МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ
ФАКУЛЬТЕТ КОМП'ЮТЕРНИХ НАУК ТА ТЕХНОЛОГІЙ

Кафедра Комп'ютерних систем та мереж

ДОПУСТИТИ ДО ЗАХИСТУ
Завідувач випускової кафедри
_____Ігор ЖУКОВ
«____» _____2023 р.

КВАЛІФІКАЦІЙНА РОБОТА
(ПОЯСНЮВАЛЬНА ЗАПИСКА)

ВИПУСКНИКА ОСВІТНЬОГО СТУПЕНЯ
«МАГІСТР»
ЗА СПЕЦІАЛЬНІСТЮ 123 «КОМП'ЮТЕРНА ІНЖЕНЕРІЯ»

Тема: «Система мережевого програмного керування верстатів з числовим програмним управлінням»

Виконавець: Олег КОСТЕНКО

Керівник: Олександр АНДРЕЄЄВ

Нормоконтролер: Василь МАЛЯРЧУК

Засвідчую, що у дипломній роботі немає
Запозичень праць інших авторів без
Відповідальних посилань
Студент _____ Олег КОСТЕНКО_

Київ 2023

НАЦІОНАЛЬНИЙ АВІАЦІЙНИЙ УНІВЕРСИТЕТ

Факультет комп'ютерних наук та технологій
Кафедра комп'ютерних систем та мереж
Спеціальність 123 «Комп'ютерна інженерія»

ЗАТВЕРДЖУЮ
Завідувач кафедри комп'ютерних систем та мереж

_____Ігор ЖУКОВ

« ___ » _____2023 р.

ЗАВДАННЯ

на виконання дипломної роботи студента

Костенко Олег Євгенійович
(студент, група, прізвище, ім'я, по батькові)

1. Тема роботи: «Система мережевого програмного керування верстатів з числовим програмним управлінням»

затверджена наказом ректора від "08" серпня 2023р. №1251/од

2. Термін виконання роботи: з "02" жовтня 2023 р. по 31 грудня 2023 р.

3. Вихідні данні до роботи: гнучка модульна архітектура пристроїв числового програмного управління, система мережевого керування верстатом на базі гнучкої модульної архітектури пристроїв числового програмного управління, дослідження проблем ЧПУ та тенденцій їх майбутнього розвитку

4. Зміст пояснювальної записки (перелік питань, що підлягають розробці): Огляд сучасних тенденцій в розвитку ЧПУ, розробка універсальної системи ЧПУ з гнучкою модульною архитектурою, дослидження сучасних проблем СЧПУ та їх вирішення

5. Перелік обов'язкового графічного матеріалу: презентація *PowerPoint*

6. Календарний план-графік

№ п/п	Етапи виконання дипломного проекту(роботи)	Термін виконання етапів	Примітка
1.	Огляд літератури	02.10.2023-14.10.2023	
2.	Розгляд платформи Arduino	15.10.2023-17.10.2023	
3.	Розгляд системи симуляції електронних схем Proteus ISIS Professional	18.10.2023-22.10.2023	
4.	Аналіз проблем сучасних архітектур ЧПУ ти дослідження шляхів їх вирішення	23.10.2023-27.10.2023	
5.	Побудова алгоритмів інтерполяторів та інтерпретатора	28.10.2023-01.11.2023	
6.	Розробка прінціпової схеми ПЧПУ за гнучкою модульною архітектурою, програмування мікроконтроллерів та тестування за допомогою Proteus ISIS Professional	02.11.2023-10.11.2023	
7.	Аналіз тенденцій у розвитку ПЧПУ	11.11.2023-13.11.2023	
8.	Оформлення пояснювальної записки	14.11.2023-24.11.2023	
9.	Виготовлення графічного матеріалу	25.11.2023-04.12.2023	
10.	Проходження нормоконтролю	05.11.2023-12.12.2023	
11.	Представити дипломну роботу на кафедру	18.12.2023	

7. Дата видачі завдання “02” жовтня 2023 р.

Керівник дипломної роботи _____ Олександр АНДРЄЄВ
(підпис)

Завдання прийняв до виконання _____ Олег КОСТЕНКО
(підпис студента)

РЕФЕРАТ

Пояснювальна записка до дипломної роботи «Система мережевого програмного керування верстатів з числовим програмним управлінням» 89 с, 6 таблиць, 32 рисунки, 4 додатки, 22 літературних джерела.

ЧИСЛОВЕ ПРОГРАМНЕ КЕРУВАННЯ, ВЕРСТАТ, ЧПУ, МОДЕРНІЗАЦІЯ ЧПУ, *CNC, DNC, ETHERNET, PCNC*

Об’єкт дослідження – сучасні тенденції в розвитку верстатів ЧПУ, проблеми та задачі ЧПУ та оптимальні способі їх вирішення.

Предмет дослідження – розробка структурної та принципої схеми універсального пристрою ЧПУ, та розробка прототипу пристрою.

Мета дипломної роботи – розробка концепції, структурної та принципової схем універсального пристрою ЧПУ, що призначений для керування верстатом або групою верстатів, які об’єднані у комп’ютерну мережу *Ethernet*.

Метод дослідження – аналіз існуючих на сьогоднішній день систем ЧПУ, їх архітектур, задач та проблем їх функціонування, та симуляція за допомогою програмного програмного засобу *Proteus*

Результати – запропонована гнучка модульна архітектура розстрою, яка може використовуватись як методика для побудови пристроїв ЧПУ. Розроблений пристрій може використовуватись для керування будь-якими верстатами, включаючи такі, що не передбачають використання пристроїв ЧПУ.

Наукова новизна одержаних результатів – запропонована концепція пропонує способи вирішення деяких проблем, що виникають при створення систем ЧПУ та надає наступні можливості при їх використанні:

– спрощує та в рази здешевлює процес побудови пристроїв ЧПУ, що відповідають за своїми можливостями сучасним світовим аналогам;

– дозволяє значно спрощувати та в десятки разів здешевлювати розробку спеціалізованого програмного *CAM*-забезпечення, тобто системи автоматизованої підготовки виробництва і робить можливим застосовувати її на будь-яких сучасних операційних системах, позбавляючи розробника ПЗ проблеми реального часу.

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

API	—	*Application Program Interface*
CAD	—	*Computer Aided Design*
CAM	—	*Computer Aided Manufacturing*
CNC	—	*Computer Numerical Control*
GUI	—	*Graphic User Interface*
Hobby-CNC	—	
EEPROM	—	*Electrically Erasable Programmable Read-Only Memory*
PCNC	—	*Personal ComputerNumerical Control*
PLC	—	*Programmable Logic Controller*
RTAPI	—	*Real Time Application Interface*
RTSS	—	*Real TimeSub System*
ДЗЗ	—	Датчик Зворотньогозв’язку
ОС	—	Операційна система
ОСРЧ	—	Операційна система реального часу
ПЗП	—	Постійний Запам’ятовуючий Пристрій
ПЕОМ	—	Персональна Електронно-Обчислительна Машина
ЧПУ	—	Числове Програмне Управління
КП	—	Керуюча програма
КД	—	Крокoвий двигун
ШІМ	—	Широтно-Імпульсна Модуляція
СЧПУ	—	Система Числового Програмного Управління
ПЧПУ	—	Пристрій Числового Програмного Управління

ВСТУП

В сучасному світі, де комп'ютерні технології стрімко розвиваються, системи комп'ютерного чисельного управління (ЧПУ) стають необхідним елементом в промисловості, відіграваючи ключову роль у підтримці та оптимізації виробничих процесів. За останні роки спостерігається неймовірний ріст у сфері ЧПУ, де флагманські моделі від провідних світових виробників встановлюють нові стандарти ефективності та точності в цій динамічній галузі.

Сучасні системи ЧПУ вражають своєю високоточною архітектурою, яка базується на двохкомп'ютерній системі, що дозволяє забезпечити найвищий рівень контролю над виробничими процесами. Вони здатні управляти до 16 осей на різних каналах управління, використовуючи інтерполяцію з нанометричною точністю. Однією з ключових переваг є здатність виконувати високошвидкісну обробку деталей, що робить ці системи вельми ефективними в умовах сучасних виробничих ліній.

Вартість сучасних систем ЧПУ від провідних світових виробників розпочинається від $15,000 за мінімальну базову конфігурацію, враховуючи їхні потужності та функціональні можливості. Однак ціни можуть зростати і перевищувати вражаючу позначку у $60,000 в залежності від обраної комплектації та додаткових функціональних опцій. Це відображає широкий спектр можливостей та гнучкість систем ЧПУ, які можуть задовольнити різні потреби та вимоги виробництва, від малих майстерень до великих промислових підприємств.

Незважаючи на великий попит, в країнах колишнього СРСР виробництво систем ЧПУ такого класу стикається з великими труднощами через їхню високу складність. Це призводить до ситуації, коли навіть при зростаючому попиті на такі технології, національні компанії не можуть їх виробляти. Більше того, системи ЧПУ класу "Hi-End" можуть підпадати під обм еження технологій подвійного призначення, через що деякі країни можуть обмежити їх експорт або навіть заборонити.

Однак, несумне положення у сфері виробництва систем ЧПУ в країнах колишнього СРСР створює можливості для розвитку і співпраці. Національні

компанії можуть спрямувати свої зусилля на розробку і вдосконалення власних систем ЧПУ, які відповідають вимогам місцевого ринку та промислових потреб. Залучення фахівців з досвідом і знаннями з даної галузі може сприяти досягненню прогресу в цьому напрямку.

Крім того, важливо звернути увагу на можливості інноваційного співробітництва з провідними світовими виробниками систем ЧПУ. Це може включати ліцензійну угоду або партнерство для спільного розвитку та виробництва передових систем ЧПУ. Такий підхід дозволить отримати доступ до передових технологій і забезпечити належну якість продукції.

Загалом, розвиток виробництва систем ЧПУ в країнах колишнього СРСР потребує системного підходу, що включає в себе науково-дослідну роботу, технічну експертизу і залучення фахівців з відповідними знаннями і навичками. Також необхідно уважно аналізувати міжнародні правила і обмеження, щоб забезпечити безперебійний доступ до передових технологій та можливостей експорту.

В умовах швидкого технологічного прогресу та зростаючого значення автоматизації виробництва, розвиток систем ЧПУ визначає новий етап у виробничій сфері та стає стратегічно важливим фактором для багатьох галузей промисловості. При цьому, важливо враховувати, що національні підприємства можуть зазнавати труднощі у виробництві високотехнологічних систем ЧПУ через їхню складність та високі технічні вимоги.

В одночасність з цим, слід зауважити, що наше виробництво ще й досі активно використовує ряд верстатів радянського виробництва, які характеризуються значним резервом міцності та надійною механічною структурою. Зазначена механічна частина цих верстатів утримує стабільний технічний стан і часто навіть перевершує за якістю та надійністю аналогічні компоненти флагманських моделей відомих світових виробників.

Проте, в умовах стрімкого технологічного прогресу та зростаючих вимог ринку, виявляється, що зазначені верстати, які не містять систем ЧПУ, не відповідають сучасним стандартам продуктивності та автоматизації. Наприклад, проблеми з ручним управлінням та застаріванням електронних компонентів систем

ЧПУ стають суттєвими обмеженнями для їх ефективного використання в сучасному виробництві. Це вимагає серйозних зусиль у модернізації та оновленні виробничих процесів для підтримки конкурентоспроможності національних підприємств у глобальному ринковому середовищі.

З іншого боку, для менших підприємств і самостійних майстрів, що діють у галузях, де вимоги до точності обробки не є критичними, використання старих або модернізованих верстатів набуває додаткової важливості. У таких галузях, де елементи творчості та індивідуальний підхід до виробництва переважають, варто враховувати, що старі верстати можуть виконувати завдання ефективно, забезпечуючи при цьому економію коштів порівняно із витратами на придбання та утримання нових верстатів із ЧПУ. Такий підхід стає особливо актуальним у галузях, де важливий не лише високий рівень автоматизації, але й збереження гнучкості та можливості швидко адаптуватися до змінних умов виробництва.

Зазначений контекст свідчить про потребу в модернізації та оновленні виробничих процесів для підтримки кон курентоспроможності підприємств у глобальному ринковому середовищі. Для досягнення цієї мети можна вжити такі заходи:

Впровадження систем ЧПУ: Установка систем числового програмування (ЧПУ) на старих верстатах значно підвищить їх продуктивність та автоматизацію. Це дозволить забезпечити точність обробки, покращити швидкість і ефективність процесу та зменшити кількість помилок.

Модернізація електронних компонентів: Застосування сучасних електронних компонентів в системах ЧПУ допоможе позбутися проблем, пов'язаних з застаріванням та недоліками старих компонентів. Це забезпечить стабільну та надійну роботу верстатів у сучасних умовах виробництва.

Навчання персоналу: Робота з новими технологіями та системами вимагає належного навчання персоналу. Інвестування в підготовку та навчання робітників допоможе їм ефективно користуватися оновленими верстатами та системами ЧПУ, забезпечуючи оптимальні результати.

Розробка індивідуальних рішень: Для менших підприємств або тих галузей, де елементи творчості є важливими, можуть бути вигідними індивідуальні рішення. Наприклад, модернізація окремих компонентів або розробка спеціалізованих програмних продуктів можуть допомогти вирішити конкретні завдання та задовольнити потреби таких підприємств.

Поступове оновлення: Враховуючи обмежений бюджет підприємства, можна розглянути планове поступове оновлення верстатів. Заміна найстарішого та найменш ефективного обладнання спочатку дозволить підприємству поліпшити продуктивність та якість, а потім виробниче середовище буде готове для більш широкого оновлення.

На сучасному етапі розвитку промисловості сфера розробки програмного забезпечення для систем числово-програмованого управління (СЧПУ) визнається як вирішальний етап у створенні ЧПУ. Зазначимо, що цей етап складає понад 80% від загального обсягу робіт і визначає технологічний прогрес у виробництві. Виробники систем ЧПУ ставлять перед собою завдання зберегти вкладені інвестиції та досягти максимальної прибутковості. Шляхом розширення функціональності та модернізації ядра систем управління вони спрямовують свої зусилля на забезпечення не лише надійності і точності, але й налаштовуваності для відповіді на різноманітні виробничі потреби.

Основною метою даного дослідження є глибокий аналіз теперішніх тенденцій у сфері СЧПУ, прогноз подальшого розвитку цих систем та визначення можливостей для подальшого удосконалення. Дослідження також передбачає розгляд сучасних проблем у галузі та розробку інноваційних методів їх вирішення.

Однією з основних завдань дослідження є створення модульної архітектури ЧПУ, яка не лише забезпечить максимальну налаштовуваність та універсальність для користувачів, але й дозволить ефективно впроваджувати зміни без залучення зовнішніх фахівців. Також передбачається розробка простої та економічно доступної системи ЧПУ, що спростить керування групами верстатів з централізованого робочого місця.

У ході реалізації дипломної роботи було вдосконалено та впроваджено інноваційну "півторакомп'ютерну" архітектуру та PCNC-систему, що виявилася ефективною та конкурентоспроможною. Цей вражаючий проривний проект успішно поєднав в собі переваги професійної двокомп'ютерної та доступної за вартістю аматорської однокомп'ютерної архітектур, створюючи ідеальне середовище для ефективного управління верстатами та їх групами.

Ця архітектура вирішує одну з ключових проблем у сфері розробки управляючого програмного забезпечення (ПЗ) – проблему реального часу в системах чисельно-програмованого управління (СЧПУ). Спроектований метод не лише ефективно вирішує цю проблему, але й значно спрощує розробку управляючого ПЗ, роблячи її більш доступною та економічно вигідною. Ця нова архітектура відкриває двері для впровадження інноваційних підходів, які раніше були важкодоступні.

Однією з ключових переваг цієї архітектури є можливість ефективного використання невеликих плагінів для великих систем автоматизованого проектування і систем 3D-моделювання, таких як AutoCad, 3ds Max та Rhinoceros3d. Цей підхід дозволяє уникнути необхідності створення окремого програмного забезпечення для кожного верстата, замість цього використовуючи універсальні інтерфейси для всіх машин.

Розроблена архітектура також надає можливість централізованого управління групами верстатів різних технологічних епох. Завдяки єдиній мережі та однаковому СЧПУ можливо ефективно керувати цими верстатами, забезпечуючи при цьому оптимальну вартість впровадження без необхідності використання додаткових пристроїв.

На даний момент прототип СЧПУ, що базується на описаній архітектурі, успішно використовується в несерійному виробництві у галузі ювелірного виробництва. Простота архітектури, легкість інтеграції програмних інтерфейсів, а також конкурентоспроможна вартість цього рішення можуть знайти широке застосування як на промислових підприємствах, так і серед індивідуальних користувачів, що прагнуть модернізувати свій верстатний парк та підняти його продуктивність на новий рівень.

РОЗДІЛ 1
ОГЛЯД СУЧАСНИХ ТЕНДЕНЦІЙ В РОЗВИТКУ ЧПУ

Терміни та визначення основних понять в області числового програмного управління металообробним устаткуванням встановлює стандарт ДСТУ 20523-80. Числове програмне управління верстатом (далі ЧПУ) – це процес управління обробкою заготовки на верстаті згідно з програмою, що здійснює управління верстатом, в якій дані задані в цифровій формі.

1.1. Класифікація архітектур систем ЧПУ

Виходячи з «Автоматизація підготовки управляючих програмч для верстатів з ЧПУ» [1], по своєму характеру рухи робочих, або виконавських органів системи ЧПУ класифікуються наступним чином: позиційні, контурні, універсальні, та синхронні. При позиційному управлінні переміщення робочих органів верстата відбувається в задані точки, причому траєкторія переміщення не задається.

Позиційні пристрої ЧПУ забезпечують автоматичне переміщення робочого органу верстата в потрібну координату, яка задану програмою, без обробки в процесі переміщення робочого органу. Ці пристрої застосовують в свердлувально-розточувальних та верстатах інших типів. Переміщення інструменту від однієї точки, або координати обробки до іншої виконується на прискорених ходах. Специфічною для цього класу ЧПУ є вимога забезпечення точності лише при зупинці в заданій координаті, при цьому вигляд траєкторії при переміщенні з однієї координати в іншу не задається. Час переміщення робочого органу верстату при цьому має бути мінімальним. Враховуючи те, що відсоток холостих ходів у верстатах з позиційними системами ЧПУ є значним, до приводу подач пред’являються вимоги високої швидкодії та забезпечення значних швидкостей переміщення при малій дискретності.

Контурне управління характеризується переміщенням робочих органів верстата по заданій траєкторії, причому це здійснюється із заданою швидкістю для виконання необхідного контура деталі. Контурне управління підрозділяється на:

контурні прямокутні системи ЧПУ, контурні криволінійні системи ЧПУ, та синхронні системи ЧПУ.

Контурні прямокутні системи ЧПУ використовуються у верстатах, в яких обробка заготовки проводиться лише при русі робочого органу по одній координаті, та оброблювана поверхня паралельна направляючим даної координати.

У більшості верстатів з ЧПУ застосовують прямокутні координати, тому такі системи отримали назву – прямокутні. В таких системах, як і в позиційних, програмуються кінцеві координати переміщення. Проте, в програмі задається швидкість руху відповідно до потрібного режимом обробки і переміщення виконується по черзі по кожній з координатних осей. У цих системах відставання або випередження за швидкістю відносно запрограмованого значення безпосередньо не викликає погрішності обробки, наприклад різання, оскільки інструмент продовжує рух по заданій траєкторії. Виникає лише порушення розрахункового режиму різання та пов’язана з цим зміна шорсткості оброблюваної поверхні і пружних деформацій системи верстат-деталь. Прямокутні системи управління використовують у фрезерних, токарних і шліфувальних верстатах.

Контурні криволінійні системи ЧПУ застосовуються у верстатах багатьох груп. Вони забезпечують формоутворення деталі при обробці в результаті одночасного погодженого руху по декількох керованих координатах. В загальному випадку число координат може бути більше трьох. Програму руху приводу, а також програму подач по окремих координатах при контурній і об’ємній обробках розраховують, виходячи із заданої форми оброблюваної поверхні деталі і результуючої швидкості руху, визначеної режимом обробки. Розузгодження приводу подач може привести до помилки обробки контура деталі. Контурні системи є найбільш складними із подібних систем, оскільки з точки зору алгоритму роботи пристрою ЧПУ (далі – ПЧПУ), так і з точки зору вимог, що пред’являються до приводу подач.

Різновидом контурних систем ЧПУ є синхронні, або синфазні системи, які переважно застосовуються в зубообробних верстатах. ПЧПУ задає постійне співвідношення швидкостей по двох, або більшому числу координатних осей

верстата, а формоутворення при цьому забезпечується завдяки конфігурації інструменту. Співвідношення швидкостей руху по осях задається програмою обробки, та зберігається протягом всього часу обробки заготовки даної деталі. В більшості випадків потрібно не лише забезпечити певне співвідношення середніх швидкостей руху по координатах, але і зберігати певне розузгодження, або синфазність в приводах координат. Одна з координат верстата ЧПУ, зазвичай головний привід, служить задаючою і на ній встановлюють вимірювальний перетворювач – датчик. Синфазна система входить як складовий елемент в ПЧПУ токарно-гвинторізних верстатів для забезпечення процесу нарізування різьблення в деталі.

Універсальне управління поєднує в собі принципи як позиційного, так і контурного управління, дозволяючи здійснювати позиціювання і рух робочих органів верстата по заданій програмою траєкторії. Таке управління є найефективнішим для багатоопераційних і багатоцільових верстатів.

По числу потоків інформації системи з верстатів ЧПУ можуть бути:

1. Розімкненими – один потік від ЧПУ до верстата. Основна перевага такої системи полягаї в її простоі.
2. Замкнуті – два потоки від ЧПУ до верстата і навпаки (датчики положення швидкості). Основна їх перевага – високоточне переміщення виконавчих органів.
3. Адаптивні, або самоналагоджувальні системи. Такі системи характеризуються управлінням, при якому забезпечується автоматичне пристосування самого процесу обробки деталі до умов обробки, які змінюються по певних критеріях. Вони, окрім основного потоку інформації, мають і додаткові, які дозволяють коректувати процес обробки з врахуванням деформації деталі, затуплення ріжучого або шліфуючого інструменту, коливання припуск і твердості заготовок та ін.

За способом принципів їх реалізації системи ЧПУ узагальнено можна класифікувати наступним чином: системи з апаратною реалізацією алгоритмів управління; системи, що побудовані на основі мікроконтроллерів; системи,

побудовані на основі ПЕОМ.

В залежності від рівня використання засобів обчислювальної техніки, системи ЧПУ класифікуються наступним чином:

1. Системи типа *NC*, або *NumericalControl*. Представляє собою числове програмне управління, яке здійснює адресацію команд, розрахунок деяких елементів геометрії деталі, інтерполяцію проміжних точок по опорним, реалізацію типових циклів по жорстко заданим алгоритмах, що реалізовані апаратним способом. Інформація в систему ЧПУ типа *NC* вводиться з програми, яка управляє кадрами, або порціями.

2. Системи типа *HNC*, або *HandNC*. Це системи з ручним завданням управляючої програми за допомогою пульту управління. Перевага таких систем в порівнянні з системами типа *MNC* полягає у відсутності необхідності з боку оператора у використанні функцій технолога-програміста.

3. Системи типа *CNC*, або *Computer Numerical Control*. Це системи управління зі вбудованими однією або декількома МікроЕОМ. або мікропроцесорами та з програмною реалізацією алгоритмів, що записуються в постійний запам’ятовуючий пристрій при виготовленні пристрою ЧПУ. Системи такого типу мають можливість формувати типові цикли обробки у відповідності до різних технологічних завдань. Системи *CNC* дозволяють програмувати логіку та режими роботи електроавтоматики силового устаткування верстата.

4. Система *DNC*, або *Direct Numerical Control*. Система, що здійснює управління групою верстатів від однієї ЕОМ, має загальну пам’ять для зберігання програм, що розподіляються по запитах, що надходять від верстатів. Такі ПЧПУ є пристроями вищого рангу і служать для організації погодженої роботи технологічних об’єктів, включених в єдиний комплекс.

5. Система *PCNC*, або *Personal Computer Numerical Control*. Це такі сучасні системи управління, що побудовані на основі ПЕОМ в індустріальному виконанні,

тобто в ударо- та віброзахищеному виконанні, а також в наявності в ньому спеціальної інтерфейсної плати, яка забезпечує сполучення ПЕОМ з приводами, датчиками, та електроавтоматикою верстата. Така концепція побудови дозволяє здешевити систему ЧПУ, легко адаптувати її до різних по функціональному призначенню верстатів. Це здійснюється шляхом корекції відповідних текстових файлів програмного забезпечення. Це дозволяє легко модернізувати застарілі системи ЧПУ *NC, MNC, SNC, HNC, CNC, DNC* до *PCNC*, що іноді успішно виконується за умови прийнятних характеристик точності модернізованого устаткування.

6. *STEPNCCNC,* або покрокова система управління. Це новітня система ЧПУ, яка побудована на основі систем *PCNC*, основна ідея якої полягає в виключенні участі людини в підготовці до процесу обробки деталі. До складу програмного забезпечення такої системи обов'язково входять пакети *CAD, CAPP* та *CAM*.

Функціонування такої системи здійснюється по наступним кроках:

1. Цей крок виконує система *CAD*. Він забезпечує автоматизацію розробки креслення оброблюваної деталі та підготовку геометричної і технологічної информації до передачі в *CAPP* і *CAM*- системи.

2. Система *CAPP* визначає технологію обробки заготовки деталі на устаткуванні: встановлює способи обробки деталі, призначає режими, встановлює необхідні для обробки інструменти, встановлює послідовність та склад переходів обробки.

3. Система *CAM* виконує за результатами попередніх кроків розрахунок траєкторії переміщень інструменту, визначення моментів дій та їх послідовність, а також послідовність подій управління приводами і електроавтоматикою верстата. Зазвичай результатом роботи *CAM*- системи є керуюча програма (КП), яка надалі відпрацьовується устаткуванням верстата. Це дозволяє легко модернізувати існуючі системи, такі як *DNC* та *PCNC* до рівня *STEPNCCNC*. В даний час здійснюється проектування *CAM*- систем, що безпосередньо управляють верстатом без формування КП.

Багато систем ЧПУ, що мають мікропроцесорне і комп'ютерне управління, виконують самодіагностування. В разі, якщо буде виявлено помилку, відповідне повідомлення буде виведено на екран ПЧПУ. Це полегшує процедуру налагодження системи. Але слід пам'ятати, що разом з новими комп'ютерними ПЧПУ в промисловості експлуатується значна кількість систем першої групи з схемною реалізацією алгоритмів. В даної роботї розглядається лише архітектуру *PCNC*-типа, як найбільш відповідаючу вимогам сучасності.

1.2. Загальні принципи побудови систем ЧПУ *PCNC*-типу

Серед розробників і виробників систем ЧПУ остаточно склалося розуміння того, що сучасні системи управління мають в повній мірі використовувати всі досягнення сучасних комп'ютерних технологій [3, 4]. В системах ЧПУ нового покоління виділяють системну платформу *PC*, або *PersonalComputer*, та прикладну компоненту*NC*, або *NumericalControl*, тобто ЧПУ – звідси і загальне позначення класу *PCNC* [5]. Системна платформа надає свої послуги модулям прикладної компоненти через прикладний інтерфейс *API*, або *ApplicationProgramInterface* кожного модуля, при цьому *API* приховує механізм реалізації будь-яких послуг.

У системі *PCNC* підтримується:

1. Мобільність прикладних модулів – тобто можливість їх переносу на інші системні платформи.

2. Комунікабельність модулів – тобто здатність їх до взаємодії через єдине комунікаційне середовище системної платформи.

3. Масштабованість системи в цілому – можливість змінювати, при необхідності, як функціональність прикладної компоненти, так і обчислювальні можливості системної компоненти.

Для прикладної компоненти можна позначити три рівні декомпозиції. Перший рівень полягає у виділенні так званих «завдань управління» [6,7], в число подібних завдань входять: геометрична, логічна, термінальна та, можливо, інші. Другий рівень декомпозиції полягає у виділенні модулів в складі завдань управління, при цьому

кожен окремий модуль відповідає фазі рішення задачі управління. Так, до складу геометричного завдання управління входять наступні компоненти: диспетчер режимів; інтерпретатор, або *ISO*-процессор [8]; інтерполятор; модуль управління приводами стеження. Деякі з цих модулів мають власний прикладний інтерфейс *API,* інші об'єднуються в групи з метою побудови загального інтерфейсу *API*. В якості прикладу подібної групи може служити так званий «канал ЧПУ», який об'єднує інтерполятор і інтерпретатор. Виділення каналу особливе корисно в багатоканальних системах ЧПУ, що працюють з декількома управляючими програмами,т а обслуговують декілька об'єктів управління – верстатів. Об'єднання модулів в групи знижує навантаження на комунікаційне середовище, але в той же час зменшується гнучкість системи ЧПУ та її здатність до реконфігурації.

Третій рівень декомпозиції є найбільш глибоким і, в принципі, не є обов'язковим: він означає виділення блоків у складі модулів. Блоки, самі по собі, не містять зовнішніх інтерфейсів *API* і, відповідно, доступу до глобального комунікаційного середовища. Внутрішня організація модуля при цьому має носити регулярний характер, а взаємодія блоків модуля між собою може здійснюватися по внутрішній локальній програмній шині. Прикладом подібної декомпозиції модуля може служити реалізація інтерполятора у вигляді таких блоків: диспетчера інтерполяції, блоку випереджаючого перегляду кадрів блоку розгону-гальмування, блоків окремих алгоритмів інтерполяції. Алгоритми інтерполяції можуть бути різними лінійним, круговим, сплайновим.

На четвертому рівні декомпозиції будується потокова модель для систематизації основних потоків даних в прикладній компоненті.

На п'ятому рівні будуєтьсяоб'єктно-орієнтована модель, яка зв'язує між собою модулі системи ЧПУ та потоки даних між собою в цілісну систему. На базі об'єктно-орієнтованої моделі здійснюється її реалізація. Цей рівень моделі дозволяє забезпечити нові можливості системи *PCNC,* як відносно архітектури программного забезпечення, так і відносно функціональних можливостей систем.

На шостому рівні будується компонентна модель, яка здійснює специфікацію інтерфейсів компонентів, та визначає рівень незалежності програмного забезпечення від особливостей апаратної реалізації.

На сьомому рівні створюється модель типа «клієнт-сервер» для визначення способу інтеграції компонентів між собою.

1.3. Модульна архітектура ЧПУ

Модульна архітектура визначає можливості і перспективи вирішення нових завдань. Кажучи про модульну архітектуру *PCNC*-системи, слід дати пояснення про саме поняття «модульність», формалізувавши структуру абстрактного модуля *PCNC,*

ЧПУ неможливо представити у вигляді єдиного монолітного блоку, в якому вирішені всі функціональні задачі. Необхідно розбити його на модулі, кожен з яких виступає як логічно закінчений блок, що реалізовує якесь конкретне призначення; що змінює свій внутрішній стан в процесі роботи; що взаємодіє з іншими модулями і здатний до конфігурації під конкретні задачі. Кожен модуль відносно є автономним, що сприяє загальній гнучкості і здатності до реконфігурурації системи в цілому.

Виходячи з перерахованих функцій, в деякому абстрактному модулі *PCNC*, що представлений на рис. 1.1, можна виділити чотири блоки: функціональний блок, блок стану машини, або *StateMachine,* блоки конфігуратора і інтерфейсу.

Функціональний блок виконує основне призначення модуля. Це, наприклад, інтерпретація кадру, інтерполяція кадру, управління електроавтоматикою, підтримка інтерфейсу з користувачем, та ін. Управління цим блоком ззовні здійснюється за допомогою команд інтерфейсу. Залежно від конкретної реалізації, функціональний блок може бути монолітним або складатися з елементів. Виконуючи ті або інші функції, модуль переходить з одного стану в інший.

Машина стану здійснює управління станами модуля. Алгоритм роботи машини стану добре піддається формалізації за допомогою графів станів. Перед початком виконання команди функціональний модуль звертається до машини стану для аналізу можливості виконання. З боку зовнішнього середовища за станом модуля можна здійснювати спостереження через його інтерфейс.

Вимоги до гнучкості і налаштування *PCNC*-системи дозволяє говорити як проконфігурацію окремого модуля, так і про конфігурацію системи в цілому.

Конфігурацію модуля здійснює блок конфігурації.

При цьому, можна здійснювати конфігурацію модуля під певний формат вхідних або вихідних даних. Конфігурацію ззовні здійснюють за допомогою інтерфейсу. Конфігурація *PCNC*-системи в цілому означає налаштування міжмодульного комунікаційного середовища (рис. 1.2).

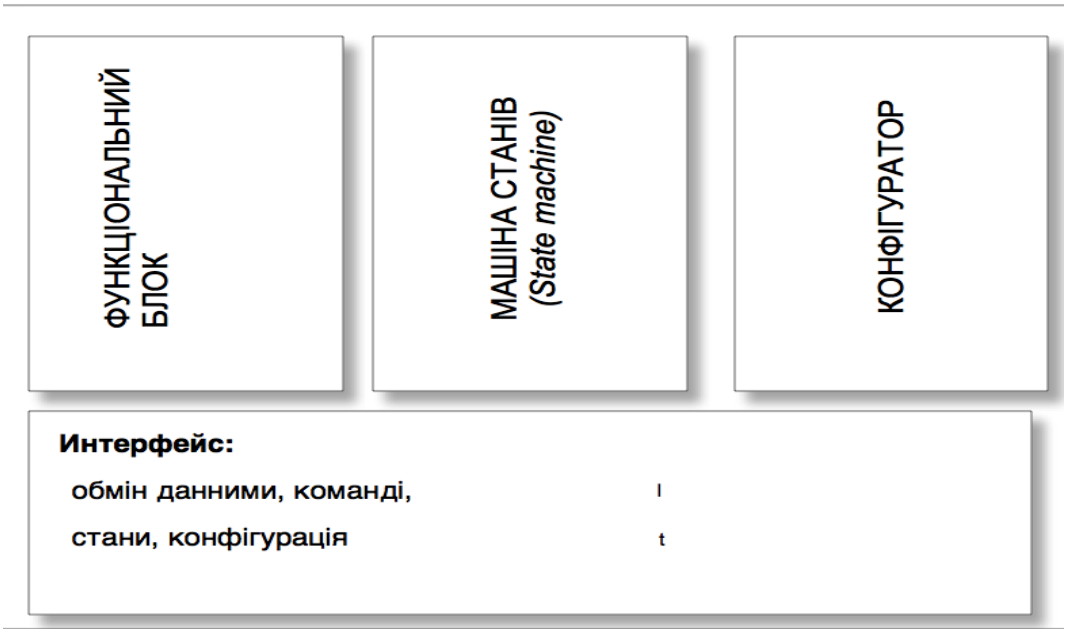


Рис. 1.2. Формальне представлення модуля СЧПУ

Вхідними даними модуля є кадри у форматі *ISO-7bit*, радіус інструменту для розрахунків еквідистантної корекції, таблиця зсувів нуля і т.д. Вихідними даними являються *IPD*-код (*InterpretatorData*), коди у вихідному форматі інтерпретатора, поточний вектор *G*-кодів і т.д. В результаті виконання перерахованих вище команд машина станів (*StateMachine*) може бути в одному з наступних станів: «Ініціалізувала», «Готова», «Працює», «Чекає», або «Помилка».

Конфігурація для інтерпретатора означає налаштування версії мови *ISO-7bit* – координатні осі, адреси, символи коментарів і т.д., а також налаштування *IPD*-кодів – вхідні дані для інтерполятора (*InterpolatorData*). Модульність слід розглядати на різних рівнях: на апаратному рівні, на рівні операційної системи і на прикладному

рівні. Якщо на апаратному рівні та на рівні операційної системи модульність підтримується використанням стандартних рішень в області апаратного і системного програмного забезпечення, то на прикладному рівні проблема реалізації модульності архітектури *PCNC*-системи залишається відкритою. Модульність апаратури означає реалізацію модулів у вигляді *PC*-плат і плат розширення – таких як мережева карта, *SERCOS*-карта управління приводами, карта управління введенням-виводом на базі протоколів *CANBUS* або *PROFIBUS* [9, 10] і т.д. Усунення будь-якої несправності в системі здійснюється простою заміною модуля, що вийшов з ладу.

Надзвичайно актуальна підтримка на рівні операційної системи таких апаратних конфігурацій, які відмінні від конфігурації звичайних ПК: багато промислових комп'ютерних систем можуть працювати без монітора або клавіатури, використовують *FLASH*-диски та спеціальні шини. Модульність на цьому рівні означає можливість конфігурувати операційні системи під конкретну конфігурацію. Наприклад, інсталюючи *Windows* 10, та більш ранніх, за допомогою продукту *ComponentIntegrator фірми VenturCom*, можна встановити оптимальний для конкретної ситуації набір із понад 50 компонентів операційної системи. Аналогічні можливості пропонує проект *LinuxCNC.*

Під модульністю на прикладному рівні розуміють розбиття системи ЧПУ на функціональні частини, які розробляються різними групами розробників, та зазвичай не передбачають загального механізму взаємодії. Труднощі при комплектації системи з'являються вже на рівні включення модулів в систему: виникає безліч хаотичних міжмодульних зв'язків. Такі системи практично не піддаються модернізації.

Новий погляд на модульність, який був запропонований в дипломній роботі, передбачає: формальну структуру модуля, наявність чітко позначеного інтерфейсу для кожного модуля і загального механізму взаємодії між модулями. Це передбачає таку архітектуру, в якій комунікаційне середовище виконане особливим чином у

вигляді каналу – магістралі для обміну командами та даними. Магістраль служить єдиним механізмом не лише для внутрішнього обміну між модулями, а і для обміну інформацією із зовнішнім середовищем. Кожен модуль, що підключений до магістралі, запрошує деякий сервіс від інших модулів або від зовнішнього середовища.

Модульність в *PCNC*-системі дозволяє здійснювати наступне:

– Здійснювати різну комплектацію *PCNC*; наприклад, використовувати програмно- або апаратно-реалізований програмований контроллер, використовувати традиційний привід подачі або автономний цифровий привід подачі з берсів- інтерфейсом і т.д.;

– компонувати *PCNC*-систему під конкретні технологічні завдання і для конкретного типа верстата – токарного, фрезерувального, шліфувального, електроерозійного, лазерного та ін.; створювати можливість операційної спеціалізації верстатів;

– збирати *PCNC*-систему з компонентів від різних виробників, обираючи при цьому не лише кращі компоненты, але і кращі їх поєднання.

– Сьогодні особливо гостро стоїть проблема забезпечення проблемної орієнтації системи ЧПУ. Реалізація цього можливо лише за рахунок багатократного використання готових рішень, які конструктивно оформлені у вигляді модулів. Таким чином, виникає можливість:

– розпаралелювати процес розробки систем;

– скоротити терміни розробки і зменщити собівартість системи за рахунок використання готових модулів;

– досягти оптимальних технологічних, цінових і експлуатаційних параметрів при компонуванні системи;

– підвищити надійність та ремонтоздатність всього пристрою при експлуатації *PCNC*-систем.

Модульний підхід до архітектури *PCNC*-систем вимагає вирішення абсолютно

нових завдань на прикладному рівні, а саме:

– створення єдиного механізму обміну між модулями;
– формування єдиного підходу до реалізації модулів.

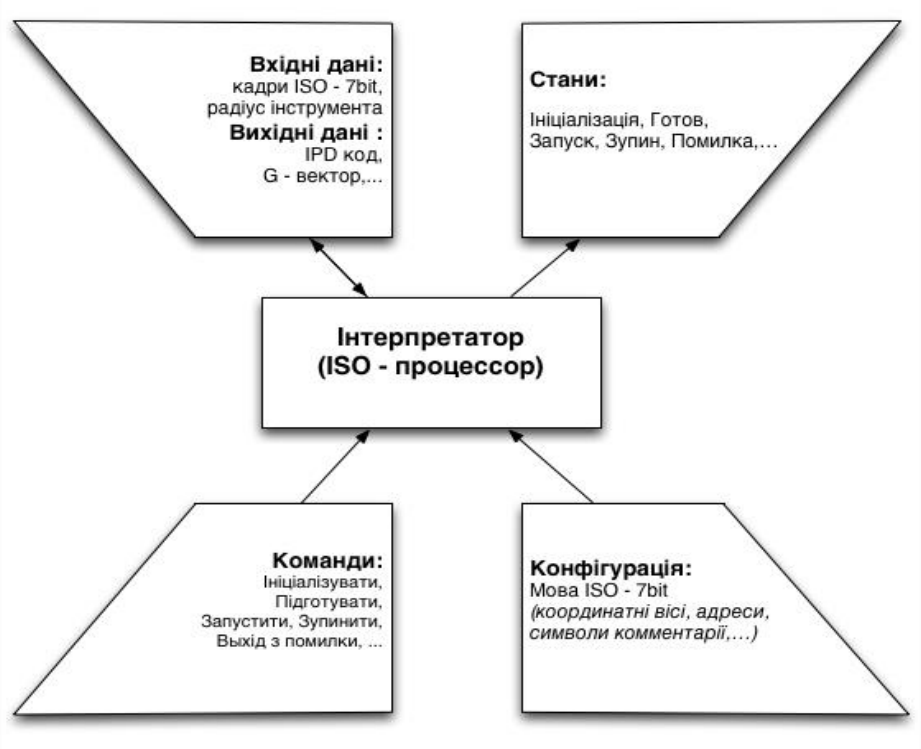


Рис 1.3. Формальне представлення інтерфейса модуля інтерпретатора

Забезпечення міжмодульної комунікації має на увазі створення середовища, яке включає в себе програмні та апаратні засоби, засоби зв’язку і інтерфейси, що використовують специфіковані формати даних та протоколи. В основі такого середовища лежать стандарти, які розвиваються, доступні і є загальновизнаними. Це середовище забезпечує високий ступінь переносимості, комунікабельності і масштабування.

Єдиний підхід до реалізації модулів вимагає стандартизацію інтерфейсів модулів, стандартизацію протоколів обміну по магістралі і реалізацію інтерфейсів на рівні, близькому до рівня операційної системи. Це означає використання клієнт-серверних стосунків при організації транзакцій, залучення об’єктно-орієнтованого підходу до визначення макроструктури і на рівні технології програмування. Викладене вище зумовлює принципово іншу в порівнянні з відомими рішеннями

організацію системи ЧПУ, в якій навіть модулі з традиційними найменуваннями мають нове функціональне і алгоритмічне наповнення, а також для цього потрібна нова програмна реалізація. Особливо важливу роль набуває прикладний рівень, який визначає призначені для користувача характеристики і рівень сервісу для оператора.

У системну платформу *PCNC* входять наступні частини: апаратна частина, операційна система і засоби підтримки міжмодульної комунікації.

1.4. Порівняння одинкомп'ютерної та двохкомп'ютерної архітектур *PCNC*

На сьогоднішній день *PCNC*-системи, зазвичай, реалізовані у вигляді одно- або двохкомп'ютерних рішень (рис. 1.4.). Класичною для сучасних професійних *PCNC*-систем є двохкомп'ютерна модель, що наведена на рис 1.3. б. В ній застосовують дві операційні системи різних типів: *Windows*10, або більш ранню версію в терміналі; одну з операційних систем реального часу в машині реального часу – *EmbeddedLinux, QNX* або інші *UNIX*-системи.

Структуру комунікаційного середовища *PCNC* (рис.1.5), зручно зіставити з класичною багаторівневою моделлю архітектури відкритих систем типу *ISO-OSI* [9]. У комунікаційному середовищі прийнято виділяти чотири рівня, згідно з *OSACA*.

На нижньому рівні знаходяться стандартні комунікаційні засоби, які охоплюють фізичний та канальний рівні моделі *ISO-OSI*. Для двохкомп'ютерного варіанту технічні засоби можуть бути реалізовані на базі стандартних протоколів *Intranet*.

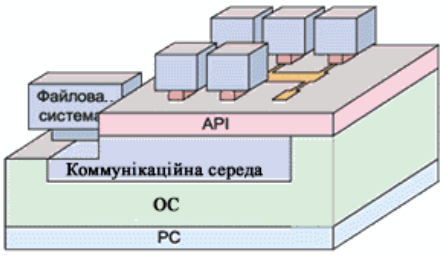
Вище ідє рівень *MTS* – *MessageTransportSystem*, система транспортування повідомлень. Цей рівень реадізує функції мережевого і транспортного рівнів моделі *ISO-OSI*. Для двохкомп'ютерної *PCNC*-системи в рівень *MTS* може бути вбудований чотирьохрівневий стек протоколів *TCP/IP*. Система транспортування повідомлень *MTS* підтримує базові транспортні функції *SEND* та *RECEIVE* для обміну різними повідомленнями між різними парами модулів. *MTS* приховує від прикладної частини *PCNC* як стандартні засоби комунікації – такі, як *TCP/IP* або ін., так і всі інші функції операційної системи, які потребує верхня частина комунікаційного середовища.

Модулі, що розташовані на різних комп'ютерах, взаємодіють за допомогою *MTS* таким чином, немов би вони знаходилися в одному комп'ютері. *MTS* використовує протокол з попереднім встановленням з'єднання – це означає, що два *MTS*-користувача, які взаємодіють між собою, мають спочатку встановити з'єднання, а вже потім користуватися їм. Розташований ще вище рівень рівень *ASS* – *ApplicationServicesSystem*, або система послуг для прикладних програм. Цей рівень відповідає сесійному, представницькому і прикладному рівням в моделі *ISO-OSI*.

Модуль *ASS* надає послуги, з попереднім встановленням з'єднання, прикладним модулям *PCNC* наступним чином. Якщо модуль потребує послуги віддаленного зв'язку, тоді система *ASS* формує повідомлення, що включає отриману від модуля інформацію.

Далі іде виклик послуги рівня *MTS* для посилки повідомлення. На стороні прийому *ASS* інтерпретує повідомлення, та передає його далі модулю-одержувачу. Перед передачею повідомлення *ASS* готує дані в тій формі, якої потребує приймаюча система. Якщо два, або декілька модулів мають загальну *ASS*, то вони не потребують *MTS* для підтримки інтерактивності.

Четвертий рівень лише умовно відноситься до комунікаційного середовища. Це прикладний інтерфейс всіх модулів *NC*-підсистемы, або *API*. Цей рівень реалізований, як диспетчер комунікаційних об'єктів *COM – Communication Object Manager*, який асоційований з кожним модулем *API*, що є в наявності. Диспетчер *COM* представляє собою оболонку, що наповнена програмами, що призначені для створення і активізації та деактивації так званих «комунікаційних об'єктів».



а

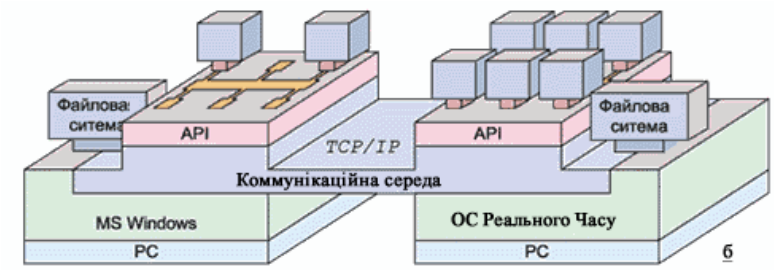


Рис. 1.3. Основні архітектури систем ЧПУ типу *PCNC*:
а – однокомпьютерна архітектура; *б* - двохкомпьютерна

Рис. 1.4. Порівняння моделей *OSACA* та *OSI*

Двохкомп'ютерна модель має щонайменше одну важливу перевагу – вона реалізує досить просте рішення проблеми роботи в реальному часі. Це реалізується за рахунок того, що всі операції, що вимагають пріоритету реального часу, такі як робота інтерполятора, управління двигунами, зворотний зв'язок, та інші, виконуються на окремому комп'ютері.

Однокомп'ютерна модель (див. рис 1.4, а) передбачає використання звичайного ПК. Стандартну апаратуру розширюють за рахунок спеціальних пристроїв, серед яких можуть бути наступні: контролери приводів стеження, програмований контроллер електроавтоматики, спеціальні пристрої для управління технологічними процесами і т.д. Перехід від двохкомп'ютерної моделі до одинкомп'ютерної здійснюється формальним перенесенням математичного забезпечення *PC*-

підсистеми в *NC*-підсистему на рівні завдань. Однокомп'ютерна модель в даний час застосовується у верстатах, що підключаються до звичайного ПК як периферія. Така модель вважається перспективним напрямом розвитку. Оскільки основна сфера застосування даної архітектури поки що переважно *Hobby CNC*-системи, розглянемо однокомп'ютерну архітектуру на основі такої *PCNC*-системи. Типова схема такого верстата (рис 1.5).

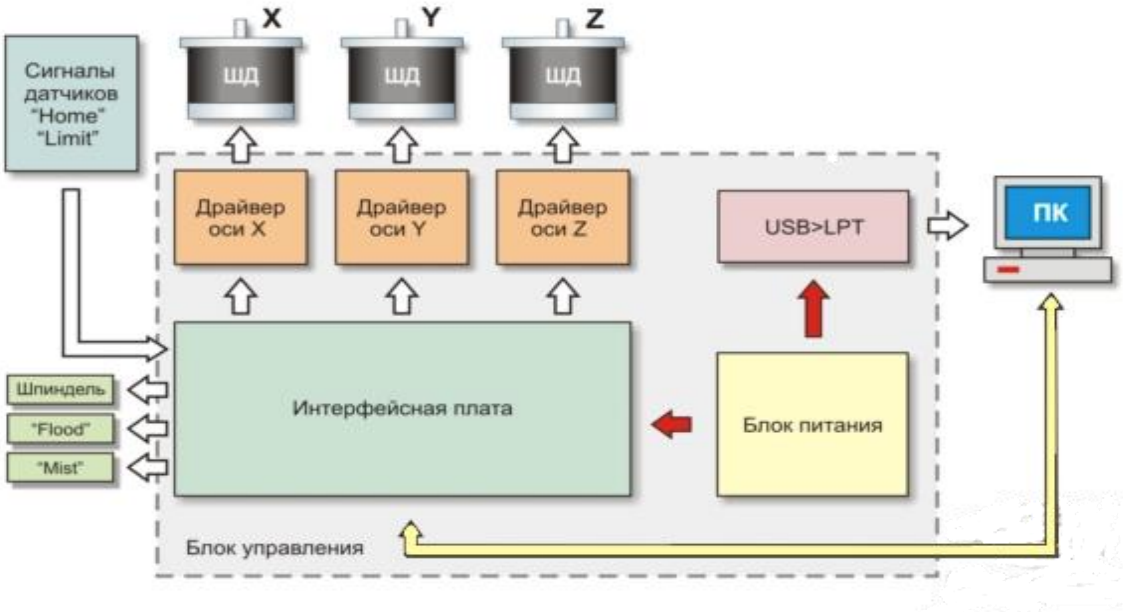


Рис 1.5. Типова схема непромислового верстата з ЧПУ на основі однокомп'ютерної архітектури *PCNC*

В більшості випадків в подібних верстатах використовуються крокові двигуни через їх дешевизну. Виходячи зі схеми на рис 1.5 видно, що типовий пристрій ЧПУ для непромислового верстата є набором драйверів крокових

двигунів в кількості, що рівна кількості координат, з якими працює верстат.

Контроллери крокових двигунів підключаються до интерфейсної плати, яка виконує суто передавальні функції. Не дивлячись на певне застаріння, велику популярність мають інтерфейсні плати з опторазв'язкою порту *LPT*, оскільки робота з *LPT* є досить простою. Так, в однокомп'ютерній системі всі завдання ЧПУ вирішуються на робочій станції і, відповідно, на неспеціалізованій ОС, що може створювати ряд проблем з надійністю. Основні переваги і недоліки однокомп'ютернихархитектур *PCNC* (табл.1.1).

Таблиця 1.1.

Порівняння архітектурних вирішень *PCNC*

Двохкомп'ютерна система	Однокомп'ютерна система
1. Підсистеми автономні. Відмова *PC*- підсистеми не приводить до відмови МС-підсистеми	1. Менша собівартість
2. Оновлення і еволюція однієї підсистеми не впливає на іншу	2. Полегшено інтеграцію модулів і управління процесами
3. Існує можливість вирішення складних технологічних завдань в окремому комп'ютері термінального завдання	3. Необхідно вирішувати проблему реального часу для десктопноїопераційноїсистеми

1.5. Програмне забезпечення для систем ЧПУ *PCNC*- типу

Основний функціонал ПЗП ЧПУ призначений для вирішення завдань управління технологічним процесом обробки деталі на верстаті. Ці завдання управління пов'язані з виконанням певного класу функцій, і їх рішення є проблемною частиною ПЗП ЧПУ. Тому при розробці базового ПЗП вони мають бути сформульовані максимально точно. Способи вирішення завдань управління роблять визначальний вплив на архітектуру системи.

Існують три глобальні задачі управління в ЧПУ (рис. 1.6.):

1. Геометрична задача управління. Забезпечечує програмою циклу обробки деталі на верстаті, тобто здійснює формоутворення деталі.

2. Логічна задача управління. Забезпечечує управління електроавтоматикою верстата.

3. Термінальна задача управління. Забезпечечує інтерфейс оператора та взаємодію з виробничим оточенням.

Далі розглядаються всі ці три глобальні задачі управління систем ЧПУ (рис.1.7).

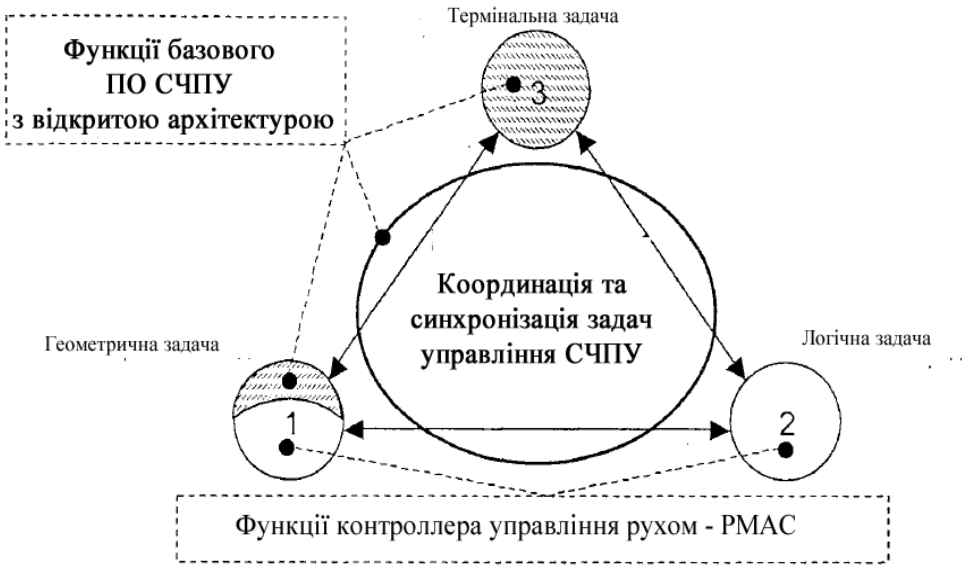


Рис. 1.6. Задачи управління в СЧПУ

1.5.1. Геометрична задача управління

Команди переміщення робочих органів верстата можуть міститися в УП, задаватися оператором вручну з пульта управління, передаватися через інформаційні обчислювальні мережі (рис. 1.7.). Геометричне завдання ЧПУ в основному визначається наступними трьома послідовними фазами перетворення команди переміщення робочих органів верстата:

1. Розпізнавання геометричної інформації – Інтерпретатор керуючих програм.
2. Інтерполяція траекторії – Інтерполятор.
3. Управління приводами робочих органів – модуль управління приводами стежження.

Розглянемо фази відпрацювання геометричного завдання, яке задається за допомогою УП, оскільки рішення цієї задачі є основним для базового ПЗ системи ЧПУ.

Перша фаза має два етапи виконання: завантаження КП в пам’ять ПК і його інтерпретація.

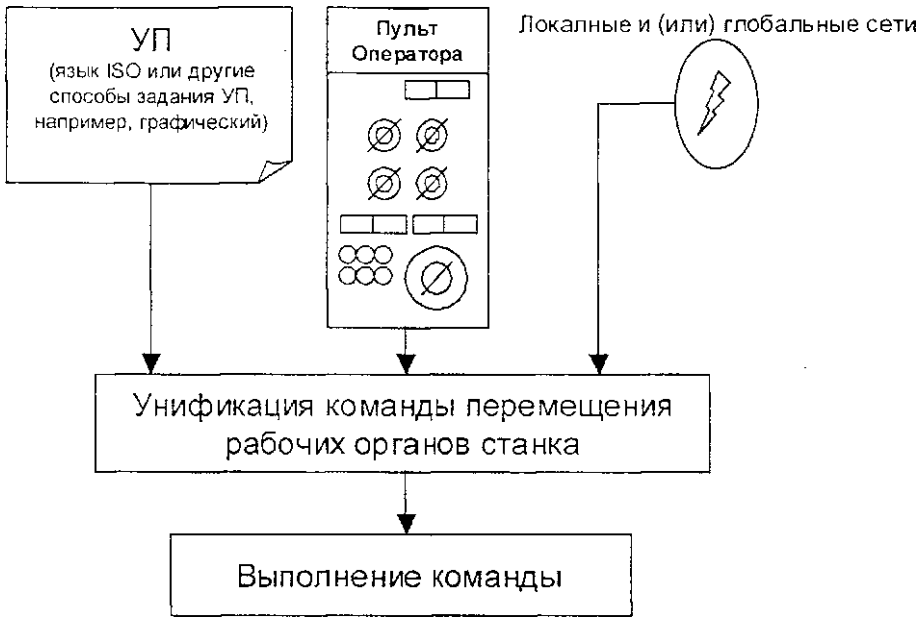


Рис. 1.7. Способи завдання команди переміщення в СЧПУ

Процес створення КП можна розглядати як окрему фазу, але будемо вважати, що КП у нас вже створена. На цьому етапі відбувається попередній перегляд КП на предмет лексичного контролю правильності завдання КП Це необхідно для виявлення можливих синтаксичних та семантичних помилок КП під час виконання нею своїх функцій, а також для запобігання можливих некоректних технологічних ситуацій. В деяких випадках повний попередній перегляд КП може бути пропущений. Наприклад, якщо КП динамічно завантажується з віддаленого терміналу, тоді попередній контроль, замість базового ПЗ СЧПУ повинно виконувати ПЗ на ввіддаленому терміналі.

На другому етапі відбувається виконання, або активізація КП. Вихідна інформація перетвориться у внутрішній формат верстата з метою передачі на наступну фазу геометричного завдання. Процес перетворення даних з одного

формату в іншій в СЧПУ проводиться, зазвичай, за допомогою інтерпретатора і називається інтерпретацією програми [11, 12].

Під час процесу інтерпретації вихідних даних інтерпретатор проводить розрахунки, виходячи з правил вхідної мови УП. В стандарті *ISO* функції в УП задаються за допомогою символів, або літер: *G*, *M*, *T*, *H* та *D*, після яких задається номер функції (додаток. А).

В одному кадрі КП не можуе бути задані більш, ніж одна *G*-функція з однієї групи. По приналежності до тієї або іншої групи *G*-функцвї мають різний пріоритет виконання. Послідовність завдання функції у вхідному кадрі не впливає на пріоритетність виконання інтерпретатором, завдяки чому досягається гнучкість мови КП.

Друга фаза геометричного завдання – інтерполяція траєкторії руху інструменту. З цієї фази функції геометричного завдання вирішуються в реальному масштабі часу. У двохкомп'ютерній архітектурі *PCNC* реалізація цієі і наступної фази геометричного завдання реалізується на машині реального часу масштабу часу, а в одинкомп'ютерній – *CAM* програмою. Існує ряд алгоритмів інтерполяторів, основні з яких будуть розглянуті в розділі 2.

Якщо здійсеюється робота з сервоприводами і датчиками зворотного зв'язку, то на виході інтерполятора, в цифро-аналоговому перетворювачі ЦАП імпульсний сигнал, який здійснює управління, перетворюється в аналоговий сигнал. Далі цей сигнал поступає на вхід приводу стеження, де елемент порівняння виробляє розузгодження, порівнюючи фз-фазу вхідного сигналу з ц- фазою – фактичним положенням вихідного валу приводу, яке визначається вимірювальним перетворювачем, тобто датчиком зворотного зв'язку ДЗЗ.

Оскільки обьем даної роботи обмежується дослідженнями, заснованими на системах ЧПУ, що мають крокові приводи, де єдиними датчиками зворотного зв'язку є кінцевики – детальний розгляд інших типів не є доцільним і в дипломній роботі розглядатись не буде.

1.5.2. Термінальна задача управління

Термінальне завдання має дві складові: перша є потоком завдань від оператора або по обчислювальній мережі до всіх завдань СЧПУ; а друга – потік інформації, що надходить від СЧПУ до оператора.

Перша складова термінального завдання призначена для вирішення комплексу завдань управління вибору режиму роботи –автоматичний або ручний, введення алфавітно-цифрових даних, редагування вводної інформації, выбору дискретності переміщень, введення виконавчих команд типа «Включити», «Вимкнути», «Старт», «Стоп», завдання елементів геометрії майбутньої деталі, активізації циклів дискретної електроавтоматики верстата та інші.

Управління процесом та об’єктом відбувається протягом всієї стаціонарної роботи системи в режимі діалогу. В нашому контексті діалог – це зміна режиму роботи з автоматичного на ручній або навпаки, завдання КП на виконання, тимчасова зупина роботи КП та її подальший пуск і тому подібне. Оператор з СЧПУ працює в основному в режимі діалогу (рис. 1.7), в якому здійснюються наступні функції: системна робота, управління об’єктом і процесом, автоматизоване проектування КП, редагування КП.

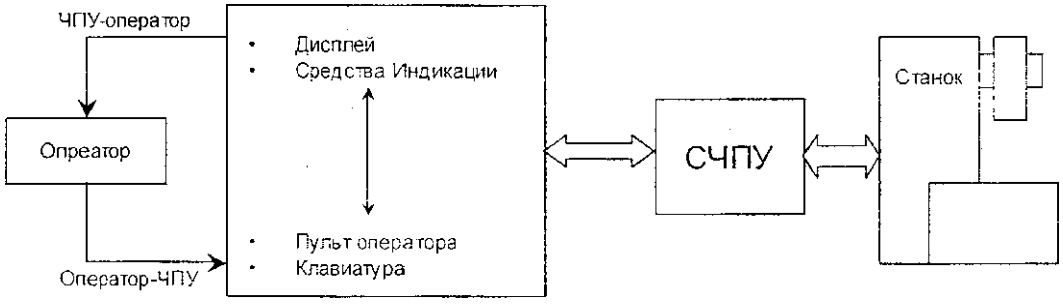


Рис. 1.7. Структурна схема організації дїалога оператора з СЧПУ

Діалогові функції системної роботи оператора включають наступні дії: маніпулювання наборами константних змінних, які необхідні для функціонування СЧПУ – це зміна таких параметрів налаштування, як геометрія переміщення ріжучих інструментів і базових точок робочих координатних систем; робота з бібліотеками

КП та з іншими базами даних; вхід в інформаційний обмін через обчислювальні мережі, такі як *Internet* і *Intranet*.

Функція автоматизованого проектування КП в режимі діалогу передбачає генерацію за допомогою спеціальних інструментальних засобів. Наприклад, для того, щоб згенерувати КП типових процедур для обробки деталі, оператор натискає спеціальні кнопки в певній послідовності, вказані на дисплеї – тобто входить в діалог з системою та задає параметрам типових процедур конкретні цифрові значення. Генерація КП для обробки складних деталей можлива за допомогою вбудованих в базову систему, або зовнішніх, набагато потужніші *CAD/CAM*-системи.

1.5.3. Логічна задача управління

В сучасних металоріжучих верстатах багато що з допоміжних функцій автоматизовано. Автоматизованими є розгін, гальмування, перемикання швидкостей, зміна інструменту та зміна заготовок, вмикання чи вимикання охолоджування та інші, управління якими в основному формується через подасу одиночного або групових дискретних імпульсів на датчики вхід/виходів системи. У цьому саме і полягає суть логічного завдання. Дискретні вхідні датчики служать основними джерелами, які активізацують ті або інші функції логічного завдання, які на верстатах з ЧПУ реалізуються у вигляді циклів дискретної електроавтоматики.

Цикли роботи електроавтоматики можуть бути активовані також за допомогою команд УП, за допомогою завдання спеціальних *M*-функцій (дви. Додаток А). По ходу виконання УП з чергового кадру вибирається *M*-функція та активізується відповідний до неї цикл дискретної електроавтоматики. До того ж, технологія обробки деталі вимагає, щоб деякі *M*-функції були активовані ще до виконання руху, що був задан в кадрі, а деякі – вже після. А в системі МБН це завдання вирішується на рівні інтерпретатора.

В архітектурі сучасних професійних СЧПУ завдання дискретної електроавтоматики вирішуються за допомогою спеціального пристрою – програмованого логічного контроллера (ПЛК). У більш простих,аматорських

системах ЧПУ вони вирішуються програмно.

1.6. Об’єднання систем ЧПУ в локальну мережу *ETHERNET* для передачі технологічної інформації і моніторингу роботи

Верстати з ЧПУ підключаються до локальної мережі *Ethernet* для прийому-передачі технологічних програм та моніторингу їх роботи, ця мережа, зазвичай, вже існує на промисловому підприємстві. Концепція запропонованої мережі для верстатів з ЧПУ, як частини комп’ютерної мережі підприємства, робить можливим встановити єдині стандарти передачі інформації в рамках підприємства, та здійснювати загальний облік роботи устаткування різного призначення, різних моделей і років випуску.

Об’єднання ЧПУ в єдмну комп’ютерну мережу дозволяє створити апаратно-програмний комплекс для централізованого управління групами верстатів з ЧПУ [14].Такий комплекс має складатися з сервера управління, в якості якого виступає ПК зі спеціалізованим ПЗ, призначеним для управління групою верстатів з ЧПУ, і клієнтів, якими є ці верстати (рис 1.8). За наявності спеціального ПЗ така система забезпечує наступні можливості:

1. Передача КП системі ЧПУ верстата з CAM програми будь-якого комп’ютера мережі, в тому числі з використанням *Internet* та *VPN* тунелів. Тут мається на увазі передача КП з комп’ютера в СЧПУ верстата, а також передача УП з СЧПУ верстата в комп’ютер. Передачу КП з комп’ютера в СЧПУ верстата повинні забезпечувати всі реалізації апаратно-програмного комплексу. Деякі реалізації комп’ютерних мереж не підтримують предачу КП з СЧПУ верстата в комп’ютер, але тоді втрачається можливість збереження КП, яка була відредагована безпосередньо на верстаті. Для збереження зміні вона записується обслуговуючим персоналом, а потім слід внести ці зміни до вихідної КП на комп’ютері. Такий спосіб редагування КП займає багато часу та підвищує вірогідність внесення помилок. Отже, відсутність можливості виведення КП з СЧПУ верстата збільшує один з елементів собівартості виробу. Існують СЧПУ, що не дозволяють редагувати КП – це системи, що не містять

власну пам’ять. Для таких систем наявність можливості виведення КП з СЧПУ в комп’ютер в апаратно-програмного комплексу є зайвим.

2. Передача системної інформації. Сюди входить передача програм діагностики, про інструмент, даних про коректори, про нульові зсуви. Ця інформація передається з сервера управління СЧПУ до СЧПУ конкретного верстата. Практично всі реалізації мережевого управління СЧПУ апаратно-програмного комплексу мають можливість передавати системну інформацію від комп’ютера до СЧПУ. В разі, коли відсутня така можливість, то системна інформація вводиться вручну, що збільшує витрати часу, а отже і вартість експлуатації такого верстата. Системи, які не мають можливості передавати КП з СЧПУ в комп’ютер, також не передають системну інформацію.

3. Моніторинг роботи устаткування. Це процес здобуття сервером управління даних про стан технологічного устаткування, які надходять від СЧПУ верстатів.

Мережу організовують декількома способами: прокладають кабелі – вита пара, коаксіальні, оптоволоконні, встановлюють біля комп’ютера і біля верстата модем та сполучають по існуючих телефонним лініям, а також без використання кабелів за допомогою берпровідного зв’язку *Wi-Fi*. При з’єднанні мережі і комп’ютера технічних труднощів не виникає, а при з’єднанні комп’ютерної мережі мережі і СЧПУ верстата іноді можуть виникнути проблеми. Все залежить від того, які інтерфейси має СЧПУ. Розглянемо детальніше.

1. Мережа організовується за допомогою кабелів типу витої пари. Зазвичай це мережа *Ethernet*. Практично у всіх сучасних СЧПУ є підтримка інтерфейсу *Ethernet*. В цьому випадку мережевий кабель безпосередньо з’єднується з СЧПУ. Якщо ж інтерфейс *Ethernet* відсутній, то, за наявності одного із стандартних інтерфейсів, що поставляються з багатьма зарубіжними СЧПУ, таких як *RS-232*, IPPR, IPPC, необхідно поставити перетворювач сигналів між СЧПУ і комп’ютерною мережею. Для такого перетворення сигналів використовується малопотужний комп’ютер, або можна використовувати додатковий пристрій з підтримкоює інтерфейсу *Ethernet*. В цьому

випадку можливі обмеження на використання можливостей, що надаються комп'ютерною мережею.

2. При організації мережі на основі оптоволоконних кабелів або за допомогою бездротових технологій *Wi-Fi*, біля верстата встановлюється комп'ютер. Автономні перетворювачі сигналів цих мереж в сигнали стандартних інтерфейсів і *Ethernet* потрібно розробляти відповідно конкретного завдання. Якщо комп'ютер біля верстата для організації перетворення сигналів, то цей комп'ютер і СЧПУ можливо поєднати між собою через практично будь-який інтерфейс – *COM, LPT, USB, Ethernet* і т.д. Схема передачі інформації за допомогою бездротової мережі *Wi-Fi* (рис. 1.9) зображена. Там цифрою 1 позначено будівлю, де знаходиться комп'ютер з архівом КП, цифрою 2 – будівлі з верстатами, 3 – пристрої приймання/передачі радіосигналів, ці пристрої сполучені з антенами, що розташовані на дахах будівель.

3. Організувати мережу також можливо, використовуючи для цього існуючі телефонні лінії за допомогою модему – з технічного боку це досить легко організувати для великої кількості різних СЧПУ, ніж в попередніх пунктах. Це пов'язано з тим, що інтерфейс *RS*-232, за допомогою якого модем здійснює зв'язок із зовнішніми пристроями, частіше за все присутній в СЧПУ. Якщо ж цього інтерфейсу в пристрої немає, то цей недолік можно легко усунути – для цього існують стандартні перетворювачі: *RS*232-ИРПР та *RS*232-ІРПС.

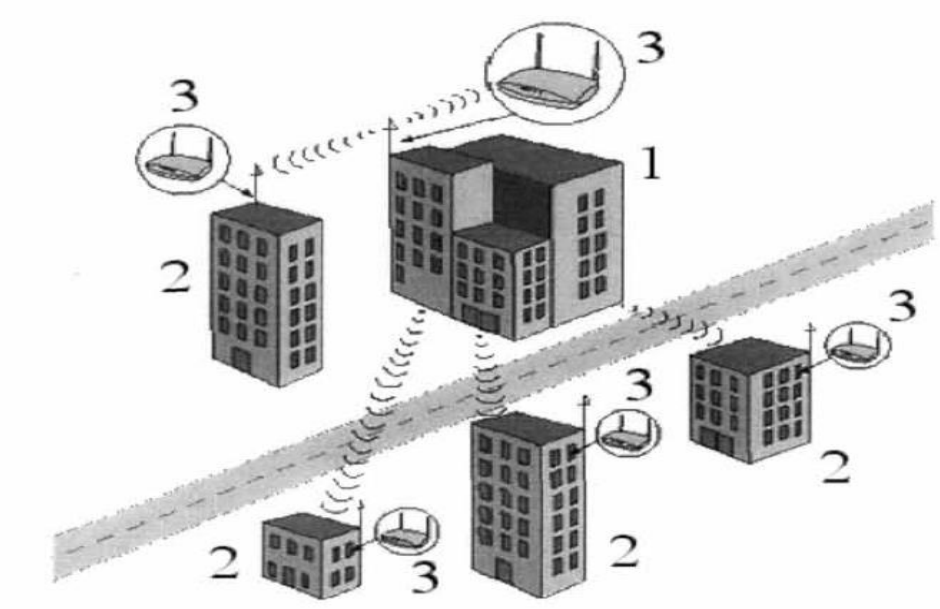


Рис. 1.8. Схема передачі інформації за допомогою бездротових технологій

Розглянемо з’єднання комп’ютера, що зберігає архів УП, та верстата з ЧПУ за допомогою комп’ютерної мережі (рис. 1.9).

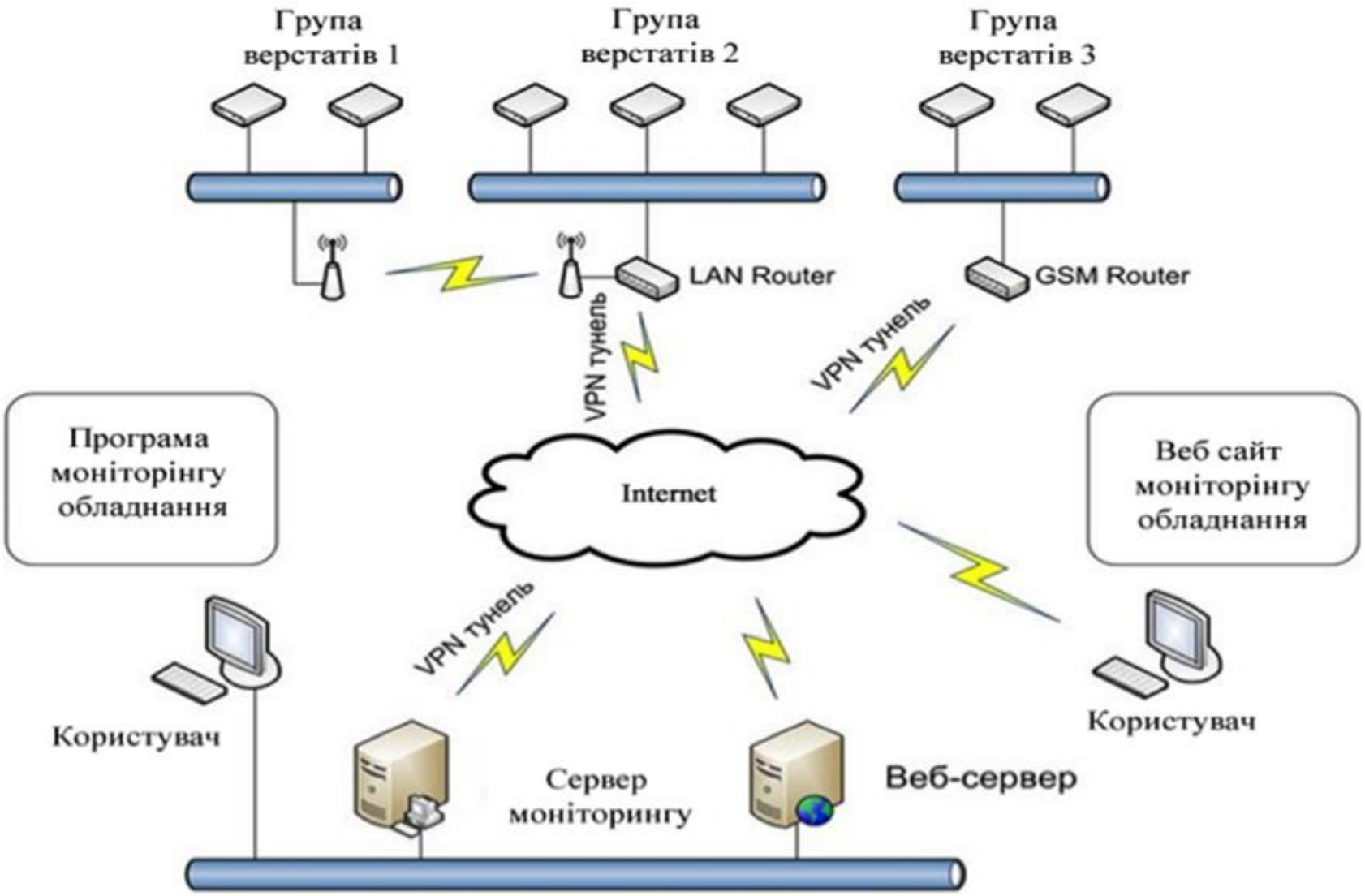


Рис 1.9. Топологія мережі апаратно прошоамного комплексу мережевого керування групами верстатів ЧПУ

При описанні способів з’єднання мережі і СЧПУ не враховувалися такі системи, які не мають стандартних інтерфейсів – наприклад ткі, що використовують для введення інформації магнітну стрічку. В цьому випадку під’єднання СЧПУ до мережі можна організувати лише за допомогою самостійно розробленого пристрою, та по суті, це буде організація відповідного інтерфейсу в СЧПУ.

При використанні комп’ютерної мережі для передачі інформації між комп’ютером і верстатом з ЧПУ відстань між ними може досягати декількох кілометрів, а при використанні додаткового устаткування, такого, як повторювачі та комутатори, відстань взагалі не обмежена.

Комп’ютерна мережа дозволяє використовувати всі можливості, що описані вище, для управління групами верстатів. Але слід пам’ятати, що в промислових масштабах обмеження може накладатися на саме середовище передачі даних. Так, наприклад, не завжди є можливість протягнути мережеві кабелі від комп’ютера до верстата з ЧПУ – в цьому випадку треба використовувати *WiFi* або існуючі телефонні лінії. Знову таки, наявність перешкод для проходження сигналу від потужних джерел струму та силових установок верстатів часто може зробити неможливим використання бездротових мереж.

Мережа розглянутого типа маєє функціональні можливості, що є необхідними для успішної роботи на великих підприємствах:

- можливість підключення верстатів різних моделей і років випуску;
- невисокою вартістю;
- простотою впровадження та обслуговування.

Так, об’єднання верстатів з ЧПУ в єдину мережу дає можливість централізованого управління цехом, наводить до збільшенні автоматизації виробництва та скороченню необхідної кількості кваліфікованих фахівців для роботи з верстатами.

Висновки за розділом

В першому розділі проведено детальний аналіз та огляд найсучасніших тенденцій у розвитку верстатів з числовим програмним керуванням (ЧПУ). Було досліджено принципи класифікації різноманітних систем ЧПУ, описано їх основні типи та архітектури, а також висвітлено їх потужності та характеристики.

Додатково, були розглянуті загальні принципи побудови верстатів з ЧПУ, зокрема, враховуючи аспекти їх конструкції та функцій. Докладно проаналізовано програмне забезпечення, призначене для ефективного управління такими системами, а також оглянуто ключові принципи керування цими системами.

Крім цього, була розглянута можливість об'єднання різних систем ЧПУ в єдину комп'ютерну мережу, а також принципи її організації. Висвітлено переваги такого об'єднання, зокрема створення централізованої системи управління та забезпечення взаємодії між верстатами з ЧПУ.

РОЗДІЛ 2
РОЗРОБКА УНІВЕРСАЛЬНОЇ СИСТЕМИ ЧПУ З ГНУЧКОЮ МОДУЛЬНОЮ АРХІТЕКТУРОЮ

На сьогоднішній день на ринку момент представлена велика кількість систем *PCNC* з різними принципами роботи, які були описані в минулих розділах. На даний момент кожна СЧПУ є авторською та по-своєму унікальною роботою, так і її установка і підключення до верстата. Нижче буде запропонована і описана концепція ЧПУ, що має гнучку модульну архітектуру, яка заснована на певних стандартних модулях, із стандартизованими інтерфейсами підключення один до одного. Подібна уніфікація використовується в світі персональних комп'ютерів, як *x*86, так і *Mac*, але ще не використовується в спеціалізованих комп'ютерах, призначених для управління верстатами. Мета створення даної архітектури – спрощення модернізації СЧПУ для дрібного виробництва, а також для створення власних верстатів з ЧПУ індивідуальними особами. Саме тому за її основу було взято системи СЧПУ, яку можно використовувати для любительських і напівпрофесійних верстатів.

В той же час мета створення даної архітектури не лише в тому, щоб уніфікувати і стандартизувати інтерфейси підключення різних модулів системи один до одного, але і в тому, що б надати користувачам любительських і напівпрофесійних верстатів можливість отримати в своє розпорядження пристрої з якостями і можливостями професійних СЧПУ, що не поступаються своїми характеристиками аналогам професійних СЧПУ провідних світових виробників.

2.1.Загальний опис системи

Запропонований підхід до створення СЧПУ має на увазі створення модульного пристрою з певним набором інтерфейсів для з'єднання його частин між собою. *PCNC*- системи реалізуються на основі одинкомп'ютерної та двохкомп'ютерної архітектури, кожна з яких має свої переваги і недоліки. Виходячи з написаного вище, ми повинні відштовхуватися від СЧПУ одинкомп'ютерної архітектури, що дозволить підключати верстат до звичайного ПК, як звичайний периферійний пристрій. Такі

пристрої часто застосовуються для самостійної збірки верстатів різного призначення в непромислових умовах, тому вони отримали назву *HobbyCNC*. Як вже було зазначено в розділі 1.3, рис. 1.6, описана система складається з драйверів крокових двигунів (далі КД), блоку живлення, інтерфейсної плати, контроллерів портів – *USB/LPT*, а також датчиків зворотного зв'язку. Також окремо потрібно згадати функції, що реалізовуються в рамках *HobbyCNC* за допомогою ПК: це – інтерпретатор УП, інтерполятори, інтерфейс користувача, та мережевий інтерфейс для здійснення обміну даними.

Спершу звернемо увагу на життєвий цикл УП. Життєвий цикл УП, від створення креслення до подачі імпульсів на КД верстата (рис. 2.1).

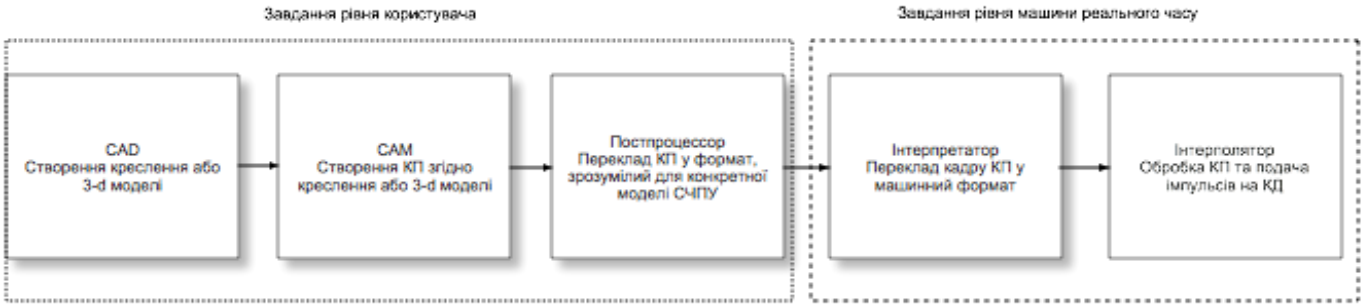


Рис 2.1. Життєвий цикл керуючої програми для СЧПУ

Як можно побачити на рис 2.1, спочатку креслення або модель створюються з використанням відповідних *CAD*-систем, наприклад, *Autocad, Kompas* або систем для 3*d* моделювання – *Autodesk* 3*ds Max, Autodesk Maya* та *Rhinoceros*.

CAM програма створює УП для верстата з ЧПУ на основі креслення для 2-х координатних верстатів, або 3*d* моделі для більш ніж 2-х координатної обробки.

Найпопулярніші на сьогоднішній день *CAM*-програми для професійного використання – *MechSoft Visual Mill* або *ArtCam*. Управляючі програми, які використовуються в пристроях типу *HobbyCNC,* намагаються поєднувати в собі функції *CAM* і власне управляючої програми – наприклад *KCam* або проект *LinuxCNC*. Тобто вони обробляють вхідні дані у форматі векторної графіки і перетворювати їх в КП (рис 2.2.).

Такі програми можуть виконуватися лише на ПК з операційною системою,

призначеною для робочих станцій. Але подібні операційні системи не мають підтримки режиму реального масштабу часу. Це піддає одинкомп'ютерну *PCNC*-системи різним ризикам, що пов'язані з проблемою реального часу, а відсутність рішення цієї проблеми повною мірою є головною перешкодою для досягнення *HobbyCNC*, зокрема і з одинкомп'ютерною архітектурою *PCNC* в цілому якостей, що відповідають професійним *PCNC* з двохкомп'ютерною архітектурою.

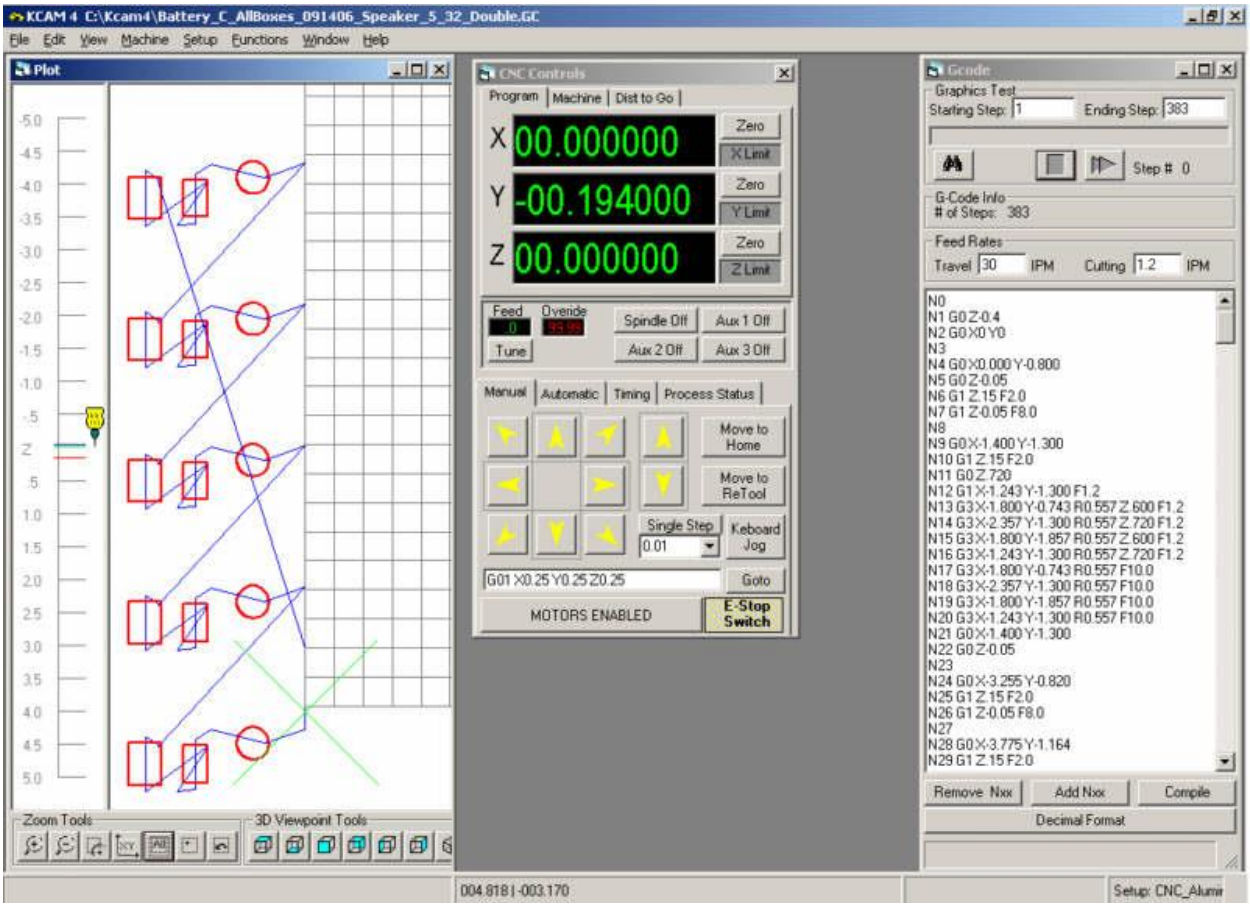


Рис. 2.2. Вікно програми *KCam*

Досвід використання подібних програм розкриває проблеми у використанні операційних систем, призначених для робочих станцій в цілях безпосереднього управління верстатом. Так, окрім головної проблеми реального часу, є ще проблема фонових завдань операційної системи, які можуть проводити неконтрольовану користувачем роботу з портами, що було вже неодноразово відмічено при роботі з операційною системою сімейства *Windows*. Опит портів самою операційною системою може генерувати імпульс, який сприймається системою управління

верстата з ЧПУ як сигнал для руху інструменту, що призводить до погрішностей і явного браку при обробці деталей. Для вирішення проблеми реального часу, зазвичай, використовуються патчи для ядра операційної системи. Але це може привести до нестабільності роботи ОС, що теж загрожує псуванням заготовки ари її обробці, якщо проблеми з ОС відбуваються під час процесу обробки деталі.

Виходячи з цього, можна зробити висновок, що одинкомп'ютерна система *PCNC* з класичному вигляді не може бути оптимальним вибором і потребує винесення функцій низькорівнєвого управління за межі ПК. Тому, інтерполятор пропонується винести за рамки програми, що управляє верстатом, в окремий пристрій, який отримує сигнали від інтерпретатора, та сигнали зворотного зв'язку з датчиків верстата. Як буде описано нижче, алгоритми інтерполяцій і інтерпретаторів являються універсальними, як і інтерфейсні плати. З цього виходить оптимальне рішення – скомпонувати інтерфейсної плати, інтерполятори і частково, інтерпретатори в один керуючий пристрій.

Можна визнати доцільним взаємодію пристрою з комп'ютером через роз'єм *Ethernet*, що може забезпечити як управління одним пристроєм через кроссоверне з'єднання, так і групою пристроїв за допомоглю комутатора. Також цей пристрій має містити окремий роз'єм для підключення плати з розширенням для драйверів КД. Така проміжна плата, яка виконує роль виключно провідника сигналу, має сенс для регулювання кількості драйверів КД та здешевлення всієї конструкції.

Пристрій повинен мати набір *API* для «спілкування» з комп'ютером. Розроблений в рамках дипломної роботи прототип має *API* функції, шо вказані в табл. 2.1. Функції, які задають переміщення для інтерполяторів, є ідентичними командам мови *g*-кодов, та приймають відносні координати точки як параметр управління. Робота з абсолютними координатами вимагає додаткові змінні для їх зберігання і цим ускладнює конструкцію. Процес перетворення їх з абсолютних координат до необхідних відносних пропонується проводити з допомогою ПЗП для ПК. Функції та передавальні команди пристрою для наглядності мають префікс *r*. Функції, що є відповідями від пристрою, мають префікс *i*.

Для того, щоб розширити сфери застосування, доцільно вбудувати в пристрій перемикач апаратного відключення інтерполяторів. Він необхідний на той випадок, якщо користувач захоче використовувати пристрій в якості звичайного контроллеру КД. Це необхідно для забезпечення сумісності з тим, що вже існує ПЗП для управління ЧПУ (табл.2.1).

Таблиця 2.1.

API функції СЧПУ для спілкування з комп'ютером

Функція	**Опис**
G02 (<*endPointX, endPointY, endPointZ, radius*>)	Функція кругової інтерполяції за годинниковою стрілкою. Приймає зсуви по координатах *x, y, z* відносно поточної точки, а також, координати центру окружності
G03 (<*endPointX, endPointY, endPointZ, I,J*>)	Функція кругової інтерполяції проти годинникової стрілки. Приймає зсуви по координатах *x, y, z* відносно поточної точки, а також, координати центру окружності
EStop()	Функція – сигнал аварійного зупину
FrameDone()	Сигнал про виконання кадру
Limit()	Сигнал про досягнення кінцевиків

Пристрій для реалізаціх цих функцій зібраний на платформі для прототипування *Arduinouno*. Необхідно зазначити, що *Arduino* сам по собі складаєтся з модулів, що стикуются між собою без застосування пайки. Можна визнати доцільним модульність *Arduino* та взяти її за основу модулів пристрою ЧПУ.

Так, в якості головного керуючого модуля буде плата *ArduinoUNO*. Опція підтримки мережі доступна через модуль *Ethernetshield*.

ArduinoUno – це контролер, що побудований на базі мікроконтролера *ATmega*328. Платформа має 14 цифрових вход/виходів, 6 з яких можуть використовуватися як виходи ШИМ, 6 аналогових входів, кварцевий генератор 16 Мгц, роз'єм *USB*, силовий роз'єм, роз'єм *ICSP* та кнопку перезавантаження. Для роботи необхідно підключити платформу до комп'ютера за допомогою *USB* кабелю, або подати живлення за допомогою адаптера *AC/DC* або електричної батареї. Характеристики плати *ArduinoUNO* (табл. 2.2).

Таблиця 2.2.

Характеристики *ArduinoUNO*

EEPROM	1 Кб
Аналоговівходи	6
Вхіднанапруга (рекомендована)	7-12 В
Вхіднанапруга (гранична)	6-20 В
ОЗП	2 кб
Постійний струм для выводу 3.3 В	50мА
Постійний струм через вхід/вихід	40мА
Робочанапруга	5 В
Тактовачастота	16 Гц
Флеш-пам'ять	32 Кб *ATmega328*, з яких 0.5 Кб використовуються для завантажувача
ЦифровіВходи/Виходи	14, 6 з якихможутьвикористовуватися як виходи ШІМ

Для підтримки мережевого інтерфейсу і карт стандарта *microsd,* використовується *arduinoethernetshield*, який дозволяє підключити плату *Arduino* до мережі. Вона заснована на *Ethernet*-микросхемі *WiznetW*5100. *WiznetW*5100 підтримує стеки *TCP* та *UDP* в *IP*-мережі, підтримує до чотирьох одночасних підключень до сокетів. Для створення скетчів, які підключаються до мережі за допомогою цієї плати, використовують бібліотеку *Ethernet*. *WiznetW*5100 з'єднується з платою *Arduino* за

допомогою довгих штирів, що проходять через неї – це дозволяє не змінювати розташування виводів та встановлювати інші плати поверх цієї. Плата *EthernetShield* має стандартний роз'єм *RJ*-45 зі вбудованим лінійний трансформатором та опцією *PoweroverEthernet*. Останні версії цієї плати мають роз'єм для карт типу *micro-SD*, який може використовуватися для зберігання файлів і роботи з ними по комп'ютерній мережі. Вона є сумісною з *ArduinoUno* та *Mega* при використанні бібліотеки *Ethernetlibrary*. Кардрідер для карт *micro-SD* доступний за допомогою бібліотеки *SD*Library, та при використанні цієї бібліотеки вивід 4 використовується для сигналу *SS* (*SlaveSelect*).

Останні версії цієї плати також мають контроллер скидання, яке дозволяє бути упевненим в правильному перезапуску *W*5100 при її включенні, оскільки попередні версії плати були не сумісні з *ArduinoMega* та вимагали ручного скидання після включення. Передача команд з ПК до ПЧПУ здійснюється за протоколом *HTTP*. Загальна принципова схема модулів керуючого пристрою та мережевого з'єднання представлена в Додатках Б та В.

2.2. Сучасні аналоги пристрою на ринку ЧПУ *PCNC*- типу

На сьогоднішній день на ринку *PCNC* представлені різноманітні пропріетарні архітектури від різних виробників, та різні типи контролерів, що дозволяють підключати верстат до ПК в ролі звичайного периферійного пристрою.

Пристрої типу *HobbyCNC* представлені контролерами крокових двигунів з *LPT*, *COM*, рідше з *USB* інтерфейсом, але останнім часом з'являються пристрої, які оснащені *Ethernet* інтерфейсом.

Такі пристрої складаються з інтерфейсної плати і драйвера для управління КД. Класична схема таких пристроїв – одна інтерфейсна плата і декілька драйверів. На сьогоднішній день прийнято об'єднувати ці два пристрої в один контроллер ЧПУ, який реалізується на програмованому мікроконтроллері. Пристрій, який запропонований в дипломній роботі, порушує цю тенденцію на користь гнучкості, а також внаслідок запропонованого шляху вирішення проблеми реального часу, саме

тому ми розглянемо класичні варіанти інтерфейсної плати і простого драйвера КД. Типова схема такої плати (рис 2.3).

Для управління КД використовуються 3 шіна *LPT* порту, на які подаються наступні сигнали:

- *Dir* – напрямок обертання;
- *Step* – крок, або швидкість;
- *Enable* – дозвіл роботи;
- *Home* (або *Limit*) – досягнення кінцевика.

Сигнал *Enable* в таких платах є загальним на всіх, тобто під час роботи верстата всі його крокові двигуни знаходяться в стані збудження.

Останнім часом набирають популярність інтерфейсні плати зі вбудованими в них контроллерами *USB*, які перетворюють сигнал з *USB* у вже описані стандартні сигнали, призначені для КД. Оскільки опис *USB* контроллера виходить за межі даної роботи, детально розглядати його ми не будемо. До інтерфейсної плати можна підключати будь-які драйвери крокових двигунів, що робить її універсальним компонентом подібних верстатів з ЧПУ.

На даний момент ШІМ-драйвери крокових двигунів э найбільш популярними, практично всі драйвери на ринкуналежать до цього типа. Ці драйвери подають на обмотку крокового мотора ШІМ-сигнал, що маэ дуже високу напругу, яка відсікається по досягненню струмом необхідного рівня. Величина сили струму, по якій відбувається це відсічення, задається або потенціометром, або *DIP*-перемикачем. Іноді ця величина програмується за допомогою спеціального ПЗП. Такі драйвери є досить інтелектуальними, та забезпечені великою кількістю додаткових функцій. Вони млжуть підтримувати різні ділення кроку, що дозволяє збільшити дискретність позиціювання та плавність ходу. Але ШІМ-драйверы також вельми сильно відрізняються один від одного своїми характеристиками. Окрім таких характеристик, як живляча напруга і максимальний струм обмотки, у них відрізняється частота ШІМ. Оптимальною є частота драйвера буде більше 20 кГц, та

взагалі, чим вона більша, тим краще. Частота нижче, ніж 20 кГц погіршує ходові характеристики двигунів і потрапляє в чутний діапазон людського вуха – крокові мотори починають видавати неприємний писк.

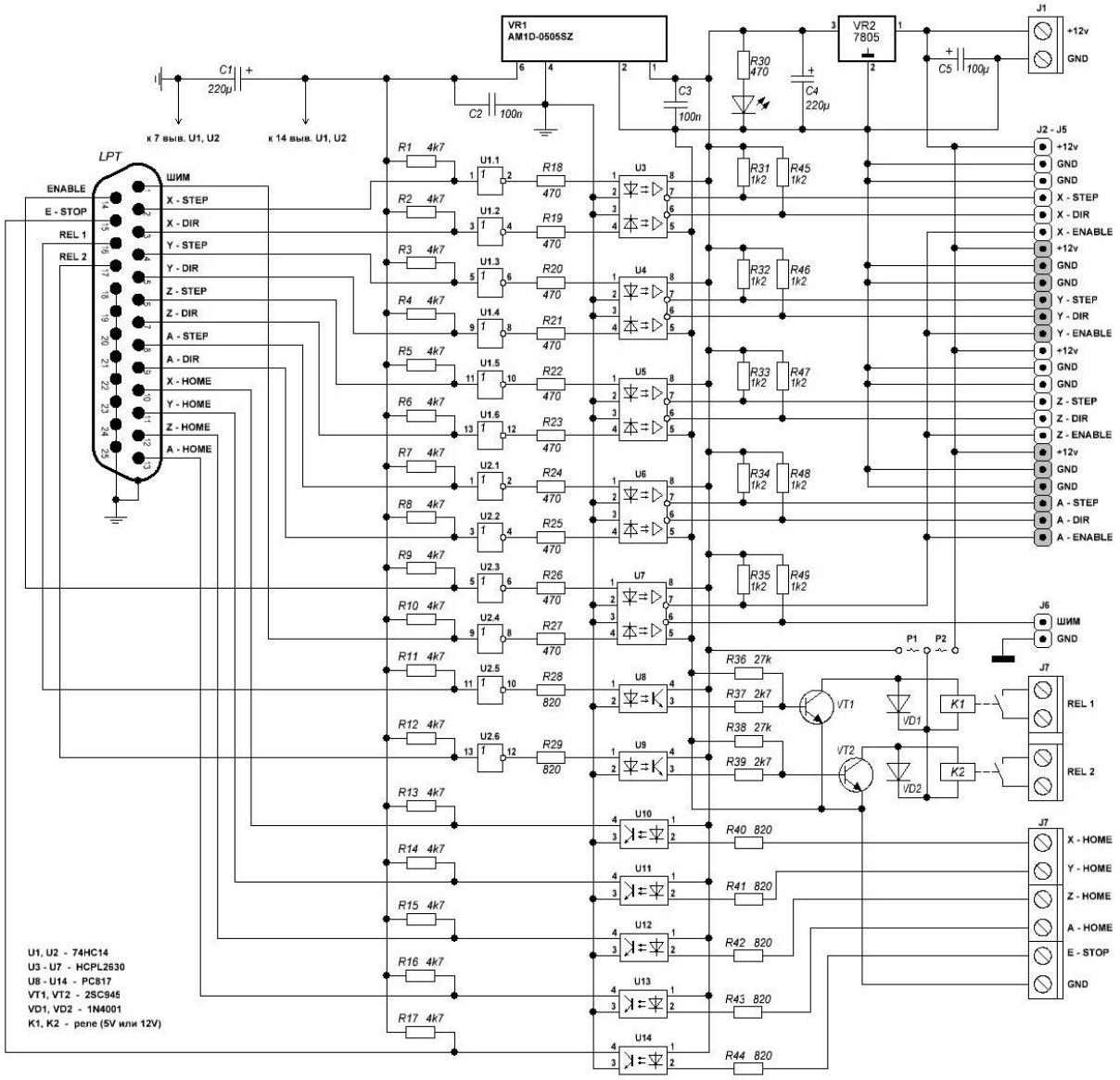


Рис 2.3. Схема типової інтерфейсної плати для систем *Hobby CNC*

Драйвери крокових двигунів, як і самі двигуни, діляться на уніполярні та біполярні.

Початківцям в галузі верстатобудуванням рекомендується не експериментувати з приводами, а обрати ті, по яких можна отримати максимальний об’єм технічної підтримки, необхідної інформації, та продукти для яких представлені найширше на ринку. Такими є драйвери біполярних гібридних крокових двигунів. В

рамках запропонованого пристрою використовуються окремі від інтерфейсної плати драйвера КД. Детальніше класичний драйвер КД буде розглянуто в розділі 2.3.4. як модуль пристрою.

2.3. Опис модулів системи

Запропонована система складається з центрального керуючого блоку, який є об'єднаний з блоком мережевого з'єднання, драйверами КД, блоком живлення, та має можливість підключення периферії , такої як дисплей, розширення пам'яті, модудь *WIFI* та ін. В описі модулів системи ми обмежимось описом модулів центрального блоку, які вирішують завдання управління ЧПУ в рамках запропонованої реалізації та драйверів КД. Усі алгоритми частин керуючого модуля ЧПУ реалізовані програмно. Код викладений у додатку Г.

2.3.1. Системи координат у верстатах

Інтерполятори можуть працювати в різних системах координат, тому, доцільним є розглядати системи координат стосовно верстата з ЧПУ та інтерполятора.

1. Кожен верстат з ЧПУ має свою систему координат, осі якої розташовані паралельно направляючим верстата. Це дозволяє в процесі обробки деталі вказувати напрям та величину переміщення заготовки, або інструменту в робочому просторі верстата. Вісь *Z* виходить з елементів верстата, що обертаються – шпинделя, фрези, або свердла. Вісь *X* – перпендикулярна до осі *Z*, паралельна площині установки заготовки, та характеризує більше, або єдино можливе переміщення робочого органу верстата. Вісь *Y* перпендикулярна до осей *Z* та *X*. Позитивними напрямами осей є такі, при яких інструмент і заготовка деталі віддаляються одне від одного. Така система називається стандартною. Початок стандартної системи координат верстата – точка *M*, зазвичай поєднують з базовою точкою вузла верстата, що несе заготовку деталі.

2. Кожна деталь має свою систему координат, в якій задані конструктором її розміри. В цій системі координат задається траєкторія руху інструменту, вказується

точка початку відліку руху інструменту в процесі обробки деталі – це так звана вихідна точка руху інструменту.

3. Система координат інструменту задає його положення відносно державки, а отже і самого верстата. Ця система координат дозволяє пов’язати координати опорних точок деталі і координати вихідної точки руху інструменту з базовою точкою верстата.

З точки зору алгоритмів інтерполяторів, актуальними є дві останні системи координат, тому ми будемо вважати, що система координат верстата і система координат деталі збігаються.

2.3.2. Опис інтерполяторів

Програма обробки деталі, що власне і здійснює управління, визначає траєкторію руху інструменту. Траєкторія руху складається з окремих, поєднаних одна з одною ділянок, лінійних або дугових. Точки, які задають траєкторію, мають назву опорних. Таким чином, можна сказати, що управляюча програма – це послідовний набір опорних точок. Опорні точки можуть лежати в одній площині, і для їх завдання використовується дві координати – двохкоординатна обробка. або в просторі – об’ємна трикоординатна обробка. На практиці для переміщення інструменту системі ЧПУ не достатньо завдання лише опорних точок, необхідно більш детальне її виставлення. Для розрахунку проміжних точок руху і видачі команд руху по лінійних осях використовується спеціальний обчислювальний пристрій – інтерполятор.

Інтерполятором називається модуль, що формує траєкторію руху інструменту по заданому закону із заданими швидкістю і точністю між двома опорними точками контура деталі, координати яких вказані в керуючий програмі. Досить часто під інтерполятором мають на увазі міні-ЕОМ, що працює за певною програмою.

У сучасних СЧПУ, зазвичай, застосовується два види інтерполяторів – лінійні і кругові, оскільки до 90 % траєкторій можуть бути представлені з достатньою мірою точності сукупністю відрізків прямих та дуг кола. Лінійні інтерполятори забезпечують формування траєкторії у вигляді прямої лінії. Кругові інтерполятори формують траєкторію руху у вигляді кола, або її частини – дуги кола.

Існують різні алгоритми інтерполяції реального масштабу часу, які умовно можна розділити на дві групи:

1. Алгоритми одиничних приростів – методи оціночнної функції, метод цифро-дифференциальных аналізаторів.

2. Алгоритми рівних часів – методи цифрової інтеграції, прогнозу і корекції, ітераційно-табличні методи.

Наш пристрій використовує інтерполятори, які працюють по методу оцінної функції. На вхід інтерполятора поступає інформація про координати опорних точок, на його виході для кожної координати формується послідовність імпульсів, необхідних для відпрацювання заданої геометрії. Лінійний інтерполятор дозволяє відпрацьовувати лише прямолінійні рухи, але забезпечити точну відповідність переміщення вздовж заданої прямої досить складно.

Підсумкова траєкторія переміщення інструменту приблизно нагадує ламану лінію, як показано на рис 2.4. В процесі роботи інтерполятор позмінно управляє включенням приводів то по осі X, то по осі Y , а так само і Z-приводу при 3-х координатній обробці, посилаючи потрібну кількість імпульсів на відповідні приводи.

Розглянемо детальніше алгоритм роботи лінійного інтерполятора. Припустимо, що задане переміщення інструменту між опорними точками A_0 та A_k в площині XY. Кожна точка площини характеризується коефіцієнтом

$$K = \frac{Y_i}{X_j},$$

де X_k та Y_k – координати кінцевої опорної точки заданої прямої.

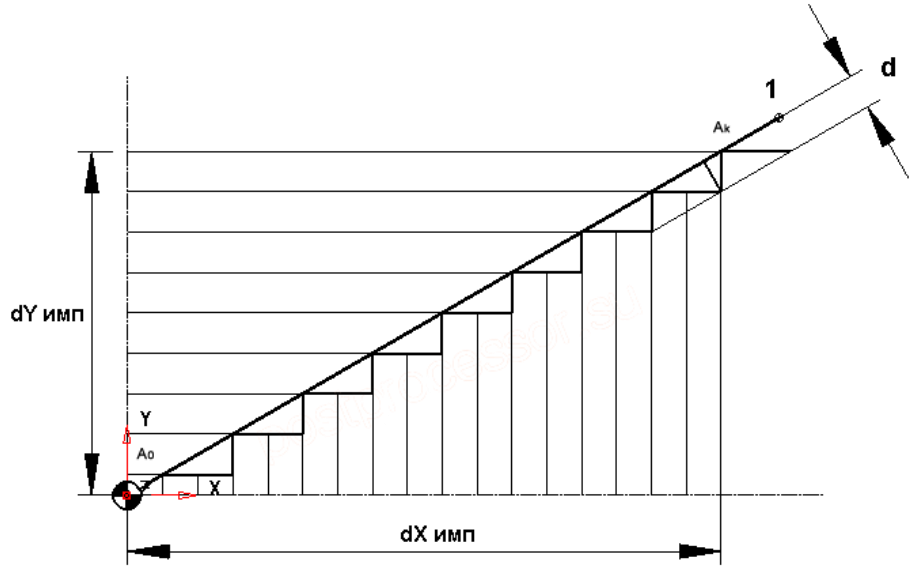


Рис 2.4. Траєкторія переміщення інструменту при лінійній інтерполяції

Залежно від знаку різниці коефіцієнтів $H_{i,j}$ площина XY ділиться на три області.

$$H_{i,j} = \frac{Y_i}{X_j} - \frac{Y_k}{X_k}$$

Перша область – над прямою $A_0\ A_k$, де $H>0$.

Друга область – під прямою $A_0\ A_k$, де $H<0$.

Третя область – на самій прямій $A_0\ A_k$, где $H=0$.

Кожен інтерполятор має свої алгоритми роботи. Будемо вважати, що даний лінійний інтерполятор працює по наступному алгоритму.

1. Якщо $H\geq 0$, то інтерполятор виробляє і посилає на подаючий привод один електричний імпульс для переміщення інструменту на одну дискрету по осі X.

2. Якщо $H<0$, то інтерполятор виробляє і посилає на подаючий привод один електричний імпульс для переміщення інструменту на одну дискрету по осі Y.

3. Після виконання кожного кроку знов і знов здійснюється розрахунок нового значення оцінної функції.

Оскільки інструмент обробки, в данному випадку ріжучий інструмент, в переміщується по двох координатах, то і СЧПУ повинне мати два подаючі приводи. Спростимо вираз 2.1., приведши його до спільного знаменника і використаємо лише

його чисельник, як носій знаку. Отримаємо наступне вираження оцінній функції вигляду

$$F_{j,i} = Y_i * X_k - Y_k * X_j \quad (2.2)$$

Проведемо спрощення виразу (2.2) в припущенні, що інтерполятор має можливість запам'ятовувати по якій координаті був зроблений попередній крок.

1. Припустимо, що попередній крок інструменту був зроблений по осі *X*. Тоді поточна координата інструменту буде дорівнювати попередній координаті плюс одна дискрета $X_{j+1} = X_j + 1$. Підставимо цей вираз в (2.2):

$$F_{j+1,i} = Y_i * X_k - Y_k * (X_j + 1) = Y_i * X_k - Y_k * X_j - Y_k = F_{j,i} - Y_k$$

Отже, після виконання чергового кроку по осі *X* нове значення оцінної функції буде розраховуватись, як різниця між попереднім значенням оцінної функції і координатою кінцевої опорної точки по осі *У*.

2. Припустимо, що попередній крок був зроблений по осі *У*. Тоді поточна координата інструменту буде дорівнювати попередній координаті плюс одна дискрета $Y_{i+1} = Y_i + 1$. Підставимо цей вираз в (2.2):

$$F_{j,i+1} = (Y_i + 1) * X_k - Y_k * X_j + 1 = Y_i * X_k - Y_k * X_j + X_k = F_{j,i} + X_k$$

Отже, після чергового кроку по осі нове значення оцінної функції буде розраховуватись, як сума попереднього значення оцінної функції і координати кінцевої опорної точки по осі *X*.

Алгоритм роботи лінійного інтерполятора, що був описан графічно (рис.2.5).

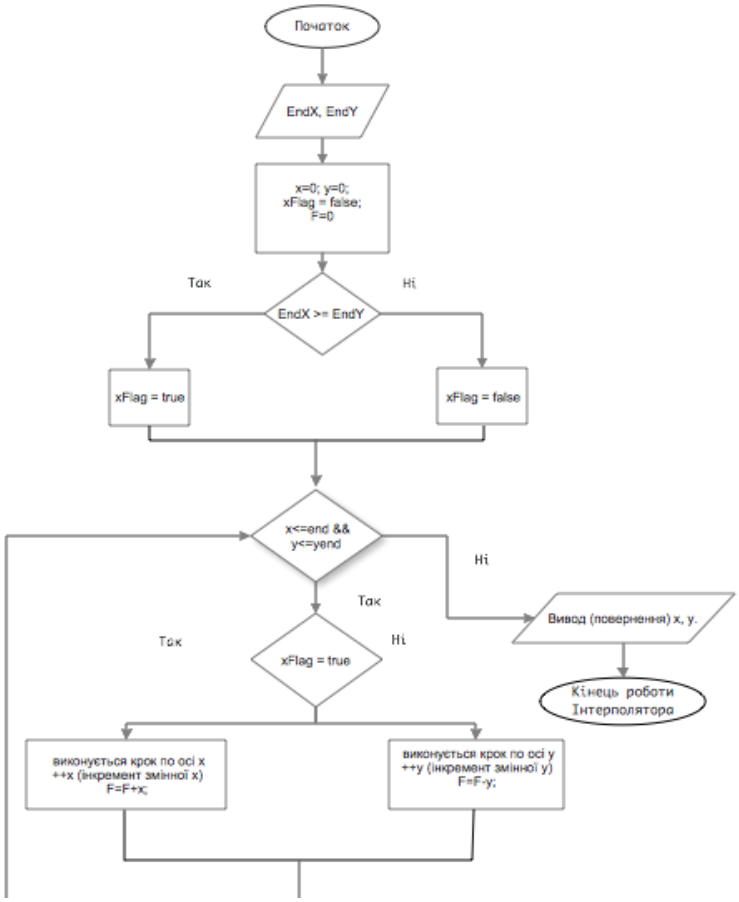


Рис. 2.5. Блок схема алгоритму лінійного інтерполятора

Круговий інтерполятор має чотири режими роботи – по кількості квадрантів системи координат. Режими роботи в тому або іншому квадранті визначаються знаками при кінцевих значеннях координат X_k, Y_k. Але при розрахунках оцінних функцій значення кінцевих координат беруть в своїх абсолютних значеннях, при цьому завжди із знаком +. Напрям руху інструменту вздовж осей координат визначається знаками + або -, які прсвоюються сигналу на виході інтерполятора.

Розглянемо роботу кругового інтерполятора. Оцінна функція кругового інтерполятора має наступний вигляд:

$$F_{j,i} = R_{j,i}^2 - R_b^2 \tag{2.3}$$

де $R_{j,i}^2 = X_j^2 + Y_j^2$ – квадрат відстані від центру системи координат XY, поєднаної з центром описуваного кола, до поточної точки ступінчастої, або дійсної траєкторії руху інструменту;

X_j, Y_i – координати поточної точки ступінчастої траєкторії руху інструменту; R_n^2 – квадрат радіусу заданої дуги кола (рис.2.6.). В залежності від знаку оцінної функції площина *XY* може бути розбита на три області.

Перша область – поза дугою, де $F>0$.

Друга область – під дугою, де $F<0$.

Третя область – на дузі, де $F=0$.

Для кругового інтерполятора застосуємо алгоритм роботи, який є аналогічним алгоритму роботи лінійного інтерполятора.

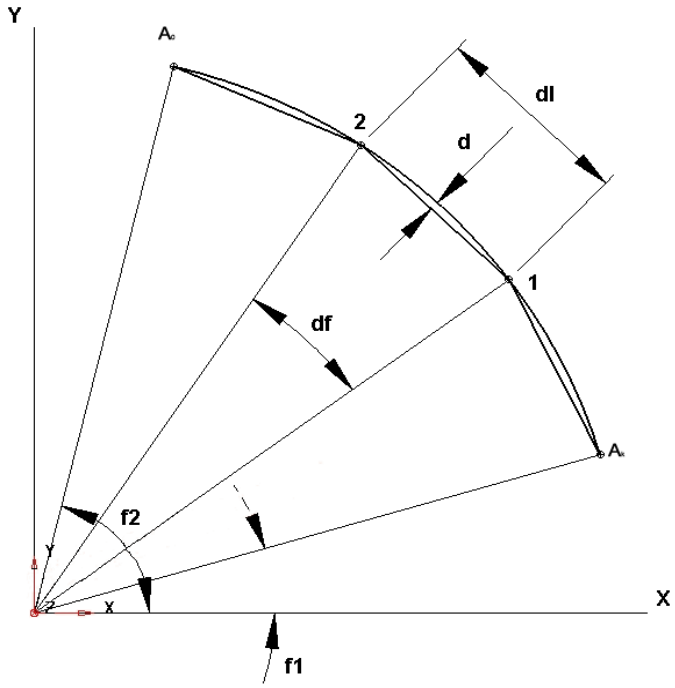


Рис. 2.6. Траєкторія переміщення інструменту при круговій інтерполяції

Круговий інтерполятор має 8 режимів роботи: чотири квадранти, і в кожному квадранті інструмент обробки може рухатись як за годинниковою стрілкою, так і проти неї.

В якості прикладу розглянемо один режим роботи, який полягає в тому, що перший квадрант з рухом інструменту проти годинникової стрілки з однієї точки в іншу (див. рис 2.6).

Якщо припустити, що круговий інтерполятор має можливість запам’ятовувати, по якій координаті був зроблений попередній крок, то вихідну оцінну функцію можна

спростити, та представити у вигляді двох функцій, як це було у випадку лінійної інтерполяції.

1. Припустимо, що попередній крок був зроблений по осі *X*. Тоді координата поточної точки траєкторії руху інструменту для даного режиму дорівнюватиме координаті попередньої точки мінус одна дискрета $X_{j+1} = X_j - 1$. Оскільки з кожним кроком координата *X* буде зменшуватися на одну дискрету, та врешті решт вона повинна стати рівною нулю. Підставимо даний вираз в рівняння.

$$F_{j+1,i} = X_{j+1}^2 + Y_i^2 - R_n^2 = \left(X_j - 1\right)^2 Y_i^2 - R_n^2 = X_j^2 + Y_i^2 - R_n^2 - 2X_j + 1 =$$
$$== F_{j,i} - 2X_j + 1 \qquad (2.3)$$

Отже, після чергового кроку по осі *X* в режимі першого квадранта при русі, що здійснюється проти годинникової стрілки. Нове значення оцінної функції розраховується як значення оцінної функції до кроку, мінус подвоєне значення поточної координати по осі *X* і плюс одна дискрета.

2. Припустимо, що попередній крок був зроблений по осі *У*. Тоді координата поточної точки траєкторії руху інструменту буде дорівнювати координаті попередньої точки плюс одна дискрета.

$$Y_{i+1} = Y_i + 1$$

Оскільки з кожним кроком координата *B* буде збільшуватись на одну дискрету, та врешті решт повинна стати рівною радіусу дуги. Підставимо це вираження в рівняння (2.3).

Таблиця 2.3.

Режими роботи кругового інтерполятора

		Крок по осі x	Крок по осі y
Перший квадрант	Проти часової стрілки	$X_{j+1} = X_j - 1$ $F_{j+1,i} = F_{j,i} - 2 * X_j + 1$	$Y_{i+1} = Y_i + 1$ $F_{j,i+1} = F_{j,i} + 2 * Y_i + 1$
Перший квадрант	По часовій стрілці	$X_{j+1} = X_j + 1$ $F_{j+1,i} = F_{j,i} + 2 * X_j + 1$	$Y_{i+1} = Y_i - 1$ $F_{j,i+1} = F_{j,i} - 2 * Y_i + 1$
Другий квадрант	Проти часової стрілки	$X_{j+1} = X_j + 1$ $F_{j+1,i} = F_{j,i} + 2 * X_j + 1$	$Y_{i+1} = Y_i - 1$ $F_{j,i+1} = F_{j,i} - 2 * Y_i + 1$
Другий квадрант	По часовій стрілці	$X_{j+1} = X_j - 1$ $F_{j+1,i} = F_{j,i} - 2 * X_j + 1$	$Y_{i+1} = Y_i + 1$ $F_{j,i+1} = F_{j,i} + 2 * Y_i + 1$
Третій квадрант	Проти часової стрілки	$X_{j+1} = X_j - 1$ $F_{j+1,i} = F_{j,i} - 2 * X_j + 1$	$Y_{i+1} = Y_i + 1$ $F_{j,i+1} = F_{j,i} + 2 * Y_i + 1$
Третій квадрант	По часовій стрілці	$X_{j+1} = X_j + 1$ $F_{j+1,i} = F_{j,i} + 2 * X_j + 1$	$Y_{i+1} = Y_i - 1$ $F_{j,i+1} = F_{j,i} - 2 * Y_i + 1$

Четвертий квадрант	Проти часової стрілки	$X_{j+1} = X_j + 1$ $F_{j+1,i} = F_{j,i} + 2 * X_j + 1$	$Y_{i+1} = Y_i - 1$ $F_{j,i+1} = F_{j,i} - 2 * Y_i + 1$
Четвертий квадрант	По часовій стрілці	$X_{j+1} = X_j - 1$ $F_{j+1,i} = F_{j,i} - 2 * X_j + 1$	$Y_{i+1} = Y_i + 1$ $F_{j,i+1} = F_{j,i} + 2 * Y_i + 1$

$$F_{j+1,i} = X_j^2 + Y_{i+1}^2 - R_n^2 = \left(Y_j + 1\right) \ + X_j^2 - R_n^2 = X_j^2 + Y_i^2 - R_n^2 + 2Y_j + 1 =$$
$$= F_{j,i} - 2Y_i + 1$$

Отже, після чергового кроку по осі *У* в режимі першого квадранта, при здійсненні руху проти годинникової стрілки, нове значення оцінної функції розраховується, як значення оцінної функції до кроку, плюс подвоєне значення поточної координати по осі *У* і плюс одна дискрета.

В таблиці 2.2. представлені всі оцінні функції для всіх восьми режимів роботи кругового інтерполятора. На їх основі ми отримали алгоритми для кругового інтерполятора , що здійснює рух за годинниковою стрілкою *G02* (рис 2.7.), та проти годинникової стрілки *G02* (рис 2.8.).

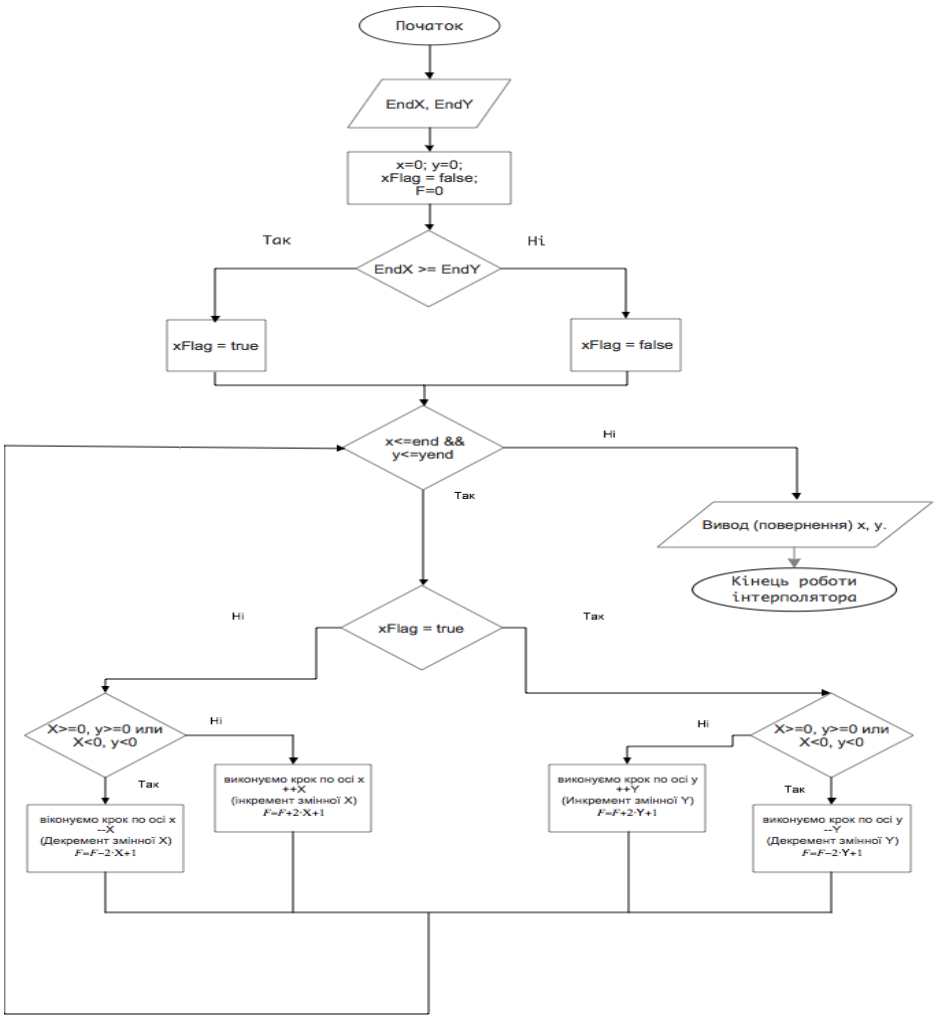


Рис. 2.7. Алгоритм кругового інтерполятора, що виконує рух за годинниковою стрілкою

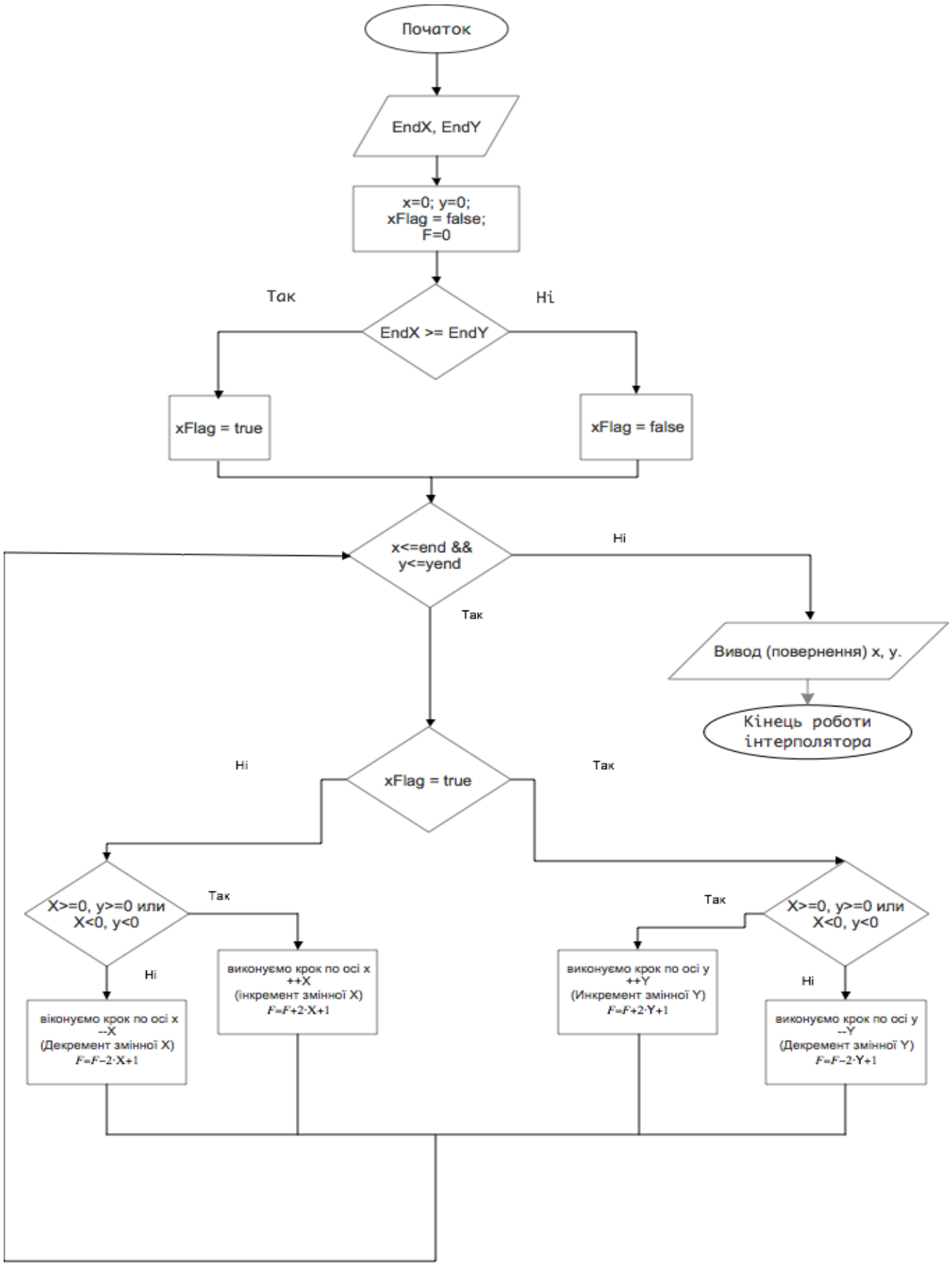


Рис. 2.8. Алгоритм кругового інтерполятора, що виконує рух проти годинникової стрілки

2.3.3. Аналіз нтерпретаторів

Інтерпретатор транслює кадри програми, що здійснює управління, представляючи їх у форматі, який є зручним для роботи інтерполятора. У фазі інтерпретації кадру система ЧПУ верстата виконує еквідистантні розрахунки і розрахунки, що пов'язані із стиковкою еквідистантних контурів; здійснює перетворення координатних систем – абсолютна чи відносна, перетворення систем виміру – мм, чи дюйми; викликає стандартні цикли та підпрограми; розділяє потоки даних геометрично, логічно та ін., (табл. 2.4.)

Крок 1. Здійснюється переклад кадру програми, що здійснює управління, у внутрішній формат. Відбувається сортування інформації за типом параметрів: *G*-функції, адреси, переміщення, коментарі.

Таблиця 2.4.

Приклад розбору кадру на першому кроці інтерпретації.

Приклад кадру	*N*20	*G*91	*G*01	*X*20,5	*Y*37,5	*F*2500	**Comment*
G-функції		*G*91	*G*01				
Переміщення				*X*20,5	*Y*37,5		
Адреси	*N*20					*F*2500	
Коментарі							**Comment*

Крок 2. Формування активного *G*-вектора, число координат якого відповідає числу груп *G*-функций. Кожна нова *G*-команда включається в ту координату *G*-вектора, яка відповідає номеру групи *G*-функциії і існуватиме, поки її не замінить інша *G*-функція з тієї ж групи.

Крок 3. Інтерпретація кадру груповими інтерпретаторами, відповідними групі *G*-функцій.

Крок 4. Призначення геометричних переміщень.

Крок 5. Еквідистантна корекція з врахуванням розмірів інструменту.

Крок 6. Стиковка сусідніх еквідистантних контурів.

Крок 7. Передача даних в інтерполятор.

Схема класичного інтерпретатора (рис. 2.8).

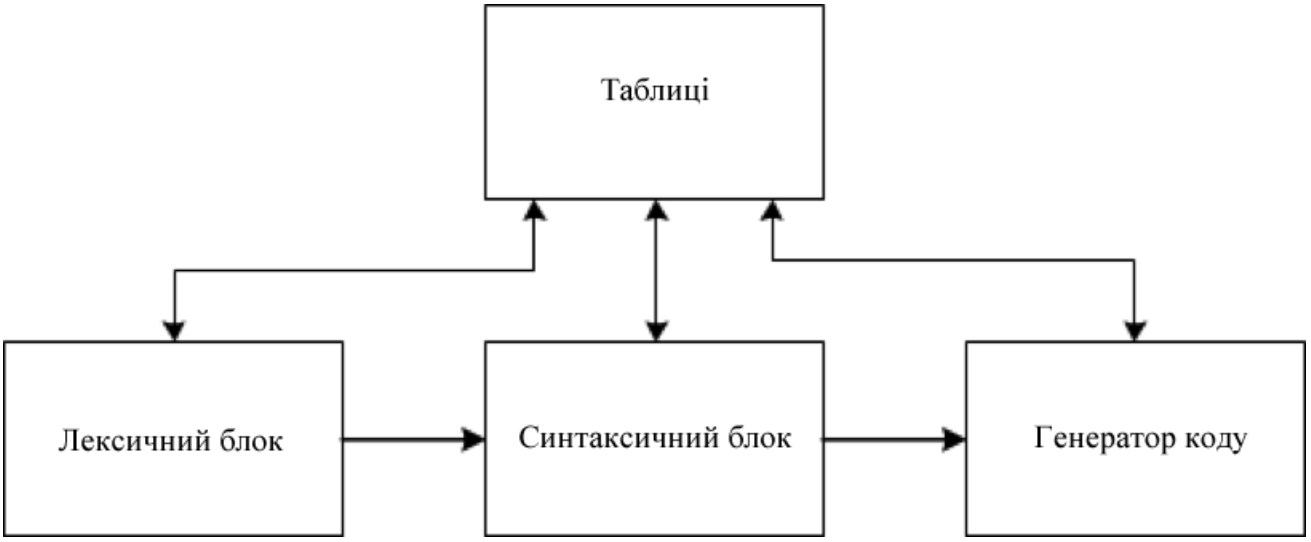


Рис. 2.8. Схема класичного інтерпретатора

Призначення блоків інтерпретатора:

– Таблиці. Цей блок зберігає набір лексем, та статичну і динамічну інформацію, пов’язану з ними. Таблиця символів транслятора має забезпечувати доступ до своїх елементів за мінімальний час, хэш-таблица забезпечує доступ до будь-якого елементу за однаковий час, тобто від кількості елементів цей час не збільшується, як, наприклад, це відбувається у випадку з бінарним деревом.

– Лексичний блок. Блок здійснює перевірку на відповідність УП алфавіту мови, виділяє лексеми і виконує їх пошук в таблиці символів.

– Синтаксичний аналізатор. Цей блок здійснює перевод послідовності лексем, шо побудована лексичним блоком, в іншу послідовність, яка безпосередньо відображає порядок, в якому повинні виконуватись операції в програмі. Мова УП є досить простою, тому не вимагає синтаксичного аналізу.

– Генератор кодів. Формує команди, що здійснюють управління, для модуля управління рухом.

Вхідними даними інтерпретатора є програма, що здійснює управління (УП). УП

складається з кадрів, які в свою чергу складаються зі слів. Слово складається з символу адреси, що являть собою латинські прописні букви, за необхідністю математичного знаку «+» або «-», та послідовності цифр. Мова не чутлива до регістра символів і не містить символів-роздільників, які відділяють один ідентифікатор від іншого, окрім кінця кадру, що містить переклад рядка. В описаному пристрої інтерпретатор приймає *G*-код, вибирає потрібний інтерполятор, передає в нього координати кінцевої точки і чекає його виконання (рис 2.8.). Програма *G*-кодів передається в інтерпретатор по мережі і потім зберігається на модулі *SD*-карти пам'яті. Деякі СЧПУ не зберігають всі координати в пам'яті, а інтерпретують кадри *G*-кодів поодинці.

Але така система має один суттєвий недолік – при передачі даних по комп'ютерній мережі може виникнути затримка через відносно низької швидкості передачі даних через мережеві інтерфейси. А це може спричинити те, що під час затримки при передачі ріжучий інструмент верстата буде продовжувати працювати, але він буде знаходиться при цьому в одному місці. Якщо, наприклад, верстат електроіскровий, то електрична дуга встигає пропалити за цей час більшу кількість металу, що приводить до неточностей у вирізаній деталі. Також під час затримок можуть виникати ривки в русі інструменту та інші проблеми. Тому з метою запобігання таких ситуацій та забезпечення плавності ходу рекомендується об'єднувати інтерпретатор і інтерполятори в один пристрій, або сполучати їх надшвидкими інтерфейсами. У даній дипломній роботі вони апаратно скомпоновані в одному пристрої та програмно виконані на платформі *Arduino*. Алгоритми і програмний код блоків інтерпретатора наведені в Додатку В.

Логічна схема запропонованого інтерпретатора та принципи його взаємодії з іншими модулями системи (рис. 2.9). При роботі в цьому модулі інтерпретатора *G*-код перетворюється в об'єкт, який містить наступні дані: тип інтерполяції, початкова та кінцева точка, та радіус кола, який для лінійного інтерполятора дорівнює нулю. Всі об'єкти поміщаються в чергу, дані якої зберігаються на вбудованому модулі пам'яті

Micro-SD. Далі кожен об’єкт із цієї черги передається в модуль інтерполятора з обов’язковим указанням типу інтерполятора, який необхідний для його виконання. Коли з інтерполятора надходить сигнал про завершення обробки кадру, то далі відбувається передача наступного кадру.

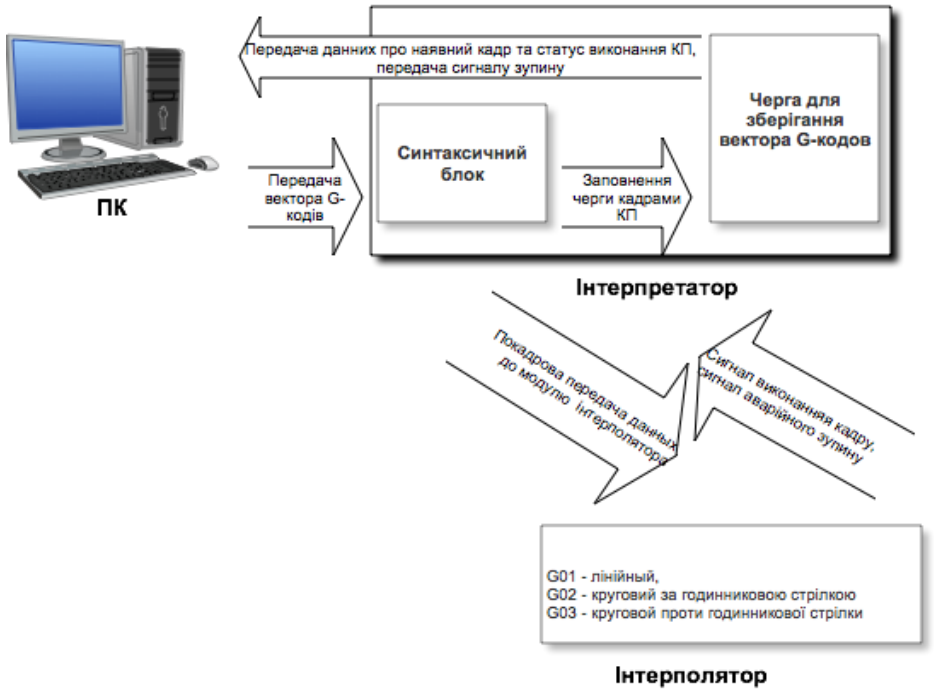


Рис. 2.9. Схема обміну даними між інтерпретатором і іншими модулями системи

2.3.4. Драйвер крокових двигунів

Драйвер крокового двигуна вирішує дві основні задачі – це забезпечення необхідного струму в обмотках двигуна для його роботи та формування необхідних часових послідовностей сигналів. В інтегральних реалізаціях цього пристрою ці завдання можуть виконуватись різними мікросхемами. Як приклад, можно привести комплект мікросхем *L*297 та *L*298, які були розроблені фірмою *SGS-Thomson*. Так. мікросхема *L*297 містить логіку формування часових послідовностей, а мікросхема *L*298 представляє собою потужний здвоєний *H*-міст. На жаль, існує деяка плутанина в термінології відносно подібних мікросхем – часто поняття «драйвер» застосовують до багатьох мікросхем, навіть якщо їх функції сильно розрізняються. Іноді мікросхеми

логіки називають «трансляторами». В дипломній роботі далі використовуватиметься наступна термінологія: «контроллер» – це мікросхема, яка відповідає за формування часових послідовностей; а «драйвер» – потужна схема живлення обмоток двигуна. Але іноді терміни «драйвер» і «контроллер» можуть також означати пристрій управління кроковим двигуном. Слід зазначити, що останнім часом все частіше контроллер і драйвер об’єднують в одній мікросхемі.

На практиці існує можливість обійтись і без спеціалізованих мікросхем – наприклад, всі функції контроллера можуть бути реалізовані програмно, а в якості драйвера застосувується набір дискретних транзисторів. При цьому підході мікроконтроллер буде сильно завантажений, а схема драйвера може вийти досить громіздкою, але в деяких випадках таке рішення буде економічно обґрунтованим та вигідним.

Для управління біполярними двигунами потрібні більш складні схеми – *H*- мости. Такі схеми можуть бути реалізовані на дискретних елементах, хоча останнім часом все частіше вони реалізуються на базі інтегральних схем. Приклад такої реалізації на дискретних (рис. 2.10).

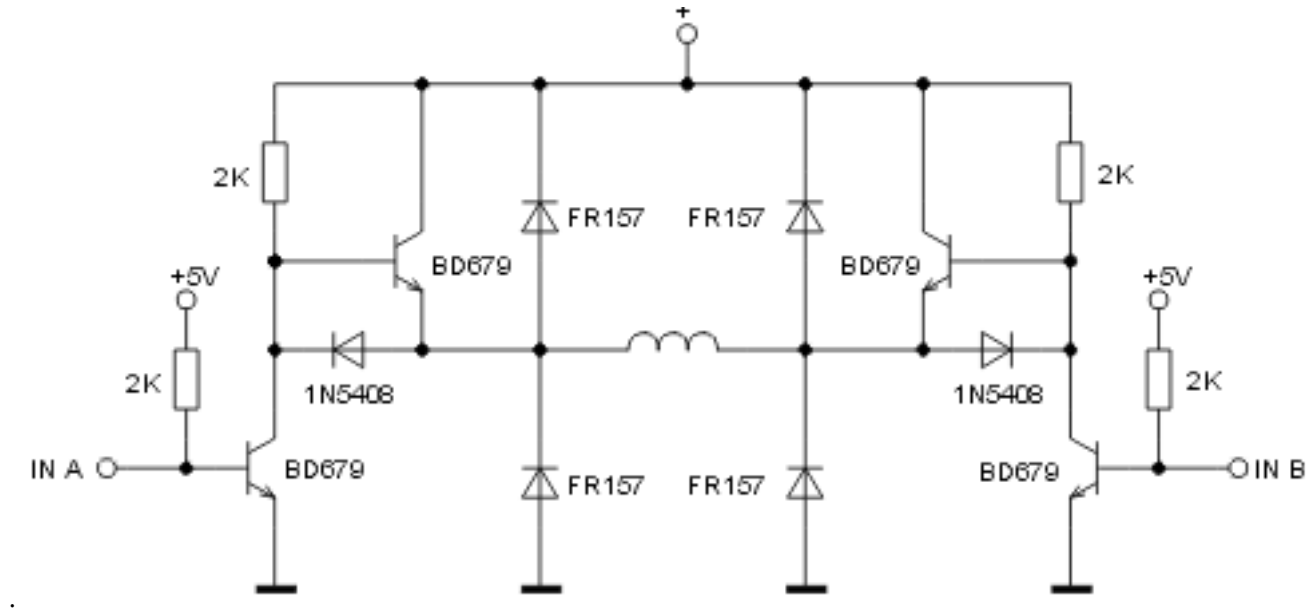


Рис. 2.10. Реалізація мостового драйвера на дискретних компонентах

Управління такийм *H*-мостом здійснюєтьсяється за допомогою двох сигналів, тому він не дозволяє забезпечити всіх можливих комбінацій. Обмотка здійснює живлення, коли рівні напруг на входах різні, та закорочена, коли рівні напруг однакові. Це дозволяє отримати лише повільний спад струму – так зване динамічне гальмування. Мостові драйвери в інтегральному виконанні випускаються багатьма фірмами, наприклад *L*293 та *L298* від фірми *SGS-Thomson*.

Як прості ключі, так і *H*-мости можуть складати частину ключового стабілізатора струму. Ця схема управління ключами може бути виконана як на дискретних компонентах, так і у вигляді спеціалізованих мікросхем. Досить популярною мікросхемою, яка призначена для реалізації ШІМ-стабилизації струму, є *L*297 від фірми *SGS-Thomson*. Спільно з мікросхемою мостового драйвера *L*293 або *L*298 вони утворюють закінчену систему управління для крокового двигуна (рис. 2.11).

Мікросхема *L*297 розвантажує керуючий мікроконтроллер, оскільки від нього вимагається лише тактова частота *CLOCK* (частота повторення кроків) і декілька статичних сигналів: *DIRECTION* – напрям (сигнал внутрішньо синхронізований, перемикання можна у будь-який момент), *HALF/FULL* – напівкроковий / повнокроковий режим, *RESET* – встановлює фази у вихідний стан (*ABCD = 0101*), *ENABLE* – дозвіл роботи мікросхеми, *V ref* – опорна напруга, яка задає пікову величину струму при ШІМ-регулировании. Крім того, є ще декілька додаткових сигналів. Так, сигнал *CONTROL* задає режим роботи ШІМ-регулятора. При його низькому рівні ШІМ-регулювання відбувається по виходах *INH1*, *INH2*, а при високому – по виходах *ABCD*. *SYNC* – це вихід внутрішнього тактового генератора ШІМ та служить для синхронізації роботи декількох мікросхем. *HOME* – сигнал початкового положення (*ABCD = 0101*). Цей сигнал використовується для синхронізації перемикання режимів роботи *HALF/FULL*. В залежності від моменту переходу в повнокроковий режим мікросхема може працювати як в режимі з однією включеною обмоткою, так і з двома включеними обмотками.

Ключове регулювання реалізують і багато інших мікросхем. Деякі з них мають ти чи інші особливості, наприклад драйвер *LMD*18*T*245 від фірми

NationalSemiconductor не вимагає застосуання зовнішнього датчика струму, оскільки він вже реалізований всередині самої мікросхеми.

Деякі мікросхеми призначені спеціально для роботи в мікрокроковому режимі. Прикладом таких схем може служити мікросхема *A*3955 від фірми *Allegro*.

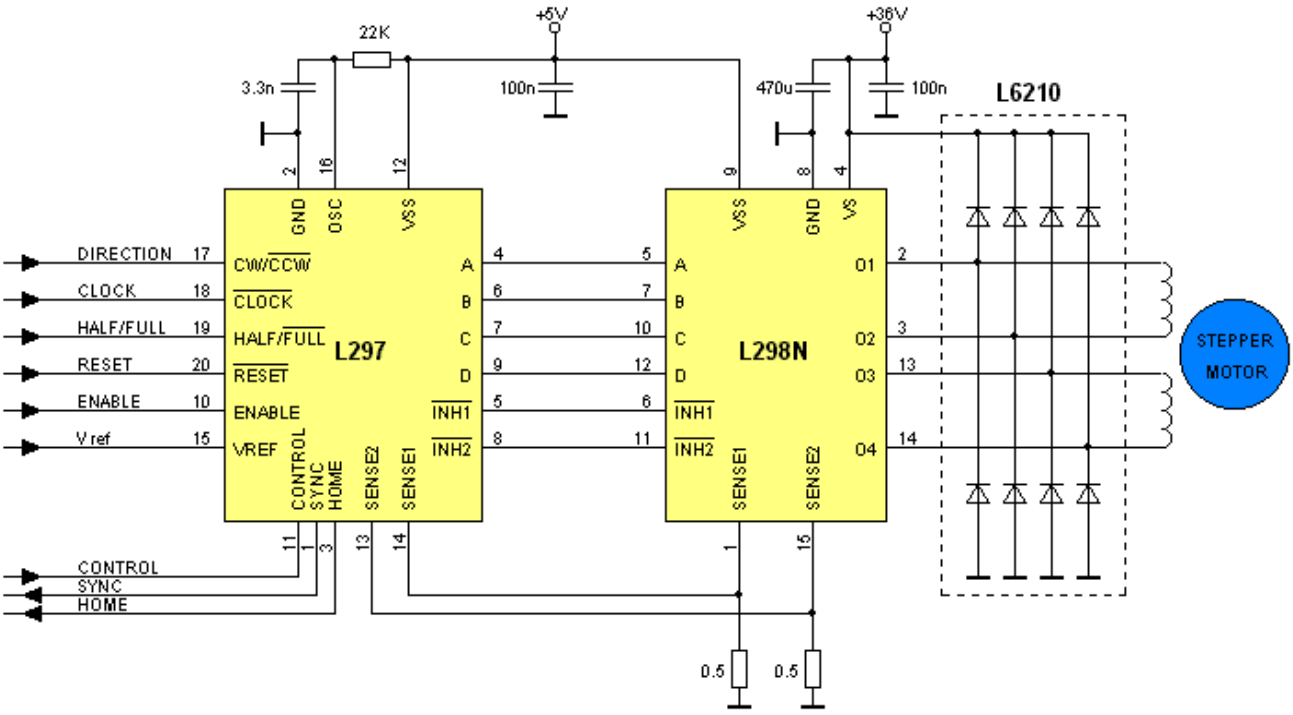


Рис 2.11. Типова схема включення мікросхем *L*297 та *L*298*N*

Висновки до розділу

У другому розділі нашої роботи ми зосередилися на розробці універсальної системи ЧПУ з гнучкою модульною архітектурою. Під час цього дослідження було ретельно вивчено як апаратну, так і програмну складову цієї системи.

Ми провели докладний опис кожного з модулів, що входять до складу системи ЧПУ. Крім того, ми розглянули і проаналізували кожен з модулів системи, а саме: інтерпретатор, інтерполятор та драйвер крокових двигунів. Детально розглянули принципи їх роботи та навели алгоритми, які використовуються для ефективного функціонування цих модулів.

Також ми дослідили сучасні аналоги системи ЧПУ і проаналізували їх переваги та недоліки. Цей аналіз допоміг нам визначити основні переваги нашої розробки порівняно з іншими системами на ринку.

Нарешті, ми вирішили реалізувати пристрій для реалізації всіх функцій системи ЧПУ на платформі Arduinouno. Ця платформа була обрана через свою надійність, доступність та широкий спектр можливостей. Ми ретельно планували проект та виконали всі необхідні з'єднання, налаштування та програмування, щоб забезпечити оптимальну роботу нашого пристрою.

В результаті нашої роботи ми створили універсальну систему ЧПУ з гнучкою модульною архітектурою, яка успішно виконує свої функції за допомогою надійних та ефективних модулів. Реалізація цієї системи на платформі Arduinouno дозволяє забезпечити широкий спектр можливостей та готовність до подальших розширень та модифікацій.

РОЗДІЛ 3
ДОСЛІДЖЕННЯ СУЧАСНИХ ПРОБЛЕМ СЧПУ І ЇХ РІШЕННЯ В РАМКАХ ДАНОЇ РОБОТИ

3.1. Проблема реального масштабу часу

При управлінні процесом програма, що здійснює управління в реальному масштабі часу, циклічно виконує певну визначену користувачем процедуру, робота якої розділяється на часові відрізки (рис. 3.1).

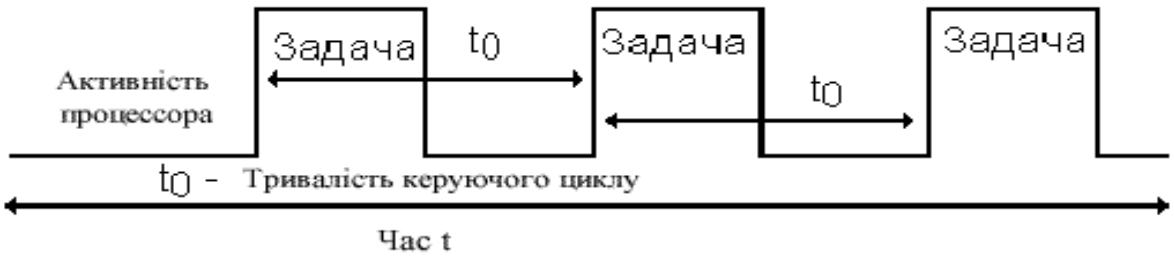


Рис. 3.1. Управління в реальному масштабі часу

Є декілька параметрів, якими можна охарактеризувати застосування реального масштабу часу:

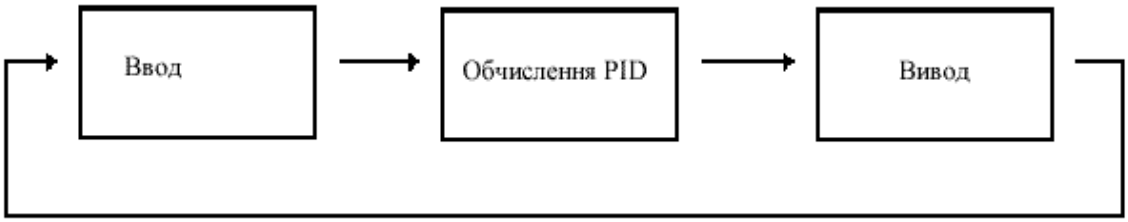


Рис. 3.2. Типовий приклад керуючої програми реального часу

1. Тривалість керуючого циклу. Це величина, що визначається інтервалом часу порівняння поточного стану системи з очікуваним станом. Виходячи з результату цього порівняння визначається те, як в подальшому поведе себе система. У інтерполяторах СЧПУ тривалість керуючого циклу є часом між імпульсами сигналів, що подаються на приводи. Для більш гнучкого налаштування подачі

імпульсів, від якихзалежать швидкість та плавність роботи приводів, що управляють інструментом, тривалість циклу управління не повинна перевищувати 1 мкс. У загальному випадку *PID*-управління в режимі реального масштабу часу відповідає (див. рис. 3.2).

2. Передбаченість. Характеризує постійність певних інтервалів часу між подіями. Багато управляючіх алгоритмів, такі як *PID*, вимагають дуже передбаченої поведінки системи. Система ОСРЧ являється передбачуваною в тому сенсі, що час, який витрачається на певну роботу, не повинен перевищувати заздалегідь встановленого ліміту. Час реакції на переривання полягає в здатності своєчасної реакції на зовнішні події, та зазвичай не перевищує 2 - 8 мкс. Час перемикання контексту використовується для передачі управління від процесу до процесу, від потоку до потоку, та знаходиться у межах 80 - 160 мкс. Час реакції планувальника – це затримка активізації процесу після відпрацювання переривання, та знаходиться в межах 4 - 16 мкс [23].

3. Розкид. Це ще один спосіб охарактеризувати непередбачуваність систем реального масштабу часу. Цей параметр розраховують як різницю між окремою часовою затримкою та її бажаною величиною.

Застосування управління в реальному масштабі часу можуте бути по-різному розподілено за шкалою швидкодії. На одному краю розташовуються системи жорсткого реального часу, які є дуже передбачуваними і ніколи не втрачають події. На іншому ж кінці шкали розташовані системи м’якого реального часу, від яких не вимагається високої передбаченості і яким дозволяється пропускати події.

Оптимальне використання обчислювальних ресурсів систем управління передбачає розподіл роботи моделі в машинному і реальному масштабах часу. Управління взаємодією модулів називають диспетчеризацією, вона використовує засоби операційних систем реального масштабу часу. Але диспетчер аж ніяк не замінює операційну систему.

Таким чином, перед розробниками СЧПУ постає завдання знайти способи, які б здійснювали організацію спільної роботи всіх модулів системи управління в цілому.

Для цього необхідне раціональне вирішення проблеми реального часу і побудова на базі цього рішення диспетчера прикладних програм. В зв'язку з цим були досліджені існуючі на сьогоднішній день пропозиції по використанню стандартних і оригінальних операційних систем реального часу, а також розширень реального часу операційних систем *Windows* та *Linux*. Крім того, були виділені типи процесів і потоки системи управління.

Традиційно системи реального часу, включаючи модуль диспетчера, будують на базі операційних систем реального часу (ОСРЧ). Операційні системи загального призначення, наприклад такі, що розраховані на багато користувачів – типа *UNIX*, орієнтовані на оптимальне використання розподілу ресурсів комп'ютера між користувачами і процесами, що виконуються. В управляючіх системах подібні завдання відходять на другий план, оскільки основна їх мета полягає в своєчасній реакції на події в об'єкті управління.

В зв'язку з цим розглянемо класифікацію можливих рішень:

1. Виконавчі системи реального масштабу часу пропонують різні платформи для розробки і виконання програмного забезпечення. Прикладну частину реального масштабу часу розробляють на хост-компьютері, потім об'єднують з ядром та завантажують в систему управління як єдине завдання. Таке рішення дає високу точність і швидкодію. Прикладом такої системи може послужити операційна система реального часу масштабу *VxWorks*.

2. Монолітні ядра реального часу мають повний набір специфічних механізмів реалізації роботи в реальному часі. Такі ядра є компактними, масштабованимі і мають модульну і добре структуровану побудову. Типовими представниками є *OS*9 (*MicrowaveSystems*) та *QNX* (*RIM*, Канада).

3. Системи управління з операційною системою *UNIX* реального масштабу часу переписують ядро стандартної операційної системи з врахуванням вимог реального часу. Такі системи підтримують весь набір *UNIX*-програм. Проте система *UNIX* реального часу має досить великий об'єм і низьку реактивність. Типовим і широко розповсюдженим представником сімейства *UNIX* єоперaційна система *Linux*.

4. Сучасні системи числового программного управління все частіше використовують операційну систему сімейства *Windows* (версії 10 та більш ранні, включаючи *NT*) з розширенням реального часу.

В дипломній роботі операційні системи використовують набір традиційних механізмів. Механізм пріоритетів і диспетчеризації забезпечує плаування завдань реального часу на основі використання деякого кванту часу.

Механізм взаємодії між різними задачами синхронізує процеси і передачу даних між ними, використовуючі різні механізми: семафори, м'ютекси, сигнали подій, пам'яті, що розділяється. Механізм роботи з таймерами генерує переривання після закінчення деякого певного інтервалу часу.

Операційна система *Windows* 10, та більш ранні версії, включаючи *NT*, не є операційною системою реального масштабу часу, оскільки вона не має достатнього діапазону пріоритетів потоків, а також не дозволяє управляти спадкоємством пріоритетів, тобто блокуючий потік повинен успадковувати пріоритет потоку, який він сам блокує. До того ж механізм синхронізації потоків є непередбачуваним, час реакції на переривання непередбачуваний.

Тим часом через зростаючу популярність в системах управління операційної системи *Windows* різних версій, ця проблема має бути вирішена будь-яким чином. Зі всіх існуючих на сьогоднішній день пропозицій по реалізації ОСРЧ на базі ОС сімейства *Windows* практичнее значення мають всього два підходи.

1. Перший підхід полягає в запуску *Windows* у вигляді низькопріоритетного завдання операційно їсистеми реального масштабу часу, тобто супервізора. При цьому передбачається використання ядра класичної ОСРЧ типа *QNX* або *VxWorks*. Існують і такі рішення, в яких в якості супервізора використовується *VxWorks*.

2. Другий підхід полягає в розширенні ОС *Windows* у сенсі реального часу. Це може бути оригінальна розробка виробника системи управління. В якості прикладу такої системи можна навести *WINCAT – BackhoffIndustrieElectronic*, ФРН. Інший варіант – використання вже готового комерційного рішення, наприклад *RTX*4.1 від фирми *VenturCom*.

Обидва ці підходи мають як свої переваги, так і недоліки. Але підхід, заснований на базі розширення реального часу для ОС сімейства *Windows* все ж має право на життя. По-перше, в розширенні використані ті ж типи об'єктів для управління завданнями, що і в самому ядрі *Windows* – м'ютексы, семафори і так далі. В протилежність цьому, *VxWorks* використовує оригінальні функції та механізми, що формують власний стиль, кардинально відмінний від стилю *Windows*. По-друге, немає необхідності в використанні другої операційної системи, що скорочує витрати і знімає проблеми установки та стиковки обох операційних систем на одному комп'ютері.

Вирішення на користь розширення реального часу *Windows* (рис 3.4) дозволяє швидко оновлювати систему управління з появою нових версій *Windows* за умови підтримки сумісності, здійснювати потужний захист застосувань, який *Windows* виконує за допомогою незалежного абстрактного рівня *HAL*, досить легко відлагоджувати коди і використовувати можливості стандартних механізмів *Microsoft* для інформаційного обміну між *Windows* і завданнями реального масштабу часу. Серед таких механізмів є наступні: *IPC* – механізм міжпроцессного зв'язку, *OLE* – механізм скріплення і впровадження об'єктів, *COM* – механізм компонентних моделей, та *RPC* – механізм віддаленого виклику процедур.

На (рис 3.3) представлені багаторівнева структура *Windows* 10 з *RTX* і розміщенням основних потоків системи управління. В самому низу знаходиться рівень апаратної абстракції реального часу *HAL*, на якому реалізовані швидкодіючий годинник та таймери, механізм розмежування переривань між *RTX* та *Windows*.

Підсистема реального масштабу часу *RTSS* виконана у вигляді драйвера, вона працює на рівні ядра системи *Windows* та забезпечує основні функції та управління ресурсами *RTX*. Ця підсистема використовує сервісні можливості *HAL* реального часу і *Windows* для роботи з швидким годинником і таймерами, та служить для обслуговування механізму переривання. Вбудований в *RTSS* менеджер потоків і планувальник, засновані на фіксованій системі пріоритетів управління прикладними завданнями [26, 27].

Підсистема *RTSS* забезпечує інтерфейс між процесами *RTX і Windows* в реальному масштабі часу за допомогою спеціального сервісного механізму *IPC – Inter Process Communication*. Треба зазначити, що *RTX* повністю відповідає концепції фірми *Microsoft*, яка допускає заміну рівня *HAL* без заміни ядра операційної системи, при цьому розширення функціональностей рівня апаратної абстракції здійснюється шляхом додавання необхідних драйверів.

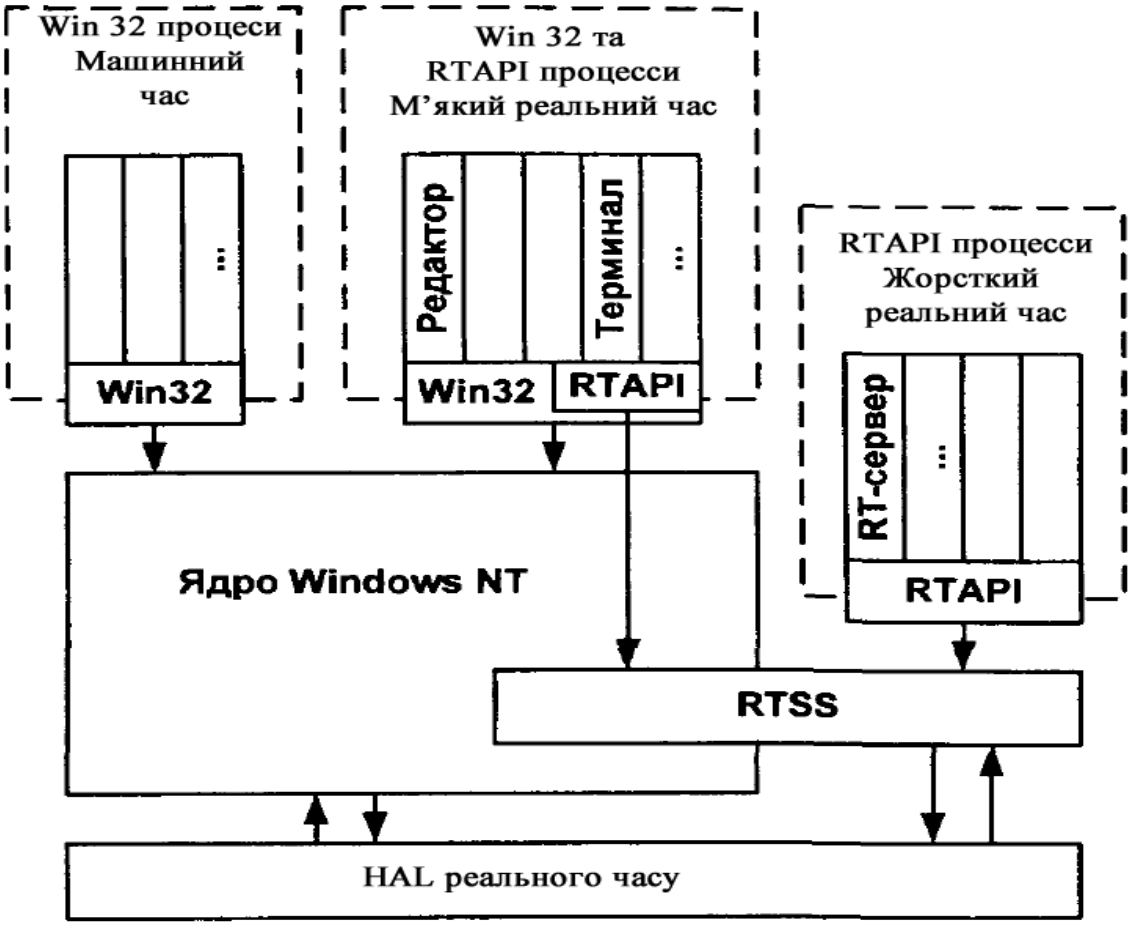


Рис. 3.3. Основні потоки системи керування з використанням *Windows* та *RTX*

В операційній системі працюють звичайний прикладний інтерфейс *Win*32 для *Windows*, який передбачений для машинного часу, а також додаткові прикладні інтерфейси реального часу *RTAPI – RealTimeApplicationInterface* та *Win*32*RT – RealTime*. Додаткові прикладні інтерфейси забезпечують два режими роботи в реальному масштабі часу – «жорсткий» та «м'який». Це дозволяє оптимізувати обчислювальні ресурси системи управління, розділивши її функціональні завдання

на три омновні групи:

1. В режимі жорсткого реального часу вирішуються критичні завдання, такі як інтерполяція кадрів КП, введення-виведення і т.д., які реалізовані в процесі роботи *RT*-сервера (рис. 3.4.).

2. В режимі м’якого реального часу вирішуються завдання, що безпосередньо пов’язані із завданнями реального часу, такі як інтерпретація кадру управляючої програми. Вони реалізовані в процесі «термінал» (рис. 3.4). На відміну від жорсткого реального часу, тут допустимі затримки потоку через стопінг, або підкачку, звернення до жорсткого диска, здійснення переривання і т.д..

3. В режимі машинного часу працюють останні стандартні прикладні модулі системи управління, такі як редактор управляючих програм, вбудована CAM система, система моделювання процесу обробки і т.д.

Стратегія диспетчеризації завдань в системі управління (рис. 3.4), де стрілками вказана послідовність подій. Спочатку в *Windows* з підсистемою реального часу *RTSS* створюється таймер. Після закінчення кванта часу стандартний механізм здійснює генерування преривання, яке обробляється прикладною *call-back* функцією, або так званою функцією зворотного виклику.

Ця функція реалізує алгоритм планерування, або диспетчеризації завдань інтерпретацій, інтерполяції, введення-виводу, комунікації та інтерфейсу оператора *MMI*. Відповідно до позначених для системи управління режимів в жорсткому реальному часі виконуються завдання диспетчеризації, інтерполяції, введення-виводу, та комунікації. У м’якому реальному часі виконуються завдання інтерпретації та оновлення екранів інтерфейсу з оператором, а у фоновому процесі здійснюється завдання інтерфейсу з оператором.

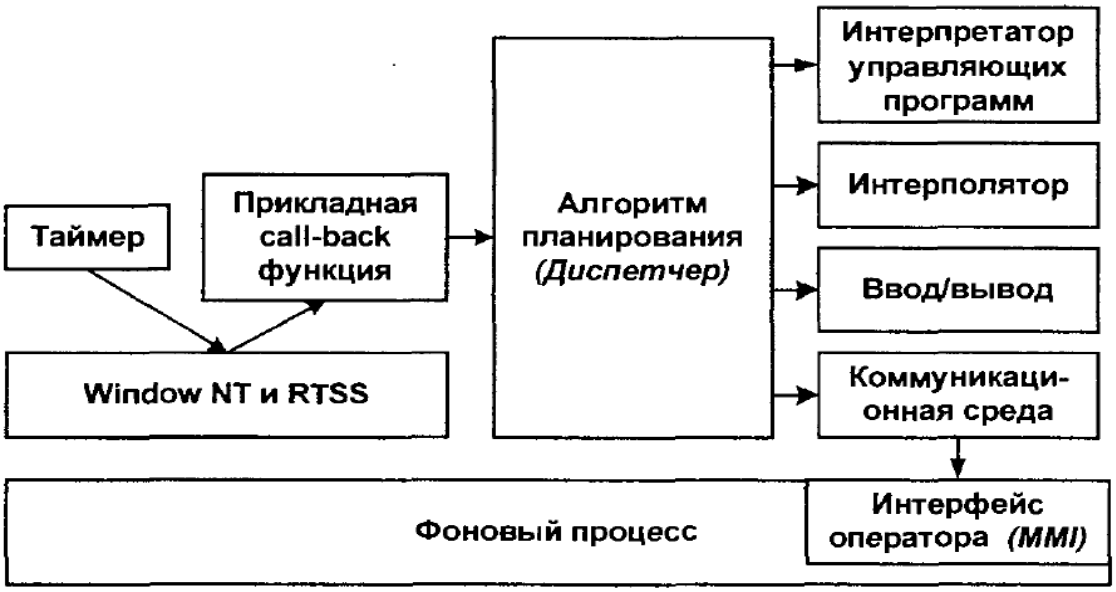


Рис. 3.4. Стратегія диспетчеризації задач реального масштабу часу в системі управління на базі *Windows*

Для вирішення проблеми реального часу в СЧПУ на базі операційних систем сімейства *Linux* використовується розширення ядра *RTLinux*. Так, це розширення використовується в проекті *LinuxCNC*. В системі *RTLinux* всі переривання обробляються ядром реального часу, який включає в себе власний планувальник завдань, обробник переривань та бібліотечний код. В разі відсутності обробника реального часу для якогось переривання, воно передається в саму операційну *Linux*. Фактично *Linux* є простоюючим завданням ОСРЧ, яке запускається лише в тому випадку, коли жодне їз завдань не виконується в реальному масштабі часу. При цьому на *Linux*-задачу накладаються деякі обмеження, які, втім, є прозорими для програміста. ОС *Linux* не здатен виконувати наступні операції:

1. блокування апаратних перериваннь;
2. оберігання себе від витіснення іншим завданнями.

Ключ до реалізації даної системи в тому, що емулюється система управління перериваннями драйвер, до якого звертається *Linux* при спробі блокування переривання. В цьому випадку драйвер перехоплює запит, зберігає його і далі повертає управління ОС *Linux*.

Всі апаратні переривання перехоплюються ядром ОСРЧ. Коли відбувається

переривання, ядро *RTLinux* вирішує, що далі треба робити. Якщо це переривання має бути оброблене обробником реального часу, то ядро здійснює виклик відповідного обробника. Інакше, або якщо обробник реального часу хоче розділяти це переривання з *Linux*, обробнику привласнюється стан чекання. Якщо *Linux* потребує вирішити переривання, то переривання, які знаходяться в стані очікування, емулюються.

Системне ядро *RTLinux* спроектовано таким чином, щоб ядро реального часу ніколи не мало потреби в очікуванні звільнення ресурсу, який зайнятий *Linux*-процесом. Для обміну даними між ОСРВ та ОС *Linux* передбачені наступні засоби:

1. Роздільні області пам’яті.
2. Псевдопристрої, які надають можливість обміну даними із застосуваннями реального часу.

Ключовий принцип побудови системи *RTLinux* – це якомога більше використовувати звичайний Linux, та якомога менше *RTLinux*. Сама ОС *Linux* піклується про ініціалізацію системи і пристроїв, а також про динамічне виділення ресурсів, якщо це необхідно. На *RTLinux* лягає задача лише планерування завдань реального часу та обробка переривань. Процеси реального масштабу часу реалізовані у вигляді завантажуваних модулів *Linux* для простоти запуску в контексті ядра, збереження модульності та розширюваності системи. Застосування реального часу з *RTLinux* зазвичай, складається з двох незалежних частин: процесу, що виконується ядром *RTLinux* і звичайної *Linux*- програми.

Такий модульний підхід до написання застосунків властивий багатьом розширенням реального часу для багатьої операційних систем, де завдання реального часу працює незалежно від самої ОС. Розробники додержуються схеми, згідно з якій критичні до часу реакції завдання програмуються за допомогою *API*-интерфейсів, які передбаченї розширенням реального часу, а всі інші функції сервісу та інтерфейсу з користувачем покладається на потужність самої операційної системи. При використанні такого підходу, програмісту потрібно вивчити лише *API*-интерфейс

обробника подій реального часу. Такий підхід передбачує наступне:

1. Застосування реального часу виконуються в самому ядрі, а отже, вони можуть переписати частину пам’яті ядра та зламати систему.

2. Взаємодія між *RT*-підсистемою та ОС *Linux* може бути лише нереалтаймовым.

3. Ядро ОС *Linux* працює в фоновому режимі, отже завдання *Linux* можуть мати великі затримки при їх виконанні.

4. Неможливо використовувати драйвери системи *Linux* в завданнях реального часу. Через це розробники застосунків реального часу вимушені переписувати драйвери пристроїв поверх *RT*-підсистеми.

Всі ці недоліки, що були вказані вище, в тій чи іншій мірі властиві всім розширенням реального часу для ядер операційних систем, що змушує прикладати величезні сили та засоби для вирішення ряду завдань ЧПУ, що вимагають роботи в режимі реального масштабу часу. Крім того (див. рис 3.3), подібні реалізації ОСРЧ вимагають великих зусиль програмістів для розробки програмного забезаечення для роботи з ними.

Для більш комплексного бачення проблеми реального часу для роботи СЧПУ проаналізуємо ті частини системи, що вимагають роботи в реальному масштабі часу.

Інтерполятор забезпечує передачу управляючіх сигналів на приводи. В разі затримки з боку операційної системи, імпульси сигналу можуть змінювати свою форму, а це може призводити до цілого ряду непередбачених наслідків. Наприклад, до зсуву інструменту або нерівномірної швидкісті його руху, що при деяких способах обробки деталей призводить до падіння точності родоти верстата і, відповідно, до зростання кількості браку.

У зв’язку з необхідністю реалізації інтерполяторів в кожній системі, що претендує на управління СЧПУ, та з тим, що інтерполятори є стандартними елементами, пропонується максимально виключити використання прикладного рівня від модуля реального часу. Як описано в розділі 2.3.2. інтерполятор є відносно

простим стандартним алгоритмом, який виконується з к ординатою стартової точки у вигляді вхідного параметра.

Сума простих операцій, що описана в алгоритмах на рис 2.6, та 2.7, може бути апаратно реалізована, про що свідчать відповідні патенти ще часів СРСР.

Так, логічна схема апаратного кругового інтерполятора, що реалізовується по методу оцінної функції, як це було описано в розділі 2, була описана в патенті [21] (рис. 3.5). Де відповідно 1 – блок визначення кроку; 2 і 3 – реверсивні лічильники; 11 і 14 – суматори-накопичувачі; 5, 6, 12 – логічні елементи «І»; 7, 9 – регістри для зберігання координат кінцевої точки; 4, 8, 10, 13, 15 – блоки порівняння; 16 – блок виведення приростів по координатах.

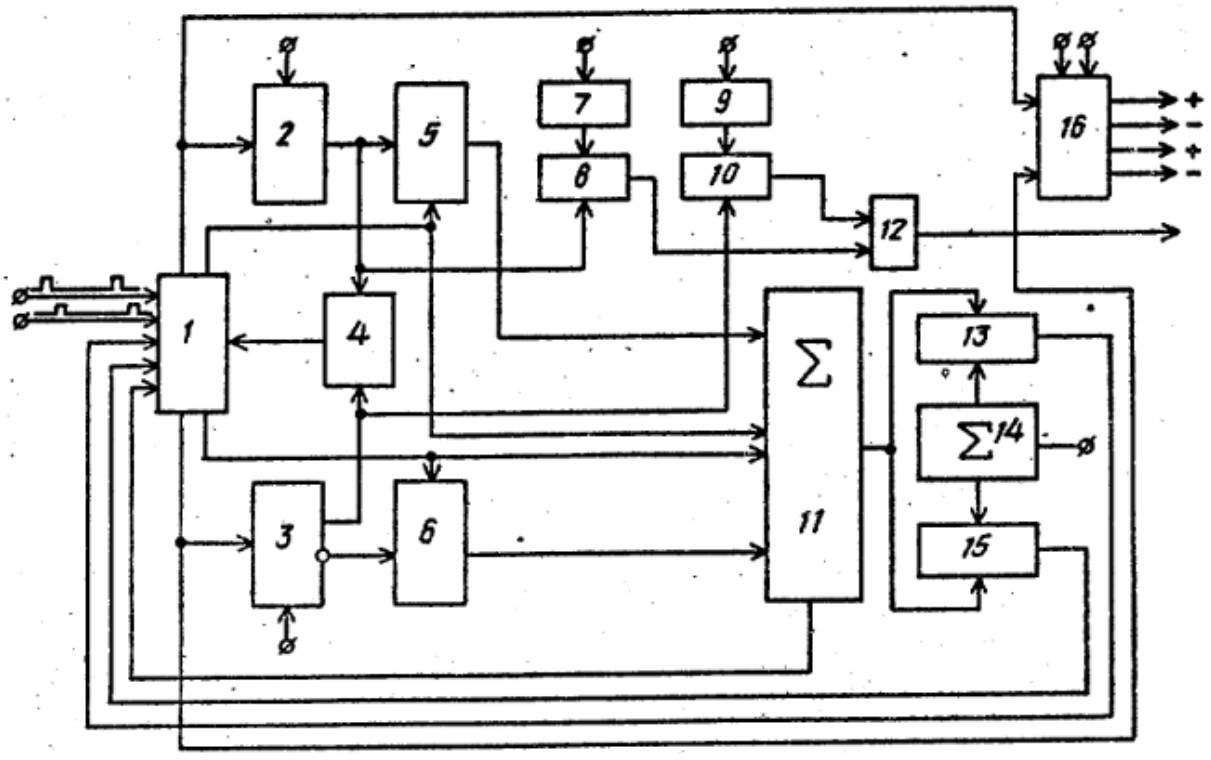


Рис. 3.5. Логічна схема апаратної реалізації кругового інтерполятора

Логічна схема апаратної реалізації лінійного інтерполятора, що навелена на (рис. 3.6), описана в патенті [22].

Цей інтерполятор містить в собі наступні складові: регістри 1, 2, 5; генератор імпульсів 4; двійковий дільник частоти 3; логічний елемент «І» 6.

Схеми інтерполяторів такого типу не використовуються при розробці пристрою на основі запропонованої концепції, оскільки на сьогоднішній день всі алгоритми інтерполяторів значно простіше реалізувати, запрограмувавши їх на мікроконтроллері. Ввони були лише наведені з метою отримати уявлення про складність апаратної реалізації таких алгоритмів інтерполяторів. Тому в дипломній роботі лише приведені загальні логічні схеми та елементи. Принцип дії розглянутих апаратних інтерполяторів можна взяти у вказаних патентах, з яких виходить, що реалізація інтерполяторів поза основним комп'ютером в рамках розширення до контроллера СЧПУ можлива за наявних технологій. Така реалізація і доступ до неї за допомогою спеціальних *API* дозволить радикально скоротити витрати на написання ПЗП для СЧПУ і прибере проблему реального часу з поля зору програмістів.

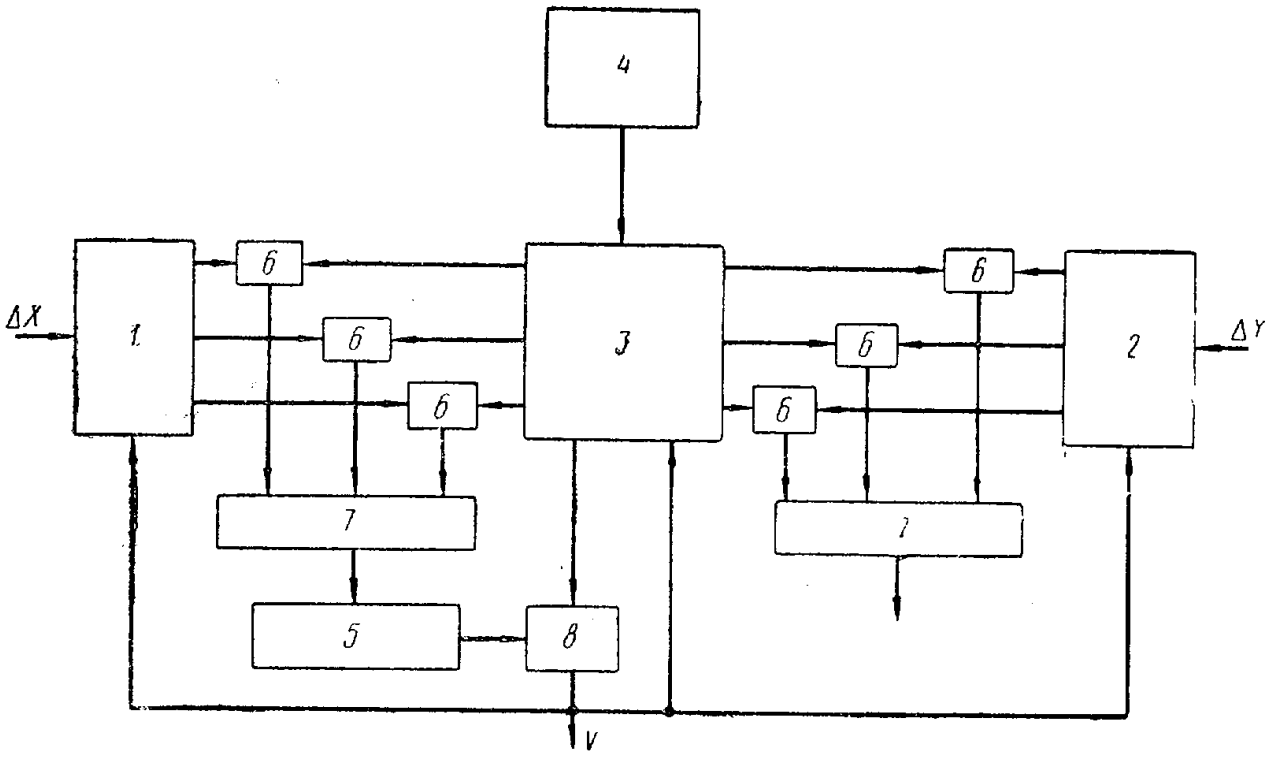


Рис. 3.6. Логічна схема апаратної реалізації лінійного інтерполятора

Тому слід визнати дане рішення простим, ефективним та доцільним вирішенням проблеми реального масштабу часу в найоптимальнішому місці СЧПУ – модулі інтерполятора.

3.2. Проблема міжмодульної комунікації

Одна із цілей при розробці систем нового покоління полягає в найбільш повнішому використанні принципів відкритої архітектури. Саме активно використнння в наш час рішення для класичних *PCNC* лежить в області використання досягнень у сфері системної інтеграції великих систем.

До складу математичного забезпечення системи ЧПУ входять оригінальні програмні компоненты виробника систем; компоненты, що замовлені в інших виробників; готові комерційні продукти; компоненты замовника і кінцевого користувача. При цьому вся система має зберігати всі ознаки відкритої архітектури.

В зв'язку з цим в архітектурі *PCNC* можуть бути використані принципи *OLE/COM*, та деякі специфікації *OPC*. Компонентний підхід та відомі принципи системної інтеграції можуть бути використаний не лише при розробці окремих модулів, а і на рівні проектування всієї системи в цілому.

Іноді виникає потреба заміни одного модуля на іншій безпосередньо в процесі роботи. Це відноситься, наприклад, до інтерпретатора управляючих програм на мові *DIN* 66025, коли міняється версія мови, при запуску КП, що була розроблена для іншої системи ЧПУ. Модулі, що бкли реалізовані у вигляді компонентів, можна відключати і міняти без перекомпіляції програмного забезпечення і без перекомпонування системи ЧПУ. При цьому змінюється поведінка системи, але не її архітектура.

COM-підхід підтримує розподілену систему функціонування – модулі системи ЧПУ можуть працювати в різних потоках і різних системах, виступати як *COM*-сервери компонентів і *COM*-клієнтів.

Введемо поняття об'єктно-орієнтованої магістралі, як засобу міжмодульної комунікації та джерела необхідних міжмодульних послуг. Магістраль є програмним, або віртуальним каналом для обміну даними між модулями, шо підключенв до каналу. Об'єктно-орієнтована магістраль передбачає наявність в її програмному забезпеченні набору об'єктів, що вирішують задачі транспортування даних,

підключення модулів, та ін. На рис. 3.8. представлено чотири типм функцій об’єктно-орієнтованої магістралі: функції запиту даних, функції управління, функції відображення, та допоміжні функції.

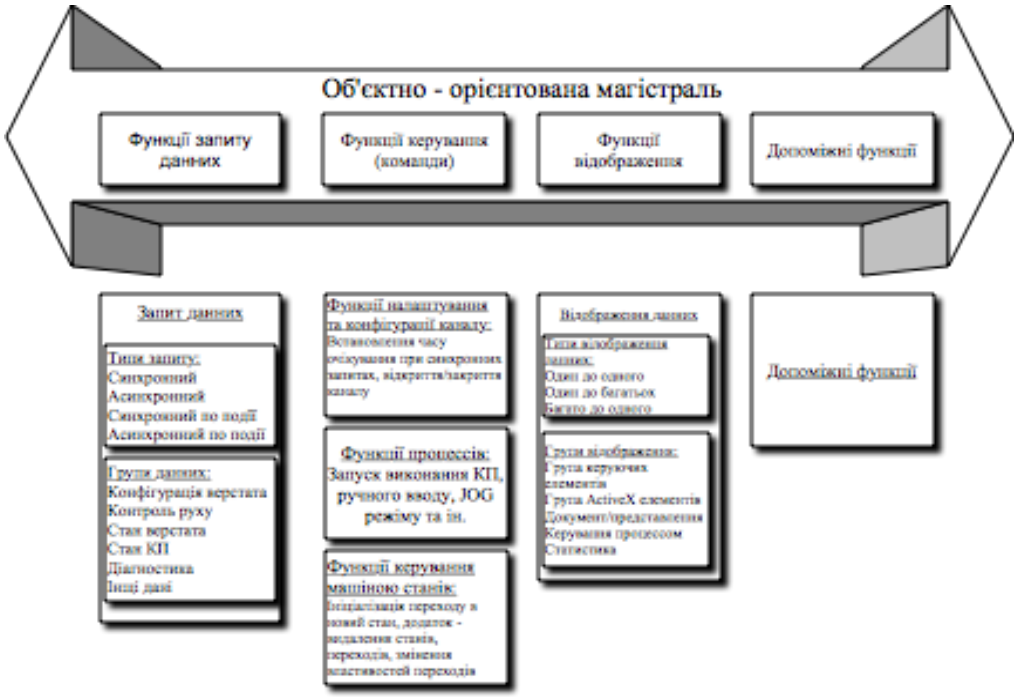


Рис. 3.8. Функції інтерфейсу об’єктно орієнтованої системи

1. Функції запиту даних. Передбачають, що в *PCNC*-системі існують дані різних типів і потреба в цих даних різна. Наприклад, дані про кількість і наіменування використовуваних на верстаті координатних осей потрібні один раз в момент ініціалізації *PCNC*-системи. Дані про поточний стан процесу обробки потрібні постійно, аби приймати коректні рішення щодо управління. Існують і інші варіанти запитів на одержання даних. Саме тому об’єктно-орієнтована магістраль передбачає п’ять їх видів: синхронний, асинхронний, синхронний по події, асинхронний по події, та асинхронний циклічний запит.

2. Функції управління. Ці фунуції можна розбити на три групи: функції управління каналом, які відкривають та закривають канал; функції процесів, що контролюють хід їх виконання, включаючи запуск та зупинку; функції управління станами (будуть розглянуті нижче). Якщо запит пов’язаний з процесом одержання

даних з модуля-джерела, тобто сервера через внутрішню структуру комунікаційного середовища, то процес перенесення і обробки цих даних в модуль-клієнт відноситься відповідно до групи функції відображення даних.

3. Функції відображення. В табл. 3.1. наведені фази обміну даними через магістраль. У фазі запиту даних визначається сервер даних. В якості даних можуть висткпати: дані про поточні координати, величину подачі, списку активних *G*-функції, або *G*-вектори і т.д. В цій же фазі встановлюється тип запиту і приймач даних, або клієнт. У фазі відображення визначається тип і формат відображення. Формат відображення передбачає, що одні і ті ж дані можуть відображувати, наприклад, в текстовому або числовому вигляді.

Таблиця 3.1.

Фази обміну даними через об’єктно-орієнтовану магістраль

Запит данних	**Відображення данних**
Джерело данних	Тип відображення
Сессія запиту	Формат відображення
Приймач данних	

Об’єктно-орієнтована магістраль передбачає три типи відображення, для підтримки кожного з яких потрібен власнй механізм.

В Залежності від способу виведення даних, відображення розділені на декілька груп: візуалізація в галереї елементів – *controlelements*; управляюча, яка будується на базі стандартних *Windows*-елементів; візуалізація в галереї *ActiveX*-элементів, яка будується на базі *OLE*-элементів *Windows*, які розширюють стандартний набір *Windows*-элементів; візуалізація в середовищі «документа-представлення» при створенні призначених для користувача застосунках на базі стандартного механізму *GUI*; візуалізація з метою управління ходом процесу в *PCNC*-системі; статистичне накопичення даних для збереження, наприклад, в базі даних, або для їх подальшого аналізу.

4. Допоміжні функції об’єктно-орієнтованої магістралі. Вони складають єдиний механізм конвертації та форматування даних, обробки помилок, формування виключень для всіх модулів, що підключені до объектно- орієнтованої магістралі. Взаємодія модулів *PCNC*-системи між собою носить клієнт-серверний характер [16]. Транзакції, або сесії між модулем-клієнтом, що запрошує послугу, і модулем-сервером, що надає цю послугу, узагальнені по їх призначенню (рис. 3.8).

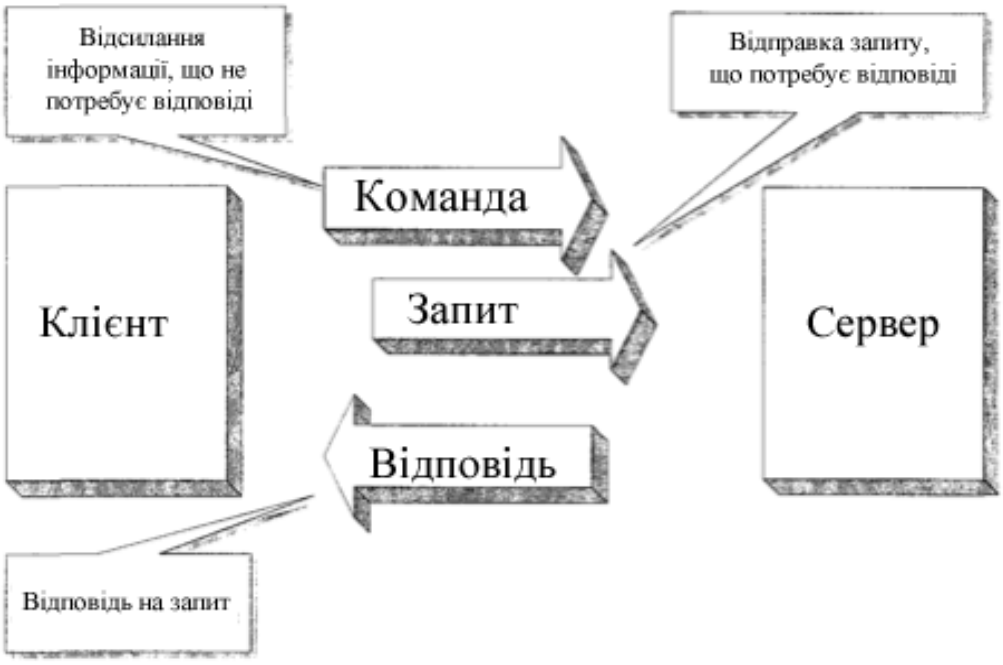


Рис. 3.8. Клієнт-серверна взаємодія в *PCNC* системі

Команда прямує до сервера для виконання певної дії, наприклад, запуску управляючої програми. Така команда не передбачає відповіді з боку сервера. Інший варіант: запит, що прямує серверу з метою здобуття деяких даних, наприклад, значення поточних координат – такий запит передбачає відповідь з боку сервера.

На наведені базові транзакції: синхронна, асинхронна та по події (рис. 3.9).

В рамках синхронної сесії, (рис. 3.9, а), клієнт направляє запит до серверу і припиняє роботу в точці запиту. При готовності сервер відповідає, після чого клієнт продовжує свою роботу.

В рамках асинхронної сесії, (рис. 3.10, б), клієнт направляє запит до серверу і продовжує свою роботу далі. Відповідь сервера обробляється спеціальною *call-back-*

функцією, що є аналогічною функції обробки переривання клієнта. Подією в системі *PCNC* служить будь-яка зміна даних, наприклад зміна стану процесу.

В рамках асинхронної сесії по події, (рис. 3.10, в) клієнт направляє запит до серверу і продовжує свою роботу. Сервер відповідає лише після того, як станеться подія, тобто зміняться дані запрошування. Відповідь сервера обробляється *call-back*-функцією клієнта.

Синхронну сесію по події (рис. 3.10 г), використовують лише для відладки *PCNC*-системи. В цьому випадку, клієнт направляє запит до серверу, та припиняє роботу в точці запиту. Клієнт продовжує свою роботу лише в тому випадку, якщо станеться подія, і сервер даст відповіь клієнту.

В рамках дипломної роботи реалізована подібніа система, що була описана вище, система комунікацій між модулями. Розроблену систему можна представити у вигляді двох модулів. Це – модуль реального часу і модуль прикладних програм. Комунікація між модулями здійснюється за схемою клієнт-сервер за допомогою комп’ютерної мережі *Ethernet*.

На основі базових транзакцій може бути реалізовано циклічне опитування даних. Запропонований блок ЧПУ є пристроєм реального масштабу часу, що здатен отримувати команди для виконання по комп’ютерній мережі.

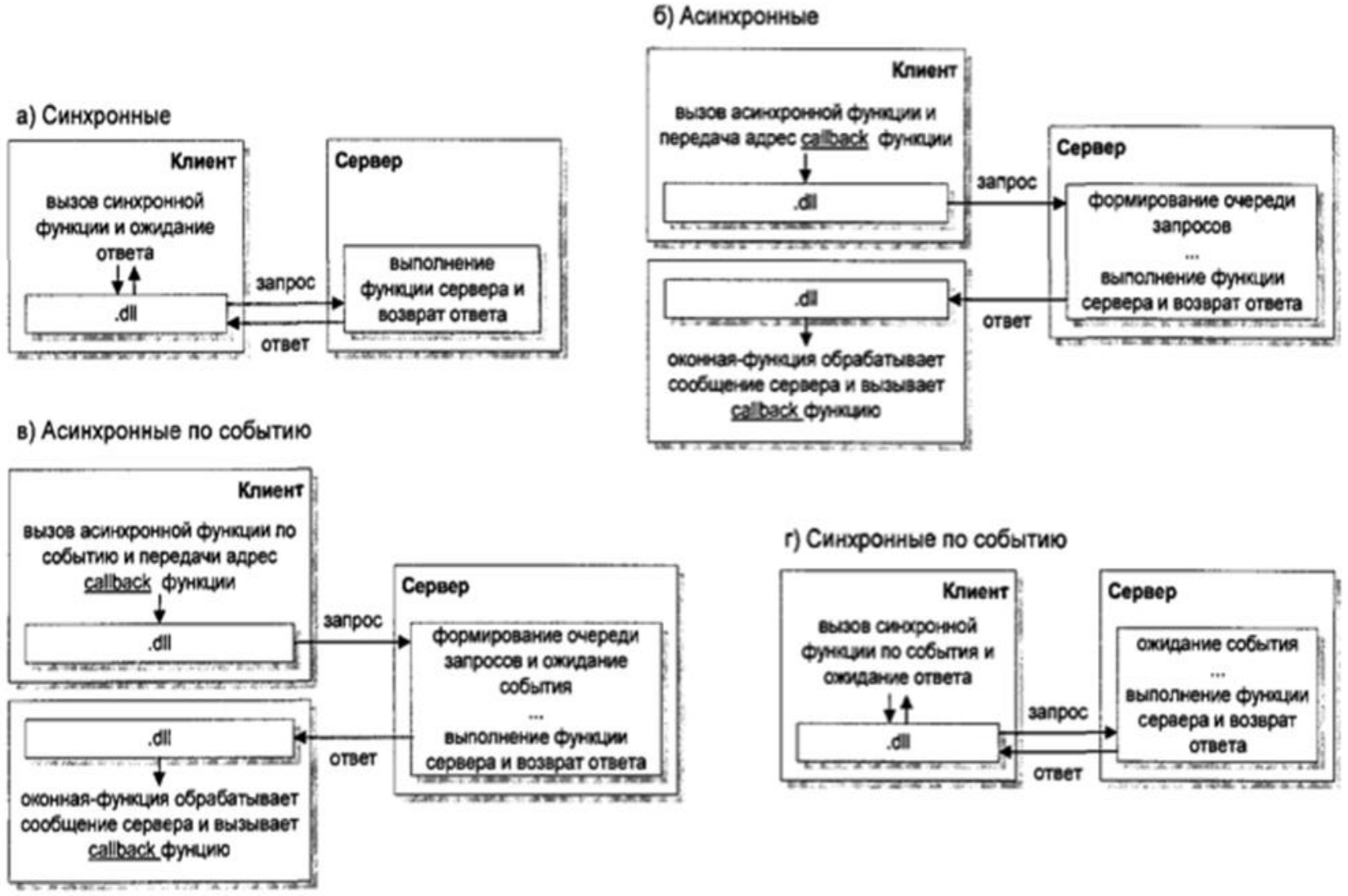


Рис. 3.9. Омновні сесії обміну даними мфж модулями СЧПУ в *PCNC* системі

На (рис 3.11) представлена схема взаємодії модулів в розробленому ПЧПУ.

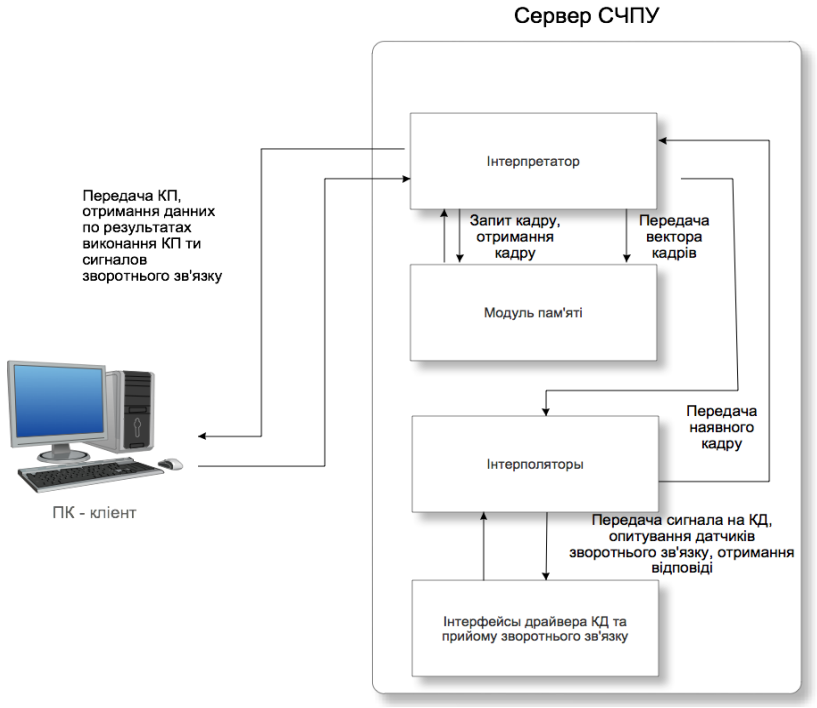


Рис. 3.11. Взаємодія між модулями ПЧПУ

3.3. Дослідження сучасних тенденцій розвитку систем ЧПУ і відповідність ним гнучкої модульної архітектури

В подальші роки розвиток верстатного устаткування ЧПУ буде йти по шляху створення:

– простіших верстатів із спрощеними пристроями ЧПУ і адаптивного управління для фрезерної і токарної обробки деталей, що мають складну форму;

– автоматичних маніпуляторів, або промислових роботів, що забезпечують встановлення деталі, її зняття, переміщення її в робочу зону і контроль;

– контрольних автоматів з ЧПУ;

– гавтоматизованих ділянок, керування якіх відбувається від однієї ЕОМ;

– верстатів третього покоління, призначених для встроювання в автоматичні ділянки.

В розвитку СЧПУ на сьогоднішній день спостерігаються наступні тенденції:

1. Збільшення інтегральної обчислювальної потужності систем, яка виражається в скороченні системного такту до сотень і навіть десятків мікросекунд. Це призводить до підвищення швидкості і точності в робочих переміщеннях верстатів з ЧПУ.

2. Збільшення числа одночасно керованих координат, або числа інтерпольованих координат і каналів управління. Це обумовлює можливість управління від однієї системи декількома верстатами та іншим технологічним устаткуванням, що об'єднується в комплексну технологічну систему.

3. Впровадження і розвиток алгоритмів адаптивного управління, з використанням в режимі жорсткого реального часу інформації від різних сенсорів. Це спияє підвищенню якостф обробки на верстатах з ЧПУ, особливо при високошвидкістній обробці і при обробці матеріалів, що важко оброблювати.

4. Розвиток систем автоматизованого програмування систем ЧПУ – мови високого рівня, пост-процесори від систем САПР.

5. Інтеграція систем ЧПУ в промислові обчислювальні мережі цехового та заводського рівня.

6. Скорочення вартості систем ЧПУ.

7. Управління від однієї системи ЧПУ різнорідними виконавчими пристроями – электро-, пневмо-, гидро, пьезоприводами і т.д.

Значний потенціал для якісного ривка в області систем ЧПУ виявляє прийнятий у всьому світі і серйозно застарілий на сьогоднішній день метод програмування систем ЧПУ на основі код *ISO* 7 *Bit*, або мова *G*-кодів. Цей метод склався ще в 70-і роки минулого сторіччя, та спочатку був розроблений для перфострічки в якості програмного носія. Розрізнені спроби впровадження проблемно-орієнтованих, інтуїтивно зрозумілих користувачеві мов програмування високого рівня – таких, що постійно робляться фірмами-розробниками систем ЧПУ, поки що не призвели до переходу на новий стандарт програмування. Але такий перехід неминуче відбудеться в майбутньому.

Слід зауважити, що світові лідери в області систем ЧПУ розвивають свої системи впродовж десятків років. А це означає, що в основу їх систем були закладені концепції, архітектурні рішення і базові принципи, відповідні рівню розвитку комп’ютерної техніки і програмних систем та технологічному рівню взагалі десятирічної і навіть більшої давності. Через велику інерційність і трудомісткість процесу початкової розробки і подальшого розвитку системи ЧПУ, що відносяться до класу «*HighEnd*» їх розробники навіть при вкладенні значних коштів та технічних засобів не встигають за швидкоплинними змінами на ринку комплектуючих виробів і інструментальних засобів. В результаті, якщо був здійснений правильний вибраної стратегії і правильно зробленого прогнозу перспектив розвитку обчислювальної техніки, виробник, що займає друге місце у розвитку, має всі можливості випередити першого.

Розроблена гнучка модульна архітектура, що дозволяє в своїй ніші диференціювати розробку СЧПУ, позбавити розробника ПЗ для СЧПУ пробеми реального часу та вивести розвиток СЧПУ на вів рівень за рахунок привабливості для ентузіастів абстрактності інтерфейсів міжмодульної комунікації.

Висновки за розділом

В третьому розділі було запропоновано вирішення проблеми управління систем реального масштабу часу, зокрема в верстатах ЧПУ, та особливості її реалізації на базі операційних систем Windows та Linux.

Для досягнення цієї мети було запропоновано реалізацію модульної взаємодії між компонентами системи ЧПУ та управління такою системою з комп'ютера. Цей підхід дозволить забезпечити ефективне управління верстатами ЧПУ, забезпечуючи точність та швидкість обробки процесів.

Крім того, в роботі було розглянуто можливість керування групами верстатів ЧПУ за допомогою комп'ютерної мережі. Ця пропозиція дає можливість координувати роботу декількох верстатів одночасно, забезпечуючи синхронізацію та оптимальне розподілення завдань між ними.

В цілому, запропоновані рішення в третьому розділі спрямовані на поліпшення управління системами ЧПУ, покращуючи їх продуктивність, ефективність та можливості координації.

ВИСНОВКИ

У магістерській дипломній роботі було проведено детальні дослідження щодо сучасних тенденцій у розробці систем числового програмування (ЧПУ), зокрема їх різновидів і найбільш перспективних архітектур.

В рамках досліджень було виявлено, що розробка ЧПУ-систем стикається з рядом проблем, які часто виникають під час процесу розробки. Однією з таких проблем є неефективність традиційної архітектури ЧПУ, що обмежує можливості системи та призводить до складнощів у реалізації нових функціональних можливостей.

В результаті досліджень була запропонована концепція гнучкої модульної архітектури ЧПУ, яка поєднує переваги основних архітектур, враховуючи вимоги сучасного програмного забезпечення. Ця концепція дозволяє створювати системи числового програмування, які можуть легко адаптуватись до різноманітних вимог та забезпечувати гнучкість у роботі з ЧПУ-машинами.

Основною ідеєю гнучкої модульної архітектури є розділення функціональності ЧПУ-системи на окремі модулі, які можуть бути розроблені незалежно один від одного. Кожен модуль виконує конкретні функції, такі як обробка puch-кодів, керування рухом осей, комунікація з оператором тощо. Модулі можуть бути додавані, видалятись або модифікуватись без необхідності переробки всієї системи, що значно спрощує процес розробки та покращує масштабованість системи.

Для практичної реалізації концепції була використана платформа Arduino, яка є широко поширеною серед розробників систем ЧПУ. В рамках роботи був створений прототип пристрою ЧПУ, що базується на гнучкій модульній архітектурі. Прототип успішно проходив тестування та продемонстрував свою ефективність у керуванні рухом осей та обробці команд G-кодів.

У дипломній роботі були представлені також структурні та принципові схеми модулів гнучкої модульної архітектури ЧПУ, що детально описують взаємодію між модулями та основні компоненти кожного модуля. Крім того, був розроблений набір

підтримуваних системою G-кодів, який забезпечує сумісність пристрою ЧПУ з різними типами ЧПУ-машин.

Одним із ключових аспектів розробленої гнучкої модульної архітектури є можливість розширення функціональних можливостей системи без необхідності повного переписування коду. Завдяки модульній структурі, нові функції можуть бути легко додані шляхом розробки нових модулів та їх інтеграції з існуючими. Це забезпечує велику гнучкість у виборі функцій та розширення можливостей ЧПУ-системи в майбутньому.

Однак, в ході досліджень також були виявлені певні виклики та обмеження. Наприклад, для забезпечення ефективної роботи системи необхідно добре продумане керування модулями та взаємодія між ними. Крім того, важливо приділити увагу безпеці данних та захисту системи від потенційних загроз.

У подальших дослідженнях можна розглянути можливості оптимізації гнучкої модульної архітектури ЧПУ, включаючи вдосконалення взаємодії між модулями та розробку нових функціональних можливостей. Також можна провести додаткові експерименти та тестування прототипу на різних типах ЧПУ-машин для оцінки його ефективності та сумісності.

Загалом, дослідження, проведені в магістерській дипломній роботі, виявилися цінним внеском у розвиток систем числового програмування (ЧПУ). Розроблена гнучка модульна архітектура ЧПУ та прототип пристрою, що базується на цій архітектурі, відкривають нові можливості для ефективного керування ЧПУ-машинами і сприяють подальшому розвитку цієї галузі.

Застосування гнучкої модульної архітектури в програмному забезпеченні для ЧПУ-систем демонструє низькі витрати розробки, що є важливим фактором для багатьох підприємств. Будучи ефективною і доступною технологією, ця нова архітектура пропонує універсальний підхід до оптимізації процесів ЧПУ та забезпечення точного керування.

Додатково, одна з ідей, запропонованих у дипломній роботі, полягає в мережевому керуванні групами верстатів з ЧПУ. Це концептуально новий рівень автоматизації виробництва, який може бути реалізований за помірну ціну. Завдяки

такому підходу, декілька верстатів можуть бути керовані через мережу в одиничному режимі, що веде до збільшення продуктивності та зниження витрат на робочу силу.

Важливо підкреслити, що розроблений пристрій ЧПУ може бути використаний з будь-якими верстатами, навіть тими, які раніше не передбачали встановлення системи числового програмування. Це означає, що обладнання, яке вже використовується на підприємстві, може бути покращено, підвищуючи його функціональність та продуктивність.

Такі інновації у галузі ЧПУ-систем мають потенціал в суттєвому поліпшенні виробничих процесів, забезпечуючи оптимальне використання ресурсів та знижуючи загальні витрати для підприємства. Додаткові дослідження і розвиток цих технологій можуть принести значний вплив на промисловість та виробництво, сприяючи створенню більш конкурентоспроможного та ефективного середовища.

Гнучка модульна архітектура представляє собою практичну реалізацію напрямку розвитку числово-програмованого керування (ЧПК), який передбачає перетворення пристроїв ЧПК на звичайну комп'ютерну периферію. Таке перетворення стає можливим завдяки росту технологій і збільшенню обчислювальної потужності комп'ютерів, що дозволяє зробити системи ЧПК більш доступними і ефективними.

Однією з ключових переваг гнучкої модульної архітектури ЧПК є її універсальність. Принципи, що лежать в основі цієї архітектури, можуть бути застосовані до різних типів верстатів і машин з ЧПК, а також до різних галузей промисловості. Такий підхід дозволяє забезпечити гнучкість і адаптованість системи до конкретних вимог і потреб користувача.

Додатковою перевагою є зниження вартості розробки програмного забезпечення для ЧПК-систем. Завдяки модульній архітектурі можна створювати загальнопризначені модулі, які можуть бути повторно використані для різних застосувань. Такий підхід дозволяє економити час і ресурси при розробці нових систем ЧПК і сприяє швидкому впровадженню нових технологій.

Окрім того, гнучка модульна архітектура ЧПК дозволяє вирішувати проблему реального часу. Завдяки оптимізації алгоритмів та використанню потужних

пристроїв, можливо досягти високої швидкості обробки команд і точності рухів. Це особливо важливо для завдань, де необхідна миттєва реакція на зміни у виробничому середовищі або при виконанні складних операцій.

В цілому, розроблення гнучкої модульної архітектури ЧПК відкриває нові перспективи у сфері автоматизації виробництва. Вона дає можливість покращити ефективність, точність та гнучкість ЧПК-систем, зменшуючи витрати і збільшуючи їх доступність. Застосування такої архітектури може мати значний вплив на промислову сферу і сприяти подальшому розвитку автоматизованих виробничих процесів. Гнучка модульна архітектура ЧПК дозволяє забезпечити більш точне керування верстатами і машинами, що призводить до зниження відхилень і помилок у виробництві.

Окрім цього, гнучка модульна архітектура ЧПК сприяє полегшенню процесу модернізації та оновлення обладнання. Замість необхідності повного заміщення старої системи ЧПК новою, можна замінювати окремі модулі, що значно знижує витрати і час на оновлення. Такий підхід також сприяє збереженню існуючих інфраструктурних рішень і запобігає зайвим витратам.

Застосування гнучкої модульної архітектури ЧПК дозволяє підвищити безпеку роботи на верстатах і машинах. Завдяки узгодженому інтерфейсу між пристроями ЧПК та комп'ютером, можна контролювати та встановлювати обмеження на роботу обладнання, що запобігає потенційним аваріям та небажаним ситуаціям.

Гнучка модульна архітектура ЧПК також сприяє ефективному управлінню ресурсами. Завдяки використанню загальнопризначених модулів і стандартизованих інтерфейсів, можна оптимізувати використання обчислювальних ресурсів і забезпечити більш економну роботу системи. Це дозволяє збільшити продуктивність та знизити енергоспоживання.

У підсумку, гнучка модульна архітектура представляє собою перспективний напрямок розвитку ЧПК, що дозволяє досягти більшої доступності, ефективності, точності та гнучкості в автоматизованому виробництві. Вона має потенціал змінити промислову сферу, оптимізувати роботу верстатів і машин, а також сприяти подальшому розвитку технологій та покращенню якості продукції.

СПИСОК БІБЛІОГРАФІЧНИХ ПОСИЛАНЬ ВИКОРИСТАНХ ДЖЕРЕЛ

1. Аверченков В.І, Жолобов А.А, Мрочек Ж.А, Аверченков А.В, Терехов М.В, Левкіна Л.Б. Автоматизація підготовки управляючих програмч для верстатів з ЧПУ: Навч. посібник. 2-е вид. – К.: «Знання», 2018. – 250 с.
2. Сосонкін В.Л., Мартинов Г.М. Системи числового програмного управління: Навч. посібник. – К.: «Либідь», 2019. – 296 с.
3. Соломенцев Ю.М., Сосонкін В.Л., Мартинов Г.М. Побудова персональных систем ЧПУ (*PCNC*) за принципом відткритих систем.-Інформаційі технології та обчислювальні системи. 1997, №3, с. 68-75.
4. Сосонкін В.Л., Мартинов Г.М. Принципи побудови систем ЧПУ з відкритою архітектурою. – Прилади та системи управління. 1996, №8, с. 18-21.
5. Сосонкін В.Л. Концепція системи ЧПУ на основі персонального комп'ютера (*PCNC*). – Верстати та інструмент. 1990, №11, с. 9-14.
6. *Pritschow G., Spur G., Weck M. Komponenten und Schnittstellen fur offene Steuerungssysteme.Munchen; Wien: Hanser*, 1996. – 216 с.
7. Сосонкін В.Л. Задачі числового програмного управління та їх архитектурна реализація. – Верстати та інструмент. 1988, №10, с. 39-40.
8. Сосонкін В.Л., Мартинов Г.М. Концепція геометричного *ISO* - процесора для систем ЧПУ. Верстати та інструмент 1994, №7, с. 17-20.
9. Щербаков А. Протоколи прикладного рівня *CAN*-мереж. Сучасні технологиї автоматизації, 1999. №3. С 6-15.
10. Гупта А., Каро Р. *FONDATION FIELDBUS* або *PROFIBUS-RA*: выбір промислової мереі для автоматизації технологічних процесів. Сучасні технології автоматизації, 1999. №3. с. 16-21.
11. *CAD/CAM computer-aided design and manufacturing / Mikell P. Groover, Emory W. Zimmers, Jr. - Rrentice Hall of India*, 2018. – 226 p.

12.Мартинов Г.М., Сосонкін В.Л. Концепція числового програмного управління мехатронними системами: реализація геометричної задачі // Мехатроніка, 2001, №1. с. 9-15.

13.Сосонкін В.Л., Мартинов Г.М. Концепція геометричного 1БО-процесора для систем ЧПУ // СТИН, 1994, №7, с. 17-20.

14.Ловигін А. А., Васильєв А. В. Сучасний верстат з ЧПУ та *CAD/CAM* система. – К.: «Вища школа», 2017. – 350 с.

15. Босинзон М. А. Сучасні системи ЧПУ та їх експлутация. Підручник для нач. проф. освіти. Вид. *Academia*, 2019 р. *ISBN* 978-5-7695-6060-6

16. *Michael Simpson, Building the KRMx01 CNC*: *The Illustrated Guide to Building a High Precision CNC. Kronos Robotics,* 2020. – 226 р.

17. *E Hess, The CNC Cookbook: An Introduction to the Creation and Operation of Computer Controlled Mills, Router Tables, Lathes, and More, Scited Publications* 2019. – 310 р.

18. Андреєв А.Г., Григорьев С. Н. Побудова комп'ютерных систем програмного управління мехатронними пристролями по модульному принципу // Мехатроніка, автоматизація, управління. 2005. №10.

19.ГОСТ 19.701-90. Схеми алгоритмів, програм, даних і систем. Умовні позначення та правила виконання.

20.ГОСТ 20999-83 Пристрої числового програмного управління для металообробного устатування. Кодування інформації управляючих програм. – К.: Видавництво стандартів, 1983.

21. Патент № 1383301, від 23.03.1988, «Функціональний інтерполятор».

22.Патент № *SU* 344415 от 01.01.1972 «Цифровий лінійній інтерполятор».

ДОДАТОК А

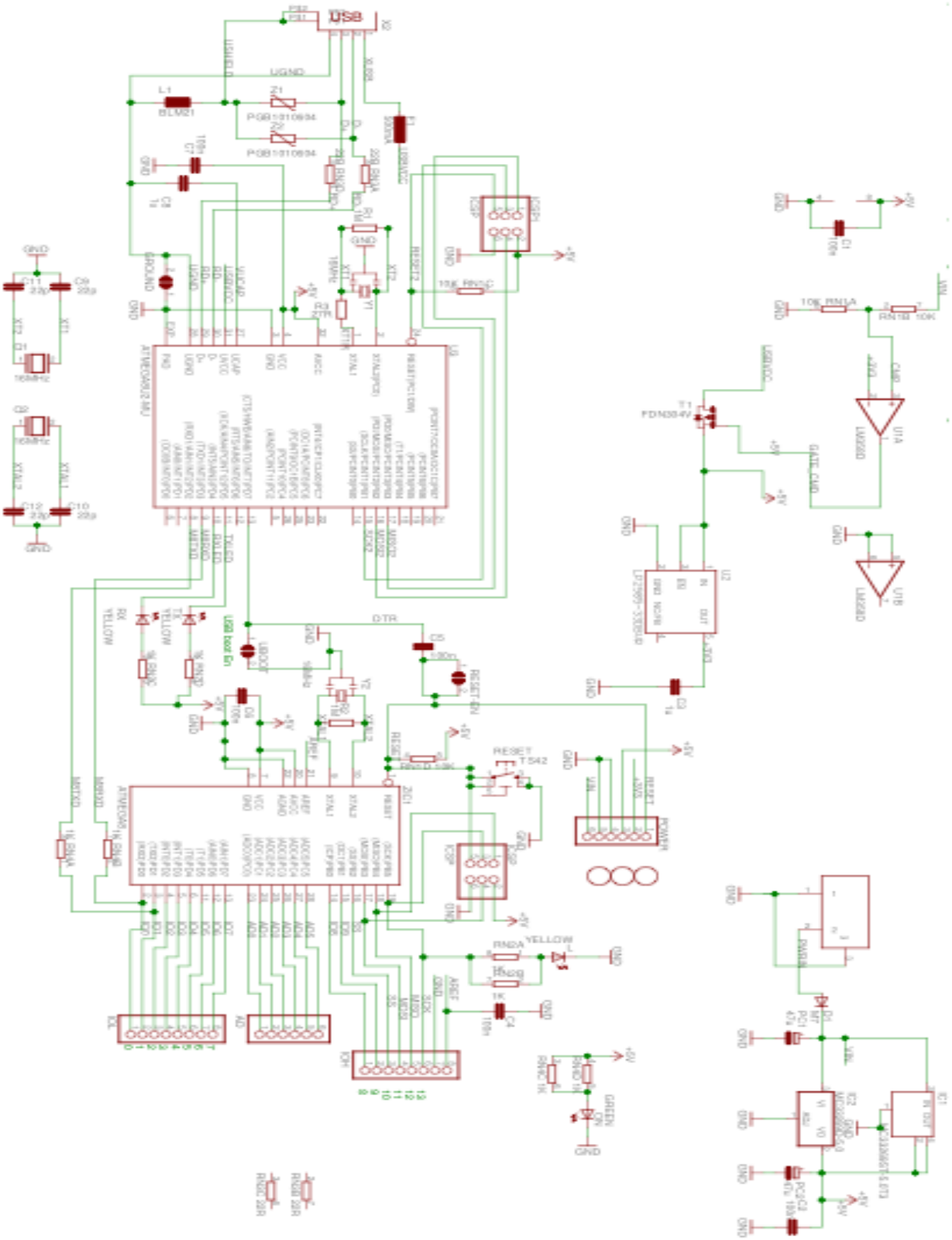
Список параметрівдля*G*-кодів*ISO* 7-*bit*

Код	Опис	Приклад
X	Координата точки траекторії по осі *X*	*G0 X100 Y0 Z0*
Y	Координата точки траекторії по осі *Y*	*G0 X0 Y100 Z0*
Z	Координа таточки траекторії по осі *Z*	*G0 X0 Y0 Z100*
P	Параметр команди	*G04 P101*
F	Швидкість робочої подачи	*G1 G91 X10 F100*
S	Швидкість обертання шпинделю	*S3000 M3*
R	Параметр стандартного цикла або радіус дуги (розширення стандарту)	*G81 R1 0 R2 −10 F50 или G2 G91 X12.5 R12.5*
D	Параметр корекції обраного інструмента	*G1 G41 D1 X10. F150.*
L	Число викликів підпрограми	*M98 L82 P10 или G65 L82 P10 X_Y_R_*
I,J,K	Параметри дуги при круговій інтерполяції	*G03 X10 Y10 I0 J0 F10*
L	Виклик підпрограми з данною міткою	*L12*

ДОДАТОК Б

Пристрій ЧПУ на базі гнучкої модульної архітектури.

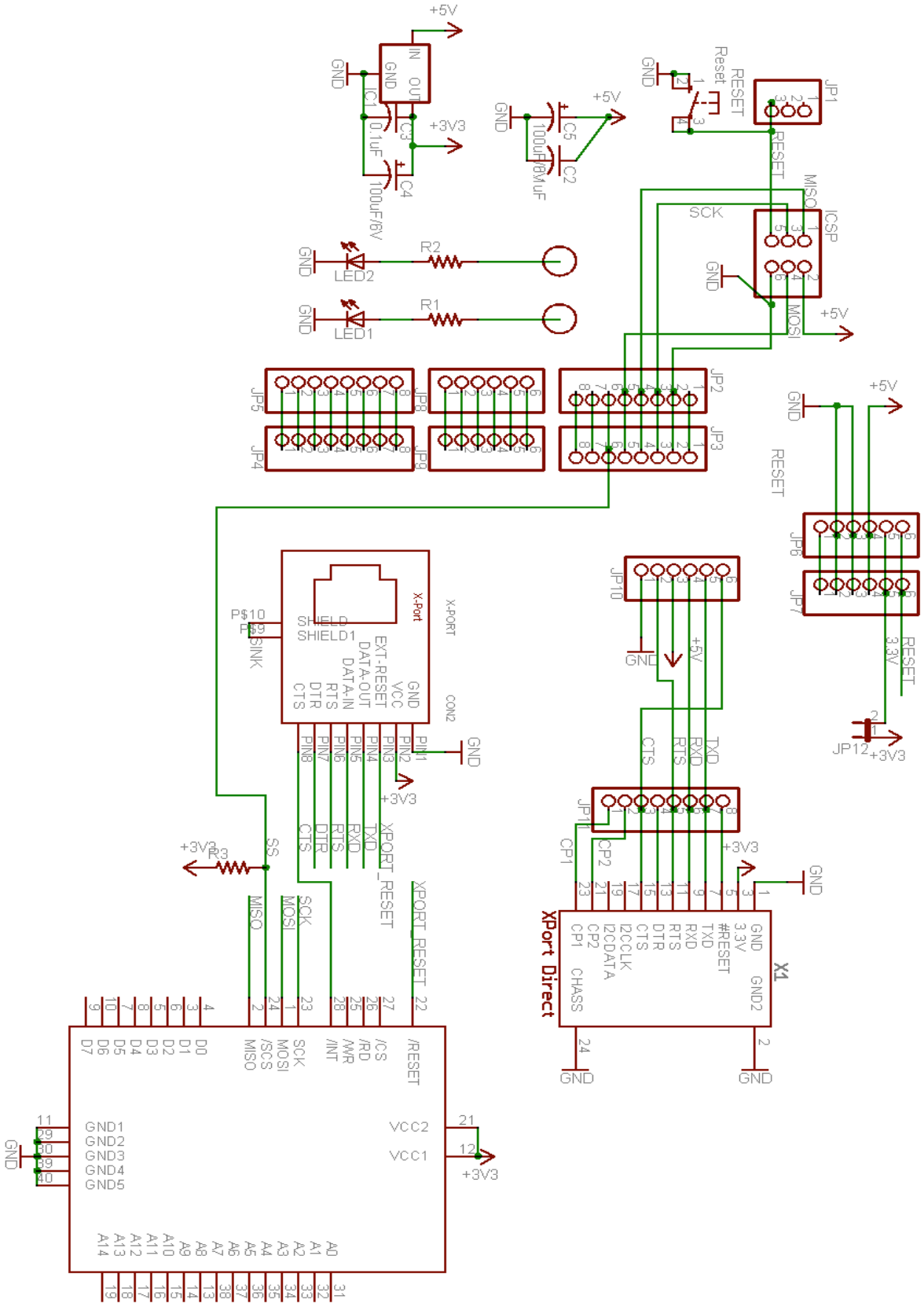
Головний керуючий модуль. Принципова схема



ДОДАТОК В

Пристрій ЧПУ на базі гнучкої модульної архітектури.

Модуль *ETHERNET*. Принципова схема



ДОДАТОК Г
Програмне забезпечення головного керуючого модуля. Код.

```
/*

G-коды должны быть преобразованы к следующему формату на стороне клиентского ПО:
  .Все измерения - в шагах (ПК преобразовывает в/из систем СИ или BES)
  .Координаты пересчитываются исходя из того, что началом и концом координат являются концевики
  .нумерация строк и неподдерживаемые g-коды должны быть отброшены парсером клиентского ПО.
  .все G, M и T коды забиваются с 2 цифрами
  .all S, F и P коды забиваются с 4 цифрами
  .all X, Y, Z, I, J, и k коды забиваются с 7 цифрами
    .Пустое пространство заполняется нулями
  .Координаты абсолютные
  .Скорость подачи определяется на стороне клиента (зависит от механики передаточного механизма)

*/
//defines
#include <SPI.h>
#include <Ethernet.h>
// MAC и IP адрес задается произвольно в зависимости от необходимости и топологии сети:
byte mac[] = { 0xDE, 0xAD, 0xBE, 0xEF, 0xFE, 0xED };
IPAddress ip( 192,168,1,102 );
IPAddress gateway( 192,168,1,1 );
IPAddress subnet( 255,255,255,0 );
//IPAddress dns( 192,168,1,1 );

// Initialize the Ethernet server library
// with the IP address and port you want to use
// (port 80 is default for HTTP):
EthernetServer server(84);


//define port pins

const byte XDir=2;
const byte XStep=5;
const byte YDir=3;
const byte YStep=6;
const byte ZDir=4;
const byte ZStep=7;
```

```
const byte spindleSpeedout=10;
const byte spindleDir=8;
const byte spindleEn=9;
const byte XLim=11;
const byte YLim=12;
const byte ZLim=13;

//define control bit positions for the movexyz(byte) function

const byte XM=0;
const byte XD=1;
const byte YM=2;
const byte YD=3;
const byte ZM=4;
const byte ZD=5;
const int baudrate=9600;
boolean go=false;
boolean spindleOn=false;
boolean spindleRev=false;
int spindleSpeed=0;
unsigned long XPos=0;
unsigned long YPos=0;
unsigned long ZPos=0;
unsigned long IPos=0;
unsigned long JPos=0;
unsigned long KPos=0;
unsigned long P=0;
byte moveMode=0;//первые 4 G-кода; 0=быстрое перемещение, 1=перемещение с заданой скоростью,
//2=дуга по часовой стрелке, 3дуга против часовой стрелки
// 4 , 9 do nout.
char plane='Z'; //plane defined by perpendicular axis
byte tool=1;
int feedRate=0;
boolean tooFast=false;
byte MCode=100;
//обьявление переменных
boolean goNew=false;
boolean spindleOnNew=false;
boolean spindleRevNew=false;
int spindleSpeedNew=0;
unsigned long XPosNew=0;
unsigned long YPosNew=0;
unsigned long ZPosNew=0;
unsigned long IPosNew=0;
```

```
unsigned long JPosNew=0;
unsigned long KPosNew=0;
unsigned long PNew=0;
byte moveModeNew=9;
char planeNew='Z';
byte toolNew=1;
int feedRateNew=0;
unsigned long stepTime;
unsigned long lastStepTime;

//some global arrays for simple pointerless passing of numbers from codes
char twoDigit[2];
char fourDigit[4];
char sevenDigit[7];
//begin code
//*****************************************************************************
**

void setup()
{
  initserial();
  initports();
  homefind();
  Serial.begin(9600);

  // set SPI SS pins on w5100 and SD
  pinMode(10,OUTPUT);
  digitalWrite(10,HIGH);
  pinMode(4,OUTPUT);
  digitalWrite(4,HIGH);

  // start the SD interface here if you want.
  // Add the SD.h library above
  // SD.begin(4);

  // Стартуем ethernet интерфейс, задаем параметры, стартуем HTTP сервер:
  Ethernet.begin(mac, ip, dns, gateway, subnet);
  // disable w5100 SPI so SD SPI can work with it
  digitalWrite(10,HIGH);
  delay(2000);
  server.begin();

  Serial.println("setup finished");

//  Serial.println("Ready");
```

```
//  Serial.flush();
}



void initports()
{
  pinMode(XDir , OUTPUT);
  pinMode(YDir , OUTPUT);
  pinMode(ZDir , OUTPUT);
  pinMode(XStep , OUTPUT);
  pinMode(YStep , OUTPUT);
  pinMode(ZStep , OUTPUT);
  pinMode(XDir , OUTPUT);
  pinMode(spindleSpeedout , OUTPUT);
  pinMode(spindleDir , OUTPUT);
  pinMode(XLim , INPUT);
  pinMode(YLim , INPUT);
  pinMode(ZLim , INPUT);
}

void initserial()
{
  Serial.begin(baudrate);
//  Serial.println("Sarting...");
}

void homefind()
{
  int i=0;
  //for X
  digitalWrite(XDir, HIGH);
  while (i<500)
  {
    digitalWrite(XStep, HIGH);
     delayMicroseconds(500) ;
    digitalWrite(XStep, LOW);
     delayMicroseconds(500) ;
    i++;
  }
  digitalWrite(XDir, LOW);

  while (digitalRead(XLim)==LOW)
  {
    digitalWrite(XStep, HIGH);
```

```
   delayMicroseconds(500) ;
  digitalWrite(XStep, LOW);
   delayMicroseconds(500) ;
}

//for Y
i=0;

digitalWrite(YDir, HIGH);
while (i<500)
{
  digitalWrite(YStep, HIGH);
  delayMicroseconds(500) ;
  digitalWrite(YStep, LOW);
   delayMicroseconds(500) ;
  i++;
}
digitalWrite(YDir, LOW);

while (digitalRead(YLim)==LOW)
{
  digitalWrite(YStep, HIGH);
   delayMicroseconds(500) ;
  digitalWrite(YStep, LOW);
   delayMicroseconds(500) ;
}
//for Z
i=0;

digitalWrite(ZDir, HIGH);
while (i<500)
{
  digitalWrite(ZStep, HIGH);
  delayMicroseconds(500) ;
  digitalWrite(ZStep, LOW);
   delayMicroseconds(500) ;
  i++;
}
digitalWrite(ZDir, LOW);

while (digitalRead(ZLim)==LOW)
{
  digitalWrite(ZStep, HIGH);
   delayMicroseconds(500) ;
  digitalWrite(ZStep, LOW);
```

```
    delayMicroseconds(500) ;
  }

}

//*************************************************************
void loop()
{
  Serial.println("> \n");
  getline();
  executeline();
}

//*************************************************************
void getline()
{
  char c;
  moveMode=9;
  MCode=100;
  while(1)
  {
    while (Serial.available() <=0);    //await data stream
    c=Serial.read();                   //get me a character
    if (c==10 || c==13 || c== '*')
    break;
    switch (c)
     {
      case 'M':                        //is M code (2 digit number allowed)
        while (Serial.available() <=0);
        twoDigit[0]=Serial.read();       //get first char after 'M' this will be a number
        while (Serial.available() <=0);
        if (Serial.peek()>= '0' && Serial.peek()<= '9')
        twoDigit[1]=Serial.read();       //get second char after 'M' this will be a number


        switch (twoToByte())
        {
          case 0:                  //Prog stop
            MCode=0;
            break;
          case 1:                  //Prog stop otional
            MCode=1;
            break;
          case 2:                  //Prog end
            MCode=2;
```

```
        break;
      case 3:                 //spindle on CW
        spindleOnNew=true;
        spindleRevNew=false;
        break;
      case 4:                 //spindle on CCW
        spindleOnNew=true;
        spindleRevNew=true;
        break;
      case 5:                 //spindle off
        spindleOnNew=true;
        break;
      case 6:                  //Tool change
        MCode=6;
        break;
      case 7:
        MCode=7;           //Mist on
        break;
      case 8:
         MCode=8;          //Flood on
        break;
      case 9:
         MCode=9;          //coolant off
        break;

        break;
    }
    break;

  case 'G':
    while (Serial.available() <=0);
    twoDigit[0]=Serial.read();       //Получение первого символа после 'G' цифра или
0
    while (Serial.available() <=0);
    twoDigit[1]=Serial.read();       //Получение Второго символа после 'G' цифра или
0
    switch (twoToByte())
    {
      case 0:                 //Rapid
        moveModeNew=0;
        break;
      case 1:                 //move
        moveModeNew=1;
        break;
      case 2:                 //arc CW
```

```
        moveModeNew=2;
        break;
      case 3:                  //arc CCW
        moveModeNew=3;
        break;
      case 4:                  //DWELL
        moveModeNew=4;
        break;
      case 17:                  //XY Plane
        planeNew='Z';
        break;
      case 18:                  //XZ Plane
        planeNew='Y';
        break;
      case 19:                  //YZ Plane
        planeNew='X';
        break;
    }
    break;


  case 'T':
    while (Serial.available() <=0);
    twoDigit[0]=Serial.read();      //Получение первого символа после T - цифра или 0
    while (Serial.available() <=0);
    twoDigit[1]=Serial.read();
    toolNew= twoToByte();
    break;

  case 'S':
    while (Serial.available() <=0);
    fourDigit[0]=Serial.read();       //Получение первого символа после 'S' цифра или
0
    while (Serial.available() <=0);
    fourDigit[1]=Serial.read();      //Получение второго символа после 'S' цифра или
0
    while (Serial.available() <=0);
    fourDigit[2]=Serial.read();       //Получение третьего символа после 'S' - цифра
или 0
    while (Serial.available() <=0);
    fourDigit[3]=Serial.read();      //Получение четвертого символа после 'S' цифра
или 0

    spindleSpeedNew=fourDigitToInt();
    break;
```

```
      case 'F':
       while (Serial.available() <=0);
       fourDigit[0]=Serial.read();       //Получение первого символа после 'S' цифра или
0
       while (Serial.available() <=0);
       fourDigit[1]=Serial.read();      //Получение второго символа после 'F' цифра или
0
       while (Serial.available() <=0);
       fourDigit[2]=Serial.read();       //Получение третьего символа после 'F' - цифра
или 0
       while (Serial.available() <=0);
       fourDigit[3]=Serial.read();      //Получение четвертого символа после 'F' цифра
или 0

       feedRateNew=fourDigitToInt();
       break;

      case 'P':
       while (Serial.available() <=0);
       fourDigit[0]=Serial.read();      //get first char after 'S' this will be a number
       while (Serial.available() <=0);
       fourDigit[1]=Serial.read();      //get second char after 'S' this will be a number
       while (Serial.available() <=0);
       fourDigit[2]=Serial.read();      //get third char after 'S' this will be a number
       while (Serial.available() <=0);
       fourDigit[3]=Serial.read();      //get fourth char after 'S' this will be a number

       PNew=fourDigitToInt();
       break;

      case 'X':
       for (int i=0; i<7; i++)
       {
         while (Serial.available() <=0);
         sevenDigit[i]=Serial.read();
       }
        XPosNew=sevenDigitTolong();
       break;

      case 'Y':
       for (int i=0; i<7; i++)
       {
         while (Serial.available() <=0);
         sevenDigit[i]=Serial.read();
```

```
        }
         YPosNew=sevenDigitTolong();
        break;

      case 'Z':
        for (int i=0; i<7; i++)
        {
          while (Serial.available() <=0);
          sevenDigit[i]=Serial.read();
        }
         ZPosNew=sevenDigitTolong();
        break;

      case 'I':
        for (int i=0; i<7; i++)
        {
          while (Serial.available() <=0);
          sevenDigit[i]=Serial.read();
        }
         IPosNew=sevenDigitTolong();
        break;

      case 'J':
        for (int i=0; i<7; i++)
        {
          while (Serial.available() <=0);
          sevenDigit[i]=Serial.read();
        }
         JPosNew=sevenDigitTolong();
        break;

      case 'K':
        for (int i=0; i<7; i++)
        {
          while (Serial.available() <=0);
          sevenDigit[i]=Serial.read();
        }
         KPosNew=sevenDigitTolong();
        break;
    }
  }
}
byte twoToByte()
{
  return(((twoDigit[0]-48)*10)+(twoDigit[1]-48));
```

```
}

int fourDigitToInt()
{
  return(((fourDigit[0]-48)*1000)+((fourDigit[1]-48)*100)+((fourDigit[2]-
48)*10)+(fourDigit[3]-48));
}

unsigned long  sevenDigitTolong()
{
  return(((sevenDigit[0]-48)*1000000)+((sevenDigit[1]-48)*100000)+((sevenDigit[2]-
48)*10000)+((sevenDigit[3]-48)*1000)+((sevenDigit[4]-48)*100)+((sevenDigit[5]-
48)*10)+(sevenDigit[6]-48));
}
void  executeline()
{
  go=goNew;
  spindleRev=spindleRevNew;
  spindleOn=spindleOnNew;
  spindleSpeed=spindleSpeedNew;
  feedRate=feedRateNew;
  plane=planeNew;
    unsigned long Xhold=XPos;
    unsigned long Yhold=YPos;
    unsigned long Zhold=ZPos;
    unsigned long XNhold=XPosNew;
    unsigned long YNhold=YPosNew;
    unsigned long ZNhold=ZPosNew;
  switch (MCode)
  {
    case 1:
      Serial.flush();
      Serial.println ("Stoped. type '$' to continue");
      Serial.println ("(note recieve buffer hs been flushed)");
      Serial.println (">");
      do
      {
        while (Serial.available() <=0);
      } while (Serial.read() != '$');
      break;

    case 6:
      ZPosNew=0;
      linear(0);
      XPosNew=0;
```

```
  YPosNew=0;
  linear(0);
  Serial.flush();
  Serial.println ("Stoped for you to change tool. type '$' to continue");
  Serial.println ("(note recieve buffer hs been flushed)");
  Serial.println (">");
  do
  {
    while (Serial.available() <=0);
  } while (Serial.read() != '$');
  tool=toolNew;
  XPosNew=Xhold;
  YPosNew=Yhold;
  linear(0);
  ZPosNew=Zhold;
  linear(0);
  XPosNew=XNhold;
  YPosNew=YNhold;
  ZPosNew=ZNhold;
  break;

case 7:
  Serial.flush();
  Serial.println ("Stoped for you to turn on mist coolant. type '$' to continue");
  Serial.println ("(note recieve buffer hs been flushed)");
  Serial.println (">");
  do
  {
    while (Serial.available() <=0);
  } while (Serial.read() != '$');
  break;
case 8:
  Serial.flush();
  Serial.println ("Stoped for you to turn on flood coolant. type '$' to continue");
  Serial.println ("(note recieve buffer hs been flushed)");
  Serial.println (">");
  do
  {
    while (Serial.available() <=0);
  } while (Serial.read() != '$');
  break;
case 9:
  Serial.flush();
  Serial.println ("Stoped for you to turn off coolant. type '$' to continue");
  Serial.println ("(note recieve buffer hs been flushed)");
```

```
    Serial.println (">");
    do
    {
      while (Serial.available() <=0);
    } while (Serial.read() != '$');
    break;
  }
  P=PNew;
  IPos=IPosNew;
  JPos=JPosNew;
  KPos=KPosNew;
moveMode=moveModeNew;
switch (moveMode)
  {
    case 0:
      linear(0);
      break;
    case 1:
      linear(feedRate);
      break;
    case 2:
      arc(1);
      break;
    case 3:
      arc(0);
      break;
    case 4:
      P=PNew;
      delay(P);
    case 5:
      break;
  }
}

//Интерполяторы.
//**********************************************************************
**********
/*
Линейный интерполятор:
Принимает координаты в формате:
X,Y (XPos,XPos,ZPos) исходное положение и  (XPosNew,XPosNew,ZPosNew) новое
положение. Для 2-х координат Z=0.
*/
void linear(int feed)
{
```

```
  int feedHold=feedRate;
  feedRate=feed;
  long  XDif=XPosNew-XPos;
  long  YDif=YPosNew-YPos;
  long  ZDif=ZPosNew-ZPos;
  unsigned long  absXDif=abs(XDif);
  unsigned long  absYDif=abs(YDif);
  unsigned long  absZDif=abs(ZDif);

  boolean xSign = (XDif>0);
  boolean ySign = (YDif>0);
  boolean zSign = (ZDif>0);
  byte data=0;
if (XDif==0 && YDif==0 && ZDif==0)
  {
    feedRate=feedHold;
    return;                          //0 Dinesional line (otherwise known as stay put!)
//    Serial.println("0D line");
  }


  if ((XDif==0 && YDif==0) || (XDif==0 && ZDif==0) || (ZDif==0 && YDif==0))
//must be a 1D line
  {
//    Serial.println("1D line");
    unsigned long *aPos;
    unsigned long aPosNew;
    byte aM;
    byte aD;

    if (XDif==0 && YDif==0)      //must be along Z axis
    {
      aPos=&ZPos;
      aPosNew=ZPosNew;
      aM=ZM;
      aD=ZD;
    }

     if (XDif==0 && ZDif==0)      //must be along Y axis
    {
      aPos=&YPos;
      aPosNew=YPosNew;
      aM=YM;
      aD=YD;
```

```
  }

  if (ZDif==0 && YDif==0)      //must be along X axis
  {
    aPos=&XPos;
    aPosNew=XPosNew;
    aM=XM;
    aD=XD;
  }

  bitWrite(data,aM,1);
  if (*aPos<aPosNew)
  {
    bitWrite(data,aD,1);
  }
  else
  {
    bitWrite(data,aD,0);
  }

  while (*aPos != aPosNew)
  {
    movexyz(data);

  }
  feedRate=feedHold;
  return;
 }


 if (XDif==0 || YDif==0 || ZDif==0)          // must be a 2D line
 {
//   Serial.println("2D line");
  unsigned long *aPos;
  unsigned long aPosNew;
  byte aM;
  byte aD;
  unsigned long *bPos;
  unsigned long bPosNew;
  byte bM;
  byte bD;

  unsigned long deltaA;
  unsigned long deltaB;
```

```
unsigned long twoDeltaA;
unsigned long twoDeltaB;

unsigned long twoDeltaAAcumulatedError;
unsigned long twoDeltaBAcumulatedError;
if (ZDif==0)
{
  aPos=&XPos;
  bPos=&YPos;
  aPosNew=XPosNew;
  bPosNew=YPosNew;
  aM=XM;
  bM=YM;

  aD=XD;
  bD=YD;

  deltaA=absXDif;
  deltaB=absYDif;

  twoDeltaA=2*deltaA;
  twoDeltaB=2*deltaB;
}

if (XDif==0)                    //2D line in ZY Plane
{
  aPos=&ZPos;
  bPos=&YPos;
  aPosNew=ZPosNew;
  bPosNew=YPosNew;
  aM=ZM;
  bM=YM;

  aD=ZD;
  bD=YD;
  deltaA=absZDif;
  deltaB=absYDif;

  twoDeltaA=2*deltaA;
  twoDeltaB=2*deltaB;
}

if (YDif==0)
{
  aPos=&ZPos;
```

```
  bPos=&XPos;
  aPosNew=ZPosNew;
  bPosNew=XPosNew;
  aM=ZM;
  bM=XM;

  aD=ZD;
  bD=XD;
  deltaA=absZDif;
  deltaB=absXDif;

  twoDeltaA=2*deltaA;
  twoDeltaB=2*deltaB;
}

if (aPosNew> *aPos)
bitWrite(data,aD,1);
if (bPosNew> *bPos)
bitWrite(data,bD,1);

if (deltaB<= deltaA)
{
  bitWrite(data,aM,1);
  twoDeltaAAcumulatedError=0;
  while( *aPos!=aPosNew)
  {
    bitWrite(data,bM,0);
    twoDeltaAAcumulatedError=twoDeltaAAcumulatedError+twoDeltaB;
    if (twoDeltaAAcumulatedError > deltaA)
    {
      bitWrite(data,bM,1);
      twoDeltaAAcumulatedError=twoDeltaAAcumulatedError - twoDeltaA;
    }
    movexyz(data);
  }
}
else
{
  bitWrite(data,bM,1);
  twoDeltaBAcumulatedError=0;
  while( *bPos!=bPosNew)
  {
    bitWrite(data,aM,0);
    twoDeltaBAcumulatedError=twoDeltaBAcumulatedError+twoDeltaA;
    if (twoDeltaBAcumulatedError > deltaB)
```

```
      {
        bitWrite(data,aM,1);
        twoDeltaBAcumulatedError=twoDeltaBAcumulatedError - twoDeltaB;
      }
      movexyz(data);
    }
  }
  feedRate=feedHold;
  return;
}
if (XDif!=0 && YDif!=0 && ZDif!=0)
{
  unsigned long *aPos;
  unsigned long aPosNew;
  byte aM;
  byte aD;
  unsigned long *bPos;
  unsigned long bPosNew;
  byte bM;
  byte bD;
  unsigned long *cPos;
  unsigned long cPosNew;
  byte cM;
  byte cD;


  unsigned long deltaA;
  unsigned long deltaB;
  unsigned long deltaC;
  unsigned long twoDeltaA;
  unsigned long twoDeltaB;
  unsigned long twoDeltaC;

  unsigned long twoDeltaAAcumulatedError;
  unsigned long twoDeltaBAcumulatedError;
  unsigned long twoDeltaCAcumulatedError;

  if (absXDif>=absYDif && absXDif>=absZDif)

  {
    aPos=&XPos;
    bPos=&YPos;
    cPos=&ZPos;

    aPosNew=XPosNew;
```

```
  bPosNew=YPosNew;
  cPosNew=ZPosNew;

  aM=XM;
  bM=YM;
  cM=ZM;

  aD=XD;
  bD=YD;
  cD=ZD;

  deltaA=absXDif;
  deltaB=absYDif;
  deltaC=absZDif;
}

if (absYDif>=absXDif && absYDif>=absZDif)  // Y is dominant
{
  aPos=&YPos;
  bPos=&XPos;
  cPos=&ZPos;

  aPosNew=YPosNew;
  bPosNew=XPosNew;
  cPosNew=ZPosNew;

  aM=YM;
  bM=XM;
  cM=ZM;

  aD=YD;
  bD=XD;
  cD=ZD;

  deltaA=absYDif;
  deltaB=absXDif;
  deltaC=absZDif;
}
if (absZDif>=absXDif && absZDif>=absYDif)  // Z is dominant
{
  aPos=&ZPos;
  bPos=&XPos;
  cPos=&YPos;

  aPosNew=ZPosNew;
```

```
  bPosNew=XPosNew;
  cPosNew=YPosNew;

  aM=ZM;
  bM=XM;
  cM=YM;

  aD=ZD;
  bD=XD;
  cD=YD;

  deltaA=absZDif;
  deltaB=absXDif;
  deltaC=absYDif;
}

twoDeltaA=2*deltaA;
twoDeltaB=2*deltaB;
twoDeltaC=2*deltaC;

if (aPosNew> *aPos)
bitWrite(data,aD,1);
if (bPosNew> *bPos)
bitWrite(data,bD,1);
if (cPosNew> *cPos)
bitWrite(data,cD,1);

bitWrite(data,aM,1);
twoDeltaBAcumulatedError=0;
twoDeltaCAcumulatedError=0;
while( *aPos!=aPosNew)
{
  bitWrite(data,bM,0);
  bitWrite(data,cM,0);
  twoDeltaBAcumulatedError=twoDeltaBAcumulatedError+twoDeltaB;
  twoDeltaCAcumulatedError=twoDeltaCAcumulatedError+twoDeltaC;

  if (twoDeltaBAcumulatedError > deltaA)
  {
    bitWrite(data,bM,1);
    twoDeltaBAcumulatedError=twoDeltaBAcumulatedError - twoDeltaA;
  }
  if (twoDeltaCAcumulatedError > deltaA)
  {
    bitWrite(data,cM,1);
```

```
      twoDeltaCAcumulatedError=twoDeltaCAcumulatedError - twoDeltaA;
     }
     movexyz(data);
    }
    feedRate=feedHold;
    return;
   }
}
void arc (byte clock)
{
     /*
    Принимает данные с центром окружности I,J, началом дуги XPos,YPos и ее концом XPosNew ,YPosNew

*
  unsigned long *aPos;
  unsigned long aPosNew;
  byte aM;
  byte aD;
  unsigned long *bPos;
  unsigned long bPosNew;
  byte bM;
  byte bD;
  long dPos;
  long ePos;
  long aCur;
  long bCur;
  long aTarg;
  long bTarg;
  byte data=0;

  if (plane == 'Z')
  {
    aPos = &XPos;
    bPos = &YPos;

    aPosNew = XPosNew;
    bPosNew = YPosNew;

    dPos = IPos;
    ePos = JPos;
    aD=XD;
    bD=YD;
    aM=XM;
    bM=YM;
```

```
  }

  if (plane == 'Y')
  {
    aPos = &XPos;
    bPos = &ZPos;

    aPosNew = XPosNew;
    bPosNew = ZPosNew;

    dPos = IPos;
    ePos = KPos;
    aD=XD;
    bD=ZD;
    aM=XM;
    bM=ZM;
  }
  if (plane == 'X')
  {
    aPos = &YPos;
    bPos = &ZPos;
    aPosNew = YPosNew;
    bPosNew = ZPosNew;
    dPos = JPos;
    ePos = KPos;
    aD=YD;
    bD=ZD;
    aM=YM;
    bM=ZM;
  }


    aTarg = (long)aPosNew - dPos;
    bTarg = (long)bPosNew - ePos;
    aCur = (long)*aPos-dPos;
    bCur = (long)*bPos-ePos;
long aSq = aCur * aCur;
    long bSq = bCur * bCur;
    long rSq = aSq + bSq;
    long aprroxR = long(sqrt(rSq));
    long twoRPlusOne = 2 * aprroxR + 1;
    byte oct;
    while (((long)*aPos<(long)aPosNew-1) || (((long)*aPos > (long)aPosNew+1) ||
((long)*bPos<(long)bPosNew-1) || ((long)*bPos > (long)bPosNew+1)))
```

```
//(!((*aPos<=aPosNew+1 )||(*aPos >= aPosNew-1) || !((bPosNew+1>= *bPos)||(*bPos
>= bPosNew-1))))
   {
   aCur = (long)*aPos - dPos;
   bCur = (long)*bPos - ePos;
   aSq = aCur * aCur;
   bSq = bCur * bCur;
   /*Определение квадранта
         (b)
          Y
          ^
       \3 | 2/
        \ | /
      4  \|/  1
       ---0--- >X (a)
      5  /|\  8
        / | \
       /6 | 7\
   */


  if (aCur>0 && bCur>0)
  {
   if (aCur>bCur)
   oct=1;
   else if (aCur<bCur)
   oct=2;
   else if (aCur==bCur && clock==1)
   oct=1;
   else if (aCur==bCur && clock==0)
   oct=2;
  }
  else if (aCur<0 && bCur>0)
  {
   if (-aCur>bCur)
   oct=4;
   else if (-aCur<bCur)
   oct=3;
   else if (-aCur==bCur && clock==1)
   oct=3;
   else if (-aCur==bCur && clock==0)
   oct=4;
  }
  else if (aCur<0 && bCur<0)
  {
```

```
  if (-aCur>-bCur)
  oct=5;
  else if (-aCur<-bCur)
  oct=6;
  else if (-aCur==bCur && clock==1)
  oct=5;
  else if (-aCur==bCur && clock==0)
  oct=6;
 }
else if (aCur>0 && bCur<0)
{
  if (aCur>-bCur)
  oct=8;
  else if (aCur<-bCur)
  oct=7;
  else if (aCur==-bCur && clock==1)
  oct=7;
  else if (aCur==-bCur && clock==0)
  oct=8;
}
else if (aCur==0 || bCur==0)
{
  if (clock==1)
  {
    if (aCur==0 && bCur>0)
    oct=2;
    else if (aCur==0 && bCur<0)
    oct=6;
    else if (bCur==0 && aCur>0)
    oct=8;
    else if (bCur==0 && aCur<0)
    oct=4;
  }
  else
  {
    if (aCur==0 && bCur>0)
    oct=3;
    else if (aCur==0 && bCur<0)
    oct=7;
    else if (bCur==0 && aCur>0)
    oct=1;
    else if (bCur==0 && aCur<0)
    oct=5;
  }
}
```

```
/*
   if ((aCur>0 && bCur>0 && (aCur>bCur || (bCur==aCur && clock==true))) ||
(bCur==0 && clock==false))
   oct=1;
   if ((aCur>0 && bCur>0 && (aCur<bCur || (bCur==aCur && clock==false))) ||
(aCur==0 && clock==true))
   oct=2;
   if ((aCur<0 && bCur>0 && (-aCur<bCur ||(bCur==-aCur && clock==true))) ||
(aCur==0 && clock==false))
   oct=3;
   if ((aCur<0 && bCur>0 && (-aCur>bCur ||(bCur==-aCur && clock==false))) ||
(bCur==0 && clock==true))
   oct=4;
   if ((aCur<0 && bCur<0 && (-aCur>-bCur ||(-bCur==-aCur && clock==true))) ||
(bCur==0 && clock==false))
   oct=5;
   if ((aCur<0 && bCur<0 && (-aCur<-bCur ||(bCur==aCur && clock==false))) ||
(aCur==0 && clock==true))
   oct=6;
   if ((aCur>0 && bCur<0 && (-aCur<-bCur ||(bCur==aCur && clock==true))) ||
(aCur==0 && clock==false))
   oct=7;
   if ((aCur>0 && bCur<0 && (-aCur>-bCur ||(bCur==aCur && clock==false))) ||
(aCur==0 && clock==true))
   oct=8;
*/

/*
  Serial.println("octant");
  Serial.print(oct,DEC);
  Serial.println("\n");
*/

   switch (oct)
   {
     case 1:

       bitWrite(data,aM,0);
       bitWrite(data,bM,1);
       if (clock==1)
       {
         bitWrite(data,aD,1);
         bitWrite(data,bD,0);
         if (aSq + bSq - rSq <  twoRPlusOne)
         bitWrite(data,aM,1);
```

```
  }
  else
  {
    bitWrite(data,aD,0);
    bitWrite(data,bD,1);
    if ((aSq + bSq - rSq) >twoRPlusOne)
    bitWrite(data,aM,1);
  }
  break;
case 2:
  bitWrite(data,aM,1);
  bitWrite(data,bM,0);
  if (clock==1)
  {
    bitWrite(data,aD,1);
    bitWrite(data,bD,0);
    if (aSq + bSq - rSq > twoRPlusOne)
    bitWrite(data,bM,1);
  }
  else
  {
    bitWrite(data,aD,0);
    bitWrite(data,bD,1);
    if (aSq + bSq - rSq < -twoRPlusOne)
    bitWrite(data,bM,1);
  }
  break;
case 3:
  bitWrite(data,aM,1);
  bitWrite(data,bM,0);
  if (clock==1)
  {
    bitWrite(data,aD,1);
    bitWrite(data,bD,1);
    if (aSq + bSq - rSq < -twoRPlusOne)
    bitWrite(data,bM,1);
  }
  else
  {
    bitWrite(data,aD,0);
    bitWrite(data,bD,0);
    if (aSq + bSq - rSq > twoRPlusOne)
    bitWrite(data,bM,1);
  }
  break;
```

```
case 4:
  bitWrite(data,aM,0);
  bitWrite(data,bM,1);
  if (clock==1)
  {
    bitWrite(data,aD,1);
    bitWrite(data,bD,1);
    if (aSq + bSq - rSq > twoRPlusOne)
    bitWrite(data,aM,1);
  }
  else
  {
    bitWrite(data,aD,0);
    bitWrite(data,bD,0);
    if (aSq + bSq - rSq < -twoRPlusOne)
    bitWrite(data,aM,1);
  }
  break;
case 5:
  bitWrite(data,aM,0);
  bitWrite(data,bM,1);
  if (clock==1)
  {
    bitWrite(data,aD,0);
    bitWrite(data,bD,1);
    if (aSq + bSq - rSq < -twoRPlusOne)
    bitWrite(data,aM,1);
  }
  else
  {
    bitWrite(data,aD,1);
    bitWrite(data,bD,0);
    if (aSq + bSq - rSq > twoRPlusOne)
    bitWrite(data,aM,1);
  }
  break;

case 6:
  bitWrite(data,aM,1);
  bitWrite(data,bM,0);
  if (clock==1)
  {
    bitWrite(data,aD,0);
    bitWrite(data,bD,1);
    if (aSq + bSq - rSq > twoRPlusOne)
```

```
    bitWrite(data,bM,1);
   }
   else
   {
    bitWrite(data,aD,1);
    bitWrite(data,bD,0);
    if (aSq + bSq - rSq < -twoRPlusOne)
    bitWrite(data,bM,1);
   }
   break;
  case 7:
   bitWrite(data,aM,1);
   bitWrite(data,bM,0);
   if (clock==1)
   {
    bitWrite(data,aD,0);
    bitWrite(data,bD,0);
    if (aSq + bSq - rSq < -twoRPlusOne)
    bitWrite(data,bM,1);
   }
   else
   {
    bitWrite(data,aD,1);
    bitWrite(data,bD,1);
    if (aSq + bSq - rSq > twoRPlusOne)
    bitWrite(data,bM,1);
   }
   break;
  case 8:
   bitWrite(data,aM,0);
   bitWrite(data,bM,1);
   if (clock==1)
   {
    bitWrite(data,aD,0);
    bitWrite(data,bD,0);
    if (aSq + bSq - rSq > twoRPlusOne)
    bitWrite(data,aM,1);
   }
   else
   {
    bitWrite(data,aD,1);
    bitWrite(data,bD,1);
    if (aSq + bSq - rSq < -twoRPlusOne)
    bitWrite(data,aM,1);
   }
```

```
      break;
    }
    movexyz(data);
  }
  linear(feedRate);
}
void movexyz(byte data)
{
  /*
  Байт с управляющей информацией имеет следующий формат:
  bit 7  -  -  -  -  -  -  0
    -- -- ZD ZM YD YM XD XM
  Где Ms - движение в шагах (двигаемся по оси координат или нет) и DS - направление
  */
  stepTime=feedRate+lastStepTime;
  if (bitRead(data,XD)==1)
  {
    digitalWrite(XDir, HIGH);
  }
  else
  {
    digitalWrite(XDir, LOW);
  }

  if (bitRead(data,YD)==1)
  {
    digitalWrite(YDir, HIGH);
  }
  else
  {
    digitalWrite(YDir, LOW);
  }

  if (bitRead(data,ZD)==1)
  {
    digitalWrite(ZDir, HIGH);
  }
  else
  {
    digitalWrite(ZDir, LOW);
  }
  if (stepTime<millis())
  tooFast=true;
  while (stepTime >= millis());
  lastStepTime=millis();
```

```
  if (bitRead(data,XM)==1)
  {
    digitalWrite(XStep, HIGH);
    digitalWrite(XStep, LOW);
  }

  if (bitRead(data,YM)==1)
  {
    digitalWrite(YStep, HIGH);
    digitalWrite(YStep, LOW);
  }
  if (bitRead(data,ZM)==1)
  {
    digitalWrite(ZStep, HIGH);
    digitalWrite(ZStep, LOW);
  }
  if (bitRead(data,XM)==1 && bitRead(data,XD)==1)
  XPos=XPos+1;
  if (bitRead(data,XM)==1 && bitRead(data,XD)==0)
  XPos=XPos-1;
  if (bitRead(data,YM)==1 && bitRead(data,YD)==1)
  YPos=YPos+1;
  if (bitRead(data,YM)==1 && bitRead(data,YD)==0)
  YPos=YPos-1;
  if (bitRead(data,ZM)==1 && bitRead(data,ZD)==1)
  ZPos=ZPos+1;
  if (bitRead(data,ZM)==1 && bitRead(data,ZD)==0)
  ZPos=ZPos-1;
}
```