НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2021 р

Дипломний проєкт

на здобуття ступеня бакалавра
за освітньо-професійною програмою “Комп’ютерні системи та мережі”
спеціальності 123 “Комп’ютерна інженерія”
на тему: “Графопобудовник для автоматичного друку векторної графіки”

Виконав:
студент IV курсу, групи ІО-71
Саяпін Юрій Львович _____

Керівник:
д.т.н, доцент,
Клименко Ірина Анатоліївна _____

Консультант з нормоконтролю:
д.т.н, професор
Сімоненко Валерій Павлович _____

Рецензент:
_____
_____ _____

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2021 року

НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
“КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО”

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалавр)

Спеціальність - 123 “Комп’ютерна інженерія”

Освітньо-професійна програма - “Комп’ютерні системи та мережі”

ЗАТВЕРДЖУЮ
Завідувач кафедри
_____ Сергій СТІРЕНКО
«___»_____2021 р

ЗАВДАННЯ
на бакалаврський дипломний проєкт студента

Саяпіна Юрія Львовича

1. Тема проєкту *Графопобудовник для автоматичного друку векторної графіки*
керівник проєкту *Клименко Ірина Анатоліївна, д.т.н, доцент*,
затверджені наказом по університету від _____ 2021 року №_____
2. Термін здачі студентом закінченого проєкту _____*20 травня 2021 р.*
3. Вихідні дані до проєкту *технічна документація, теоретичні дані*
4. Зміст розрахунково-пояснювальної записки (перелік питань, які розробляються)
аналіз предметної області, вибір й обґрунтування засобів реалізації, розробка програмного й апаратного продукту, результати тестування.
5. Перелік графічного матеріалу (з точним позначенням обов’язкових креслень)
структурна схема апаратної частини пристрою, схема алгоритму роботи програмного забезпечення персонального комп’ютера, схема алгоритму роботи програмного забезпечення мікроконтролера.

6. Консультанти проєкту, з вказівкою розділів проєкту, які до них вносяться

Розділ	Консультант	Підпис, дата	
		Завдання видав	Завдання прийняв
Нормоконтроль	*д.т.н, проф. Сімоненко В. П*		

7. Дата видачі завдання_____

Календарний план

№ п/п	Найменування етапів дипломного проєкту	Терміни виконання етапів проєкту	Примітки
1.	*Затвердження теми проєкту*	*10.12.2021-15.12.2021*	
2.	*Вивчення та аналіз завдання*	*15.12.2021-15.03.2021*	
3.	*Розробка архітектури та загальної структури пристрою*	*15.03.2021-25.03.2021*	
4.	*Розробка структур окремих підсистем програмного забезпечення*	*25.03.2021-31.03.2021*	
5.	*Розробка апаратної частини пристрою*	*1.04.2021 – 6.04.2021*	
6.	*Програмна реалізація системи*	*7.04.2021-15.04.2021*	
7.	*Оформлення пояснювальної записки*	*15.04.2021-20.05.2021*	
8.	*Захист програмного продукту*	*25.04.2021*	
9.	*Передзахист*	*23.05.2021*	
10.	*Захист*	*15.06.2021*	

Студент-дипломник _____

(підпис)

Керівник проєкту _____

(підпис)

Анотація

В бакалаврському дипломному проєкті було реалізовано зовнішній пристрій для персонального комп’ютера: графопобудовник для автоматичного друку векторної графіки, що призначений для друку монохроматичних векторних зображень.

Пристрій і супровідне програмне забезпечення дозволяє виконувати друк векторних зображень, що переведено до формату G-коду. Програмне забезпечення для персонального комп’ютера створено на мові програмування Python 3.8 з використанням бібліотеки с відкритим початковим кодом для спілкування по протоколу USBHID – pyhidapi. Програмне забезпечення для мікроконтролера написано на мові C++ з використанням фреймворку Mbed OS 6.10.

Annotation

The Bachelor's Thesis project implements an external device for a personal computer: a graph plotter for automatic printing of vector graphics, designed for printing monochromatic vector images.

The device and the accompanying software allow you to print vector images that have been converted to G-code. The software for the personal computer is created in the Python 3.8 programming language using an open-source library for communication via the USBHID protocol - pyhidapi. The software for the microcontroller is written in C ++ using the Mbed OS 6.10 framework.

ОПИС АЛЬБОМУ

до дипломного проєкту

на тему: «Графопобудовник для автоматичного друку векторної графіки»

Київ – 2021 року

№ строки	Формат	Обозначение	Наименование	Кол.	№ экз.	Приме-чание
1			Документація загальна			
2			Розроблена по-новому			
3						
4	А4	ІАЛЦ.466500.002 ТЗ	Графопобудовник для	3		
5			автоматичного друку векторної			
6			графіки			
7			Технічне завдання			
8						
9	А4	ІАЛЦ.466500.003 ПЗ	Графопобудовник для	68		
10			автоматичного друку векторної			
11			графіки			
12			Пояснювальна записка			
13						
14	А3	ІАЛЦ.466500.004 Д1	Графопобудовник для	1		
15			автоматичного друку векторної			
16			графіки			
17			Схема електрична структурна			
18						
19	А4	ІАЛЦ.466500.005 Д2	Графопобудовник для	1		
20			автоматичного друку векторної			
21			графіки			
22			Функціональна схема взаємодії			
23			користувача із пристроєм			
24						
25						
26						
27						

Зм.	Арк.	№ докум.	Підп.	Дата	***ІАЛЦ.466500.001 ОА***		
Розроб.		Саяпін Ю.Л.			*Графопобудовник для автоматичного друку векторної графіки*	Літ.	Арк. 1 / Аркушів 2
Перевір.		Клименко І. А.					
Реценз.					**ОПИС АЛЬБОМУ**	***НТУУ «КПІ» ФІОТ ІО-71***	
Н. контр.		Симоненко В. П.					
Затверд.		Клименко І. А.					

№ строки	Формат	Обозначение	Наименование	Кол.	№ экз.	Приме-чание
28	А4	ІАЛЦ.466500.006 Д3	Графопобудовник для	1		
29			автоматичного друку векторної			
30			графіки			
31			Схема алгоритму роботи			
32			програмного забезпечення			
33			персонального комп’ютера			
34						
35	А4	ІАЛЦ.466500.007 Д4	Графопобудовник для	1		
36			автоматичного друку векторної			
37			графіки			
38			Схема алгоритму роботи			
39			програмного забезпечення			
40			мікроконтролера			
41						
42	А4	ІАЛЦ.466500.008 Д5	Графопобудовник для	6		
43			автоматичного друку векторної			
44			графіки			
45			Лістинг програмного забезпечення			
46			персонального комп’ютера			
47						
48	А4	ІАЛЦ.466500.009 Д6	Графопобудовник для	3		
49			автоматичного друку векторної			
50			графіки			
51			Лістинг програмного забезпечення			
52			мікроконтролера			
53						
54						
55						
56						

ТЕХНІЧНЕ ЗАВДАННЯ

до дипломного проєкту

на тему: «Графопобудовник для автоматичного друку векторної графіки»

Київ – 2021 року

ЗМІСТ

1.НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Найменування: «Графопобудовник для автоматичного друку векторної графіки». Область застосування: графопобудовник є спеціалізованим засобом виведення графічної інформації на зовнішні носії.

2.ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для розробки є завдання на виконання бакалаврського дипломного проекту, затверджене кафедрою обчислювальної техніки Національного технічного університету України «Київський політехнічний інститут ім. Ігоря Сікорського».

3.МЕТА ТА ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою розробки є створення графопобудовника для автоматичного друку векторної графіки та допоміжного програмного забезпечення для персонального комп’ютера для керування пристроєм.

4.ДЖЕРЕЛА РОЗРОБКИ

Джерелом розробки є науково-технічна література, публікації в спеціалізованих періодичних виданнях, довідники по платформах дистанційного навчання, публікації в мережі Інтернет по даній темі.

5.ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробленого продукту

- Можливість друку простих монохроматичних векторних зображень, що надаються у форматі G-коду;
- Незалежність апаратної частини від архітектури комп’ютера, з якого виконується керування пристроєм;
- Незалежність працездатності пристрою від операційної системи, з якою працює персональний комп’ютер.

5.2. Вимоги до програмного забезпечення

- Інтерпретатор мови Python версії 3.8 та вище зі встановленою бібліотекою pyhidapi і її залежностями;
- Середа розробки Mbed Studio;
- Для апаратної частини пристрою версія Mbed OS не нижче за 6.10;
- Будь-яке програмне забезпечення, здатне перетворювати векторне зображення до G-коду, наприклад Inkscape.

5.3. Вимоги до апаратної частини

- Наявність інтерфейсу USB версії 2.0 та вище.

6. ЕТАПИ РОЗРОБКИ

	Дата
6.1. Вивчення необхідної літератури	21.02.2021
6.2. Складання і узгодження технічного завдання	08.03.2021
6.3. Написання вступної частини та огляд рішень	14.03.2021
6.4. Розробка архітектури додатку	22.03.2021
6.5. Написання програмної частини	11.04.2021
6.6. Тестування та виправлення помилок	04.05.2021
6.7. Оформлення документації дипломного проекту	17.05.2021

ПОЯСНЮВАЛЬНА ЗАПИСКА

до дипломного проєкту

на тему: «Графопобудовник для автоматичного друку векторної графіки»

Київ – 2021 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

САПР	Система автоматизованого проєктування
USB	(Universal serial bus) Універсальна послідовна шина
COM	(Communication port) Послідовний порт інтерфейсу RS-232
ШІМ	Широтно-імпульсна модуляція
UART	(Universal asynchronous receiver/transmitter) Універсальний асинхронний приймач/передавач
SPI	(Serial peripheral interface) Послідовний периферійний інтерфейс
МК	Мікроконтролер
ПК	Персональний комп’ютер
ПЗ	Програмне забезпечення
HID, USBHID	(Human Interface Device) Клас пристроїв USB для взаємодії з людиною
RTOS, ОСРЧ	(Real time operating system) Операційна система реального часу

Mbed OS	Фреймворк та однойменна операційна система реального часу
ЧПК	Числове програмне керування

ВСТУП

В останні роки все більш для проєктування використовуються спеціалізоване програмне забезпечення САПР, що дозволяє виробляти, наприклад, креслення за допомогою комп'ютерів. Але ці створенні креслення все одно треба переносити на паперовий носій. Зараз у більшості випадків для цього використовують звичайні пристрої для друку: лазерні та струменеві принтери – що не зовсім підходить для друку креслень та іншої технічної документації.

Для вирішення ситуації з неякісним друком технічної документації треба використовувати спеціальні принтери для друку саме векторної графіки – графопобудовники, що також мають назву плоттер (від англ. "plot" - графік).

При цьому всьому область використання графопобудовника не обмежується лише друком креслень: за допомогою плоттеру можна надрукувати як векторне зображення, так і звичайне растрове зображення.

У наш час існує вже багато рішень зі сторони ринку в області створення плоттерів та іншої техніки для відтворення зображень на паперовому чи іншому носії, але все вони обмежені в конструкції та мають внутрішні обмеження керуючого програмного забезпечення, та програмного забезпечення контролера, що керує процесом друку на самому пристрої.

Актуальність проєкту полягає в тому, що сьогоденні рішення, що є на ринку або не зовсім підходять для використання в конкретній області або мають суттєві недоліки, що закладаються виробником під час виробництва. Основною ціллю дипломного проєкту є створення пристрою, що підходить для використання саме для друку креслень та векторних зображень, та не має значних обмежень, або вони можуть бути змінені за допомогою програмного забезпечення пристрою.

РОЗДІЛ 1. АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ

1.1. Загальні відомості про графопобудовники.

Графопобудовник – пристрій для автоматичного друку з великою точністю малюнків, схем, креслень, мап та іншої графічної інформації на папері [1].

На цей час графопобудовники в основному використовують на підприємствах, в проєктувальних інститутах, університетах, у невеликих виробництвах для друку креслень та векторних зображень.

Графопобудовники можна класифікувати за різними категоріями [2, 3]:

- Спосіб формування креслення:
  - Растрові (друк відбувається як у звичайних принтерах, але з підвищеною точністю);
  - Векторні (друк відбувається за допомогою побудови векторних елементів: точки, відрізки, кола та ін.).
- Спосіб переміщення носія:
  - Планшетні (рис. 1.1) (при друку відбувається переміщення друкуючої головки у двох напрямках вздовж двох осей водночас);
  - Барабанні (рис 1.2) (папір подається під час процесу друку, пишуча головка рухається лише вздовж однієї осі, при цьому можна надрукувати неформатні зображення, що обмежені лише шириною барабану та довжиною аркуша; великим недоліком таких пристроїв є зниження якості друку, але з’являється можливість кольорового друку).

- Тип друкуючої головки:
  - Пишуча;
  - Фотопобудівна;
  - Фрезерна та інші.

Рис 1.1 - Планшетний графопобудовник [4]

Рис 1.2 - Барабанний графопобудовник [5]

При виконанні дипломного проєкту було прийнято рішення створити планшетний пишучий графопобудовник з векторним формуванням зображення, що значно спрощує конструкцію пристрою, при цьому не погіршуючи якості результату друку кінцевого продукту.

1.2. Аналіз існуючих засобів спілкування з пристроєм.

Зазвичай пристрої для друку графічної інформації працюють не як окремі пристрої, а як зовнішній пристрій персонального комп'ютера. Деякі моделі підтримують підключення напряму та завантаження зображення прямо до пам'яті плоттера, але з врахуванням того, що у більшості випадків друк відбувається одразу ж після проєктування на комп'ютері, цей функціонал не має сенсу [1, 3].

Старі моделі графопобудовників підключалися до комп'ютерів за допомогою паралельного інтерфейсу або за допомогою COM-портів, що зазвичай вже не підтримуються сучасними комп'ютерами напряму, а лише через спеціальні перетворювачі. У наші часи все більш використовується інтерфейс USB [2, 3].

Для керування пристроєм треба звернути увагу на протокол, за яким буде відбуватися спілкування за комп'ютером. Основним недоліком сучасних графопобудовників є те, що вони використовують закриті протоколи зв'язку, власні драйвери пристроїв, та закодовані команди, що заважають швидкому перенесенню пристрою з одного комп'ютера на інший.

1.3. Огляд способів представлення зображень.

Зображення для ПК зазвичай зберігаються двома способами растровим та векторним [6].

Растрове зображення – зображення, що складається з матриці кольорових точок (наприклад, пікселі для дисплеїв). Растрові зображення не масштабуються, або масштабуються з пошкодженнями зображення.

Векторні зображення – зображення, що описується елементарними графічними об’єктами, що мають назву примітив: точка, пряма, коло та інші. Векторне зображення легко масштабується, та придатне для друку на векторних графопобудовниках.

Але графопобудовники не приймають векторне зображення в його початковому вигляді, саме в описі кіл, прямих, прямокутників, тому зображення спочатку потрібно перетворити. Для цього використовують такий опис зображень для ЧПУ-пристроїв, що називається G-кодом, який описує рух інструмента під час друку.

1.4. Огляд наявних аналогів у галузі.

В цільовій галузі небагато вже існуючих на ринку рішень, через специфічність самого пристрою та області його використання.

Більшість існуючих рішень використовують в якості типу графопобудовника растрові або векторні барабанні системи з друкуючими головками через те, що вони займають менше місця. Але складність їх конструкції заважає його розробці та зменшують кількість сценаріїв використання з універсальних пристроїв до пристроїв для друку неформатних або кольорових зображень.

В якості прикладів розглянемо декілька усереднених представників пристроїв, що є на ринку.

Epson SureColor SC-T5200:

Растровий барабанний пишучий графопобудовник, що має можливість кольорового друку. При цьому використовуються технологія струменевого друку [7] (Рис 1.3).

Рис 1.3 - Epson SureColor SC-T5200 [7]

Makeblock XY-Plotter:

Навчальний векторний планшетний пишучий графопобудовник для однокольорового друку (рис 1.4), Основними недоліками є розміри, що обумовлюються типом графопобудовника, та закрите програмне забезпечення, що дозволяє пристрою працювати лише з операційною системою Microsoft Windows. Точність друку пристрою складає 0,1 мм, що є достатньо великою для пристрою такого цінового сегмента. Дозволяє модифікацію у вигляді заміни друкуючої головки на лазерну для виконання гравірування [8].

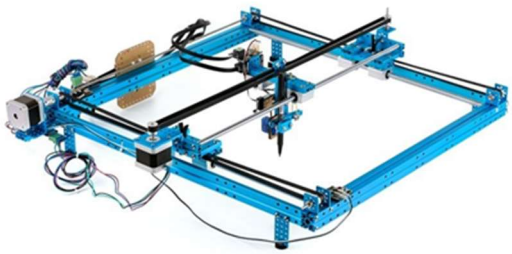


Рис 1.4 - Makeblock XY-Plotter [8]

ВИСНОВОК ДО РОЗДІЛУ 1

У цьому розділі було розглянуто предметну область проєкту, зроблено огляд типів та технологій сучасних графопобудовників та було розглянуто деякі наявні на українському ринку рішення.

Було визначено оптимальну структуру та тип пристрою, що буде розроблено під час дипломного проєктування та задано напрямок для вибору основних технологій, що будуть використані при створенні пристрою.

РОЗДІЛ 2. АНАЛІЗ ТА ВИБІР ТЕХНОЛОГІЙ

2.1. Огляд та порівняння технологій для реалізації апаратної частини.

Графопобудовник, як і будь-який інший зовнішній пристрій складається не тільки з програмної частини, але ще має апаратну частину: сам пристрій.

Апаратна частина зовнішнього пристрою зазвичай керується або програмним забезпеченням, що виконуються на основному комп'ютері, або керується програмним забезпеченням вбудованого в пристрій мікроконтролера, що виконує саме керування та спілкування с основним пристроєм. Цей підхід значно простіше за керування напряму, та дозволяє абстрагуватися від внутрішніх інтерфейсів комп'ютера, тому він зараз використовується у більшості сучасних пристроїв.

Для того, щоб обрати необхідний для побудови графопобудовника мікроконтролер необхідно розібратися з деякими основними параметрами апаратної частини:

- Якими саме апаратними пристроями він буде керувати, розглянути алгоритми керування;
- За допомогою якого інтерфейсу буде відбуватися спілкування зовнішнього пристрою та комп'ютера, на я кому буде виконуватися керуюче програмне забезпечення;
- Формат команд, за яким буде виконуватися саме керування та час необхідний на обробку команди;
- Наявні та необхідні фреймворки та бібліотеки.

2.1.1. Апаратна частина графопобудовника.

Графопобудовник, як й інші типи принтерів має в собі рухомі частини, які мають бути призведені до руху. Для цього зазвичай використовують різни типі електродвигунів. Опис та порівняння різних видів електродвигунів для

вибору не є основною темою дипломної роботи, тому його буде пропущено. Треба лише зазначити, що для керування багатьма видами двигунів використовуються системи зворотного зв'язку.

Для того щоб правильно визначити позицію друкуючої головки, треба знати стан двигуна в часі, для чого саме й використовують системи зворотного зв'язку [9]: знаючи лише початкове положення під час початку руху та швидкість руху двигуна складно розрахувати точне положення, тому вимірюють інші показники, наприклад кількість обертів які відбулися з моменту початку руху.

Як можна зрозуміти, система зворотного зв’язку ускладнює конструкцію, та збільшує кількість обчислень, що потрібно провести для одного переміщення, яких під час друку може бути більше ста. Для уникнення проблеми з необхідністю використання систем зворотного зв'язку має сенс використовувати двигуни зі строго детермінованою поведінкою під час руху, наприклад, крокові двигуни (Рис 2.1). В залежності від того, який струм на які обкладки двигуна було подано, двигун виконую один крок в потрібному напрямку [10].

Рис 2.1 - Крокoвий двигун [10]

Керувати будь яким типом двигуна напряму з мікроконтролера неможливо, тому що вони потребують дуже великі токи та напруги для своєї роботи, тому для них використовуються драйвери [10] (Рис 2.2). Драйвером двигуна називається така спеціалізована мікросхема, що виконує керування двигуном з відносно великими токами та напругами за допомогою спеціальної послідовності керуючих сигналів [11].

Рис 2.2 - Драйвер крокового двигуна [11]

Драйвери крокових двигунів мають три керуючих сигнали, а саме:

- EN, сигнал дозволу роботи. Без наявності цього сигналу драйвер ігнорує всі інші сигнали;
- DIR, сигнал напрямку руху. За тим, який логічний рівень встановлено на цьому вході, драйвер визначає в яку сторону має обертатися двигун;
- STEP, сигнал необхідності виконати крок. За наявністю перепаду сигналу з 0 до 1 драйвер розуміє, що потрібно згенерувати керуючий сигнал для пересування двигуна на один крок.

Опис рівнів сигналів, їх величини та часу можна знайти в документації на конкретний драйвер двигуна.

Друкуюча головка також має власний двигун для керування режимом переміщення головки: друкує чи ні. Для цього потрібно використати двигун з як мінімум двома детермінованими станами. Найбільш для цього підходять

серводвигуни (Рис 2.3). Це звичайні двигуни постійного струму, але вони мають в себе вбудовану схему зворотного зв'язку та керуються напряму з мікроконтролера за допомогою широтно-імпульсної модуляції. Чім більше коефіцієнт заповнення ШІМ-сигналу, тим на більший кут здвигається серводвигун [12].

Рис 2.3 - Серводвигун

Також для визначення початкової позиції друкуючої головки та для запобігання руйнування механізму під час роботи потрібно використовувати кінцеві вимикачі, що мають бути розміщені на опорах осей, по яких рухається друкуюча головка.

З цього можна зробити висновок, що для повноцінної роботи графопобудовника необхідно обрати такий мікроконтролер, що підтримує не тільки вільне керування портами введення-виведення, але й може генерувати ШІМ-сигнал та обробляти переривання з портів введення-введення.

2.1.2. Інтерфейс зв’язку з комп’ютером.

Сучасні мікроконтролери підтримують багато різних стандартів для передачі даних с одного пристрою на інший.

Для підключення до персонального комп'ютера підходять не усі з них, через те, що деякі інтерфейси не реалізуються у звичайних ПК, такі як SPI, I2C (Inter-Integrated Circuit, послідовна шина даних для зв'язку інтегральних схем). Але все ж таки деякі інтерфейси можуть бути підключені напряму або через спеціалізовані перетворювачі, серед них:

1. UART

Може бути підключено через перетворювач з UART на RS-232 до COM порту ПК, та через перетворювач з UART на USB (Рис 2.4). Інтерфейс COM є досить старим, тому зазвичай ці роз'єми не встановлюються у сучасні ПК. В такому разі є доцільним використання перетворювача з UART на USB [13].

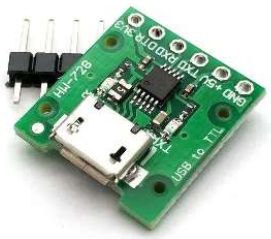


Рис 2.4 - USB-UART перетворювач

Переваги інтерфейсу UART:

- Просте підключення через недорогий та простий у використанні перетворювач;
- Підтримка інтерфейсу практично усіма сучасними мікроконтролерами;
- Наявність вбудованих в операційну систему драйверів. Простий алгоритм роботи з пристроєм (фактично читання та запис у спеціальний файл).

Недоліки:

- Необхідність додаткової мікросхеми для перетворення інтерфейсу;
- Неможливість автоматично визначити пристрій та підключитися до нього, що зумовлено самим інтерфейсом.

2. Ethernet

Всім звичний мережевий інтерфейс фізичного рівня. Повна реалізація цього інтерфейсу в сучасному мікроконтролері є не зовсім частим явищем, та у більшості випадків в мікроконтролері реалізовано протоколи канального рівня та потрібне підключення спеціальної мікросхеми інтерфейсу фізичного рівня. Ті мікроконтролери, що мають в собі реалізацію канального рівня мають спеціальний інтерфейс MII (або його різновид - RMII) ([Reduced] Media Independent Interface, незалежний від середи передачі інтерфейс) для підключення мікросхеми фізичного рівня. Для мікроконтролерів, що не мають реалізації фізичного рівня можливе підключення спеціальних модулів, що реалізують в собі не тільки фізичний, але й канальний, мережевий та транспортний рівні. Для підключення таких модулів зазвичай використовують швидкісний інтерфейс SPI, що є в багатьох сучасних мікроконтролерах (Рис 2.5).

Рис 2.5 - SPI Ethernet модуль

Переваги:

- Висока швидкість передачі даних;
- Можливість віддаленої роботи з пристроєм через мережу інтернет.

Недоліки:

- Необхідність додаткової мікросхеми для перетворення інтерфейсу;
- Складність роботи з інтерфейсом. Через цей інтерфейс не можна просто переслати дані на ПК: необхідно реалізовувати повний стек протоколів TCP/IP для пересилання даних;
- Необхідність у мережевому обладнанні.

3. WiFi

Інтерфейс WiFi повторює вищесказане про Ethernet, з деякими застереженнями.

Реалізація цього інтерфейсу є дуже рідкою в самих мікроконтролерах, та їх можна перерахувати на пальцях однієї руки. Можливе підключення зовнішнього модуля WiFi через інтерфейс UART, або запрограмувати саме мікросхему WiFi модуля для виконання потрібних задач. Також у деяких модулів WiFi є можливість підключення по інтерфейсу SDIO (Secure Digital Input/Output, захищений цифровий ввід-вивід) (Рис 2.6), але його підтримка наявна не в усіх мікроконтролерах.

Рис 2.6 - SDIO WiFi модуль

Переваги:

- Висока швидкість передачі даних;
- Можливість віддаленої роботи з пристроєм через мережу інтернет.

Недоліки:

- Необхідність другого мікроконтролера або використання зовнішнього модуля;
- Якщо програмувати сам мікроконтролер, то реалізовувати повний стек протоколів TCP/IP для пересилання даних;
- Необхідність у мережевому обладнанні.

4. USB

USB є одним з найбільш поширених інтерфейсів сучасних ПК (саме для цього створювався цей інтерфейс) [14, 19]. Сучасні мікроконтролери мають в собі реалізацію фізичного рівня інтерфейсу та мають необхідність лише у підключенні до них роз'єму USB, або навіть підключення провідників кабелю напряму до контролера. Не всі контролери підтримують цей інтерфейс, але існують спеціальні перетворювачі фізичного рівня для реалізації USB-інтерфейсу (Рис 2.7).

Переваги:

- Висока швидкість передачі даних;
- Підтримка усіма сучасними ПК;
- Універсальність;

- Можливість передавати дані практично у будь-якому вигляді;
- Наявність фізичної реалізації у мікроконтролерах.

Недоліки:

- Складність програмної реалізації інтерфейсу;
- Деякі мікроконтролери не мають фізичної реалізації. тому необхідно використовувати зовнішній перетворювач.

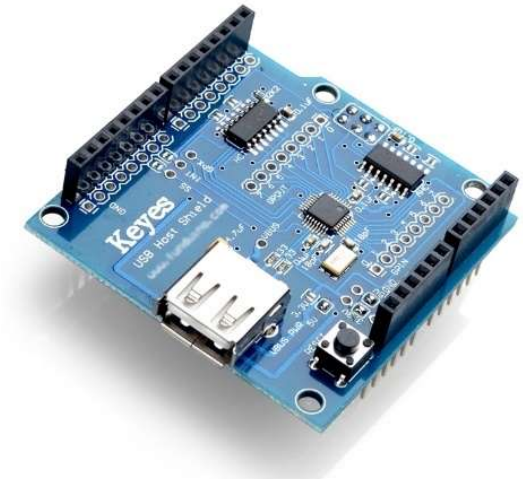


Рис 2.7 - USB-Host модуль

Але з інтерфейсом USB не все так просто. Через велику поширеність інтерфейсу є багато програмних стандартів, що працюють поверх нього [14, 15, 18]. З точки зору операційної системи будь-який USB-пристрій є зовнішнім пристроєм (для того ж UART пристрій перетворюється у файл для читання-запису), тому для його роботи потрібно використання драйверів. При цьому є два варіанти вирішення проблеми:

- Створення пропрієтарного драйверу

Для роботи з пристроєм необхідно написати спеціальний драйвер, що буде описувати специфікацію спілкування с пристроєм: набір команд, їх структура, розмір, кінцева точка пристрою, до якої спрямована ця команда.

Написання драйвера такого роду є дуже складною задачею, та ця задача ускладняється через те, що принцип написання драйвера буде змінюватися в залежності від операційної системи, що в основному зумовлює, чому виробники роблять драйвери, наприклад, лише для ОС Windows.

- Універсальні драйвери пристроїв для взаємодії з людиною

Для вирішення цих вищеперерахованих проблем є універсальні драйвери (драйвери USB-HID). Звичайно ці драйвери будуть працювати повільніше, аніж пропрієтарні, але використання технології HID значно покращує кросплатформеність пристрою, спрощуючи з ним роботу. Драйвери HID описуються специфікацією стандарту USB [15, 17], тому вони ведуть себе однаково на усіх операційних системах. Звичайно ж ОС мають в себе вбудовані драйвери HID, миші та клавіатури, що підключають по USB, теж є HID-пристроями.

З усього вищезазначеного можна розставити пріоритети для використання інтерфейсу у готовому пристрої:

1. USB-HID;
2. UART;
3. Мережеві інтерфейси (обидва Ethernet та WiFi);
4. USB драйвер.

В якості інтерфейсу зв'язку пристрою було обрано USB-HID, через свою гнучкість та розповсюдженість.

2.1.3. Набор команд для керування.

На відміну від кількості інтерфейсів, що було розглянуто, варіантів наборів команд не так багато. В якості основних можна розглянути два види наборів команд:

1. Пропрієтарні команди

В основному виробники принтерів використовують звичайні пропрієтарні драйвери зовнішніх пристроїв через інтерфейс USB. При цьому, звичайно ж, використовується власний набір команд для керування пристроєм, вони можуть бути закодовані або зашифровані.

Переваги:

- Можливість закодувати дані у зручному для обробки кінцевим пристроєм форматі;
- Відсутність прив'язки до існуючих стандартів.

Недоліки:

- Відсутність універсальності та відкритості.
- Необхідність перетворення мови опису креслення на набір команд.

2. G-код

G-код - умовна назва спеціалізованого язика програмування для пристроїв з числовим програмним керуванням [20, 21, 22, 23]. Саме в G-кодах генерують свої файли спеціалізовані САПР та програми для перетворення векторних зображень для друку на станках з числовим програмним керуванням.

Переваги:

- Стандартизований набір команд;
- Підтримка напряму САПР та спеціалізованим ПЗ

Недоліки:

- Необхідність виконувати дешифрування та декодування команд на самому мікроконтролері.

З вищезазначеного можна зробити висновок, що краще реалізувати передачу даних на пристрій у пропрієтарному вигляді, тому що це зменшить навантаження на мікроконтролер. Перетворення G-кодів на пропрієтарні команди можна виконати в програмному забезпеченні ПК.

2.1.4. Необхідні фреймворки та бібліотеки.

Для друку зображення у векторному форматі треба водночас змінювати положення друкуючої головки по двох осях, що звісно має відбуватися паралельно. Для цього можна використати потоки. При програмуванні мікроконтролера, на відміну від програмування звичайного комп'ютера, не можна просто підключити бібліотеку потоків, тому що на мікроконтролері немає такого поняття - потік, тому що на контролері відсутня операційна система. Цей ланцюжок роздумів призводить нас до використання спеціальних бібліотек, що формують операційну систему реального часу.

Для мікроконтролерів існує багато операційних систем реального часу. Розглянемо в якості прикладу дві ОСРЧ для мікроконтролерів:

- FreeRTOS

FreeRTOS - операційна система реального часу для вбудованих систем, що на цей час була реалізована для багатьох ядер мікроконтролерів, таких як Xtensa та Cortex-M. Доступна під ліцензіями MIT, а також для комерційного використання. FreeRTOS розробляли як просту і легку операційну система для вбудованих систем. Основною мовою реалізації є C. Ця ОСРЧ є саме лише системою реального часу. Вся робота по написанню драйверів пристроїв, протоколів взаємодії, ініціалізації обладнання покладається на програміста. [24, 25]

- Mbed OS

Mbed OS - програмно-апаратна платформа, та однойменна операційна система реального часу [26, 27]. вона створена спеціально для мікроконтролерів сімейства Cortex-M компанією ARM на базі власної ОСРЧ -

CMSIS-OS. Mbed оптимізована для роботи з ядрами Cortex-M та написана на мові C/C++.

Mbed OS є не тільки ОСРЧ - це повноцінна програмна-апаратна платформа, що має в собі зручний функціонал для роботи з портами введення-виведення, налаштування периферійних пристроїв, і що є дуже важливим, має вбудовану підтримку інтерфейсу USB-HID, абстрагуючи нас від роботи з дескрипторами та кінцевими точками інтерфейсу [27, 28].

Також є варіант не використовувати потоки взагалі. Можна зробити імітацію паралельного виконання за допомогою апаратних таймерів, тому що апаратна частина пристрою (двигуни та датчики) набагато повільніші за процесорну частину. Такий вид можна розглянути для контролерів, що не підтримують ОСРЧ.

Так як ми розглянули Mbed OS не тільки в якості ОСРЧ, а як й фреймворк, тоді розглянемо іншій розповсюджений фреймворк - Arduino.

Arduino фреймворк є надзвичайно популярним в середі починаючих розробників електроніки. Ядро цього фреймворку має відкритий початковий код, та розповсюджується за вільною ліцензією, тому воно було переписано для багатьох інших мікроконтролерів. Велика кількість бібліотек, що створені іншими розробниками в якості відкритого коду [29, 30].

Звичайно ж будь-який мікроконтролер підтримує розробку на самому низькому рівні - з регістрами, та на рівні апаратних абстракцій, але це дуже ускладнює розробку, особливо при реалізації інтерфейсу USB, тому даній варіант може бути використаний поза фреймворком для оптимізації критичних ділянок коду.

Вибір операційної системи реального часу та фреймворку в основному буде залежати від вибору мікроконтролеру, для якого буде писатися програмне забезпечення.

2.1.5. Вибір мікроконтролера.

Розглянемо розповсюджені мікроконтролери, та порівняємо їх по параметрам, що нам необхідні, та були визначені в попередніх розділах.

- Arduino

Arduino - надзвичайно популярна програмно-апаратна платформа основними компонентами якої є плата мікроконтролера з елементами вводу/виводу та середовище розробки Processing/Wiring на мові програмування, що є спрощеною підмножиною C/C++ [29, 30].

Найпопулярнішою платою з цього сімейства - Arduino UNO, що побудовано на мікроконтролері ATmega328p.

Відповідність контролера виставленим вимогам [31]:

- Просте керування портами введення-виведення, підтримка апаратних переривань, наявність генерації ШИМ-сигналу
- Підтримуємі інтерфейси - UART, Ethernet через SPI, WiFi через UART, USB через зовнішні модулі
- Не підтримує ОСРЧ. Можлива реалізація через таймери. Фреймворк Arduino.
- Велика кількість бібліотек від інших користувачів.

- ESP

Сімейство контролерів ESP від китайського виробника Espressif. Основною властивістю контролера є вбудований WiFi. Популярними контролерами є ESP8266 та ESP32.

Розглянемо ESP32, як найбільш потужний.

Відповідність вимогам [32]:

- Просте керування портами введення-виведення, підримка апаратних переривань, наявність генерації ШИМ-сигналу
- Підримуємі інтерфейси - UART, Ethernet через SPI, WiFi
- FreeRTOS, фреймворк Arduino.

- Велика кількість бібліотек від інших користувачів.

➢ STM32

Сімейство контролерів від STMicroelectronix є дуже великим, вони відрізняються один від одного за кількістю вбудованої переферії, типом ядра, пам'яттю та іншим. Всі ці контролери побудовані на ядрі Cortex-M різних версій.

Найпопулярнішим контролером з цих є контролер STM32F103CCU6, але я розгляну два контролери, цей та інший, що встановлено на спеціальну відлагоджувальну плату.

STM32F103CCU6 [33]:

- Просте керування портами введення-виведення, підримка апаратних переривань, наявність генерації ШИМ-сигналу
- Підтримуємі інтерфейси - USB, UART, Ethernet через SPI, WiFi через UART
- Arduino, Mbed OS, FreeRTOS
- Бібліотеки від виробників пристроїв, та інших користувачів.

STM32F767ZI на відлагожувальній платі Nucleo-F767ZI [34]:

- Просте керування портами введення-виведення, підримка апаратних переривань, наявність генерації ШИМ-сигналу
- Підтримуємі интерфейси - USB, UART, Ethernet, WiFi через UART та SDIO
- Arduino, Mbed OS, FreeRTOS
- Бібліотеки від виробників пристроїв, та інших користувачів.

В ході порівняння цих мікроконтролерів. перевірки вимог та наявних технічних рішень, було прийнято рішення використати відлагоджувальну плату Nucleo-F767ZI (Рис 2.8) за рядом причин:

- Наявність фізичного інтерфейсу USB

- Наявність фреймворку зі вбудованими драйверами роботи USB-HID

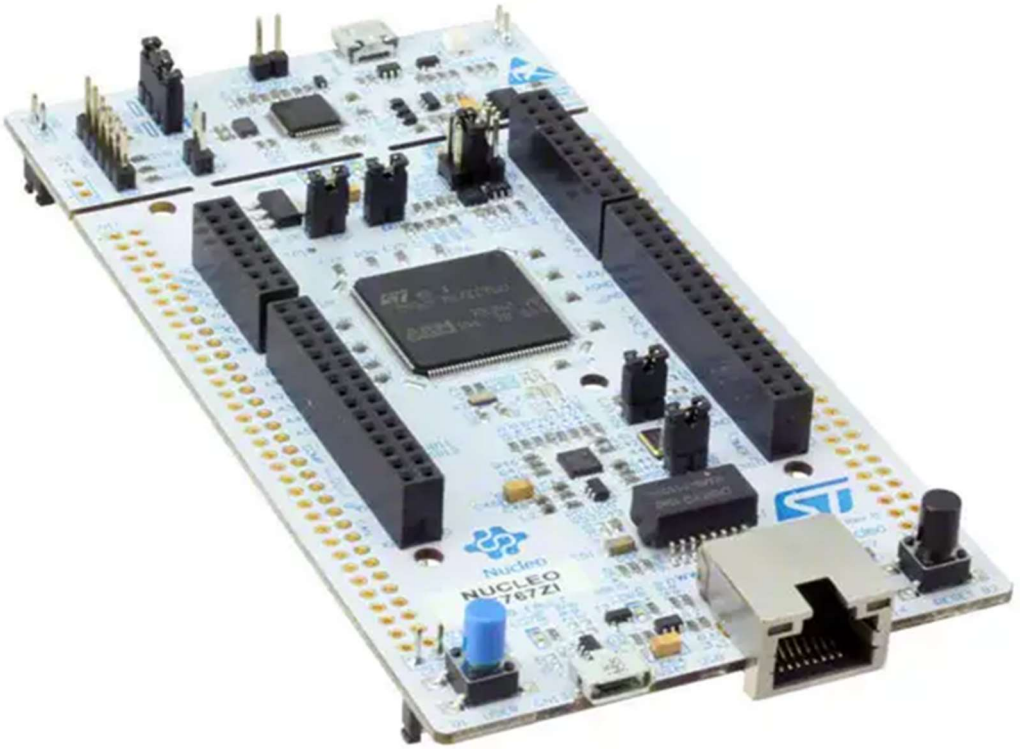


Рис 2.8 - Плата Nucleo-F767ZI [34]

Для програмування мікроконтролера був обрано фреймворк та операційну систему реального часу Mbed OS. Середовище розробки Mbed Studio. Мова програмування C++.

2.2. Вибір засобів для розробки програмного забезпечення на ПК.

Основними властивостями, які мають бути у мови для написання частини ПЗ, що буде виконуватися на ПК, є:

- Кросплатформеність
- Наявність бібліотек для роботи з USB-HID
- Можливість обробки великих масивів даних для перетворення G-кодів

Для задовольняння першому пункту потрібно, щоб виконання програми було незалежно від операційної системи. В такому випадку доцільно використовувати мови, що інтерпретуються, наприклад, чисто скриптові мови або мови, що використовують технологію Just-in-time.

Мабуть практично усі мови програмування мають в себе бібліотеки для роботи з USB-HID, хоча не завжди вони вбудовані в стандартну бібліотеку.

Мова повинна мати в себе можливості або бібліотеки для обробки великих масивів даних.

З урахуванням попередніх міркувань було обрано мову програмування Python. Середовище розробки не має значення, можливо навить написання програми в простому текстовому редакторі.

ВИСНОВОК ДО РОЗДІЛУ 2

У цьому розділі було розглянуто та порівняно технології, що можна використати для створення графопобудовника. Було визначено необхідні спроможності мікроконтролера, на якому буде базуватися пристрій; визначено інтерфейс взаємодії з комп’ютером, тип команд, за якими буде відбуватися керування. Було обрано необхідні фреймворки, та визначено мову написання програмного забезпечення для мікроконтролера.

Також були розглянути загальні риси, якім має відповідати мова програмування, на якій буде написано керуюче програмне забезпечення, та обрано саму мову програмування.

РОЗДІЛ 3. РОЗРОБКА АПАРАТНОЇ ТА ПРОГРАМНОЇ ЧАСТИНИ

3.1. Розробка апаратної частини

Як вже було зазначено раніше в другому розділі, графопобудовник, як зовнішній пристрій, має в собі не тільки програмну, але й апаратну частину, що складається з приводів осей друку, друкуючої головки та мікроконтролерної частини, що контролює виконує керування приводами.

3.1.1. Опори приводів осей.

Більша частина пристрою складається з готових модулів, таких як плата мікроконтролера, серводвигун, драйвери крокових двигунів, кінцеві вимикачі, але саме опори осей приводу є нестандартними елементами, тому вони були розроблені окремо.

Розробка моделей цих елементів проводилась в програми Autodesk Fusion 360 (Рис 3.1). Після розробки деталей їх було надруковано з використанням 3D-принтеру.

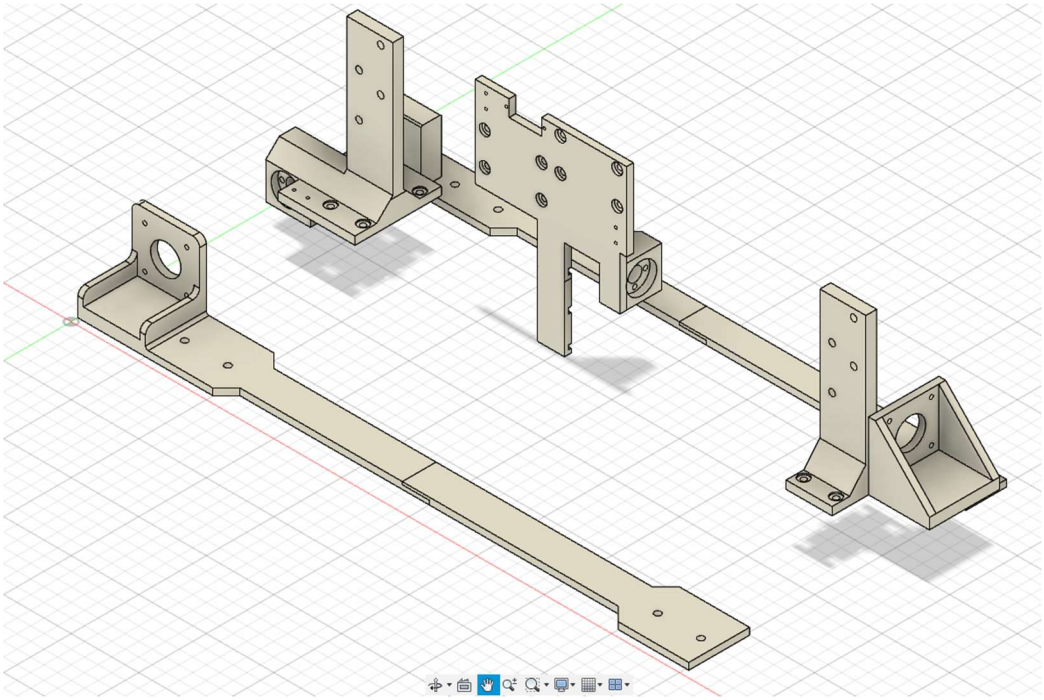


Рис 3.1 – Модель опорів приводів осей

3.1.2. Механізм приводу осей

Для того щоб призвести до руху друкуючу головку, необхідно перетворити обертальних рух осі двигуна в поступовий рух приводів осей та друкуючої головки. Для цього можливі два варіанти:

- Використовувати ремінну передачу (Рис 3.2), як найбільш очевидний спосіб передачі руху. Цей метод ускладнює конструкцію через велику кількість додаткових деталей для кріплення ременів.

Рис 3.2 – Приклад ремінної передачі

- Використовувати приводні вали (Рис 3.3 та 3.4). Цей спосіб дозволяє перетворити рух в наперед заданому співвідношенні кількості кроків на одиницю відстані, що проходить друкуюча головка, за простим співвідношенням. Проста реалізація без додаткових деталей в опорі приводу осей.

Рис 3.3 – Приводной вал з втулкою

Рис 3.4 – З’єднання валу з віссю двигуна

Як було зазначено, використання приводного валу дозволяє визначити точне відношення кількості кроків на одиницю відстані зсуву друкуючої головки. Розрахуємо це за формулою (1).

$$\Delta l = \frac{\Delta s * \varphi}{360°} \quad (1)$$

, де Δl – зсув за один крок, мм;

Δs – прохід приводного валу за один повний оберт, мм;

φ – кут, на який обертається вал крокового двигуна за один крок, °.

В даному випадку було використано кроковий двигун з кроком $\varphi = 1.8°$, приводной вал з проходом $\Delta s = 8$ мм, тому зсув на один крок складає 0,04 мм.

В графопобудовнику використовуються дві приводні осі, та обох з них розрахунки зсуву за один крок є справедливими через використання одного й того ж типу двигуна та приводних валів.

3.1.3. Підключення драйверів крокових двигунів.

Як було зазначено в другому розділі для керування кроковими двигунами необхідно в будь-якому разі використовувати драйвери крокових двигунів.

Для створення графопобудовника було обрано двигуни з напругою живлення від 5 до 24 вольт, та номінальним струмом обмотки 1,7 ампер, тому потрібно використати драйвер двигуна, що дозволяє працювати з ним. Було обрано драйвер TB6600 (Рис 3.5), що підтримує напруги до 45 вольт та струм до 4,5 ампер [37].

Рис 3.4 – Драйвер TB6600

Для керування драйвером потрібні три керуючих сигнали, що не потребують використання ШІМ, тому на платі мікроконтролера можемо обрати будь-які порти введення виведення. Нехай для першого драйвера це будуть EN – PF_13, DIR – PE_9, STEP – PE_11, та для другого: EN – PF_14, DIR – PE_13, STEP – PF_15 [35, 36]. Значення назв керуючих сигналів було описано в розділі 2 пункт 1.1. Вибір цих портів введення-виведення базується лише на тому, що вони знаходяться близько один до одного на платі, тому їх буде зручно підключати.

Для того, щоб підключити драйвер на нього також треба подати живлення 5 вольт, що буде передано з плати мікроконтролера, та напруга живлення двигунів, що буда надаватися з зовнішнього потужного блока живлення. Та не слід забувати про те, що мікроконтролер та драйвер двигуна мають мати спільну землю живлення для правильної інтерпретації логічних рівнів.

Для підключення крокового двигуна на драйвері є спеціальні клеми з назвами A+, A-, B+, B-, що відповідають контактам обмоток двигуна. Підключення відбувається згідно з документацією на двигун, та неправильне підключення може призвести до того, що двигун буде обертатися не в той бік, або не буде обертатися взагалі.

3.1.4. Підключення серводвигуна, кінцевих вимикачів та роз’ємів USB.

Для підключення та керування серводвигуном потрібен порт, що має підтримку генерації ШІМ-сигналу, візьмемо для цього порт PD_15 [36]. Напруга живлення складає 5 вольт, тому живлення буде підключено до плати мікроконтролера.

Для підключення кінцевого вимикача потрібно використовувати лише два контакти: спільну землю, та нормально відкритий контакт. Для того, щоб обробити подію знаходження початкової точки потрібно створити переривання на мікроконтролері.

Мікроконтролери сімейства STM32 мають можливість налаштувати

переривання на будь-якому порту введення-виведення, з деякими внутрішніми обмеженнями [34]: переривання встановлюються на лінії портів введення-виведення, по одному на кожну лінію з шістнадцяти, тому ми не можемо налаштувати більш ніж шістнадцять переривань з зовнішніх портів одночасно, що в нашому випадку не заважатиме.

Для того, щоб визначити початкове положення друкуючої головки, потрібно мати лише два кінцевих вимикача, по одному на кожну вісь з боку початкової точки, тому буде підключено лише два вимикачі. Порти для переривання - PA_5 – для осі X, PA_6 – для осі Y.

Однією з основних причин, за якою було обрано саме відлагоджувальну плату було те, що вона має в собі встановлений роз'єм USB для роботи в режимі пристрою [35, 36]. USB є дуже складним протоколом не тільки з програмного боку, а й з апаратного: USB для передачі даних використовує диференціальні пари, що мають складні вимоги для розробки підключення, та розведення друкованої плати [16, 17, 18]. Звичайно ж можна підключити роз'єм напряму, та це навіть буде працювати, але диференціальний сигнал використано не просто так, я для зменшення впливу зовнішнього електромагнітного випромінювання на сигнал [38], тому пряме підключення роз'єму до контролера не є доцільним без використання складної обв'язки інтерфейсу USB.

Для того, щоб налагодити пристрій буде використано другий роз'єм USB, що встановлено на програматорі ST-Link v2.1, вбудованому в плату мікроконтролера [35, 36]. Через цей роз'єм буде відбуватися прошивка мікроконтролера, та спілкування з ним через віртуальний COM-порт, для визначення стану пристрою під час розробки.

3.2. Розробка програмної частини. Програмне забезпечення для ПК.

3.2.1. Поняття G-коду. Основні команди, необхідні для роботи.

В другому розділі було визначено, що краще за все буде виконати перетворення G-коду, що описує зображення для роботи на ЧПУ-пристрої, на набір команд для роботи зовнішнього пристрою на самому комп'ютері, для того щоб зменшити навантаження на контролер.

Перш за все розглянемо саму систему запису програми для ЧПУ-пристроїв у форматі G-кодів. Зазвичай виробники пристроїв використовують основну систему G-коду зі своїми доопрацюванням під конкретні пристрої [20, 21].

G-код де-факто є своєрідною мовою програмування. Програми, що написані цією мовою мають певну структуру: всі команди об'єднуються в групи по декілька (зазвичай 4) команд в кадри, що в свою чергу розділяються за допомогою перенесення строки [20].

Перший та останній кадри команди складаються з однієї команди - %. В основному програма завершується командами M02 або M30.

Коментарі розташовуються після коду всієї програми та в окремих кадрах в самій програмі. Вони обмежуються круглими дужками.

Порядок команд в кадрі не є строгим, але традиційно йдуть в такому порядку: команда підготовки, переміщення, вибір режиму та технологічні команди.

Розглянемо основні команди, що будуть підтримуватися програмним забезпеченням пристрою [21]:

- G00 – холостий хід до заданої в параметрах координати;
- G01 – лінійна інтерполяція (друк прямої);
- G02 – інтерполяція за колом в напряму годинникової стрілки;
- G03 - інтерполяція за колом проти напряму годинникової стрілки.

Примітка – в стандарті G-коду існує більше команд, але в межах даного пристрою вони не мають сенсу, такі як нарізання різьби, зміна системи координат або зміна координатної площини.

- M02, M30 – кінець програми;
- M03, M04 – почати роботу інструмента;
- M05 – завершити роботу інструмента.

Примітка – команд класу M також існує більше, але вони теж не всі мають сенс для роботи пристрою, наприклад, заміна інструмента, додаткове охолодження. Команди M02 та M30 відрізняються тим, що M02 завершує роботу без скидання налаштувань. В даному випадку ніяких додаткових налаштувань не передбачено, тому команди не відрізняються. Команди M03 та M04 відрізняються напрямом обертання інструмента під час роботи, що в даному випадку також не несе ніякої різниці.

Також слідує загострити увагу на том, що команди M можуть бути лише одна на один кадр, та й на тому, що старші нулі в номері команди може бути опущено, тобто команди M02 та M2 є однаковими.

Команди типу G мають в себе параметри з якими вони працюють [21]:

- X – кінцева координата осі X;
- Y – кінцева координата осі Y;
- Z – кінцева координата осі Z;
- I – відстань від центра дуги окружності по осі X;
- J – відстань від центра дуги окружності по осі Y;
- K – відстань від центра дуги окружності по осі Z;
- F – швидкість руху інструмента.

Хоча і здається, що параметри Z та K зайві, але вони використовуються для того, щоб надати нам інформацію про те, чи ми друкуємо, або лише рухаємося.

Одиниці виміру для параметрів можна міняти, але це не має сенсу, тому що при генерації коду в САПР, або генераторах кодів з зображень можна обрати одиниці виміру. Стандартною одиницею є міліметр. Числа пишуться одразу після позначення параметра без пробілів. Між параметрами встановлюється пробіл.

Приклад програми (Рис 3.6), що друкує зображення (Рис 3.5).

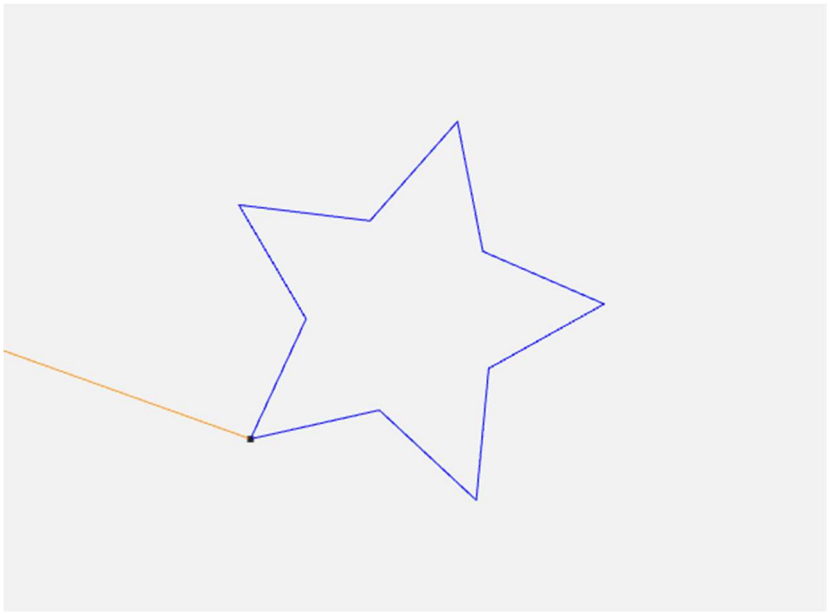


Рис 3.5 – Зображення, що описано G-кодом

GCode File

```
%
(Header)
(Generated by gcodetools from Inkscape.)
(Using default header. To add your own header create file "head
M3
(Header end.)
G21 (All units in mm)

(Start cutting path id: path10)
(Change tool to Cylindrical cutter)

G00 Z5.000000
G00 X77.508164 Y217.086413

G01 Z-1.000000 F100.0(Penetrate)
G01 X55.178286 Y227.441617 Z-1.000000 F400.000000
G01 X33.988766 Y214.917320 Z-1.000000
G01 X36.936838 Y239.354230 Z-1.000000
G01 X18.477602 Y255.636440 Z-1.000000
G01 X42.629489 Y260.384077 Z-1.000000
G01 X52.410573 Y282.971334 Z-1.000000
G01 X64.389188 Y261.468625 Z-1.000000
G01 X88.893467 Y259.146107 Z-1.000000
G01 X72.144771 Y241.109065 Z-1.000000
G01 X77.508164 Y217.086413 Z-1.000000
G00 Z5.000000

(End cutting path id: path10)

(Footer)
M5
G00 X0.0000 Y0.0000
M2
(Using default footer. To add your own footer create file "foot
(end)
%
```

Рис 3.6 –G-код програма, що описує попереднє зображення

3.2.2. Перетворення G-коду на керуючі команди

Як можна було помітити під час розгляду синтаксису G-коду, він має спільні риси з мовами асемблера, тому найбільш логічним рішенням буде написати транслятор з мови ЧПК на мову, яку зрозуміє контролер графопобудовника, тобто створити таку мову опису руху та запитів до ПЗ комп'ютера.

Описана вище підмножина мови G-коду є дуже спрощеною, тому для реалізації транслятора можна буде використати лише основні елементи компіляторів: лексер, парсер та кодогенератор. Також можна додати до цього препроцесор, що видалить коментарі та вони не будуть заважати трансляції програми на мову графопобудовника.

Таким чином, буде виконано трансляцію з файлу інструкцій G-коду до легкого у використанні для обробки мікроконтролером набору команд – свого роду система Just-in-time компіляції, в якій транслятор з G-коду, який може бути виконано на будь-якій машині, відправить на виконання зовнішньому пристрою набір команд. Подібний принцип можна побачити, наприклад, при роботі з мовою Java.

Зауваження: тут та далі вважається, що файл G-коду для друку зображення генерується автоматично у спеціалізованому програмному забезпеченні, тому він не може мати лексичних або синтаксичних помилок.

3.2.2.1. Препроцесор

Препроцесором є така програма, або частина програми, що виконує попередню підготовку програми до трасляції або компіляції. В нашому випадку препроцесор буде використано для видалення коментарів, та зайвих перенесень строк, для того щоб спростити етапи лексичного та синтаксичного аналізу коду.

Алгоритм роботи препроцесору є дуже простим:

1. Знайти за допомогою регулярного виразу весь текст, що знаходиться в дужках (коментарі) та видалити його;

2. Знайти в коді пусті строки, та видалити їх.

Після виконання препроцесором обробки вхідного тексту отримуємо готовий до лексичного аналізу текст програми.

3.2.2.2. Лексичний аналіз

Лексичний аналіз, він же токенізація, - процес обробки вхідного тексту, розбиття його на окремі групи – лескеми, для отримання послідовності токенів, та визначення типів токенів [39, 40].

Токеном називається така група символів, що відповідає певним шаблонам.

Згідно з визначеною раніше потрібною для роботи пристрою підмножиною мови G-код, отримуємо такі типи токенів:

1. Токени початку та кінця програми: «%»;
2. Токен роздільника кадрів: «\n»;
3. Токен команди, що відповідає шаблону: «A00», де A – тип команди (G або M), 00 – код команди;
4. Токен параметру, що відповідає шаблону: «P000.00», де P – ідентифікатор параметра (X, Y, Z, I, J, K або F), 000.00 – числове значення відповідного параметра, що може бути як цілим числом, так і числом з плаваючою комою, з символом десяткового роздільника – «.», у випадку від’ємного числа можлива наявність знаку «-» перед числовим значенням.
5. Токен, що не підтримується – Такий токен, що не входить до підмножини, необхідної для роботи програми.

Результатом роботи лексичного аналізатора є послідовний набір токенів, що повністю описує зображення у вигляді G-коду, а також є готовим для обробки синтаксичним аналізатором.

3.2.2.3. Синтаксичний аналіз

Синтаксичний аналіз, також має назву парсінг, - процес аналізу граматичних структур вхідного тексту для пошуку граматичних структур. Під час синтаксичного аналізу послідовний набір токенів перетворюється в зручну для обробки структуру, зазвичай дерево синтаксичного розбору [39, 40].

G-код не має строгої ієрархії команд, як наприклад у мовах високого рівня, лише розділення команд по кадрах, тому процес синтаксичного аналізу є дуже простим:

1. Визначити за типом токена клас команди, якщо клас M, тоді додати команду у дерево і перейти да наступного кадру, якщо команда не підтримується, то пропустити її.
2. Якщо команда належить до класу G, тоді проаналізувати параметри команди, записати параметри в словник, за їх типом.

В нашому випадку доцільніше буде генерувати не дерева синтаксичного розбору, а масив словників, тому що, як вже було сказано, G-код не має строгої ієрархії команд та параметрів. Наприклад перетворення команди “G00 X0 Y0” буде виглядати так: {cmd:”G00”, x:”0”, y:”0”}, а команда “M02” – {cmd:”M02”}

3.2.2.4. Генерація кода

Після того, як початковий G-код було перетворено у масив словників команд, згенерувати набір команд для роботи графопобудовника є простою задачею:

1. Пройти по кожному елементу масиву команд.
2. Для кожної команди зі словнику поставити у відповідність набор

байтів.

3. Якщо команда не має параметрів, тоді заповнити повідомлення до кінця нулями.

4. Якщо команда є складною для виконання графопобудовником напряму (друк кола), тоді виконати її лінійну інтерполяцію, перетворивши дугу кола на прямі відрізки. Таким чином команди інтерполяції по колу буде перетворено у набір команд лінійної інтерполяції.

5. Якщо параметри наявні, то записати їх послідовно в такому порядку: X, Y, Z, F – при цьому параметри з міліметрів перетворюються на кроки відносно кінцевої точки попередньої команди для того, щоб прибрати дробі з параметрів, заповнити повідомлення до кінця нулями.

Після завершення процесу кодогенерації, набор команд для виконання графопобудовником є готовим до передачі через USB.

3.2.3. Кодування команд для графопобудовника.

Для того щоб передати дані на пристрій потрібно визначити кодування, за яким буде відбуватися передача даних. У свою чергу для цього треба визначити кількість команд, що буде передаватися й розміри параметрів, для визначення кінцевої довжини повідомлення.

Підмножина G-коду, яку було визначено, складає дев’ять команд, але при перетворенні на команди ми замінюємо команди інтерполяції за колом і деякі команди мають однаковий сенс, тому набір команд зменшується до 5. Для того, щоб повідомити ПК інформацію про готовність прийняти наступну команду на друк, потрібна одна команда, таким чином набір команд складатиме 6 команд. Закодуємо одну команду одним байтом.

Максимальна довжина поля параметра складатиме:

$$L = 1 + \lceil \log_2 l/\Delta l \rceil \qquad (2)$$

, де L – довжина команди, біт,

l – довжина аркуша (розмір поля для друку), мм,

Δl – розмір одного кроку, мм.

За розрахунками по формулі (2) отримаємо розмір поля 14 біт, для зручності округлюємо до двох байтів. Один біт у формулі додається для передачі знаку результату.

Таким чином одна команда для графопобудовника матиме довжину дев’ять байт.

Таблиця 3.1 – Відповідність команд і коду команди

Команда	Код
G00	0000 0000
G01	0000 0001
M02, M30	0000 0010
M03, M04	0000 0011
M05	0000 0100
ANS	1111 0000

Наприклад, команду “G01 X10 Y10 Z0 F400” буде перетворено у вигляд: “0000 0001 0000 0000 0000 1010 0000 0000 0000 1010 0000 0000 0000 0000 0000 0001 1001 0000” або 0x01000A000A00000190.

3.2.4. Пересилання даних.

Для того, щоб отримати доступ до системного драйвера USB-HID потрібно використати спеціальну бібліотеку, тому що вбудованих засобів для роботи з зовнішніми пристроями HID в мові Python немає. Для реалізації даного функціоналу було обрано бібліотеку hid [41, 42], яку можна легко встановити командою:

$pip install hid

Увага, ця бібліотека має залежність, що дозволяє їй працювати кросплатформено! Повну інформацію щодо встановлення залежності можна знайти за посиланням https://github.com/libusb/hidapi [42].

Ця бібліотека є дуже простою для роботи з зовнішніми пристроями. Для того, щоб відкрити з'єднання з пристроєм потрібно задати конструктору об'єкта *hid.Device()* vid (ідентифікатор виробника) та pid (ідентифікатор пристрою) пристрою. Після створення об'єкту для того щоб переслати дані на пристрій необхідно використати метод *write()*, для отримання даних, відповідно, *read()* [42].

3.3. Розробка програмної частини. Програмне забезпечення для МК.

Для роботи програмної частини на МК потрібно, як вже було зазначено, потрібно використання операційної системи реального часу. В якості ОСРЧ та фреймворку було обрано MbedOS.

3.3.1. робота за портами введення-виведення

MbedOS дає розробникам дуже зручний інтерфейс роботи з периферійними пристроями процесора [43], та фактично є заздалегідь налаштованою обгорткою для набору бібліотек HAL.

Для роботи графопобудовника нам потрібно налаштувати порти у режимі цифрового виведення, ШІМ-генератора та введення з перериванням.

Режим цифрового виведення налаштовується за допомогою класу *DigitalOut()* [44], що приймає ідентифікатор порту, наприклад PF_15. Для того щоб вивести значення достатньо присвоїти об'єкту порту введення виведення потрібне значення, наприклад:

DigitalOut pin(PF_15); //налаштувати PF_15 як цифровий вивід

pin = 1; // вивести логічну одиницю

Для генерації ШІМ-сигналу потрібно використати інший клас – *PwmOut()* [45], що ініціалізується також як цифровий вивід. Для роботи з ШІМ-виходом використовуються спеціальні методи: *period()* і *write()*. Але так как ми використовуємо ШІМ-вивід для контролювання серводвигуна, ми можемо використати бібліотеку для роботи з серводвигунами, наприклад, https://os.mbed.com/cookbook/Servo [46].

Приклад роботи:

```
Servo myservo(PD_10);
myservo = 0.4;
```

Для того щоб реагувати на переривання потрібно використати третій клас – *InterruptIn()* [47], що також приймає порт, на який потрібно налаштувати переривання. Для того щоб встановити переривання, необхідно викликати функцію *fall()* або *rise()*, що приймають вказівник на функцію, яка обробляє переривання.

Приклад роботи з офіційної документації [47]:

```
#include "mbed.h"

InterruptIn button(SW2);
DigitalOut led(LED1);
DigitalOut flash(LED4);

void flip()
{
   led = !led;
}

int main()
{
   button.rise(&flip);
   while (1) {
      flash = !flash;
       ThisThread::sleep_for(250);
   }
}
```

3.3.2. Потоки в MbedOS

Потоки в MbedOS маюсь синтаксис подібний до потоків в мові програмування C++ [48]. Для створення потоку використовується клас під назвою *Thread* [49]. Для того почати виконання потоку потрібно передати йому вказівник на функцію, яку потрібно запустити у потоці, за допомогою метода *start()*. Якщо потрібно зачекати на виконання потоку, тоді потрібно викликати метод *join()*.

Приклад роботи з потоками з офіціальної документації [49]:

```
#include "mbed.h"

DigitalOut led1(LED1);
DigitalOut led2(LED2);
Thread thread;

void led2_thread()
{
    while (true) {
        led2 = !led2;
        ThisThread::sleep_for(1000);
    }
}
int main()
{
    thread.start(led2_thread);
    while (true) {
        led1 = !led1;
        ThisThread::sleep_for(500);
    }
}
```

3.3.3. USB-HID

З використанням класу USBHID зі стандартного набору бібліотек MbedOS можна з легкістю перетворить пристрій на основі MbedOS в зовнішній пристрій HID, та пересилати й отримувати повідомлення через шину USB [50].

Для ініціалізації USBHID потрібно створити об’єкт відповідного класу. Конструктор цього класу потребує наступних параметрів:

- Режим блокування
- Довжина вхідного повідомлення
- Довжина вихідного повідомлення
- VID
- PID
- Версія пристрою

Прийом і передачу даних слід починати тільки після того, як пристрій ініціалізується, для цього треба використати метод *wait_ready()*, що припинить роботу пристрою поки він не запустить інтерфейс.

Прийом і відповідно передача даних можуть виконуватися у двох режимах: блокуючому та ні. У блокуючому пристрій призупиняє свою роботу, поки не відбудеться повна передача пакета даних, у неблокуючому, відповідно, ні.

Ця бібліотека також має розширені функції, такі як ручне редагування дескриптору пристрою, налаштування кінцевих точок та інше, але вони не потрібні для виконання дипломного проєкту [50].

Приклад роботи з бібліотекою з офіційної документації [50]:

```
#include <stdio.h>
#include "mbed.h"
#include "usb/USBHID.h"
// Declare a USBHID device
USBHID HID(true, 8, 8, 0x1234, 0x0006, 0x0001);
```

```
HID_REPORT output_report = {
    .length = 8,
    .data = {0}
};
HID_REPORT input_report = {
    .length = 0,
    .data = {0}
};
DigitalOut led_out(LED1);
int main(void)
{
    while (1) {

        // Fill the report
        for (int i = 0; i < output_report.length; i++) {
            output_report.data[i] = rand() & UINT8_MAX;
        }

        // Send the report
        HID.send(&output_report);

        // Try to read a msg
        if (HID.read_nb(&input_report)) {
            led_out = !led_out;
            for (int i = 0; i < input_report.length; i++) {
                printf("%d ", input_report.data[i]);
            }
            printf("\r\n");
```

```
        }
    }
}
```

3.3.4. Бібліотека для роботи з драйвером крокового двигуна

Як вже було сказано, мікроконтролер не може працювати з двигунами напряму через великий струм та напругу, що потребує двигун. А саме: двигун потребує від 5 до 24 вольт і 1,7 ампера для нормальної роботи, а мікроконтролер на своєму порту введення-виведення видає 3,3 вольт і не більш ніж 25 міліампер на один порт, та не більше 120 міліампер взагалі. Через все це ми використовуємо спеціальні мікросхеми (в даному випадку модулі) драйверів крокового двигуна.

Для виконання дипломного проєкту було обрано драйвер TB6600, якій згідно зі своїми характеристиками підтримує двигуни, що було встановлено в пристрій. За документацією на цей драйвер стає відомо, що активним рівнем для цього драйверу є рівень землі, тобто його логіка роботи інверсна.

Фреймворк Mbed є дуже поширеним і має велику підтримку з боку інших користувачів та виробників пристроїв, тому в репозиторії фреймворку вже є потрібна бібліотека для керування саме таким драйвером двигунів [51].

Бібліотека має дуже простий інтерфейс для взаємодії з драйвером, розглянемо її заголовний файл, для того щоб вивчити методи роботи з драйвером [51]:

```
#ifndef STEPPERTB_H
#define STEPPERTB_H

#include "mbed.h"

class StepperTB {
  public:
  /*ENA = ENA-
```

```
*DIR = DIR-
*PUL = PUL-
*/
StepperTB(PinName ENA, PinName DIR, PinName PUL, int Microstep, int StepPerRev);
void MoveOneStep(bool Direction, int Interval);
void MoveStep(int StepAmt, int Interval);
void MoveRev(int RevAmt, int Interval);
//Getter
//StepPerRev
int getSPR();
//Microstep
int getMstep();

private:
//Pins
DigitalOut ENA_;
DigitalOut DIR_;
DigitalOut PUL_;

int Microstep_;
int StepPerRev_;

};

#endif
```

Дійсно бібліотека є дуже простою для використання: для того щоб створити об’єкт драйвера необхідно задати лише три керуючих порти, кількість мікрокроків на один крок (в нашому випадку не використовуються

мікрокроки, тому встановлюємо одиницю) і кількість кроків на один оберт двигуна (в нашому випадку 200).

Для того, щоб виконати один крок необхідно використати метод *MoveOneStep()*, що приймає напрямок обертання і часовий інтервал, з яким робиться один крок. Щоб виконати одразу ж серію кроків у потрібному напрямі є метод *MoveStep(),* що приймає кількість кроків (позитивна, якщо за годинниковою стрілкою, та негативна, якщо проти) і часовий інтервал на один крок. Для виконання одразу повного оберту є метод *MoveRev(),* який, відповідно, приймає кількість обертів і часовий інтервал [51].

3.3.5. Декодування команд

Через те, що перетворення G-коду було виконано одразу ж до передачі даних на пристрій, то процес декодування значно спрощується.

При отриманні повідомлення ми маємо доповідь, що складається з дев’ятьох байтів зміст яких було описано в пункті 3.2.3. Для виконання друку залишається лише передати потрібні параметри в методи роботи з драйвером, розташувавши їх у двох незалежних потоках, для того щоб паралельно рухати друкуючу головку за двома осями. Після виконання команди необхідно в зворотному репорті відправити байт відповіді 0xF0.

ВИСНОВОК ДО РОДІЛУ 3

В цьому розділі було виконано створення апаратної та програмної частини пристрою, а саме розроблено опори приводів осей, обрано оптимальній варіант приводу, визначено порти введення-виведення мікроконтролера, до яких буде підключено кожен елемент пристрою.

Було розглянуто програмні засоби, що використано в написанні програмного забезпечення для персонального комп’ютера, з якого виконується керування пристроєм, а також для мікроконтролера, що керує апаратною частиною графопобудовника.

РОЗДІЛ 4. РОБОТА С ПРИСТРОЄМ. ТЕСТУВАННЯ

Роботу на графопобудовником завершено (Рис 4.1). Виконаємо тестування роботи пристрою на простому прикладі, та розглянемо процес роботи з ним по етапах, для того щоб надрукувати просте векторне зображення.

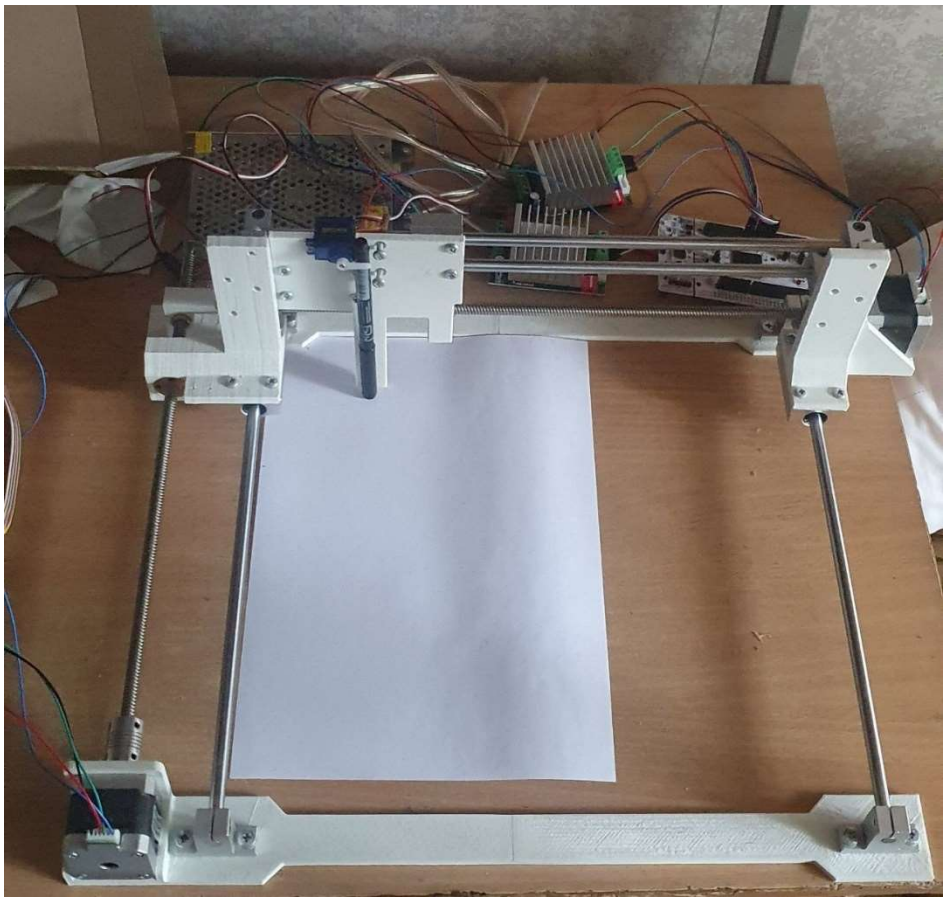


Рис 4.1 – Пристрій у зібраному стані

4.1. Підготовка векторного зображення. Генерація G-коду.

Векторне зображення, яке треба надрукувати на графопобудовнику необхідно по перше перетворити на G-код. Для цього зазвичай використовують спеціалізоване програне забезпечення по типу САПР, або редакторів векторної графіки. Розглянемо генерацію на прикладі ПЗ Inkscape.

Нехай в нас є зображення (Рис 4.2).

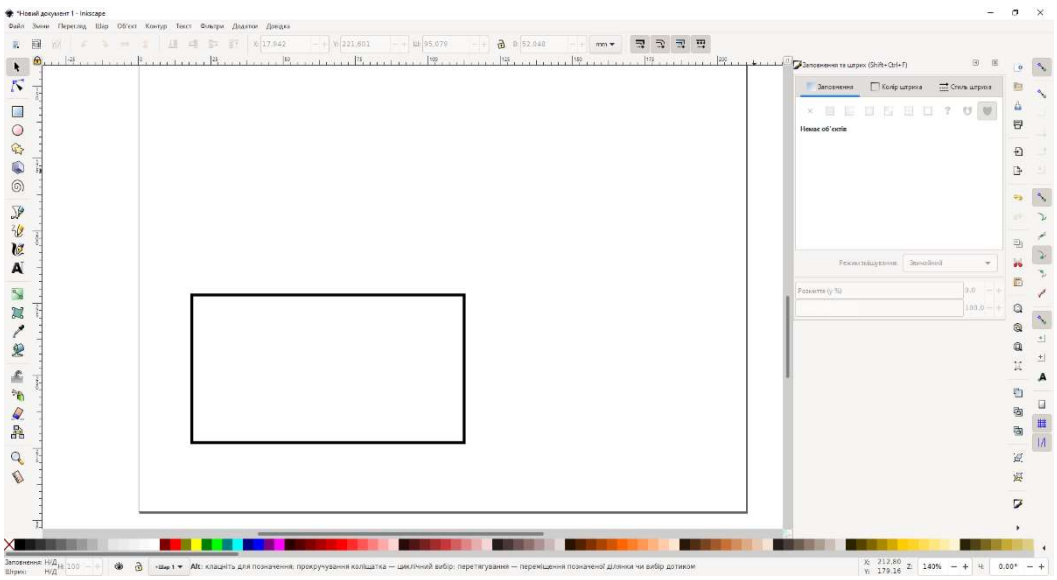


Рис 4.2 – Скріншот програми Inkscape з тестовим зображенням

Для генерації G-коду необхідно виконати наступні етапи:

1. Виконати команду «Об’єкт у контур», що знаходиться в меню «Контур» (Рис 4.3), для цього потрібно заздалегідь виділити зображення.

2. Згенерувати точки орієнтації. Для цього виділити зображення, і в меню «Додатки» - «Інструменти Gcode» обрати команду «Точки орієнтації» (Рис 4.4). Застосувати зміни за замовчуванням.

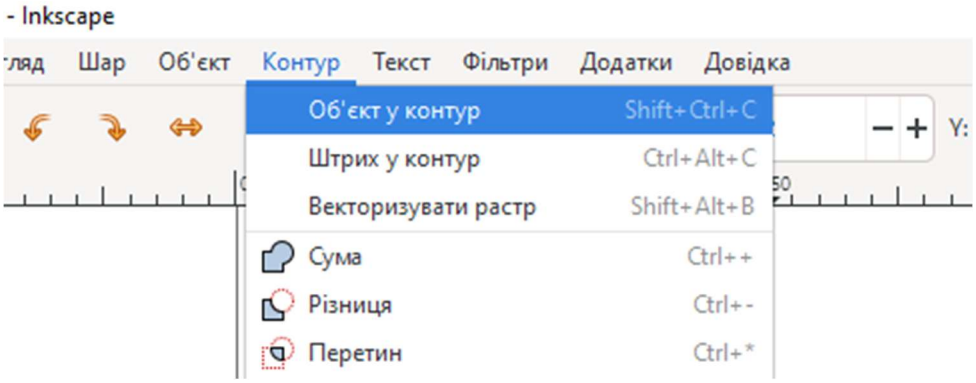


Рис 4.3 – Команда «Об’єкт у контур»

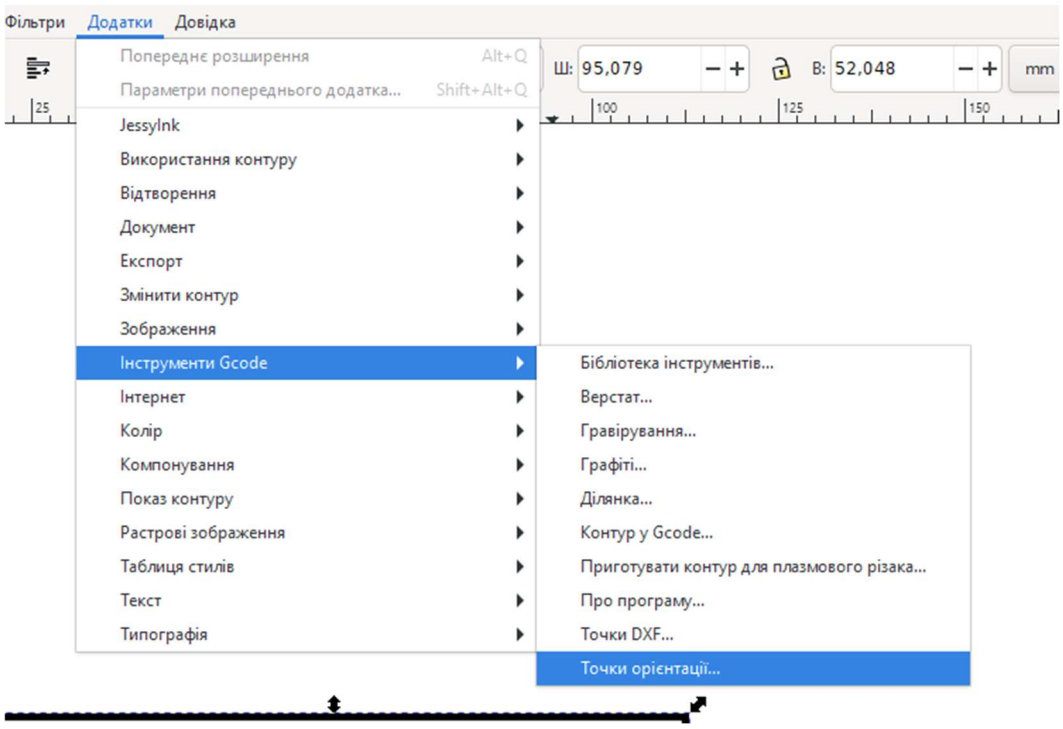


Рис 4.4 – Команда «Точки орієнтації»

3. Обираємо інструмент в меню «Додатки» - «Інструменти Gcode» командою «Бібліотека інструментів». Обираємо «Циліндр» та застосовуємо. Після виконання команди отримуємо такий результат (Рис 4.5). Зображення підготовлено до генерації G-коду.

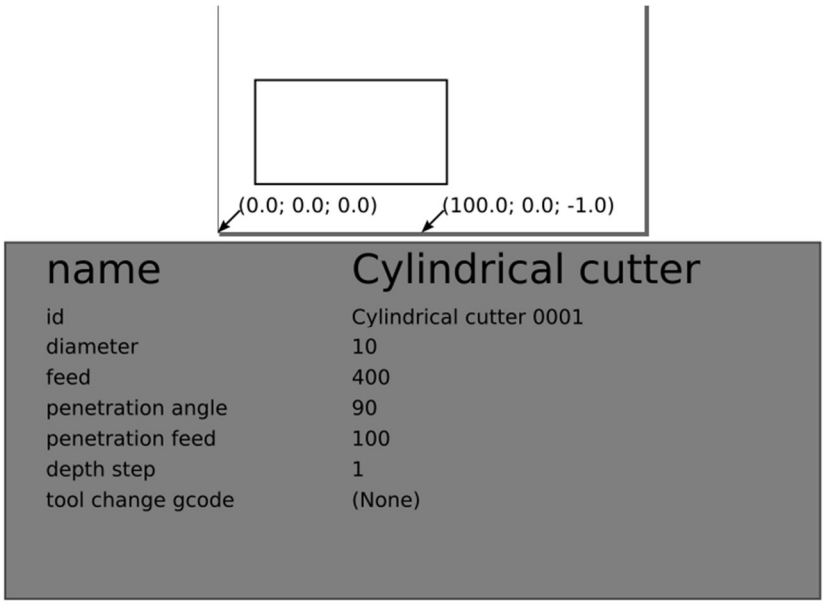


Рис 4.5 – Зображення готово до генерації коду

4. Генеруємо G-код. Для цього в меню «Додатки» - «Інструменти Gcode» обираємо команду «Контур у Gcode» (Рис 4.6) і на вкладці «Налаштування» задаємо назву вихідного файлу «code.cnc» і путь, за яким знаходиться скрипт, що виконує перетворення коду та відправляє команди на графопобудовник. Після налаштувань переходимо на вкладку «Контур у Gcode» і натискаємо «Застосувати» (Рис 4.7). Файл G-коду згенеровано.

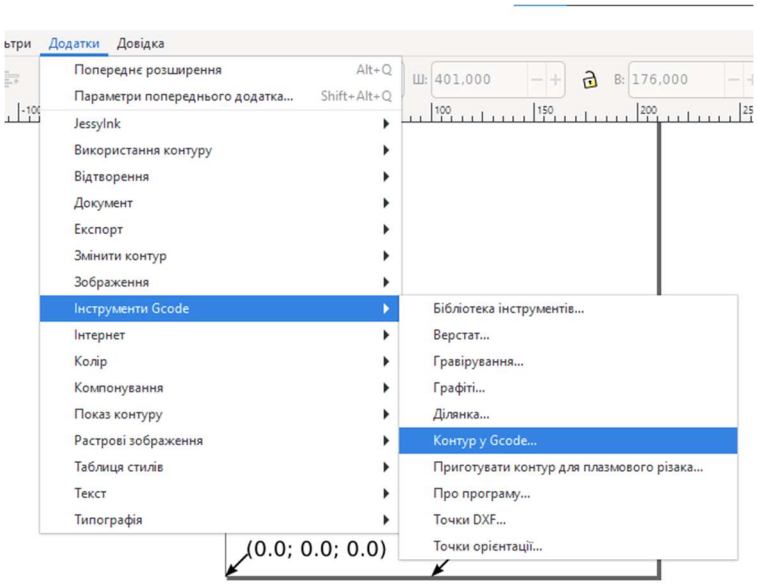


Рис 4.6 – Команда «Контур у Gcode»

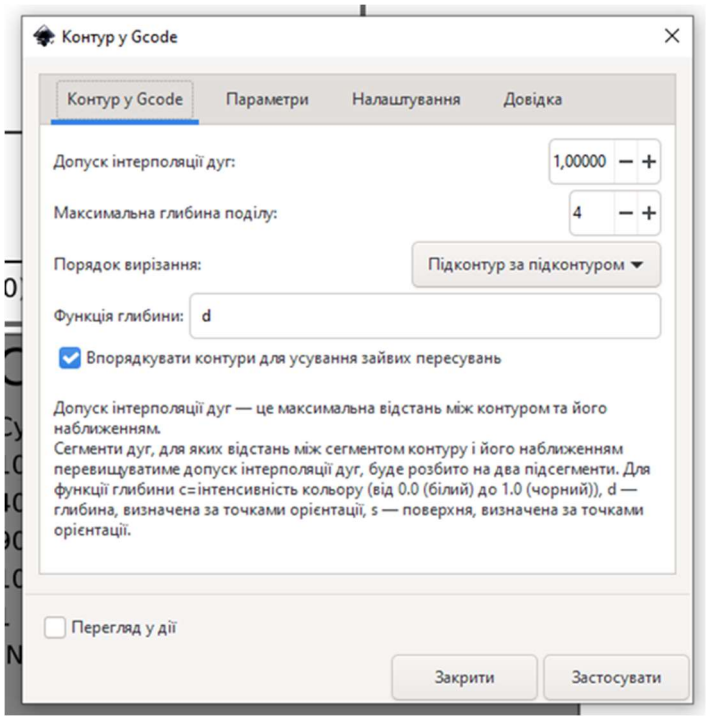


Рис 4.7 – Генерація G-коду

Після генерації году отримаємо такий результат:

```
%
(Header)
(Generated by gcodetools from Inkscape.)
(Using default header. To add your own header create file "header" in the output dir.)
M3
(Header end.)
G21 (All units in mm)

(Start cutting path id: rect833)
(Change tool to Cylindrical cutter)

G00 Z5.000000
G00 X18.441597 Y74.899020

G01 Z-1.000000 F100.0(Penetrate)
G01 X112.520470 Y74.899020 Z-1.000000 F400.000000
G01 X112.520470 Y23.850540 Z-1.000000
G01 X18.441597 Y23.850540 Z-1.000000
G01 X18.441597 Y74.899020 Z-1.000000
G00 Z5.000000

(End cutting path id: rect833)

(Footer)
M5
G00 X0.0000 Y0.0000
M2
(Using default footer. To add your own footer create file "footer" in the output dir.)
(end)
%
```

4.2. Виконання скрипту, аналіз результатів роботи.

Виконаємо скрипт, що перетворює G-код на команди графопобудовника з виведенням проміжних результатів для наочності та тестування роботи транслятора на кожному етапі.

Результати роботи скрипту:

```
#Code after preprocessing

%
M3
G21
G00 Z5.000000
G00 X18.441597 Y74.899020
G01 Z-1.000000 F100.0
G01 X112.520470 Y74.899020 Z-1.000000 F400.000000
G01 X112.520470 Y23.850540 Z-1.000000
```

```
G01 X18.441597 Y23.850540 Z-1.000000
G01 X18.441597 Y74.899020 Z-1.000000
G00 Z5.000000
M5
G00 X0.0000 Y0.0000
M2
%

#List of lexems

['%']
['M3']
['G21']
['G00', 'Z5.000000']
['G00', 'X18.441597', 'Y74.899020']
['G01', 'Z-1.000000', 'F100.0']
['G01', 'X112.520470', 'Y74.899020', 'Z-1.000000', 'F400.000000']
['G01', 'X112.520470', 'Y23.850540', 'Z-1.000000']
['G01', 'X18.441597', 'Y23.850540', 'Z-1.000000']
['G01', 'X18.441597', 'Y74.899020', 'Z-1.000000']
['G00', 'Z5.000000']
['M5']
['G00', 'X0.0000', 'Y0.0000']
['M2']
['%']

#List of frame dictionaries

{'cmd': 'M3'}
{'cmd': 'G21'}
{'cmd': 'G00', 'z': '5.000000'}
{'cmd': 'G00', 'x': '18.441597', 'y': '74.899020'}
{'cmd': 'G01', 'z': '-1.000000', 'f': '100.0'}
{'cmd': 'G01', 'x': '112.520470', 'y': '74.899020', 'z': '-1.000000', 'f': '400.000000'}
{'cmd': 'G01', 'x': '112.520470', 'y': '23.850540', 'z': '-1.000000'}
{'cmd': 'G01', 'x': '18.441597', 'y': '23.850540', 'z': '-1.000000'}
{'cmd': 'G01', 'x': '18.441597', 'y': '74.899020', 'z': '-1.000000'}
{'cmd': 'G00', 'z': '5.000000'}
{'cmd': 'M5'}
{'cmd': 'G00', 'x': '0.0000', 'y': '0.0000'}
{'cmd': 'M2'}

#List of commands

030000000000000000
0000000000003e0000
0000e603a800000000
0100000000104b0064
0104970000000000190
0100000127f00000190
0114980000000000190
0100000027e00000190
0000000000004b0190
0400000000000000190
0010e713a900000190
0200000000000000190

Connecting to device VID:0x7122 PID:0x1111
[3, 0, 0, 0, 0, 0, 0, 0, 0]
Sending cmd - 030000000000000000
Waiting for answer
[0, 0, 0, 0, 0, 0, 62, 0, 0]
Sending cmd - 0000000000003e0000
Waiting for answer
[0, 0, 230, 3, 168, 0, 0, 0, 0]
Sending cmd - 0000e603a800000000
Waiting for answer
[1, 0, 0, 0, 0, 16, 75, 0, 100]
Sending cmd - 0100000000104b0064
Waiting for answer
[1, 4, 151, 0, 0, 0, 0, 1, 144]
Sending cmd - 0104970000000000190
Waiting for answer
[1, 0, 0, 18, 127, 0, 0, 1, 144]
Sending cmd - 0100000127f00000190
```

```
Waiting for answer
[1, 20, 152, 0, 0, 0, 0, 1, 144]
Sending cmd - 011498000000000190
Waiting for answer
[1, 0, 0, 2, 126, 0, 0, 1, 144]
Sending cmd - 010000027e00000190
Waiting for answer
[0, 0, 0, 0, 0, 0, 75, 1, 144]
Sending cmd - 0000000000004b0190
Waiting for answer
[4, 0, 0, 0, 0, 0, 0, 1, 144]
Sending cmd - 040000000000000190
Waiting for answer
[0, 16, 231, 19, 169, 0, 0, 1, 144]
Sending cmd - 0010e713a900000190
Waiting for answer
[2, 0, 0, 0, 0, 0, 0, 1, 144]
Sending cmd - 020000000000000190
Waiting for answer
Print finished
```

Відповідь графопобудовника через послідовний порт для відлагодження:

```
VID = 0x7122 PID = 0x1111
Starting USBHID…
HID configured
Received: 030000000000000000
Processing…
Answering…
Received: 0000000000003e0000
Processing…
Answering…
Received: 0000e603a800000000
Processing…
Answering…
Received: 0100000000104b0064
Processing…
Answering…
Received: 010497000000000190
Processing…
Answering…
Received: 010000127f00000190
Processing…
Answering…
Received: 011498000000000190
Processing…
Answering…
Received: 010000027e00000190
Processing…
Answering…
Received: 0000000000004b0190
Processing…
Answering…
Received: 040000000000000190
Processing…
Answering…
Received: 0010e713a900000190
Processing…
Answering…
Received: 020000000000000190
Processing…
Answering…
```

Результат роботи пристрою (Рис 4.8)

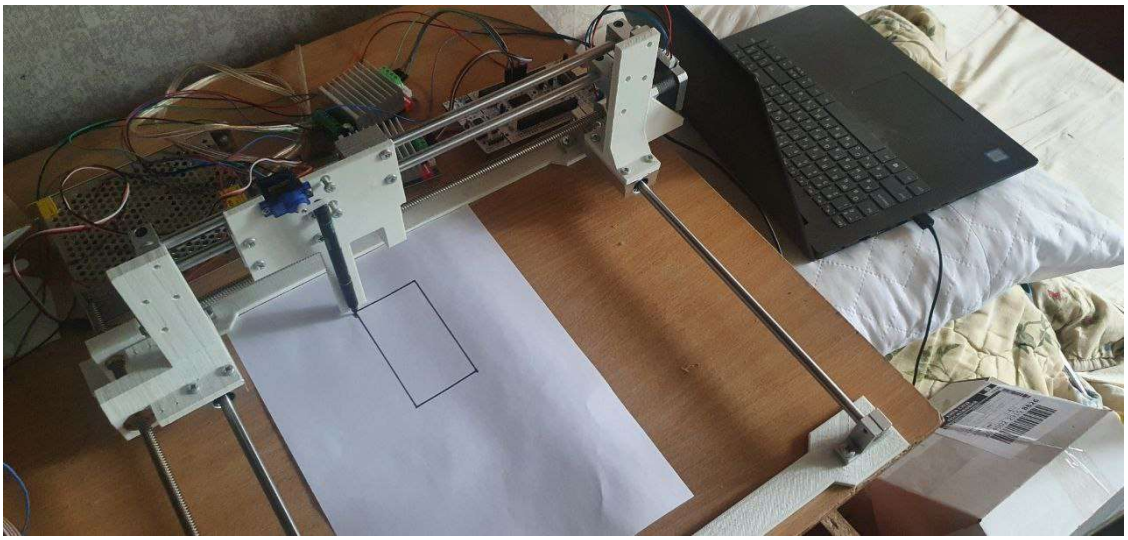


Рис 4.8 – Результат роботи пристрою

ВИСНОВОК ДО РОЗДІЛУ 4

В цьому розділі було проведено тестування пристрою за допомогою тестового зображення, і було розказано як підготувати зображення до друку на графопобудовнику.

В ході тестування було визначено що пристрій працює без помилок та виконує свої функції. Надруковане тестове зображення відповідає тому, яке було відправлено на друк.

ВИСНОВОК

В першому розділі було розглянуто предметну область, та опрацьовано основи технічної бази, а саме існуючи способи зберігання зображень, способи роботи з зовнішніми пристроями. Було вивчено існуючи типи графопобудовників та обрано найбільш оптимальний для реалізації. Також було виконано огляд існуючих рішень у даній області. Була обрана область та подальший напрямок роботи в рамках даного дипломного проєкту.

В другому розділі було виконано пошук найбільш оптимальних та необхідних технологій для реалізації графопобудовника, визначено спосіб спілкування с пристроєм, обрано мікроконтролер на базі якого буде побудовано пристрій. Визначено фреймворки та бібліотеки необхідні для реалізації проєкту. Була обрана необхідна мова програмування для реалізації програмного забезпечення на персональному комп'ютері. Було виконано обґрунтування обраних технологій.

В третьому розділі було показано процес реалізації пристрою. Описано необхідні етапи у створенні, а саме створення апаратної частини, вибір конкретних портів введення-виведення. Було показано процес розробки програмного забезпечення для персонального комп'ютера та мікроконтролера з покроковим роз'ясненням для чого використовується кожен програмний компонент та коротким описам алгоритму його роботи.

В четвертому розділі біло проведено тестування роботи пристрою та демонстрацію його роботи з покроковим поясненням принципів роботи з ним. Було описано алгоритм генерації G-коду зі файлу зображення та його друк на графопобудовнику.

Результатом виконання даного дипломного проєкту є працюючий зовнішній пристрій, що показує успішне застосування великого стеку протоколів та технологій при проєктуванні та реалізації цього пристрою.

Графопобудовники є дуже специфічними пристроями з невеликою кількістю кінцевих користувачів. Основною причиною, за якою вони не є такими розповсюдженими є те, що вони використовуються лише у деяких конкретних задачах та не потрібні багатьом користувачам у повсякденному житті. Найбільшим недоліком можна вважати те, що придатні для друку на графопобудовниках тільки зображення, що знаходяться у форматі G-коду.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. Вікіпедія – електронна енциклопедія: Графопостроитель [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%93%D1%80%D0%B0%D1%84%D0%BE%D0%BF%D0%BE%D1%81%D1%82%D1%80%D0%BE%D0%B8%D1%82%D0%B5%D0%BB%D1%8C.

2. Графопостроитель - История развития устройств вывода [Електронний ресурс] – Режим доступу до ресурсу: https://sites.google.com/site/sinkevichslava/home/grafopostroitel.

3. ГРАФОПОСТРОИТЕЛЬ ВИДЫ. ЧТО ТАКОЕ ПЛОТТЕР, ЕГО ПРЕДНАЗНАЧЕНИЕ И ПРИНЦИП УСТРОЙСТВА. [Електронний ресурс] – Режим доступу до ресурсу: https://whatsappss.ru/ofisnye-programmy/grafopostroitel-vidy-chto-takoe-plotter-ego-prednaznachenie-i.html.

4. ПЛАНШЕТНЫЙ РЕЖУЩИЙ ПЛОТТЕР FC4060 [Електронний ресурс] – Режим доступу до ресурсу: https://www.forsign.ru/catalog/oborudovanie/planshetnye_rezhushchie_plottery/planshetnye_plottery_s_flyugernym_nozhom/x_cut_knr/planshetnyy_rezhushchiy_plotter_fc4060/.

5. Плоттер: что это за устройство, назначение, принцип работы, характеристики [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://tehnolev.ru/tsifrovaya-tehnika/plotter/plotter-chto-eto-za-ustroystvo-naznachenie-printsip-raboty-harakteristiki.html.

6. Растровая и векторная графика [Електронний ресурс]. – 2016. – Режим доступу до ресурсу: https://htmlacademy.ru/blog/boost/frontend/rastr-vector.

7. Epson SureColor SC-T5200 [Електронний ресурс] – Режим доступу до ресурсу: https://epson.ua/catalog/lfp/epson-surecolor-sc-t5200/.

8. Makeblock XY-Plotter [Електронний ресурс] – Режим доступу до ресурсу: https://www.moyo.ua/obuchayushchiyi_konstruktor_makeblock_xy-plotter_robot_kit_v2_0/276718.html.

9. Большая российская энциклопедия - Обратная связь [Електронний ресурс] – Режим доступу до ресурсу: https://bigenc.ru/physics/text/2674121.

10. Stepper Motors Basics: Types, Uses, and Working Principles [Електронний ресурс] – Режим доступу до ресурсу: https://www.monolithicpower.com/en/stepper-motors-basics-types-uses.

11. Драйвер шагового двигателя - устройство, виды и возможности [Електронний ресурс] – Режим доступу до ресурсу: http://electricalschool.info/elprivod/1872-drajjver-shagovogo-dvigatelja.html.

12. Вікіпедія – електронна енциклопедія: Серовпривод. [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%A1%D0%B5%D1%80%D0%B2%D0%BE%D0%BF%D1%80%D0%B8%D0%B2%D0%BE%D0%B4.

13. Вікіпедія – електронна енциклопедія: UART. [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/UART.

14. Вікіпедія – електронна енциклопедія: USB. [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://uk.wikipedia.org/wiki/USB.

15. HUMAN INTERFACE DEVICE (HID) PROFILE [Електронний ресурс]. – 2003. – Режим доступу до ресурсу: https://web.archive.org/web/20101027225304/http://www.bluetooth.com/SiteCollectionDocuments/HID_SPEC_V10.pdf.

16. USB 2.0 USB-IF [Електронний ресурс] – Режим доступу до ресурсу: https://www.usb.org/usb2.

17. Document Library USB-IF [Електронний ресурс] – Режим доступу до ресурсу: https://www.usb.org/documents.

18. Gowdy S. List of USB ID's [Електронний ресурс] / Stephen J. Gowdy – Режим доступу до ресурсу: http://www.linux-usb.org/usb.ids.

19. USB in a NutShell. Making sense of the USB standard [Електронний ресурс] – Режим доступу до ресурсу: https://www.beyondlogic.org/usbnutshell/usb1.shtml.

20. G-код (NC-код) [Електронний ресурс] – Режим доступу до ресурсу: https://www.dreambird.ru/useful/definitions/g-code/.

21. Справочник кодов и специальных символов программирования [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: http://planetacam.ru/college/learn/16-1/.

22. Описание G и M кодов для программирования ЧПУ (CNC) станков [Електронний ресурс] – Режим доступу до ресурсу: https://3d-stanki.ru/spravochnik/programmnoe-obespechenie-dlya-stankov-s-chpu/opisanie-g-i-m-kodov-dlya-programmirovaniya-chpu-cnc-stankov-2/.

23. Вікіпедія – електронна енциклопедія: G-code. [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/G-code.

24. FreeRTOS. Real-time operating system for microcontrollers [Електронний ресурс] – Режим доступу до ресурсу: https://www.freertos.org/.

25. FreeRTOS: введение [Електронний ресурс] – Режим доступу до ресурсу: https://habr.com/ru/post/129105/.

26. Вікіпедія – електронна енциклопедія: Mbed. [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Mbed.

27. Mbed OS [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/mbed-os/.

28. Применение Arm Mbed OS. Тонкая настройка [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://habr.com/ru/post/422413/.

29. Arduino Documentation [Електронний ресурс] – Режим доступу до ресурсу: https://docs.arduino.cc/.

30. Вікіпедія – електронна енциклопедія: Arduino. [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/Arduino.

31. ATmega328P DATASHEET [Електронний ресурс]. – 2015. – Режим доступу до ресурсу: https://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-7810-Automotive-Microcontrollers-ATmega328P_Datasheet.pdf.

32. ESP32 Series Datasheet Version 3.6 Espressif Systems [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf.

33. STM32F103x8 STM32F103xB Medium-density performance line ARM®-based 32-bit MCU with 64 or 128 KB Flash, USB, CAN, 7 timers, 2 ADCs, 9 com. interfaces. Datasheet - production data [Електронний ресурс] – Режим доступу до ресурсу: https://www.st.com/resource/en/datasheet/stm32f103c8.pdf.

34. STM32F767ZI High-performance and DSP with FPU, Arm Cortex-M7 [Електронний ресурс] – Режим доступу до ресурсу: https://www.st.com/en/microcontrollers-microprocessors/stm32f767zi.html.

35. STM32 Nucleo-144 development board with STM32F767ZI MCU [Електронний ресурс] – Режим доступу до ресурсу: https://www.st.com/en/evaluation-tools/nucleo-f767zi.html.

36. NUCLEO-F767ZI [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/platforms/ST-Nucleo-F767ZI/.

37. TB6600 Datasheet Analog driver model TB6600 [Електронний ресурс]. – 2018. – Режим доступу до ресурсу: https://www.mcielectronics.cl/website_MCI/static/documents/TB6600_data_sheet.pdf.

38. Вікіпедія – електронна енциклопедія: Дифференциальный сигнал. [Електронний ресурс]. – 2021. – Режим доступу до ресурсу: https://ru.wikipedia.org/wiki/%D0%94%D0%B8%D1%84%D1%84%D0%B5%D1%80%D0%B5%D0%BD%D1%86%D0%B8%D0%B0%D0%BB%D1%8C%D0%BD%D1%8B%D0%B9_%D1%81%D0%B8%D0%B3%D0%BD%D0%B0%D0%BB.

39. Павлов В. Г. Конспект лекцій з предмету "Системне програмування, - 2" / В. Г. Павлов. – Київ, 2019.

40. Павлов В. Г. Методичні вказівки для виконання лабораторних робот з предмету "Системне програмування, - 2" / В. Г. Павлов. – Київ, 2019.

41. Python hid library [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://pypi.org/project/hid/.

42. hidapi library [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://github.com/apmorton/pyhidapi.

43. Mbed OS API Reference: I/O APIs [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/docs/mbed-os/v6.11/apis/i-o-apis.html.

44. Mbed OS API Reference: Digital out [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/docs/mbed-os/v6.11/apis/digitalout.html.

45. Mbed OS API Reference: PWM APIs [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/docs/mbed-os/v6.11/apis/pwmout.html.

46. Mbed Servo library [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/cookbook/Servo.

47. Mbed OS API Reference: Intrerrupt APIs [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/docs/mbed-os/v6.11/apis/interruptin.html.

48. Mbed OS API Reference: Scheduling APIs [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/docs/mbed-os/v6.11/apis/scheduling-rtos-and-event-handling.html.

49. Mbed OS API Reference: Thread APIs [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/docs/mbed-os/v6.11/apis/thread.html.

50. Mbed OS API Reference: USBHID APIs [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/docs/mbed-os/v6.11/apis/usbhid.html.

51. Stepper driver TB6600 library [Електронний ресурс] – Режим доступу до ресурсу: https://os.mbed.com/teams/Dagozilla-to-RoboCup/code/StepperTB/.

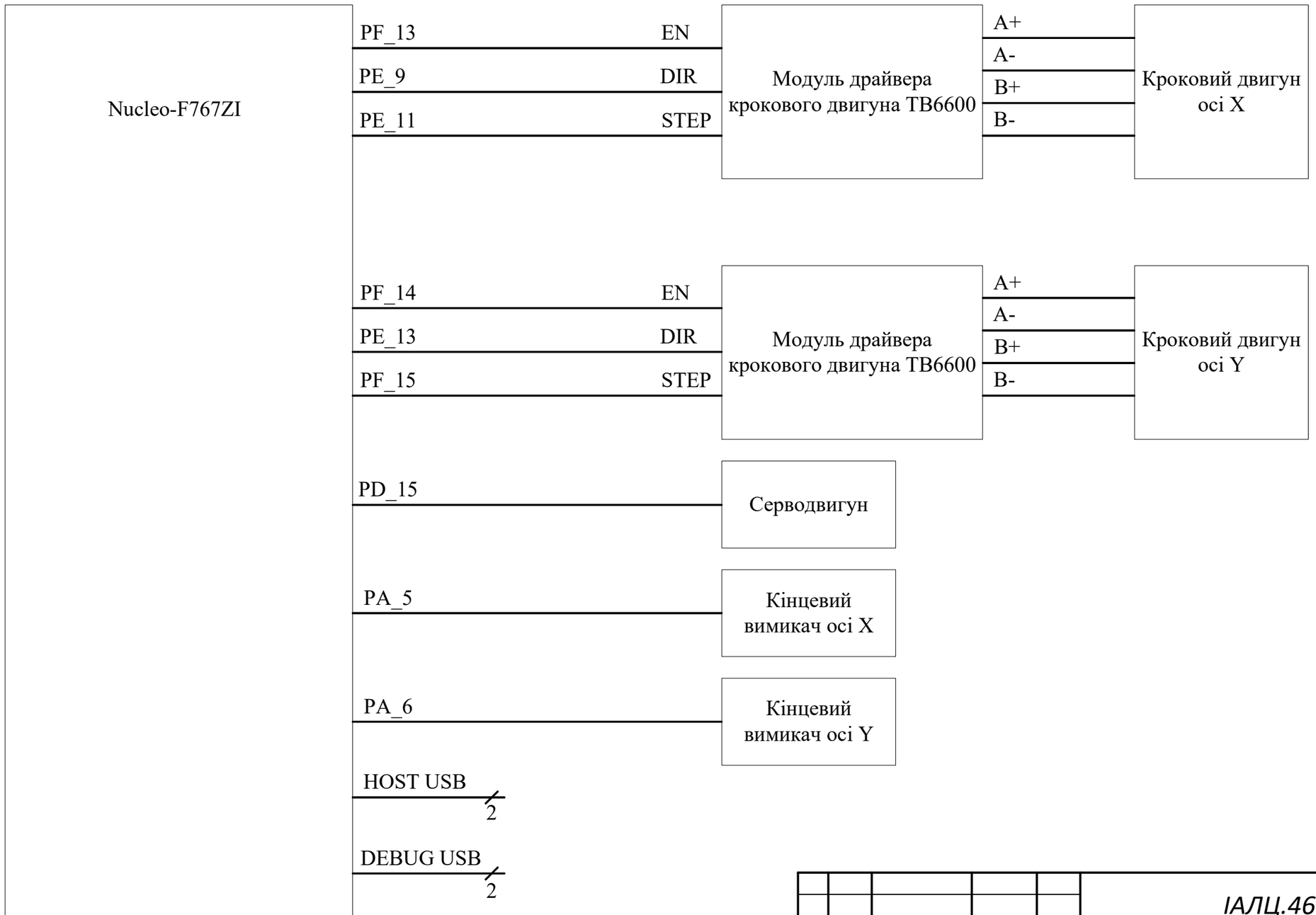
Додаток А

СХЕМА ЕЛЕКТРИЧНА СТРУКТУРНА

до дипломного проєкту

на тему: «Графопобудовник для автоматичного друку векторної графіки»

Київ – 2021 року



					ІАЛЦ.466500.004 Д1	
Зм.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Саяпін Ю.Л.			Схема електрична структурна Графопобудовник	Літ. / Маса / Масштаб
Перевір.		Клименко І. А.				Арк. / Аркушів
Н. контр.		СимоненкоВ. П.			Дипломний проект	КПІ ФІОТ кафедра ОТ гр. ІО-71
Затверд.		Клименко І. А.				

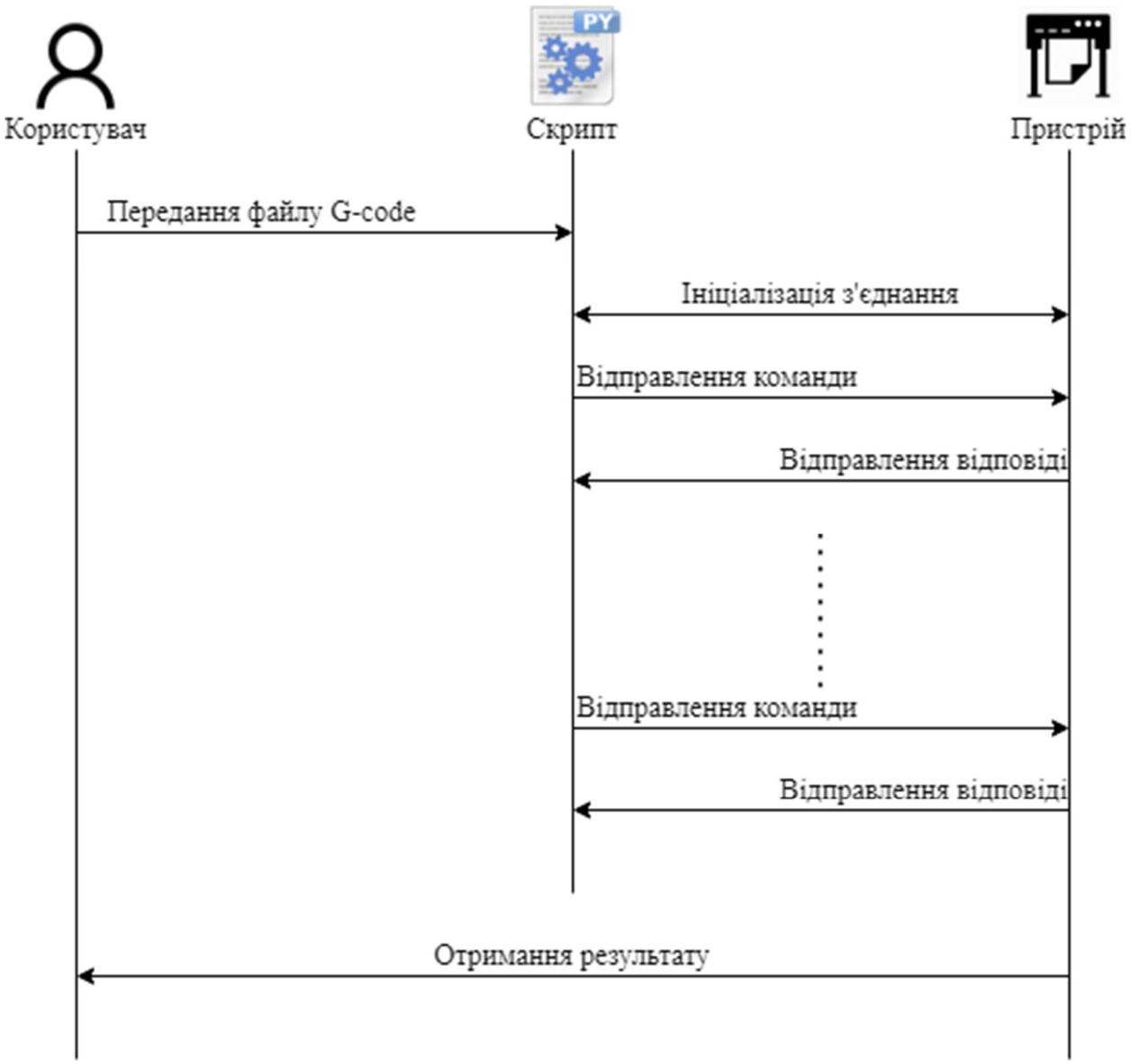
Додаток Б

СХЕМА ВЗАЄМОДІЇ КОРИСТУВАЧА ІЗ ПРИСТРОЄМ ФУКЦІОНАЛЬНА

до дипломного проєкту

на тему: «Графопобудовник для автоматичного друку векторної графіки»

Київ – 2021 року



					ІАЛЦ.466500.005 Д2		
Зм.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Саяпін Ю.Л.			*Графопобудовник для автоматичного друку векторної графіки* *Функціональна схема взаємодії користувача із пристроєм*	Літ.	Арк. 1 / Аркушів 1
Перевір.		Клименко І. А.					
Реценз.						**НТУУ «КПІ» ФІОТ ІО-71**	
Н. контр.		Симоненко В. П.					
Затверд.		Клименко І. А.					

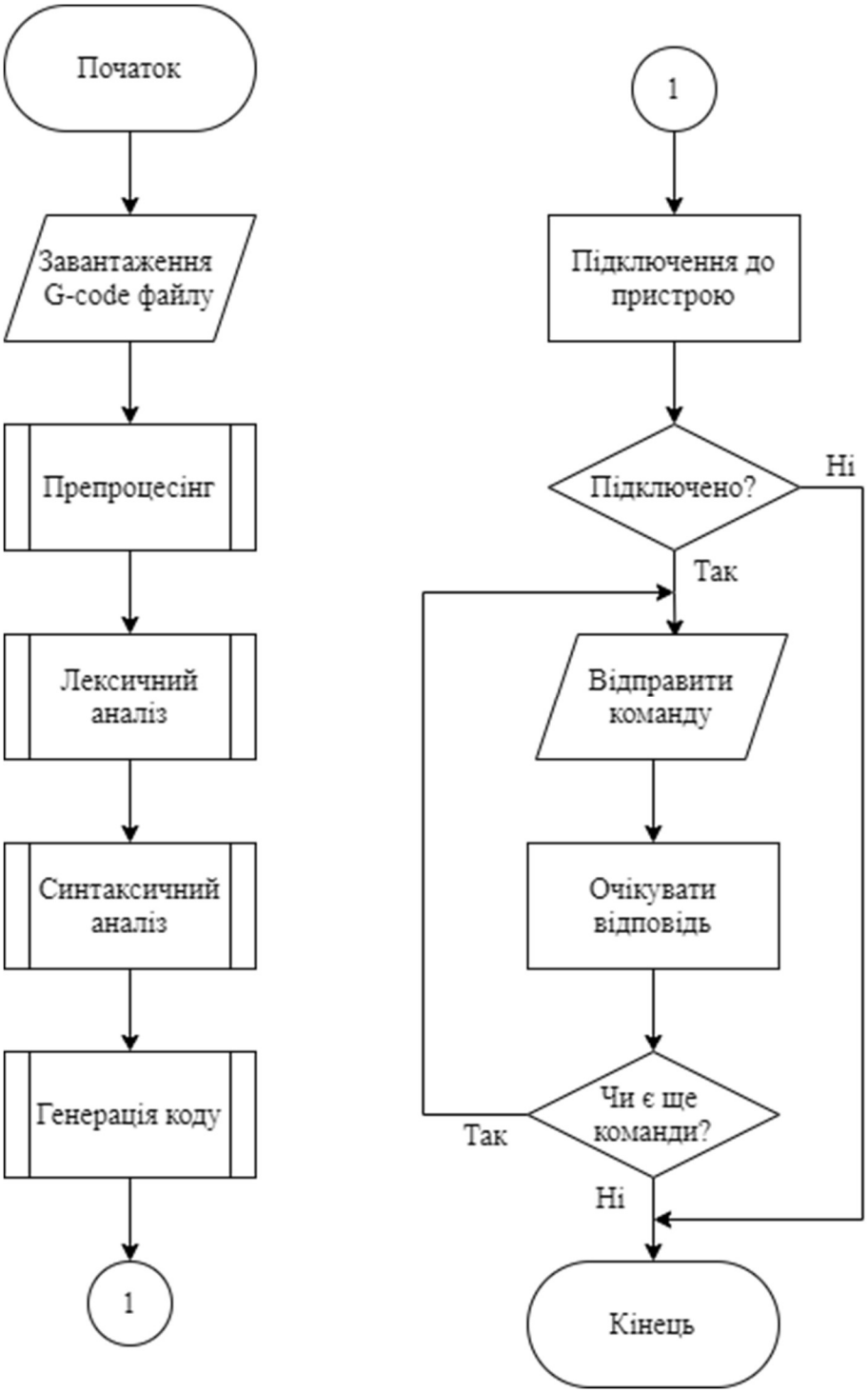
Додаток В

СХЕМА АЛГОРИТМУ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА ПРИНЦИПОВА

до дипломного проєкту

на тему: «Графопобудовник для автоматичного друку векторної графіки»

Київ – 2021 року



					ІАЛЦ.466500.006 ДЗ		
Зм.	Арк.	№ докум.	Підп.	Дата			
Розроб.		Саяпін Ю.Л.			*Графопобудовник для автоматичного друку векторної графіки*	Літ.	Арк. 1 / Аркушів 1
Перевір.		Клименко І. А.			*Схема алгоритму роботи програмного забезпечення персонального комп'ютера*		
Реценз.						**НТУУ «КПІ» ФІОТ ІО-71**	
Н. контр.		Симоненко В. П.					
Затверд.		Клименко І. А.					

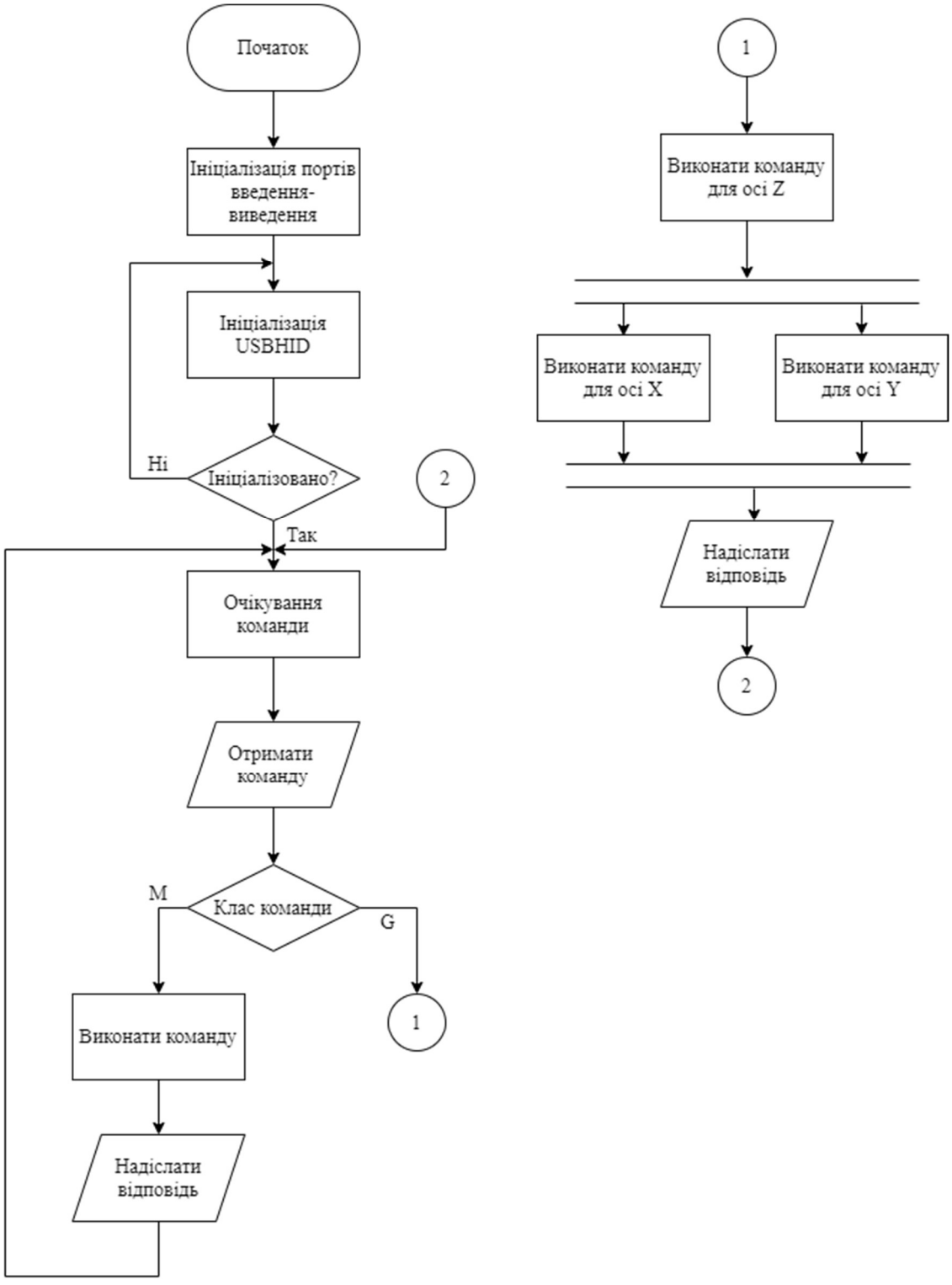
Додаток Г

СХЕМА АЛГОРИТМУ РОБОТИ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МІКРОКОНТРОЛЕРА ПРИНЦИПОВА

до дипломного проєкту

на тему: «Графопобудовник для автоматичного друку векторної графіки»

Київ – 2021 року



Зм.	Арк.	№ докум.	Підп.	Дата	ІАЛЦ.466500.007 Д4		
Розроб.		Саяпін Ю.Л.			Графопобудовник для автоматичного друку векторної графіки Схема алгоритму роботи програмного забезпечення мікроконтролера	Літ.	Арк. 1 / Аркушів 1
Перевір.		Клименко І. А.					
Реценз.						НТУУ «КПІ» ФІОТ ІО-71	
Н. контр.		Симоненко В. П.					
Затверд.		Клименко І. А.					

Додаток Д

ЛІСТИНГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ПЕРСОНАЛЬНОГО КОМП'ЮТЕРА

до дипломного проєкту

на тему: «Графопобудовник для автоматичного друку векторної графіки»

Київ – 2021 року

ЛІСТИНГ ПРОГРАМИ

```
import hid
import re
from math import floor


def Preprocessing(code):
    no_comms = re.sub("[\(\[].*?[\)\]]", "", code)
    filtered = "\n".join([ll.rstrip() for ll in no_comms.splitlines() if ll.strip()])
    return filtered


def Lexical(code):
    code = code.split('\n')
    for i in range(len(code)):
        code[i] = code[i].split(' ')
    return code


def Syntax(lexems):
    structure = []
    for frame in lexems:
        if frame[0] == '%':
            continue
        frm = {}
        frm.setdefault('cmd', frame[0])
```

```
        if frame[0][0] == 'M':
            structure.append(frm)
            continue
        for i in range(1, len(frame)):
            if frame[i][0] == 'X':
                frm.setdefault('x', frame[i][1:])
            if frame[i][0] == 'Y':
                frm.setdefault('y', frame[i][1:])
            if frame[i][0] == 'Z':
                frm.setdefault('z', frame[i][1:])
            if frame[i][0] == 'I':
                frm.setdefault('i', frame[i][1:])
            if frame[i][0] == 'J':
                frm.setdefault('j', frame[i][1:])
            if frame[i][0] == 'K':
                frm.setdefault('k', frame[i][1:])
            if frame[i][0] == 'F':
                frm.setdefault('f', frame[i][1:])
        structure.append(frm)
    return structure

def Interpolate():
    return

def Generate(struct):
    commands = []
    current_x = 0
    current_y = 0
```

```
    current_z = 0
    speed = 0
    for frame in struct:
        cmd = ''
        if frame.get('cmd') == 'M02' or frame.get('cmd') == 'M2' or frame.get('cmd') == 'M30':
            cmd += '02'
        if frame.get('cmd') == 'M03' or frame.get('cmd') == 'M3' or frame.get('cmd') == 'M04' or frame.get('cmd') == 'M4':
            cmd += '03'
        if frame.get('cmd') == 'M05' or frame.get('cmd') == 'M5':
            cmd += '04'
        if frame.get('cmd') == 'G00':
            cmd += '00'
        if frame.get('cmd') == 'G01':
            cmd += '01'
        if len(cmd) == 0:
            continue
        if frame.get('x', 0) != 0:
            dx = float(frame.get('x')) - current_x
            current_x = float(frame.get('x'))
            dxs = floor(dx / 0.08)
            if dxs >= 0:
                cmd += '0' + '{:03x}'.format(dxs)
            else:
                dxs = -dxs
                cmd += '1' + '{:03x}'.format(dxs)
        else:
```

```
        cmd += '0000'
    if frame.get('y', 0) != 0:
        dx = float(frame.get('y')) - current_y
        current_y = float(frame.get('y'))
        dxs = floor(dx / 0.08)
        if dxs >= 0:
            cmd += '0' + '{:03x}'.format(dxs)
        else:
            dxs = -dxs
            cmd += '1' + '{:03x}'.format(dxs)
    else:
        cmd += '0000'
    if frame.get('z', 0) != 0:
        dx = float(frame.get('z')) - current_z
        current_z = float(frame.get('z'))
        dxs = floor(dx / 0.08)
        if dxs >= 0:
            cmd += '0' + '{:03x}'.format(dxs)
        else:
            dxs = -dxs
            cmd += '1' + '{:03x}'.format(dxs)
    else:
        cmd += '0000'
    if frame.get('f', 0) != 0:
        speed = floor(float(frame.get('f')))
        cmd += '{:04x}'.format(speed)
    else:
        cmd += '{:04x}'.format(speed)
```

```
            commands.append(cmd)
        return commands

    def Print(cmd_list):
        vid = 0x7122
        pid = 0x1111

        print("\nConnecting to device VID:" + hex(vid) + " PID:" + hex(pid))
        plotter = hid.Device(vid, pid)
        for cmd in cmd_list:
            byte_list = list(bytes.fromhex(cmd))
            print(byte_list)
            print("Sending cmd - " + cmd)
            plotter.write(byte_list)
            ans = plotter.read(9)
            print("Wainting for answer")
            while(ans[0] != 0xf0):
            #    ans = plotter.read(9)
        print("Print finished")
        return

    def main():
        g_code = open("code.cnc", 'r').read()

        preprocessed = Preprocessing(g_code)
        print("#Code after precprocessing\n")
        print(preprocessed)
```

```
    lexems = Lexical(preprocessed)
    print("\n#List of lexems\n")
    for frame in lexems:
        print(frame)
    structure = Syntax(lexems)
    print("\n#List of frame dictionaries\n")
    for frame in structure:
        print(frame)
    cmds = Generate(structure)
    print("\n#List of commands\n")
    for frame in cmds:
        print(frame)
    Print(cmds)

if __name__ == "__main__":
    main()
```

Додаток Ж

ЛІСТИНГ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ МІКРОКОНТРОЛЕРА

до дипломного проєкту

на тему: «Графопобудовник для автоматичного друку векторної графіки»

Київ – 2021 року

ЛІСТИНГ ПРОГРАМИ

```
#include "mbed.h"
#include "USBHID.h"
#include "Servo.h"
#include "StepperTB.h"

InterruptIn ax_x(PA_5);
InterruptIn ax_y(PA_6);

USBHID hid(true, 9, 1, 0x7122, 0x1111, 0x0001);
StepperTB step_x(PF_13, PE_9, PE_11, 1, 200);
StepperTB step_y(PF_14, PE_13, PE_15, 1, 200);
Servo ax_z(PD_15);

HID_REPORT output_report = {
   .length = 1,
   .data = {0}
};
HID_REPORT input_report = {
   .length = 9,
   .data = {0}
};

int steps_x, steps_y;
```

```
void move_x() {
   step_x(steps_x, 1);
}

void move_y() {
   step_y(steps_y, 1);
}

// main() runs in its own thread in the OS
int main()
{
   hid.wait_ready();

   while (true) {
      hid.read(&input_report);

      Thread thread_x;
      Thread thread_y;

      auto cmd = input_report.data[0];
      steps_x = (input_report.data[2] * 16 * 16 + input_report.data[3]) *
(input_report.data[1] == 0 ? 1 : -1);
      steps_y = (input_report.data[5] * 16 * 16 + input_report.data[6]) *
(input_report.data[4] == 0 ? 1 : -1);
      switch(cmd) {
         case 0: // G00
            ax_z = 1;
            thread_x.start(move_x);
```

```
            thread_y.start(move_y);
            thread_x.join();
            thread_y.join();
        case 1: // G01
            ax_z = 0;
            thread_x.start(move_x);
            thread_y.start(move_y);
            thread_x.join();
            thread_y.join();
        case 2: // M02 M30
            ax_z = 1;
        case 3: // M04 M05
            ax_z = 0;
        default:
            printf("Wrong command!");
    }
    output_report.data[0] = 240;
    hid.send(&output_report);
  }
}
```