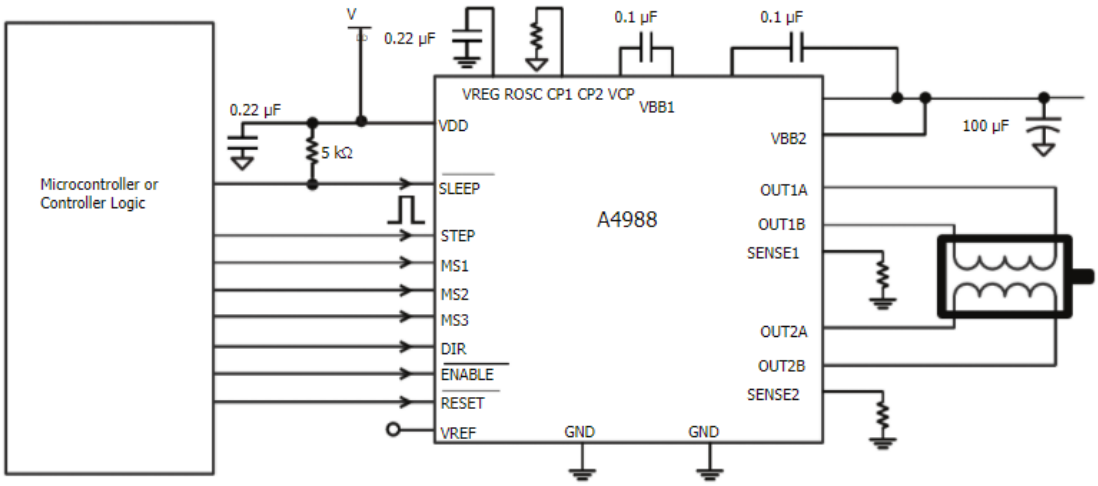

Рис. 2.5. Схема підключення *A4988*

В *Altium Designer* створюємо новий проект та додаємо в нього файл принципової схеми. [11] На схему розміщуємо мікроконтролер та драйвери (рис. 2.6).

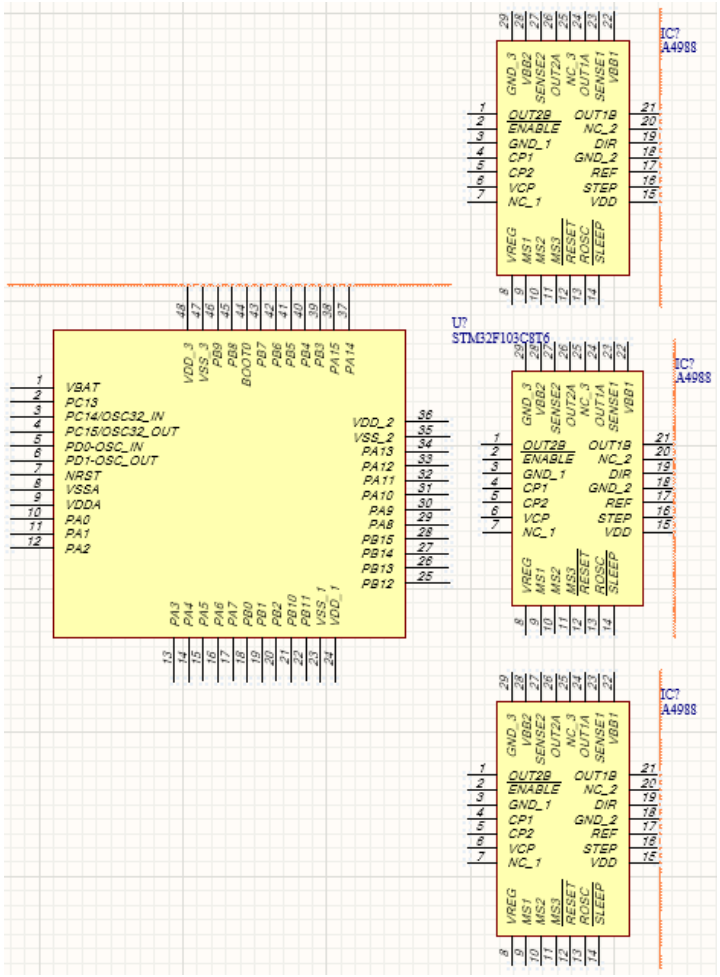

Рис. 2.6. Розміщені компоненти

Для живлення мікроконтролера та драйверів будемо використовувати лінійний регулятор напруги *Texas Instruments LM317* з вихідним діапазоном 1.25-32B, максимальний вихідний струм 0.1A, робочий діапазон температур від -40 до +125°C. [8] Вихідна напруга розраховується за формулою (2.1).

$$V_{OUT} = V_{REF} \times \left(1 + \frac{R_2}{R_1}\right) + \left(I_{ADJ} \times R_2\right) \quad (2.1)$$

де R_1 – 470 *Ом;*

V_{REF} – 1.25B.

I_{ADJ} зазвичай становить 50 *мкА*, у більшості застосувань він незначний, оскільки вихідна напруга відома – 3.3B, розраховується опір резистора R_2, з формули (2.1) виводимо розрахунок опору резистора R_2 в формулу (2.2).

$$R_2 = \frac{V_{OUT}}{V_{REF}} \times R_1 - R_1 \quad (2.2)$$

Розрахунковий опір резистора R_2 – 770.7 *Ом*, звідки виходить, що для R_2 потрібно встановити резистор з опором 820 *Ом*. Створюємо схему для під'єднання живлення до плати та реалізуємо лінійний регулятор для живлення мікросхем (рис. 2.7).

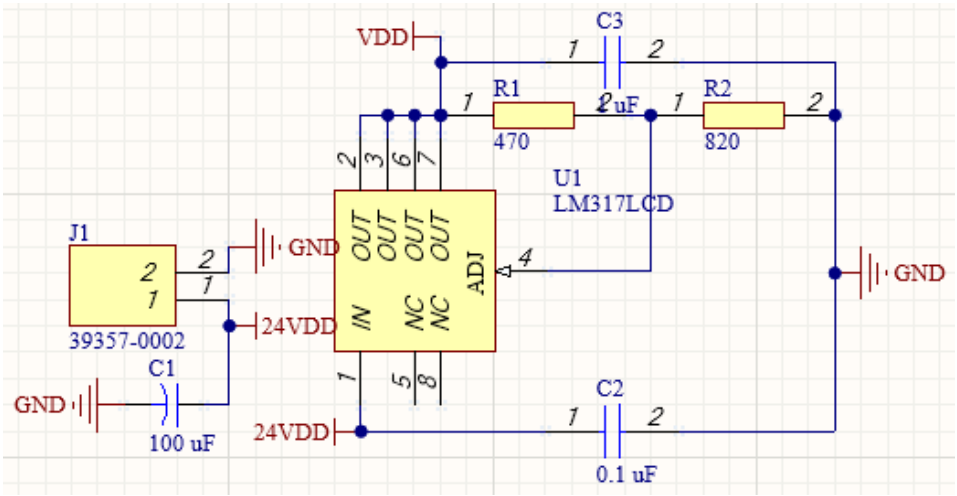

Рис. 2.7. Схема регулятора живлення

Далі додаємо кварцевий резонатор, під'єднуємо живлення та робимо фільтрацію живлення за допомогою керамічних конденсаторів, додаємо роз'єм *USB* та під'єднуємо його до мікроконтролера (рис. 2.8).

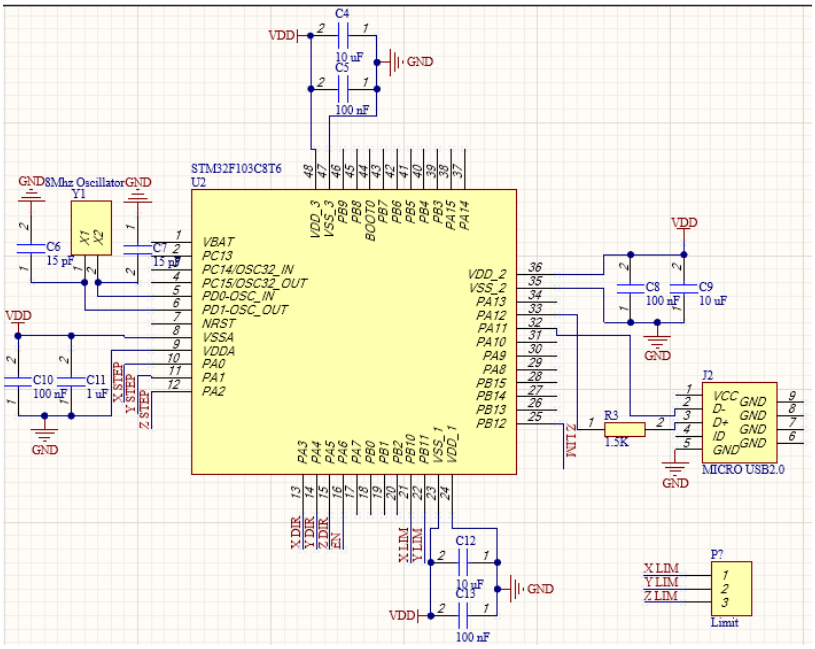

Рис. 2.8. Обв'язка та *USB* мікроконтролера

Створюємо схеми для трьох драйверів крокових двигунів, підводимо живлення та робимо фільтрацію, робимо з'єднання з клемними терміналами для під'єднання крокових двигунів (рис. 2.9).

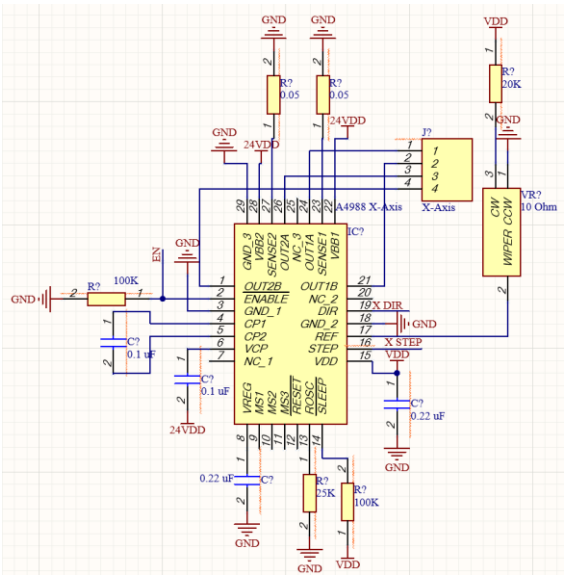

Рис. 2.9. Схема драйверу

Повторюємо схему драйверу ще три рази та з’єднуємо драйвери з мікроконтролером (рис. 2.10).

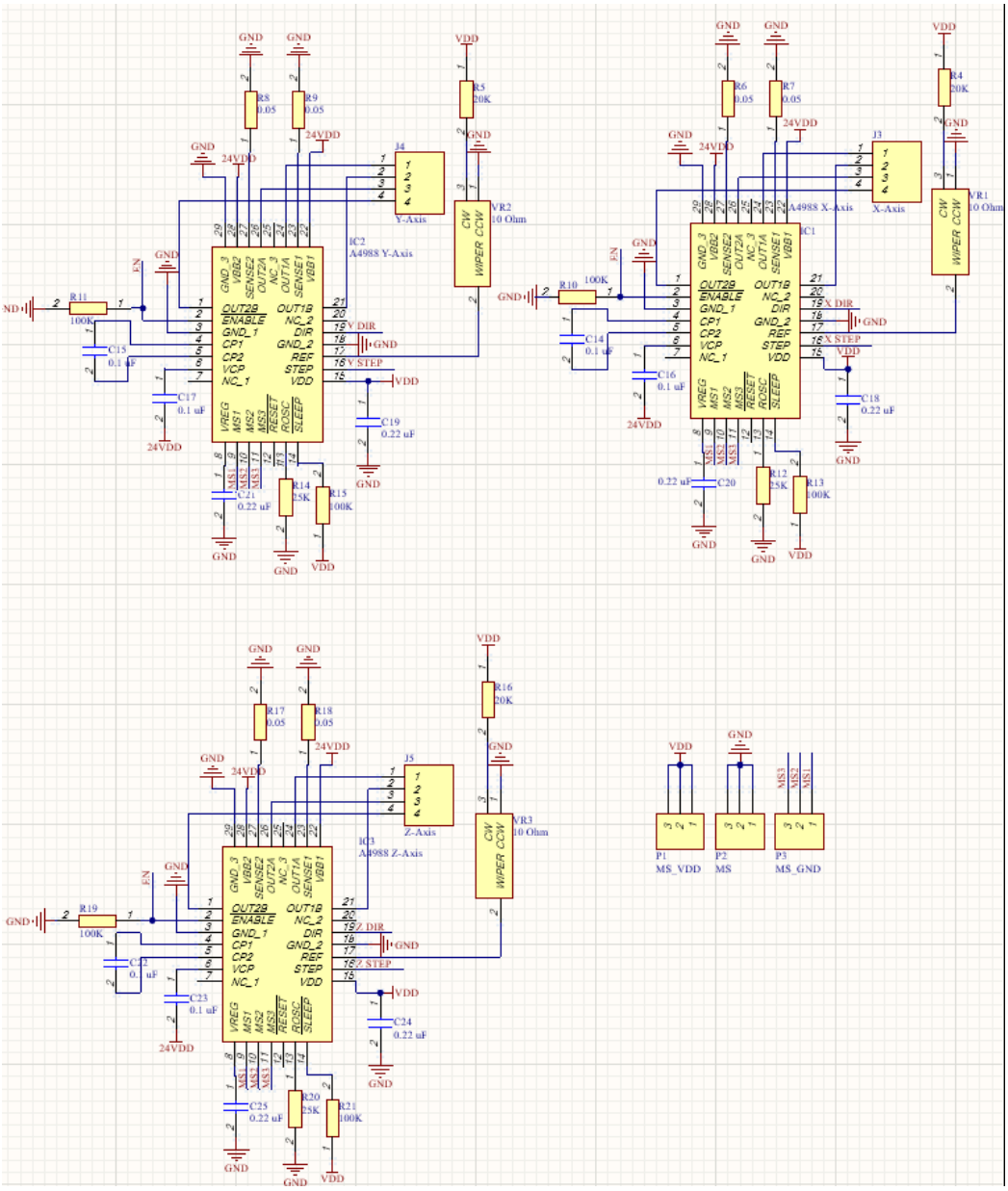

Рис. 2.10. Загальна схема драйверів

Після чого створену принципову схему потрібно імпортувати в файл друкованої плати та розмістити компоненти на платі (рис. 2.11).

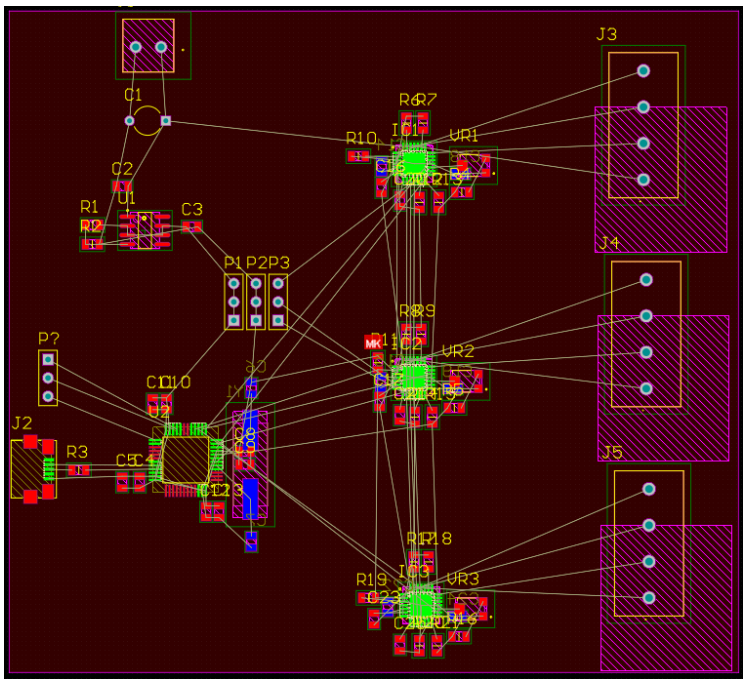

Рис. 2.11. Розміщення компонентів

Далі робимо з’єднання компонентів за допомогою друкованих доріжок та додаємо земляний полігон для кращої електропровідності та створення вбудованого екрану, який зменшує рівень шумів та наводок. [12] Готова друкована плата зображена на рис. 2.12.

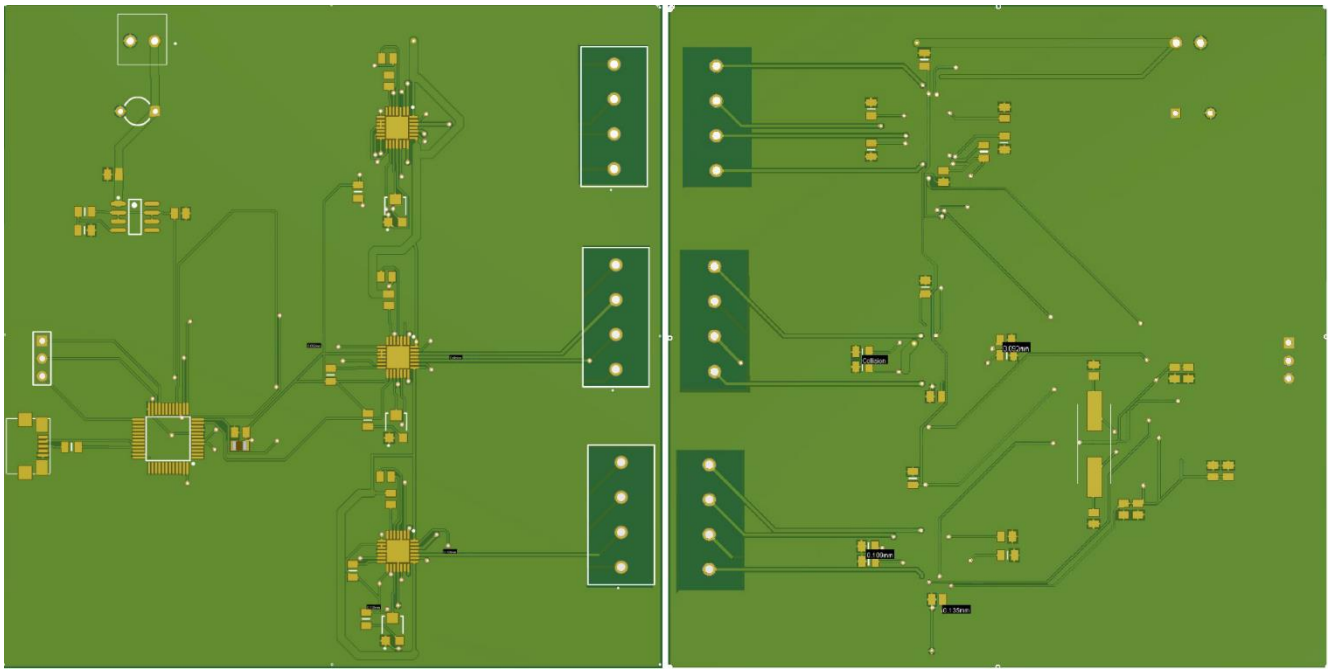

Рис. 2.11. Друкована плата

2.5. Висновки до розділу

Сформовані наступні функціональні та структурні вимоги до підсистеми:

– з’єднання з *CAM* системою через *USB*;

– керування кроковими двигунами в вісях *X*, *Y*, *Z* за допомогою модулів дешифрування та перетворення керуючих команд;

– опитування датчиків та модуль зворотного зв’язку.

Описано структуру системи відтворення креслень та підсистема керування виконуючими пристроями.

Розроблено та описано наступні схеми та діаграми:

– структурна схема системи відтворення креслень;

– діаграма послідовності підсистеми керування виконуючими пристроями;

– схема алгоритму підсистеми керування виконуючими пристроями.

Описані наступні інструментальні засоби розробки:

– *Visual Studio Code*;

– *STM32CubeMX*;

– *STM32F103*.

Розроблено принципову схему на основі мікроконтролера *STM32F103T8C6* та з використанням драйверів для керування кроковими двигунами *Allegro A4988*, на основі принципової схеми розроблено друковану плату.

РОЗДІЛ 3
ПРОГРАМА ПІДСИСТЕМИ ЧИСЛОВОГО ПРОГРАМНОГО КЕРУВАННЯ

3.1. Обґрунтування вибору мови програмування

Нижче наведено кілька факторів, які враховано при виборі мови програмування для розробки вбудованих систем.

– розмір: пам'ять, яку займає програма, дуже важлива, оскільки вбудовані процесори, такі як мікроконтролери, мають дуже обмежену кількість *ROM* (програмної пам'яті);

– швидкість: програми повинні бути дуже швидкими, тобто вони повинні працювати якомога швидше. Апаратне забезпечення не повинно сповільнюватися через повільну роботу програмного забезпечення;

– переносимість: одна і та ж програма може бути складена для різних процесорів;

– простота впровадження;

– простота обслуговування;

– легкість читання.

Архітектура *ARM*, як і більшість 32-розрядних архітектур, добре підходить для використання компілятора *C* або *C++*. Більшість контрольних кодів написані з використанням мов програмування високого рівня, таких як *C* та *C++*, замість мови асемблера. На це є вагомі причини. Мови програмування високого рівня за своєю суттю безпечніші та менш схильні до помилок, ніж програмування у збірці. Код, написаний мовами програмування високого рівня, також може бути написаний для перенесення в різні архітектури.

Мова *C* є однією з найпотужніших мов комп'ютерного програмування за всю історію, *C* є фактичним вибором для програмування вбудованих систем. Низьке використання пам’яті складеної програми на *C* також є критерієм прийняття C на найкращу мову програмування вбудованих систем.

Крім того, ця статично типізована мова також пропонує швидкість, яка не досягається на жодній іншій мові програмування.

– *C* надає доступ навіть до самих низькорівневих системних компонентів через вбудований покажчик;

– розробники можуть створити компілятори *C* для своїх вбудованих систем досить швидко, завдяки його широкій популярності;

– *C* поєднує в собі низькорівневу функціональність мови асемблеру дуже акуратно із сучасними умовами програмування;

– політика слабкої типізації даних *C* робить його надзвичайно придатним для програмування вбудованих систем;

– переносити вбудовані програми на різні пристрої набагато простіше, ніж програми, написані іншими мовами.

C++ – це об'єктно-орієнтована мова програмування заснована на *C*, *C++* стає все більш популярною серед вбудованих програмістів. Всі з основних функцій мови такі ж, як *C*, але *C++* додає нову функціональність для кращої абстракції даних та більш об’єктно-орієнтований стиль програмування. Ці нові функції дуже корисні розробники програмного забезпечення, але деякі з них знижують ефективність виконуваної програми. Отже, *C++* має тенденцію бути найбільш популярною мовою серед великих команд розробників, де вигоди для розробників перевищують втрати ефективності програми.

Багато функцій *C++*, такі як класи, автоматичне очищення ресурсів, параметричний поліморфізм та додаткова безпека типу, настільки ж корисні для *RTOS* або безпосередньо на апаратному забезпеченні, як і на робочому столі з універсальною операційною системою.

Однак «авто» магія *C++* – це палиця з двома кінцями. Деякі мовні функції залежать від системних функцій, які не підходять для вбудованих середовищ.

Налаштування ланцюга інструментів часто є складним. *libgcc* та *libstdc++* не слід повністю відкидати, оскільки вони забезпечують такі важливі функції, як *memcpy*, атомарні операції та функціональна плаваюча точка; однак певних його ділянок слід уникати.

Різниця між *C* та *C ++* полягає, як правило, у тому, що мова є процедурною, призначеною для використання в системному програмуванні, і є більш «легкою» (вимагає менше пам'яті), тоді як *C++* є більш загальною та об'єктно-орієнтованою.

Мови асемблеру були розроблені для надання мнемоніки або символів для інструкцій коду машинного рівня. Програми асемблерної мови складаються з мнемонік, тому їх слід перекласти у машинний код. Програма, яка відповідає за це перетворення, відома як асемблер. Мову асемблеру часто називають мовою низького рівня, оскільки вона безпосередньо працює з внутрішньою структурою центрального процесора. Щоб програмувати мовою асемблера, програміст повинен знати всі регістри ЦП.

Сьогодні насправді мало причин використовувати мову асемблеру для цілих проектів, оскільки якісні оптимізаційні компілятори для мов високого рівня (особливо *C* та *C++*) легко доступні як безкоштовне програмне забезпечення з відкритим кодом і тому, що архітектура *ARM* спеціально оптимізована для мов високого рівня. Однак знання мови асемблеру все ще корисні для налагодження певних проблем, написання програм низького рівня, таких як завантажувачі та ядра операційної системи, та зворотньої розробки програм, для яких немає вихідного коду. Іноді доводиться вручну оптимізувати деякий розділ коду критичний до продуктивності.

На основі даних обґрунтувань було обрано мову програмування *C*.

3.2. Кроки створення програми

Розробка більшості проектів для мікроконтролерів *STM32* на основі архітектури *ARM* починається з ініціалізатора коду *STM32QubeMX*. В

STM32QubeMX обирається мікроконтролер, на основі якого буде проводитись розробка, в даному випадку буде використовуватись *STM32F103C8T6*, після вибору мікроконтролеру створюється проект і з'являється вікно налаштування ніжок мікроконтролера. В даному меню потрібно налаштувати інтерфейс *USB* та обрати ніжки для вводу датчиків та ніжки для керування кроковими двигунами. Ніжки *GPIOB* 10-12 будуть використовуватись для вводу датчиків кінця руху тому їх потрібно налаштувати як *GPIO_Input*, *GPIOA6* налаштовується на вихід для вимкнення крокових двигунів, *GPIOA* 3-5 налаштовуються на вихід для керування напрямом крокових двигунів, *GPIOA* 0-2 налаштовуються на вихід для команди руху крокових двигунів та встановити на ніжки *PD* 0-1 зовнішній резонатор (рис. 3.1).

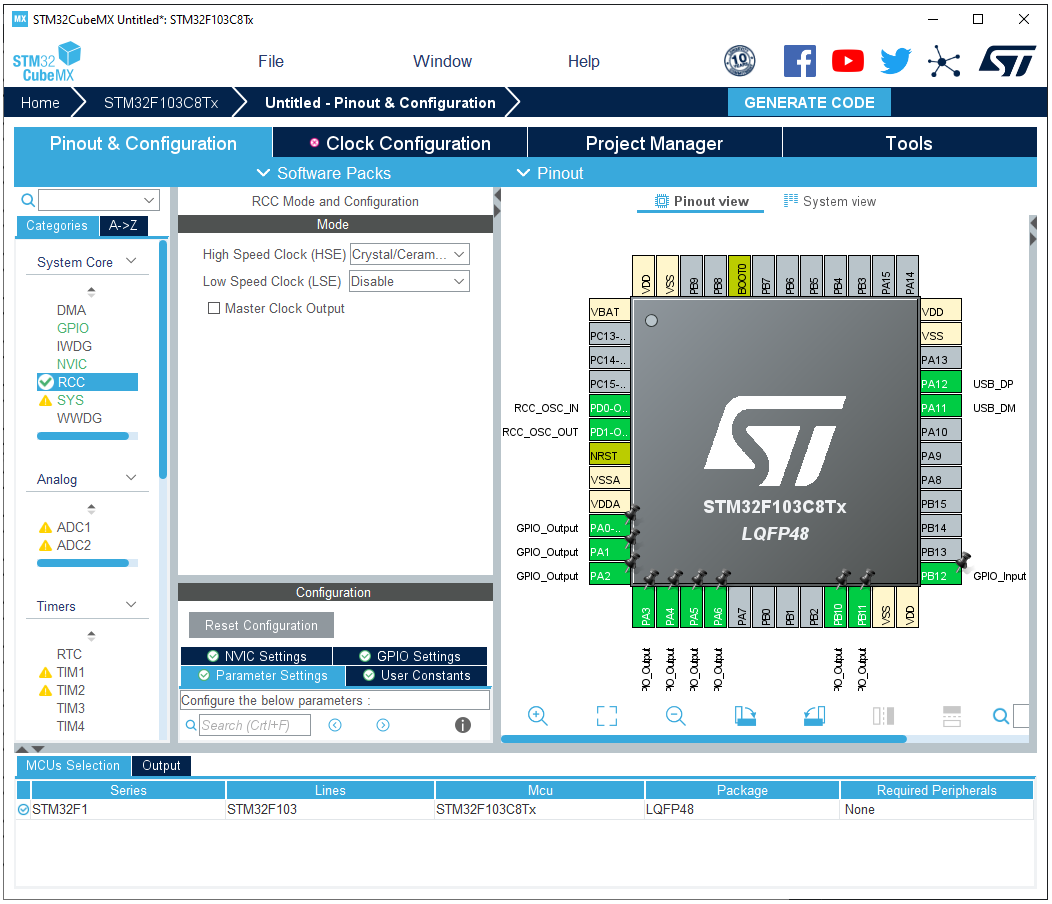

Рис. 3.1. Налаштування ніжок мікроконтролера

Після чого потрібно налаштувати тактові генератори та множники мікроконтролера. Так як для мікроконтролера використовується зовнішній

кварцовий резонатор, то потрібно встановити *HSE* на частоту 8 *МГц*, далі потрібно досягти частоти 48 *МГц* для універсальної шини, оптимальними налаштуваннями будуть мультиплікатор *PLL* на 8 та дільник *USB* встановлений на 1.5. Далі налаштовуємо периферійні шини, вмикаємо *CSS* та перемикаємо джерело тактування на *PLLCLK* та для того щоб частота шини *APB1* не виходила за рамки дозволеного встановлюємо дільник *APB1* на 2 (рис. 3.2).

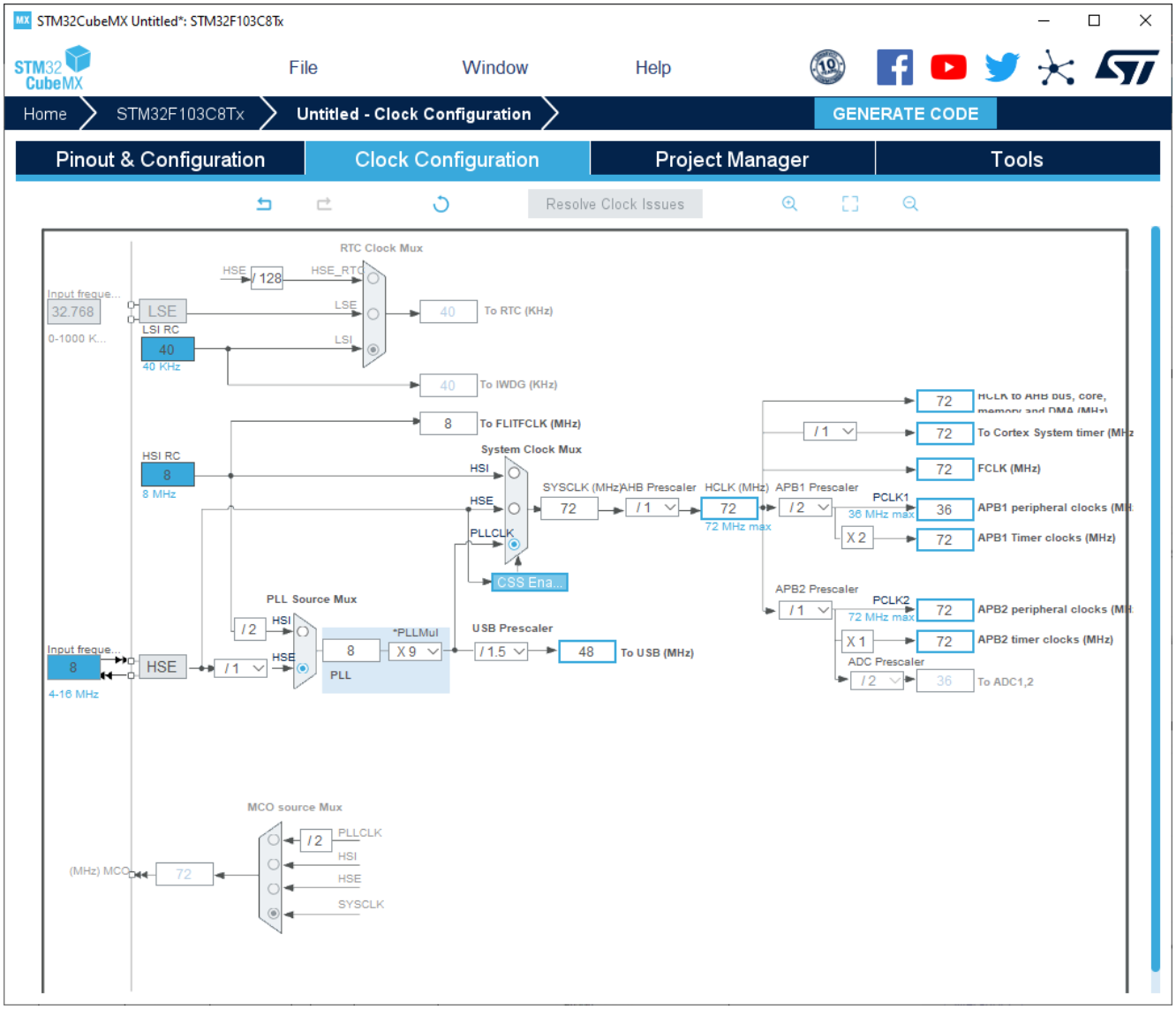

Рис. 3.2. Налаштування тактування мікроконтролера

Після зроблених налаштувань потрібно згенерувати код та відкрити його в середовищі розробки.

Перш за все створимо заголовний файл *cpu_map.h* в якому розмістимо директиви препроцесора для більш зручного звернення до ніжок мікроконтролера та макроси:

```
#define STEP_PORT       GPIOA
#define RCC_STEP_PORT   RCC_APB2Periph_GPIOA
#define X_STEP_BIT      0
#define Y_STEP_BIT      1
#define Z_STEP_BIT      2
#define STEP_MASK ((1<<X_STEP_BIT) | (1<<Y_STEP_BIT) | (1<<Z_STEP_BIT))
#define DIRECTION_PORT      GPIOA
#define RCC_DIRECTION_PORT   RCC_APB2Periph_GPIOA
#define X_DIRECTION_BIT   3
#define Y_DIRECTION_BIT   4
#define Z_DIRECTION_BIT   5
#define DIRECTION_MASK ((1<<X_DIRECTION_BIT) | (1<<Y_DIRECTION_BIT) | (1<<Z_DIRECTION_BIT))
#define STEPPERS_DISABLE_PORT   GPIOA
#define RCC_STEPPERS_DISABLE_PORT RCC_APB2Periph_GPIOA
#define STEPPERS_DISABLE_BIT    6
#define STEPPERS_DISABLE_MASK (1<<STEPPERS_DISABLE_BIT)
#define SetStepperDisableBit() GPIO_SetBits(STEPPERS_DISABLE_PORT,STEPPERS_DISABLE_MASK)
#define ResetStepperDisableBit() GPIO_ResetBits(STEPPERS_DISABLE_PORT,STEPPERS_DISABLE_MASK)
#define LIMIT_PIN       GPIOB
#define LIMIT_PORT      GPIOB
#define RCC_LIMIT_PORT   RCC_APB2Periph_GPIOB
#define GPIO_LIMIT_PORT  GPIO_PortSourceGPIOB
#define X_LIMIT_BIT     10
```

#define Y_LIMIT_BIT 11

#define Z_LIMIT_BIT 12

Створюємо заголовний файл *cnc.h* в якому будуть розміщуватись всі імпортовані файли проекту та деякі константи. Створюємо файл *defaults.h* в якому будуть розміщуватись стандартні налаштування для ЧПК:

#define X_MICROSTEPS 16

#define Y_MICROSTEPS 16

#define Z_MICROSTEPS 8

#define STEPS_PER_REV 200.0f

#define MM_PER_REV 4.0

*#define DEFAULT_X_STEPS_PER_MM (STEPS_PER_REV * X_MICROSTEPS / MM_PER_REV)*

*#define DEFAULT_Y_STEPS_PER_MM (STEPS_PER_REV * Y_MICROSTEPS / MM_PER_REV)*

*#define DEFAULT_Z_STEPS_PER_MM (STEPS_PER_REV * Z_MICROSTEPS / MM_PER_REV)*

#define DEFAULT_X_MAX_RATE 3000.0f

#define DEFAULT_Y_MAX_RATE 3000.0f

#define DEFAULT_Z_MAX_RATE 500.0f

*#define DEFAULT_X_ACCELERATION (50.0f * 60 * 60)*

*#define DEFAULT_Y_ACCELERATION (50.0f * 60 * 60)*

*#define DEFAULT_Z_ACCELERATION (10.0f * 60 * 60)*

#define DEFAULT_X_MAX_TRAVEL 280.0f

#define DEFAULT_Y_MAX_TRAVEL 377.0f

#define DEFAULT_Z_MAX_TRAVEL 53.0f

#define DEFAULT_STEP_PULSE_MICROSECONDS 4

#define DEFAULT_STEPPER_IDLE_LOCK_TIME 25

Далі створюємо файл *main.c* в якому буде розміщуватись функція для точки входу *main* з нескінченним циклом та функція затримки:

int main(void){

```
        Set_USBClock();
        USB_Interrupts_Config();
        USB_Init();
        SysTick->CTRL &= 0xfffffffb;
   serial_init();
  settings_init();
  stepper_init();
  system_init();
  memset(sys_position,0,sizeof(sys_position));
for(;;) {
    uint8_t prior_state = sys.state;
    memset(&sys, 0, sizeof(system_t));
    sys.state = prior_state;
    sys.f_override = DEFAULT_FEED_OVERRIDE;
    sys.r_override = DEFAULT_RAPID_OVERRIDE;
    sys_rt_exec_state = 0;
    sys_rt_exec_alarm = 0;
    sys_rt_exec_motion_override = 0;
    sys_rt_exec_accessory_override = 0;
    serial_reset_read_buffer();
    gc_init();
    limits_init();
    plan_reset();
    st_reset();
    plan_sync_position();
    gc_sync_position();
    protocol_main_loop();
  }
  return 0;
}
```

```
void _delay_ms(uint32_t x){
  u32 temp;
  SysTick->LOAD = (u32)72000000 / 8000 * x;
  SysTick->VAL = 0x00;
  SysTick->CTRL = 0x01;
  do{
    temp = SysTick->CTRL;
  } while ((temp & 0x01) && !(temp&(1 << 16)));
  SysTick->CTRL = 0x00;
  SysTick->VAL = 0X00;
}
```

Далі створюємо файл *stepper.c* який буде відповідати за крокові двигунами:

```
typedef struct {
  uint32_t counter_x,
           counter_y,
           counter_z;
  uint8_t execute_step;
  uint16_t step_pulse_time;
  PORTPINDEF step_outbits;
  PORTPINDEF dir_outbits;
  uint16_t step_count;
  uint8_t exec_block_index;
  st_block_t *exec_block;
  segment_t *exec_segment;
} stepper_t;
static stepper_t st;
```

Дешифрування *G*-кодів буде виконуватись в файлі *gcode.c*, повинні підтримуватись наступні команди:

Функціональні команди: *G4*, *G10L2*, *G10L20*, *G28*, *G30*, *G28.1*, *G30.1*, *G53*, *G92*, *G92.1*.

Режими руху: *G0*, *G1*, *G2*, *G3*, *G38.2*, *G38.3*, *G38.4*, *G38.5*, *G80*.

Режими системи координат: *G54*, *G55*, *G56*, *G57*, *G58*, *G59*.

Режими подачі: *G93*, *G94*.

Координатні режими: *G90*, *G91*.

Потік програми: *M0*, *M1*, *M2*, *M30*.

Розшифрування керуючих команд виконується подібним чином:

```
case 4: case 53:
        word_bit = MODAL_GROUP_G0;
        gc_block.non_modal_command = int_value;
        if ((int_value == 28) || (int_value == 30) || (int_value == 92)) {
          if (!((mantissa == 0) || (mantissa == 10))) { FAIL(STATUS_GCODE_UNSUPPORTED_COMMAND); }
          gc_block.non_modal_command += mantissa;
          mantissa = 0;
        }
        break;
case 80:
        word_bit = MODAL_GROUP_G1;
        gc_block.modal.motion = int_value;
        if (int_value == 38){
          if (!((mantissa == 20) || (mantissa == 30) || (mantissa == 40) || (mantissa == 50))) {
            FAIL(STATUS_GCODE_UNSUPPORTED_COMMAND);
          }
          gc_block.modal.motion += (mantissa/10)+100;
          mantissa = 0;
        }
        break;
```

Для зчитування керуючих команд від комп'ютера використовується універсальна послідовна шина, а функція *serial_read* здійснює зчитування інформації з порту:

```
uint8_t serial_read(){
  uint8_t tail = serial_rx_buffer_tail;
  if (serial_rx_buffer_head == tail) {
    return SERIAL_NO_DATA;
  } else {
    uint8_t data = serial_rx_buffer[tail];
    tail++;
    if (tail == RX_RING_BUFFER) { tail = 0; }
    serial_rx_buffer_tail = tail;
    return data;
  }}
```

Відповідно для відправки даних зворотного зв'язку також використовується універсальна послідовна шина, а дані на порт записує наступна функція:

```
void serial_write(uint8_t data) {
  uint8_t next_head = serial_tx_buffer_head + 1;
  if (next_head == TX_RING_BUFFER) { next_head = 0; }
  while (next_head == serial_tx_buffer_tail) {
    if (sys_rt_exec_state & EXEC_RESET) { return; }
  }
  serial_tx_buffer[serial_tx_buffer_head] = data;
#ifdef USB_CHANGED_DEBUG
  while(txUsbLock) {
  if (sys_rt_exec_state & EXEC_RESET) { return; }
  }
#endif
  serial_tx_buffer_head = next_head;
}
```

3.3. Інструкція користувача

Для прошивання мікропрограми потрібно мати плату на основі мікроконтролера *STM32F103C8*, зображено на рис. 3.3.

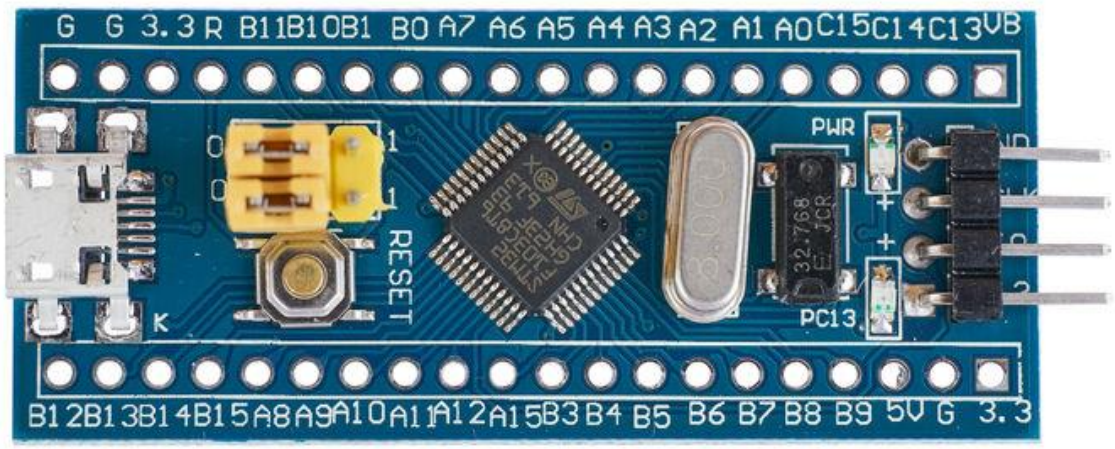

Рис. 3.3. Плата «*blue pill*»

Та для завантаження мікропрограми з комп'ютера на мікроконтролер використовується програматор *ST-LINK V2*, зображено на рис. 3.4.

Рис. 3.4. Програматор *ST-LINK V2*

Далі потрібно з’єднати програматор з платою мікроконтролера(рис. 3.5) для цього потрібно з’єднати наступні контакти:

1) Контакт програматора 2(*SWCLK*) з контактом 2(*SWCLK*) на платі мікроконтролера.

2) Контакт програматора 4(*SWDIO*) з контактом 3(*SWIO*) на платі мікроконтролера.

3) Контакт програматора 6(*GND*) з контактом 1(*GND*) на платі мікроконтролера.

4) Контакт програматора 8(*3.3V*) з контактом 4(*3.3V*) на платі мікроконтролера.

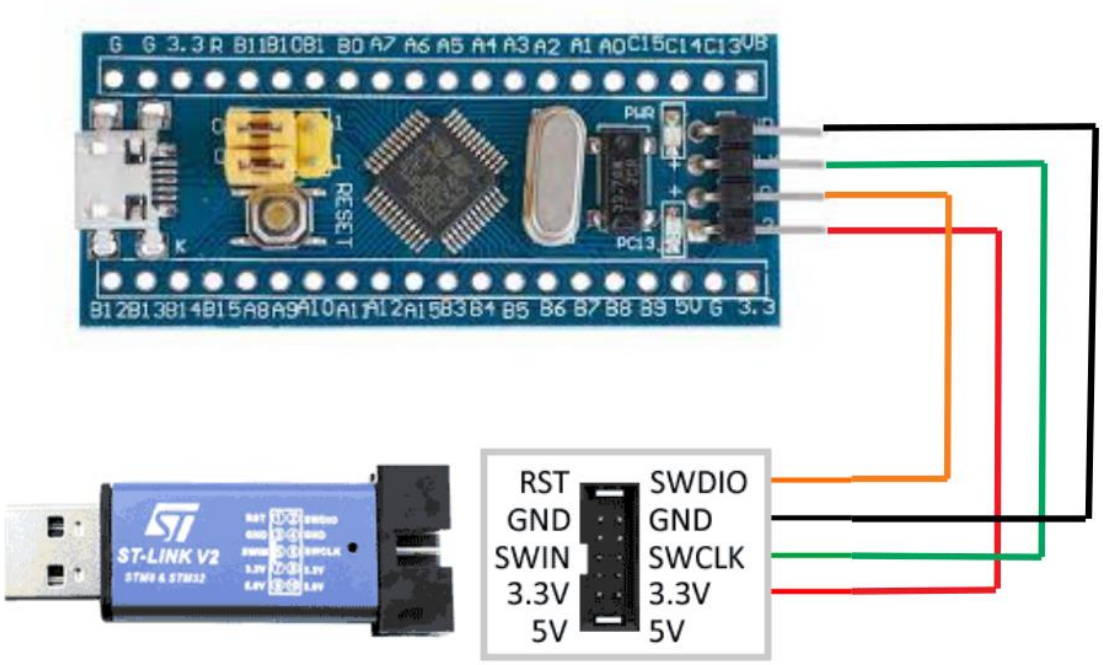

Рис. 3.5. Підключення програматора

Далі завантажуємо утиліту для програматора *STM32CubeProg* з офіційного сайту та встановлюємо її, та драйвери, які ідуть в комплекті з утилітою. [9]

Далі потрібно перевести плату в режим програмування, для цього потрібно контакт *BOOT0* підтягнути до *3.3V*, як зображено на рис. 3.6.

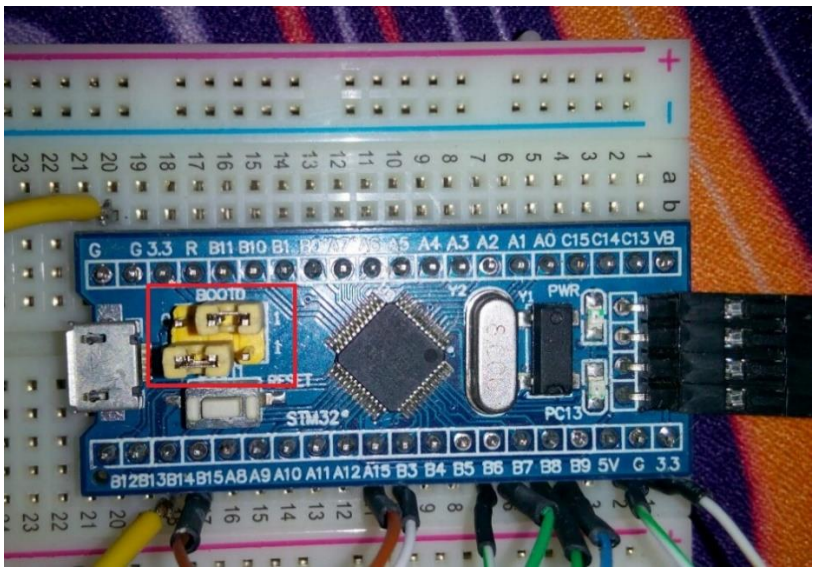

Рис. 3.6. Переведення в режим програмування

Після чого програматор потрібно підключити до комп'ютера, після під'єднання програматора до *USB* порту в диспетчері пристроїв повинен відобразитись програматор (рис. 3.7).

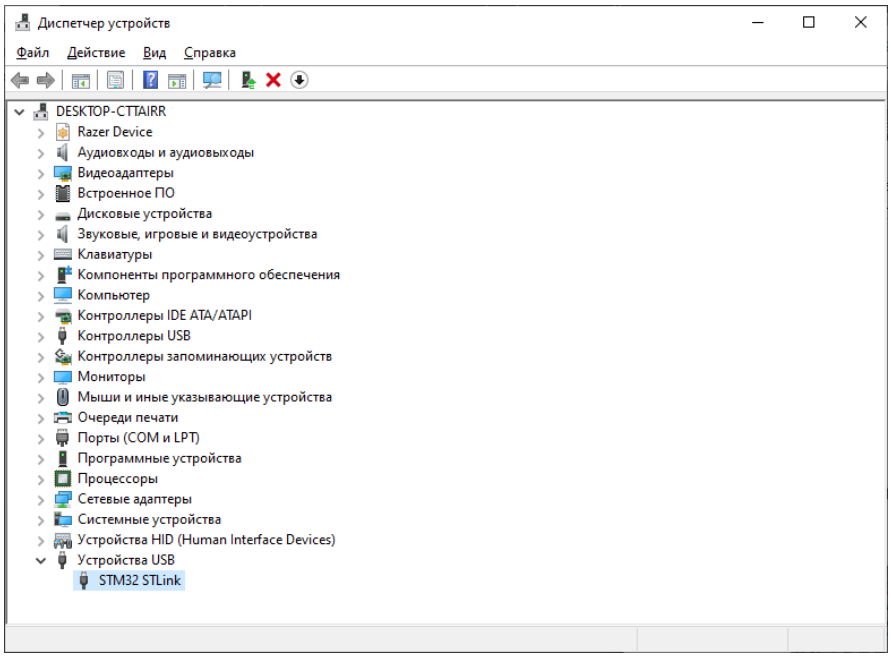

Рис. 3.7. Перевірка підключення програматора

Після чого потрібно запустити *STM32CubeProgrammer*, в правій частині вікна повинен відобразитись зелений індикатор і повідомлення, що

мікроконтролер під’єднано, також в правій нижній частині вікна відображено інформацію про мікроконтролер (рис. 3.8).

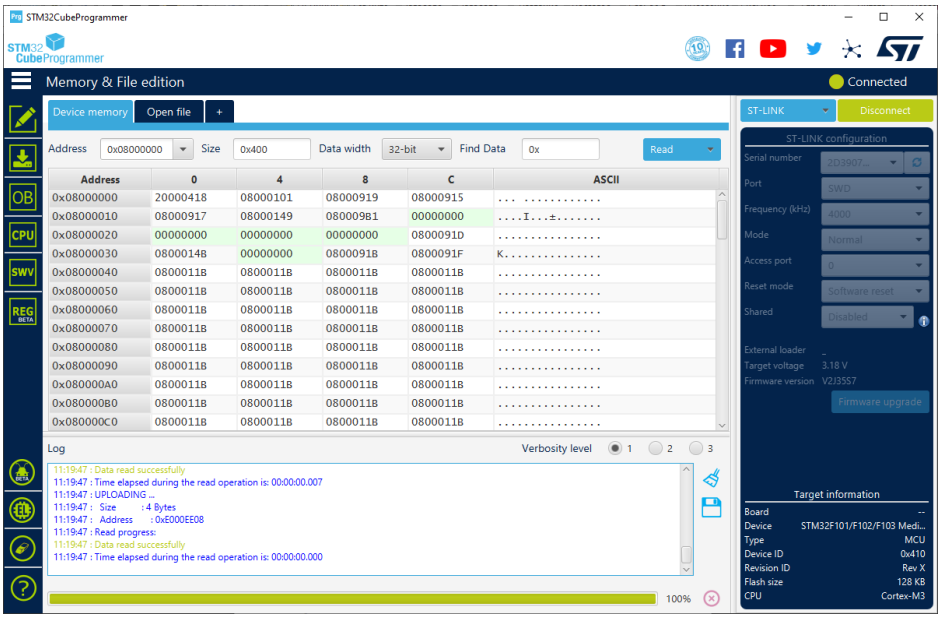

Рис. 3.8. *STM32CubeProgrammer*

Далі потрібно натиснути на вкладку «*Open file*» та вказати *hex*-файл з прошивкою та натиснути кнопку «*Download*» після чого почнеться завантаження прошивки до мікроконтролеру, по закінченню завантаження повинно відобразитись сповіщення про закінчення завантаження (рис. 3.9).

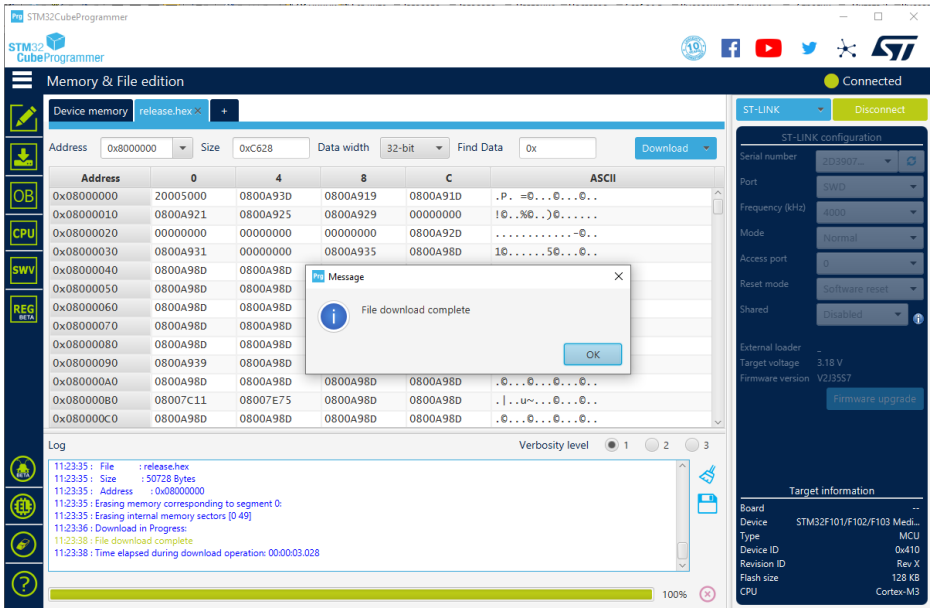

Рис. 3.9. Успішне завантаження

Тепер потрібно від’єднати програматор від комп’ютера та мікроконтролера та перевести контакт *BOOT0* назад в положення 0 (рис. 3.10).

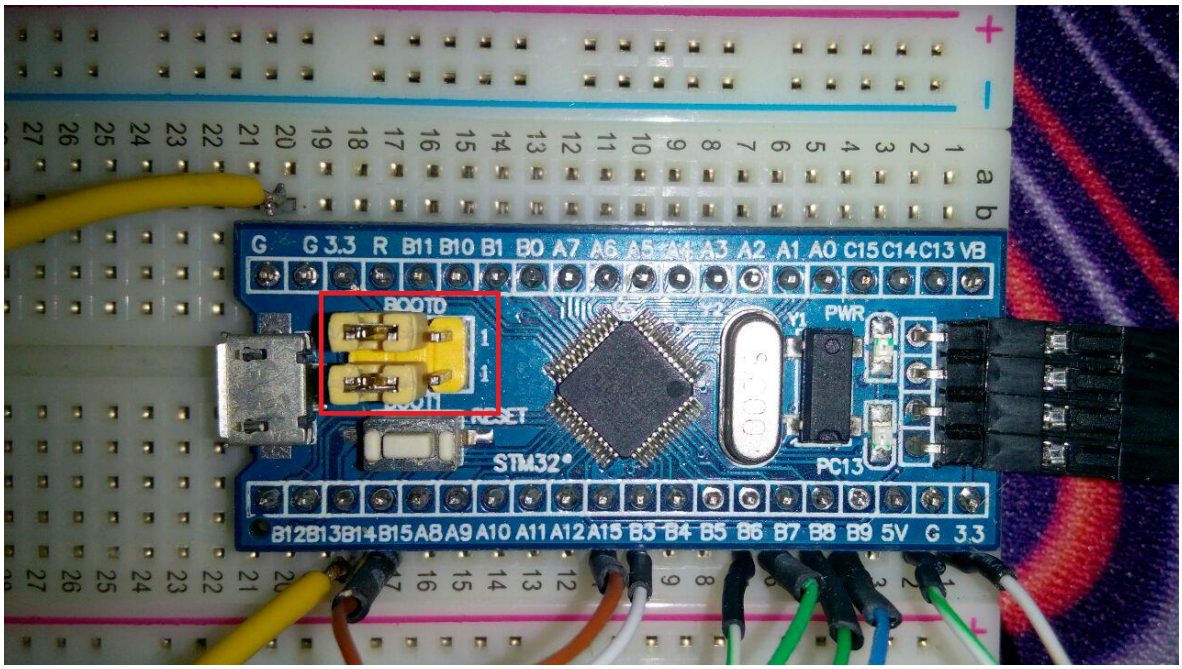

Рис. 3.10. Робочий режим

Після чого мікроконтролер готовий до роботи.

3.4. Тестування

Вимоги до додатку, що будуть тестуватись:

1) з’єднання мікроконтролера з комп’ютером;
2) отримання команд через *USB*;
3) відправка даних через *USB*;
4) дешифрування команд;
5) відправка команди на драйвер;
6) виконання команди драйвером;
7) зчитування кінцевих датчиків.

Методи тестування.

За ступенем автоматизації – ручне. Тестувальник має відігравати роль користувача програми й використовувати властивості програми для знаходження помилок у роботі програми.

За знанням системи – *white box*. Тестувальник має доступ до коду програм і може писати код, який пов’язаний з бібліотеками програмного забезпечення, що тестується.

За критерієм запуску програми – динамічне. Динамічні методи застосовуються в процесі безпосереднього виконання програми. Коректність програмного засобу перевіряється на безлічі тестів або наборів підготовлених вхідних даних. При прогоні кожного тесту збираються та аналізуються дані про відмови та збої в роботі програми.

По об’єкту тестування – функціональне:

а) *USB*:

1) з’єднання з комп’ютером;

2) отримання даних через *USB*;

3) відправка даних через *USB*.

б) *G*-коди:

1) розшифрування кодів;

2) генерація команд на основі кодів.

в) датчики:

1) зчитування інформації з кінцевих датчиків.

г) драйвери:

1) передача команд на драйвери;

2) виконання команди драйвером.

Вимоги до середовища тестування. Тестування додатку відбувалось на платі «*blue pill*» з мікроконтролером *STM32F103C8T6*.

Необхідні ресурси. Для виконання поставлених навчальних завдань необхідно мати наступні знання та вміння:

Знання і вміння застосування на практиці стандарту *IEEE-829*.

Знання різних типів тестування в тому числі функціонального.

Тестові випадки:

Для тестування з’єднання універсальної послідовної шини плата під’єднується до комп’ютера та виконується передача даних (табл. 3.1).

Таблиця 3.1

Тестування *USB*

Дія	Очікуваний результат	Результат тесту
Під’єднати мікроконтролер	Мікроконтролер під’єднано.	Пройдено
Очікувати комутації *USB* з комп’ютером.	На комп’ютері відобразиться новий пристрій.	Пройдено
Відправити код з комп’ютера.	Мікроконтролер отримав дані.	Пройдено
Відправити дані з мікроконтролера.	Мікроконтролер відправив дані і вони надійшли через *USB*.	Пройдено.
Від’єднати.	Мікроконтролер від’єднано і пристрій зник з ОС комп’ютера.	Пройдено

Мікроконтролер успішно під’єднався до комп’ютера та виконав операції відправки/отримання даних, отже тест пройшов успішно.

Для тестування модуля обробки *G*-кодів, на контролер відправляється тестова керуюча команда (табл. 3.2).

Таблиця 3.2

Тестування *G*-кодів

Дія	Очікуваний результат	Результат тесту
Під’єднати мікроконтролер	Мікроконтролер під’єднано.	Пройдено
Очікувати комутації *USB* з комп’ютером.	На ПК відобразиться новий пристрій.	Пройдено
Відправити код з комп’ютера.	Мікроконтролер отримав дані.	Пройдено
Дешифрувати код.	МК дешифрував *G*-код в команду для драйверу.	Пройдено.
Від’єднати.	Мікроконтролер від’єднано і пристрій зник з ОС комп’ютера.	Пройдено

Тест пройшов успішно, так як програма отримала керуючу команду та дешифрувала її.

Для тестування системи моніторингу та зворотного зв’язку до контролера під’єднується датчик кінця руху та виконується його спрацювання (табл. 3.3).

Таблиця 3.3

Тестування датчиків

Дія	Очікуваний результат	Результат тесту
Під’єднати МК	МК під’єднано.	Пройдено
Очікувати комутації *USB* з комп’ютером.	На комп’ютері відобразиться новий пристрій.	Пройдено

Закінчення таблиці 3.3

Натиснути на кінцевий датчик.	Спрацює зчитування датчика і дані про кінець шляху відправляться на комп’ютер.	Пройдено
Вийти.	Вихід здійснюється.	Пройдено

Тест пройдено успішно, датчик опрацьовано контролером та відправлено повідомлення про кінець шляху.

Для тестування модуля керування кроковими двигунами відправляється тестова команда на переміщення крокового двигуна і очікується його виконання (табл. 3.4).

Таблиця 3.4

Тестування драйверів

Дія	Очікуваний результат	Результат тесту
Під’єднати МК	МК під’єднано.	Пройдено
Очікувати комутації *USB* з комп’ютером.	На ПК відобразиться новий пристрій.	Пройдено
Відправити код з ПК.	Мікроконтролер отримав дані.	Пройдено
Дешифрувати код.	МК дешифрував *G*-код в команду для драйверу.	Пройдено.
Відправити команду на драйвер.	Команда отримана драйвером.	Пройдено
Виконання команди	Двигун виконав команду	Пройдено
Від’єднати.	МК від’єднано і пристрій зник з ОС комп’ютера.	Пройдено

Тестування модуля керування кроковими двигунами пройдено успішно, так як контролер опрацював команду і кроковий двигун виконав задану кількість кроків. Всі модулі пройшли тестування, отже програмний модуль є правильно функціонуючим.

3.5. Висновки до розділу

Проведено обґрунтування вибору мови програмування серед найбільш придатних мов для програмування мікроконтролерів *STM32* на основі архітектури *ARM*, а саме *C*, *C++* та мови асемблеру, за наступними критеріями: пам’ять яку займає скомпільована програма, швидкість роботи програми, переносимість, простота впровадження, простота обслуговування, легкість читання. На основі цього обґрунтування обрано мову програмування *C*.

Проведена ініціалізація проекту та конфігурацію тактових генераторів для системи та периферії, налаштовано порти вводу-виводу загального призначення, напрями та режими, налаштована універсальна послідовна шина та налаштовані таймери за допомогою *STM32CubeMX*. Проведена покрокова розробка підсистеми керування виконуючими пристроями.

Створено керівництво користувача для підготовки плати та прошивання програми в мікроконтролер.

Успішно проведено наступні етапи тестування:

1) з’єднання мікроконтролера з комп’ютером;

2) отримання команд через *USB*;

3) відправка даних через *USB*;

4) дешифрування команд;

5) відправка команди на драйвер;

6) виконання команди драйвером;

7) зчитування кінцевих датчиків.

ВИСНОВКИ

Результатом дипломного проекту була реалізація програмного забезпечення для автоматизованого виготовлення виробів за допомогою верстатів з числовим програмним керуванням, а саме:

– реалізовано стандартизований протокол з’єднання та передачі ЧПК-верстату з комп’ютером;

– створено модулі для обробки, обробки та зворотного зв’язку керуючих програм.

Розглянуто основні поняття та терміни, які використовуються в системах з числовим програмним керуванням.

Проаналізовано сучасні програмно-апаратні рішення в області проектування систем з числовим програмним керуванням та виявлено основні особливості до структури побудови систем числового програмного керування.

Розглянуто особливості структури систем числового програмного керування, розглянуті задачі виконувані контролером, розглянуто процес розробки керуючих програм, класифіковано типи виконавчих органів системи числового програмного керування та проведено їх огляд.

Розглянуто та порівняно мікроконтролери *ATmega328* та *STM32F103C8* для побудування системи числового програмного керування, порівняні між собою наступні характеристики:

– частота;

– розрядність;

– кількість пам’яті;

– кількість інтерфейсів вводу-виводу.

Для розкриття теми дипломного проекту виконано наступне:

1) розроблено структурну схему системи відтворення креслень;

2) розроблено схему алгоритму підсистеми керування виконуючими пристроями;

3) проведено програмну реалізацію розробленого алгоритму;

4) розроблено інструкцію користувача;

5) проведено тестування підсистеми керування виконуючими пристроями.

Сформовані наступні функціональні та структурні вимоги до підсистеми:

– з’єднання з *CAM* системою через *USB*,

– керування кроковими двигунами в вісях *X*, *Y*, *Z* за допомогою модулів дешифрування та перетворення керуючих команд,

– опитування датчиків та модуль зворотного зв’язку.

Описано структуру системи відтворення креслень та підсистема керування виконуючими пристроями.

Розроблено та описано наступні схеми та діаграми:

– структурна схема системи відтворення креслень;

– діаграма послідовності підсистеми керування виконуючими пристроями;

– схема алгоритму підсистеми керування виконуючими пристроями.

Описані наступні інструментальні засоби розробки:

– *Visual Studio Code*;

– *STM32CubeMX*;

– *STM32F103*.

Розроблено принципову схему на основі мікроконтролера *STM32F103T8C6* та з використанням драйверів для керування кроковими двигунами *Allegro A4988*, на основі принципової схеми розроблено друковану плату.

Проведено обґрунтування вибору мови програмування серед найбільш придатних мов для програмування мікроконтролерів *STM32* на основі архітектури *ARM*, а саме *C*, *C++* та мови асемблеру, за наступними критеріями: пам’ять яку займає скомпільована програма, швидкість роботи програми, переносимість, простота впровадження, простота обслуговування, легкість читання. На основі цього обґрунтування обрано мову програмування *C*.

Проведена ініціалізація проекту та конфігурацію тактових генераторів для системи та периферії, налаштовано порти вводу-виводу загального призначення, напрями та режими, налаштована універсальна послідовна шина та налаштовані

таймери за допомогою *STM32CubeMX*. Проведена покрокова розробка підсистеми керування виконуючими пристроями.

Створено керівництво користувача для підготовки плати та прошивання програми в мікроконтролер.

Успішно проведено наступні етапи тестування:

1) з’єднання мікроконтролера з комп’ютером;

2) отримання команд через *USB*;

3) відправка даних через *USB*;

4) дешифрування команд;

5) відправка команди на драйвер;

6) виконання команди драйвером

7) зчитування кінцевих датчиків.

Практична цінність полягає у можливості створення простого для користувача верстата з числовим програмним керуванням для автоматизації механічної обробки виробів з високим ступенем їх повторюваності, з використанням сучасних компонентів для елементної бази та сучасних та зручних інтерфейсів передачі даних.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНИХ ДЖЕРЕЛ

1) Ковальов В.А. Конструктивні особливості та основи програмування верстатів з числовим програмним керуванням. / В.А. Ковальов, А.Ю. Гаврушкевич, Н.В. Гаврушкевич; – КПІ ім. Ігоря Сікорського. – Київ, 2020. 158с.

2) Лозінський Д.О. Системи автоматизованого проектування верстатів з ЧПК. / Д.О. Лозінський; ВНТУ. – Вінниця, 2018. 84 с.

3) Джемелінський В.В. Методичні вказівки з лабораторних і практичних робіт до вивчення дисципліни «Основи програмування на верстатах з числовим програмним керуванням» / В.В. Джемелінський, О.Д. Кагляк, Д.А. Лесик. ; НТУУ «КПІ». Київ, 2011, 56 с.

4) Обзор популярных систем автоматизированного проектирования (*CAD*) [Електронний ресурс]. – 2017. *www.pointcad.ru* – Режим доступу: *https://www.pointcad.ru/novosti/obzor-sistem-avtomatizirovannogo-proektirovaniya* (дата звернення 18.05.2021) – Назва з екрана.

5) *CNC Machine Tools Market Size; COVID-19 Impact Analysis, By Type, By Application and Regional Forecast*. [Електронний ресурс]. – 2019. *www.fortunebusinessinsights.com* – Режим доступу: *https://www.fortunebusinessinsights.com/industry-reports/computer-numerical-controls-cnc-machine-tools-market-101707* (дата звернення 18.05.2021) – Назва з екрана.

6) *AN2586 Application note*., *STMicroelectronics* – 2014. 28 с.

7) *A4988 DMOS Microstepping Driver with Translator and Overcurrent Protection., Allegro MicroSystems* – 955 *Perimeter Road Manchester, NH* 03103-3353 *U.S.A.*, 2020. 20 с.

8) *LM317 3-Terminal Adjustable Regulator datasheet (Rev. Y)*., *Texas Instruments – Dallas, Texas* 75265, 2020. 32 с.

9) *STM32CubeProgrammer software for all STM32*. [Електронний ресурс].– 2021. *www.st.com* – Режим доступу: *https://www.st.com/en/development-tools/stm32cubeprog.html* (дата звернення 20.05.2021) – Назва з екрана.

10) *Klipper documentation – features* [Електронний ресурс] – 2018. *www.klipper3d.org* – Режим доступу: *https://www.klipper3d.org/Features.html* (дата звернення 18.05.2021) – Назва з екрана.

11) Лопаткин А. Проектирование печатных плат в *Altium Designer*. – М.: ДМК Пресс, 2016. – 400 с.

12) *Circuit Board Layout Techniques / Bruce Carter – Texas Instruments*, 2008. – 32 с.

13) *ISO 6983-1:2009. Automation systems and integration – Numerical control of machines – Program format and definitions of address words – Part 1:Data format for positioning, line motion and contouring control systems.* – Введ. 1983, Ред. 2009, 26 с.

14) ДСТУ 3008-95 Документація. Звіти у сфері науки і техніки. Структура і правила оформлення.

15) Бойченко С.В., Іванченко О.В. Положення про дипломні роботи (проекти) випускників Національного авіаційного університету. – К.: НАУ, 2017. – 63 с.

ДОДАТОК А

Лістинг коду програмних модулів

main.c

```
#include "cnc.h"

system_t sys;
int32_t sys_position[N_AXIS];
volatile uint8_t sys_rt_exec_state;
volatile uint8_t sys_rt_exec_alarm;
volatile uint8_t sys_rt_exec_motion_override;
volatile uint8_t sys_rt_exec_accessory_override;
#ifdef DEBUG
volatile uint8_t sys_rt_exec_debug;
#endif

#include "usb_lib.h"
#include "usb_desc.h"
#include "hw_config.h"
#include "usb_pwr.h"
#include "stm32eeprom.h"

int main(void)
{

#ifdef LEDBLINK
        GPIO_InitTypeDef GPIO_InitStructure;

        RCC_APB2PeriphClockCmd(RCC_APB2Periph_GPIOC, ENABLE);
```

```
        GPIO_InitStructure.GPIO_Speed = GPIO_Speed_2MHz;
        GPIO_InitStructure.GPIO_Mode = GPIO_Mode_Out_PP;
        GPIO_InitStructure.GPIO_Pin = GPIO_Pin_13;
        GPIO_Init(GPIOC, &GPIO_InitStructure);
#endif

        Set_USBClock();
        USB_Interrupts_Config();
        USB_Init();

#ifndef NOEEPROMSUPPORT
        FLASH_Unlock();
        eeprom_init();
#endif
        SysTick->CTRL &= 0xfffffffb;

  serial_init();
  settings_init();
  stepper_init();
  system_init();

  memset(sys_position,0,sizeof(sys_position));

  #ifdef FORCE_INITIALIZATION_ALARM

    sys.state = STATE_ALARM;
  #else
    sys.state = STATE_IDLE;
  #endif
```

```
        #ifdef HOMING_INIT_LOCK
          if (bit_istrue(settings.flags,BITFLAG_HOMING_ENABLE)) { sys.state =
STATE_ALARM; }
        #endif

        for(;;) {

          uint8_t prior_state = sys.state;
          memset(&sys, 0, sizeof(system_t));
          sys.state = prior_state;
          sys.f_override = DEFAULT_FEED_OVERRIDE;
          sys.r_override = DEFAULT_RAPID_OVERRIDE;
          sys_rt_exec_state = 0;
          sys_rt_exec_alarm = 0;
          sys_rt_exec_motion_override = 0;
          sys_rt_exec_accessory_override = 0;

          serial_reset_read_buffer();
          gc_init();
          limits_init();
          plan_reset();
          st_reset();

          plan_sync_position();
          gc_sync_position();

          protocol_main_loop();

        }
        return 0;
```

```
}

void _delay_ms(uint32_t x)
{
	u32 temp;
	SysTick->LOAD = (u32)72000000 / 8000 * x;
	SysTick->VAL = 0x00;
	SysTick->CTRL = 0x01;
	do
	{
		temp = SysTick->CTRL;
	} while ((temp & 0x01) && !(temp&(1 << 16)));
	SysTick->CTRL = 0x00;
	SysTick->VAL = 0X00;
}
void LedBlink(void)
{
	static BitAction nOnFlag = Bit_SET;
	GPIO_WriteBit(GPIOC, GPIO_Pin_13, nOnFlag);
	nOnFlag = (nOnFlag == Bit_SET) ? Bit_RESET : Bit_SET;
}
```

serial.c

```
#include "cnc.h"
#include "stm32f10x.h"
#include "core_cm3.h"
#include "usb_regs.h"

#define RX_RING_BUFFER (RX_BUFFER_SIZE)
#define TX_RING_BUFFER (TX_BUFFER_SIZE)
```

```
uint8_t serial_rx_buffer[RX_RING_BUFFER];
uint8_t serial_rx_buffer_head = 0;
volatile uint8_t serial_rx_buffer_tail = 0;

uint8_t serial_tx_buffer[TX_RING_BUFFER];
uint8_t serial_tx_buffer_head = 0;
volatile uint8_t serial_tx_buffer_tail = 0;

uint8_t serial_get_rx_buffer_available()
{
  uint8_t rtail = serial_rx_buffer_tail;
  if (serial_rx_buffer_head >= rtail) { return(RX_BUFFER_SIZE -
(serial_rx_buffer_head-rtail)); }
  return((rtail-serial_rx_buffer_head-1));
}

uint8_t serial_get_rx_buffer_count()
{
  uint8_t rtail = serial_rx_buffer_tail;
  if (serial_rx_buffer_head >= rtail) { return(serial_rx_buffer_head-
rtail); }
  return (RX_BUFFER_SIZE - (rtail-serial_rx_buffer_head));
}

uint8_t serial_get_tx_buffer_count()
{
  uint8_t ttail = serial_tx_buffer_tail;
  if (serial_tx_buffer_head >= ttail) { return(serial_tx_buffer_head-ttail); }
  return (TX_RING_BUFFER - (ttail-serial_tx_buffer_head));
```

```
}

void serial_init(){
}

void serial_write(uint8_t data) {

  uint8_t next_head = serial_tx_buffer_head + 1;
  if (next_head == TX_RING_BUFFER) { next_head = 0; }
  while (next_head == serial_tx_buffer_tail) {
    if (sys_rt_exec_state & EXEC_RESET) { return; }
  }

  serial_tx_buffer[serial_tx_buffer_head] = data;
#ifdef USB_CHANGED_DEBUG
  while(txUsbLock) {
    if (sys_rt_exec_state & EXEC_RESET) { return; }
  }
#endif
  serial_tx_buffer_head = next_head;
}

uint8_t serial_read()
{
  uint8_t tail = serial_rx_buffer_tail;
  if (serial_rx_buffer_head == tail) {
    return SERIAL_NO_DATA;
  } else {
    uint8_t data = serial_rx_buffer[tail];
```

```
    tail++;
    if (tail == RX_RING_BUFFER) { tail = 0; }
    serial_rx_buffer_tail = tail;

    return data;
  }
}

void OnUsbDataRx(uint8_t* dataIn, uint8_t length)
{
  uint8_t next_head;
  uint8_t data;

  while (length != 0)
  {
      data = *dataIn ++;

  switch (data) {
    case CMD_RESET:         mc_reset(); break;
    case CMD_STATUS_REPORT: system_set_exec_state_flag(EXEC_STATUS_REPORT); break;
    case CMD_CYCLE_START:   system_set_exec_state_flag(EXEC_CYCLE_START); break;
    case CMD_FEED_HOLD:     system_set_exec_state_flag(EXEC_FEED_HOLD); break;
    default :
      if (data > 0x7F) {
        switch(data) {
          case CMD_SAFETY_DOOR:   system_set_exec_state_flag(EXEC_SAFETY_DOOR); break;
```

```
            case CMD_JOG_CANCEL:
              if (sys.state & STATE_JOG) {
                system_set_exec_state_flag(EXEC_MOTION_CANCEL);
              }
              break;
            case CMD_FEED_OVR_RESET: system_set_exec_motion_override
_flag(EXEC_FEED_OVR_RESET); break;
            case CMD_FEED_OVR_COARSE_PLUS: system_set_exec_motion_
override_flag(EXEC_FEED_OVR_COARSE_PLUS); break;
            case CMD_FEED_OVR_COARSE_MINUS: system_set_exec_motio
n_override_flag(EXEC_FEED_OVR_COARSE_MINUS); break;
            case CMD_FEED_OVR_FINE_PLUS: system_set_exec_motion_ove
rride_flag(EXEC_FEED_OVR_FINE_PLUS); break;
            case CMD_FEED_OVR_FINE_MINUS: system_set_exec_motion_o
verride_flag(EXEC_FEED_OVR_FINE_MINUS); break;
            case CMD_RAPID_OVR_RESET: system_set_exec_motion_override
_flag(EXEC_RAPID_OVR_RESET); break;
            case CMD_RAPID_OVR_MEDIUM: system_set_exec_motion_overr
ide_flag(EXEC_RAPID_OVR_MEDIUM); break;
            case CMD_RAPID_OVR_LOW: system_set_exec_motion_override_f
lag(EXEC_RAPID_OVR_LOW); break;
          }

        } else {
          next_head = serial_rx_buffer_head + 1;
          if (next_head == RX_RING_BUFFER) { next_head = 0; }


          if (next_head != serial_rx_buffer_tail) {
            serial_rx_buffer[serial_rx_buffer_head] = data;
```

```
        serial_rx_buffer_head = next_head;
       }
      }
  }
   length--;
  }
}

void serial_reset_read_buffer()
{
 serial_rx_buffer_tail = serial_rx_buffer_head;
}
```

protocol.c

```
#include "cnc.h"

#define LINE_FLAG_OVERFLOW bit(0)
#define LINE_FLAG_COMMENT_PARENTHESES bit(1)
#define LINE_FLAG_COMMENT_SEMICOLON bit(2)

static char line[LINE_BUFFER_SIZE];
#ifdef LEDBLINK
void LedBlink(void);
#endif

static void protocol_exec_rt_suspend();

void protocol_main_loop()
{
 #ifdef CHECK_LIMITS_AT_INIT
```

```
  if (bit_istrue(settings.flags, BITFLAG_HARD_LIMIT_ENABLE)) {
    if (limits_get_state()) {
      sys.state = STATE_ALARM;
      report_feedback_message(MESSAGE_CHECK_LIMITS);
    }
  }
#endif

if (sys.state & (STATE_ALARM | STATE_SLEEP)) {
  report_feedback_message(MESSAGE_ALARM_LOCK);
  sys.state = STATE_ALARM;
} else {

  sys.state = STATE_IDLE;
  if (system_check_safety_door_ajar()) {
    bit_true(sys_rt_exec_state, EXEC_SAFETY_DOOR);
    protocol_execute_realtime();
  }

  system_execute_startup(line);
}
uint8_t line_flags = 0;
uint8_t char_counter = 0;
uint8_t c;
for (;;) {
  while((c = serial_read()) != SERIAL_NO_DATA) {
    if ((c == '\n') || (c == '\r')) {
      protocol_execute_realtime();
      if (sys.abort) { return; }
```

```
        line[char_counter] = 0;
#ifdef LEDBLINK
        LedBlink();
#endif
        #ifdef REPORT_ECHO_LINE_RECEIVED
          report_echo_line_received(line);
        #endif

        if (line_flags & LINE_FLAG_OVERFLOW) {

          report_status_message(STATUS_OVERFLOW);
        } else if (line[0] == 0) {

          report_status_message(STATUS_OK);
        } else if (line[0] == '$') {

          report_status_message(system_execute_line(line));
        } else if (sys.state & (STATE_ALARM | STATE_JOG)) {

          report_status_message(STATUS_SYSTEM_GC_LOCK);
        } else {

          report_status_message(gc_execute_line(line));
        }

        line_flags = 0;
        char_counter = 0;

      } else {
```

```
            if (line_flags) {

              if (c == ')') {

                if (line_flags & LINE_FLAG_COMMENT_PARENTHESES) { line_
flags &= ~(LINE_FLAG_COMMENT_PARENTHESES); }
              }
            } else {
              if (c <= ' ') {
              } else if (c == '/') {
              } else if (c == '(') {
                line_flags |= LINE_FLAG_COMMENT_PARENTHESES;
              } else if (c == ';') {
                line_flags |= LINE_FLAG_COMMENT_SEMICOLON;
              } else if (char_counter >= (LINE_BUFFER_SIZE-1)) {
                line_flags |= LINE_FLAG_OVERFLOW;
              } else if (c >= 'a' && c <= 'z') {
                line[char_counter++] = c-'a'+'A';
              } else {
                line[char_counter++] = c;
              }
            }
          }
        }
        protocol_auto_cycle_start();
        protocol_execute_realtime();
        if (sys.abort) { return; }
      }
      return;
    }
```