МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ХАРКІВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ МІСЬКОГО ГОСПОДАРСТВА імені О. М. БЕКЕТОВА

А. І. Колесник

ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ

КОНСПЕКТ ЛЕКЦІЙ

(для здобувачів першого (бакалаврського) рівня вищої освіти денної та заочної форм навчання зі спеціальності 141 – Електроенергетика, електротехніка та електромеханіка)

Харків
ХНУМГ ім. О. М. Бекетова
2025

УДК 621.3

Колесник А. І. Програмування мікроконтролерів : конспект лекцій для здобувачів першого (бакалаврського) рівня вищої освіти денної та заочної форм навчання зі спеціальності 141 – Електроенергетика, електротехніка та електромеханіка / А. І. Колесник ; Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова. – Харків : ХНУМГ ім. О. М. Бекетова, 2025. – 97 с.

Автор
канд. техн. наук А. І. Колесник

Рецензент
Л. А. Назаренко, доктор технічних наук, професор кафедри світлотехніки і джерел світла (Харківський національний університет міського господарства імені О. М. Бекетова)

Рекомендовано кафедрою світлотехніки і джерел світла, протокол № 1 від 12 вересня 2023 р.

Конспект лекцій складено з метою допомогти бакалаврам спеціальності 141 – Електроенергетика, електротехніка та електромеханіка у підготовці до аудиторних занять і самостійної роботи, екзамену з дисципліни «Програмування мікроконтролерів»

ЗМІСТ

ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

ДБЖ – джерело безперебійного живлення
ПЯЕ – показник якості електроенергії
ІБЖ – імпульсний блок живлення
ЕП – електроприлади
LED (Light emitting diode) – світлодіоди
HID (High Intensity Discharge) – газорозрядна лампа високої інтенсивності
ШІМ – широтно-імпульсна модуляція
ЧІМ – частотно-імпульсна модуляція
АЧР – автоматичне частотне розвантаження
IoT (Internet of things) – інтернет речей
МК – мікроконтролер
ЕОМ – електронно-обчислювальна машина
ЦП – центральний процесор
МПС – мікропроцесорна система
ПКВМ – програмована користувачем вентильна матриця
ПЗ – програмне забезпечення
СЕ – схемний емулятор
ПЗП – постійний запам’ятовувальний пристрій
МП – мікропроцесор
МПП– мікропроцесорний пристрій
ОЗП – оперативний запам’ятовувальний пристрій
ОП – операційний підсилювач
ПВЗ – пристрій вибірки-зберігання
РЕП – радіоелектронний пристрій
СБ – старший байт вхідного слова
АЦП – аналогово-цифровий перетворювач
ЦАП – цифро-аналоговий перетворювач
AB – Adress Bus (шина адреси)
ADC – аналого-цифровий перетворювач
BP – Breakpoint (точка зупинки програми)
DB – Date Bus (шина даних)
EEPROM – електрично-зтираний програмований постійний запам’ятовувальний пристрій
MC – мікроконтролер
MCU – мікропроцесор
PN – Programmers Notepad

ВСТУП

Сучасна освіта вимагає нових підходів до навчання, особливо в контексті розвитку цифрових технологій. Зокрема, навички програмування стають невід'ємною частиною майбутніх професій, що зумовлює необхідність їхнього впровадження вже сьогодні у шкільну програму.

Однією з найпопулярніших платформ для навчання основам програмування, автоматики та робототехніки є Arduino – відкрита апаратно-програмна платформа. Вона дозволяє легко працювати з різноманітними електронними компонентами, такими як давачі, світлодіоди, двигуни тощо. Завдяки доступності, гнучкості та простоті використання, Arduino стала широко поширеною серед розробників і початківців у всьому світі.

Навчання з використанням цієї платформи сприяє розвитку логічного мислення, креативності та інженерних навичок. Студенти можуть створювати інтерактивні проєкти, роботизовані системи, моделі «Розумного будинку» або автоматизовані пристрої. Головною перевагою такого підходу є можливість безпосередньо спостерігати результати своєї роботи, що значно підвищує мотивацію до навчання.

Сучасні системи освітлення вимагають автоматизованого управління для підвищення ефективності, енергозбереження та адаптації до умов експлуатації.

Програмування мікроконтролерів відіграє ключову роль у створенні таких систем, дозволяючи реалізувати керування яскравістю освітлення, автоматичне ввімкнення / вимкнення світла за розкладом або показниками давачів освітлення, руху, температури тощо, інтелектуальне управління кольором у світлодіодних системах RGB та RGBW, дистанційне керування освітленням через різні протоколи, аналіз та контроль енергоспоживання для підвищення ефективності роботи системи.

Метою викладання навчальної дисципліни «Програмування мікроконтролерів» є формування знань щодо основних видів та принципів керування системами освітлення за допомогою мікроконтролера; оволодіння елементною базою та типами мікроконтролерів, використання їх у системах керування освітленням та світлотехнічними пристроями; ознайомлення з мовою програмування та апаратною підтримкою мікроконтролерних пристроїв.

Курс «Програмування мікроконтролерів» базується на знаннях загальної фізики, електротехніки та основ світлотехніки, що є основою для програмування мікроконтролерів у світлотехнічних системах. Вивчення алгоритмів керування, методів обробки даних із давачів та інтеграції з електронними модулями дозволяє реалізовувати ефективні та інноваційні рішення у світлотехніці.

Застосування мікроконтролерів у освітлювальних системах не лише покращує якість світлового середовища, а й сприяє економії енергії, зниженню експлуатаційних витрат та розширенню функціональності світлотехнічних пристроїв. У цьому конспекті лекцій розглянуто основи програмування мовою C++, а також наведено приклади практичного використання Arduino для створення власних електронних проєктів.

1 ЧИННИКИ ВИНИКНЕННЯ ПЕРЕШКОД В ЕЛЕКТРОМЕРЕЖІ

Електрична мережа, у якій присутня перешкода, може піддаватися відхилення і коливання напруги. Електрична мережа піддається і імпульсним напруженням. Причиною можуть слугувати природні явища у вигляді грози або комутаційні операції, що проводяться в мережі.

Кратні гармоніки – це синусоїдальний струм або напруга. Різниця частоти такого явища буде у багато разів відрізнятися від основної частоти. Якщо електромережа володіє нелінійною вольт-амперної характеристикою, то виникає такий вид перешкоди. Основні джерела: телевізори, люмінесцентні лампи, перетворюючі установки, індукційні печі.

Електромережа з некратними гармоніками може підключатися до трансформатора через статичні перетворювачі частоти. Періодичність і тривалість гармонік буде залежати від того, яка вихідна частота у перетворювача.

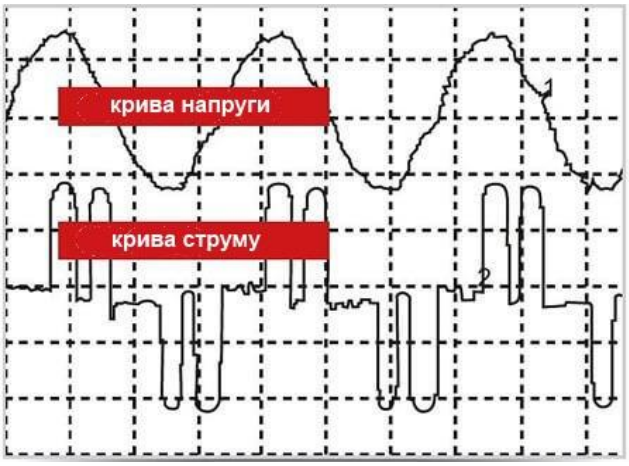


Рисунок 1.1 – Приклад відхилень в електромережі

Відхилення частоти з’являється через те, що потужність генераторів, які виробляють електроенергію, не відповідає споживаному навантаженню. Електромережа, у якій підвищується потужність навантаження, підвищує частоту і швидкість генератора. Якщо електрична мережа отримала несподіване і різке зниження напруги, то це означає, що виникла така перешкода, як короткі провали напруги. Електромережа відновлює нормальну роботу через певний час. Таке явище виникає в енергосистемах через комутаційні процеси, які пов’язані з запуском і роботою двигунів сильних потужностей, а також із коротким замиканням.

Споживачі повинні враховувати той факт, що усунути або зменшити кількість перешкод, які викликані роботою енергосистем щодо усунення коротких замикань, неможливо.

Способи захисту. Виникнення перешкод в електричній мережі може статися в будь-який момент, що призведе до неприємних моментів і втрат. Наприклад, якщо працюєш за комп’ютером, то важливі текстові дані можуть

зникнути. Щоб цього уникнути, необхідний захист від подібних явищ. Відмінним рішенням в цьому випадку буде захист за допомогою джерела безперебійного живлення (ДБЖ). Після того як електромережа вимкнулася, батарея залишається працездатною не менше десяти хвилин. Цього буде цілком достатньо, щоб зберегти всі важливі документи і програми. Також таке джерело живлення слугує захистом від перепадів напруги. Про те, як вибрати ДБЖ, рекомендується звернутись до провідних виробників таких технічних рішень. Захист від перешкод у мережі може здійснюватися і дешевшим способом: застосування мережевих фільтрів. Такий пристрій зможе врятувати прилади, які підключені до електромережі, від вимкнень живлення і перешкод. Захист такими способами дозволить уберегти прилади й перешкода в мережі їм буде безпечна.

1.1 Показники якості електроенергії в електричних мережах

Під терміном «якість електроенергії» розуміється відповідність основних параметрів енергосистеми встановленим нормам виробництва, передачі і розподілу електроенергії [1, 4].

До основних показників якості електроенергії (ПЯЕ) відносять:

– усталене відхилення напруги dUy;
– розмах зміни напруги dUf;
– доза флікера Pt;
– коефіцієнт спотворення синусоїдності кривої напруги K_u;
– коефіцієнт n-ї гармонійної складової напруги K_u;
– коефіцієнт несиметрії напруг за зворотною послідовністю K_{2u};
– коефіцієнт несиметрії напруг за нульовою послідовністю K_{0u};
– відхилення частоти Δf;
– тривалість провалу напруги Δtn;
– імпульсна напруга $U_{імп}$;
– коефіцієнт тимчасової перенапруги $K_{\Delta U}$.

1.2 Види перешкод в електромережі

Існує два типи перешкод в електричній мережі: імпульсні і високочастотні. Перші виникають під час увімкнення або вимкнення приладу в електромережу. Вони є небезпечними, оскільки можуть за короткий час вивести з ладу всю електронну техніку в будинку. Високочастотні перешкоди існують в мережі завжди, але вважаються не такими небезпечними, як імпульсні [2].

На рисунку 1.2 зображена осцилограма процесу, викликаного провалом напруги, що полягає в зниженні напруги хоча б однієї фази нижче порогового значення. Такі події відносять до випадкових, воно виникає через несправності в електричній мережі або в електроустановках споживача, під час увімкнення потужного навантаження або у разі виникнення аварійних ситуацій.

На рисунку 1.3 зображено переривання напруги, яке характеризується значеннями напруги в кожній із фаз нижче 5 % від опорної напруги.

Розглядаючи побутові навантаження, до основних технічних засобів, здатних компенсувати провал та переривання напруги, можна віднести імпульсні блоки живлення (ІБЖ).

На промислових підприємствах з великою потужністю навантажень допомогти уникнути згубних впливів провалів та переривань напруги здатні системи накопичення електроенергії, які під час виникнення провалу перетворять накопичену електричну енергію на напругу необхідного рівня.

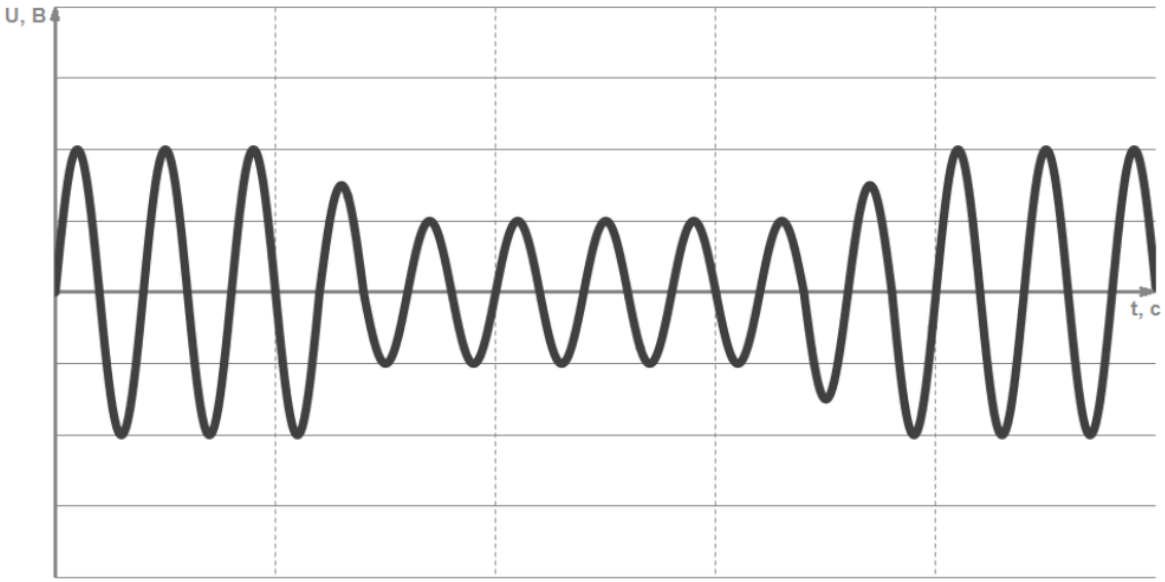


Рисунок 1.2 – Провал напруги

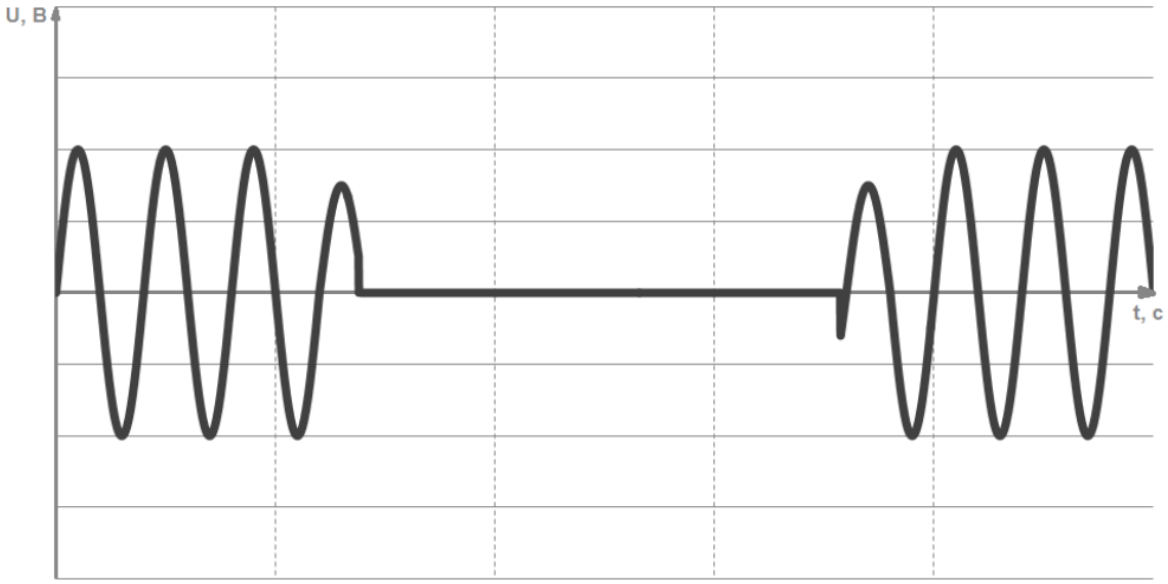


Рисунок 1.3 – Переривання напруги

1.3 Захист від перешкод

Всі пропозиції, перелічені вище, є достатньо дієвими, але залишається ще одна з найважливіших проблем – це перенавантаження мережі. Ця проблема є доволі важливою для навчальних закладів, тому що для їхнього безперервного функціонування необхідне безперебійне електропостачання. Аварії в робочий час є достатньо критичними. Такі аварії виникають частіше в холодну та жарку пору, коли паралельно з комп'ютерною технікою вмикаються кондиціонери та обігрівачі. Таких проблем можливо уникнути і для цього існують варіанти. Для уникнення цієї ситуації, доведеться контролювати кількість одночасно підключених приладів, контролювати їхню потужність, і не включати одночасно кілька потужних споживачів. Тому, щоб не замислюватися, що і якої потужності одночасно підключено, були створені спеціальні реле управління навантаженням – реле пріоритету (рис. 1.4) (їх ще називають обмежувачі потужності).

Рисунок 1.4 – Приклад реле контролю [3]

Реле постійно відстежує струм, який протікає через нього (і відповідно споживану навантаженням потужність), і якщо струм перевищує встановлене значення, реле вимикає першу не пріоритетну групу, якщо цього виявилося недостатньо, тоді вимикається друга не пріоритетна група. Залишається увімкненим тільки навантаження, об'єднане в не відключену групу. Таке реле можливо вдало інтегрувати в наші заклади, завдяки тому, що зазвичай будь-яка електрична мережа підключається через декілька електричних ліній. На головній лінії залишити комп'ютери, серверну частину та головне освітлення, на другій лінії – оргтехніку, побутові прилади, кондиціонери та обігрівачі, а до найнижчої підключити розетки, світло в технічних приміщеннях та іншу периферію.

Така система вмонтовується в серце енергопостачання в електричний щит. Невід'ємною частиною будь-якої електричної системи є стабілізатор напруги. Його головне завдання – перетворення електричної енергій на задану вихідну напругу. Завдяки йому всі прилади можуть бути у безпеці та не боятися перепадів напруги, які часто трапляються. У всіх приладів є піковий та мінімальний діапазон вхідної напруги для правильного функціонування. Стабілізатор дозволяє скорегувати вхідну напругу і видати необхідну. Також у наш час є моделі, в які вбудовані акумулятори, що дозволяє після вимкнення всього світла вимкнути безпечно всю техніку. Від таких перепадів внутрішні запобіжники в техніці часто не спасають, через що техніка не підлягає ремонту, а для державних установ це є дуже великою проблемою, бо купівля нової інколи доволі проблематична. У час, коли приміщення переповнені електроприладами, не варто забувати про економічні фактори та безпеку. Необхідно стежити за тим, скільки ми витрачаємо електроенергії та перешкоджати пожежам через короткі замикання та перенавантаження системи. Навчальні заклади, безумовно, потребують покращень, але навіть при обмеженому бюджеті можна зробити значні зміни. Система відстеження якості електричної енергії з додатковим використанням Arduino може бути реалізована за мінімальних витрат, і її компоненти повністю відповідають поставленим цілям. Вона є універсальною завдяки платформі Arduino, до якої підключено Wi-Fi модуль. Arduino виконує функцію лічильника, що дозволяє знімати показники спожитої енергії на місці та передавати їх на сервер. Звідти дані можуть бути транслюванні на телефон та використовуватись для побудови звітності. Крім того, система легко інтегрує різні типи давачів для контролю вологості, температури або функцій охорони.

2 ІМПУЛЬСНІ ТА ВИСОКОЧАСТОТНІ ПЕРЕШКОДИ

Всі мережеві відхилення можна класифікувати за двома ознаками: походженням шумів і видом електромагнітної аномалії.

Причиною виникнення спотворень є:

– природні явища (гроза, іонізація повітря сяйвами тощо);

– техногенні впливи (аварії на лініях, комутація потужних пристроїв тощо);

– електромагнітні хвилі природного та техногенного походження.

Перелічені причини можуть викликати серію імпульсних перешкод або хвилі гармонійних спотворень, накладені поверх синусоїдального струму.

Наявність імпульсних струмів у мережі дуже шкідливо впливає на роботу сучасних побутових приладів, часто насичених електронікою. Якщо не застосовувати прилади захисту (рис. 2.1), електронні пристрої можуть вийти з ладу, не кажучи вже про якість їхньої роботи. Зрозуміло, чутливе обладнання розробники захищають запровадженими схемами придушення перешкод, але нерідко потрібні додаткові зовнішні прилади, наприклад, безперебійні джерела живлення, мережеві фільтри та інші.

Рисунок 2.1 – Захисні імпульсні фільтри [4]

Імпульсні перенапруги (рис. 2.2) в точці передачі електричної енергії споживачеві викликаються переважно блискавковими розрядами або комутаційними процесами в електричній мережі або електроустановках споживача електричної енергії. Час наростання імпульсної напруги може змінюватися в широких межах (від значень менше 1 мікросекунди до декількох мілісекунд).

На середній та високій напрузі для захисту від згубних впливів перенапруг використовуються обмежувачі перенапруги, принцип їхньої дії заснований на напівпровідникових елементах, що мають нелінійну вольт-амперну характеристику.

Для захисту від перенапруг у низьковольтній мережі використовуються реле напруги, стабілізатори напруги або джерела безперебійного живлення, в яких передбачена відповідна функція. Також є модульні пристрої захисту від імпульсних перенапруг, призначені для установки в домашній розподільний щиток.

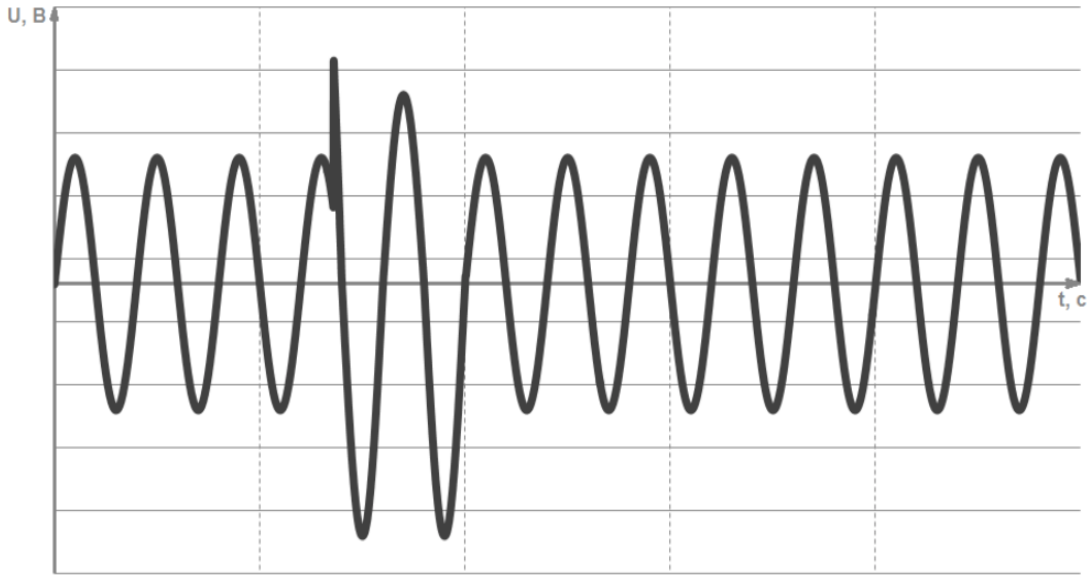


Рисунок 2.2 – Імпульсна перенапруга

Причини виникнення такого явища:

– коливання і відхилення напруги;

– імпульсні напруги;

– гармоніки;

– відхилення частоти;

– короткі провали напруги.

2.1 Основні типи спотворень якості електричної енергії за роботи освітлювальної установки

Ступінь впливу частоти на продуктивність ряду механізмів може бути виражена через споживану ними активну потужність:

$$P = a \cdot f^n, \tag{2.1}$$

де a – коефіцієнт пропорційності, що залежить від типу механізму;

f – частота мережі;

n – показник ступеня.

Залежно від значень показника ступеня n, електроприлади (ЕП) можна розбити на такі групи:

1) механізми з постійним моментом опору – поршневі насоси, компресори, металорізальні верстати та ін.; для них n = 1;

2) механізми з вентиляторним моментом опору – відцентрові насоси, вентилятори, димососи та ін.; для них n = 3; на різних електростанціях зазвичай це двигуни насосів живильної води, циркуляційних насосів, димових вентиляторів, маслонасосів і т. д.;

3) механізми, для яких n = 3,5–4 – відцентрові насоси, що працюють з великим статичним напором (протитиском), наприклад, живильні насоси котелень.

ЕП 2-ї і 3-ї груп, найбільш схильних до впливу частоти, мають регулювальні можливості, завдяки яким споживана ними потужність із мережі залишається практично незмінною.

Найбільш чутливі до зниження частоти двигуни власних потреб електростанцій. Зниження частоти призводить до зменшення їхньої продуктивності, що супроводжується зниженням потужності генераторів і подальшим дефіцитом активної потужності і зниженням частоти (наявна лавина частоти).

Такі ЕП, як лампи розжарювання, печі опору, дугові електричні печі на зміну частоти практично не реагують.

Відхилення частоти негативно впливають на роботу електронної техніки: відхилення частоти більше +0,1 Гц призводить до зміни яскравості і геометричних фонових спотворень телевізійного зображення, зміни частоти

від 49,9 Гц до 49,5 Гц тягне за собою майже чотириразове збільшення допустимого розмаху телевізійного сигналу до фонової завади. Зміна частоти до 49,5 Гц потребує суттєвого посилення вимог до відношення сигнал / фонова перешкода у всіх ланках телевізійного тракту – від обладнання апаратно-студійного комплексу до телевізійного приймача, виконання яких пов'язане зі значними матеріальними витратами.

Крім цього, знижена частота в електричній мережі впливає і на термін використання обладнання, що містить елементи із сталлю (електродвигуни, трансформатори, реактори зі сталевим магнітопроводом), шляхом збільшення струму намагнічування в таких апаратах і додаткового нагріву сталевих сердечників.

Їх роблять деякі види побутової техніки, що генерують випромінювання частотою від 2,4 ГГц. Це можуть бути мікрохвильові печі, радіотелефони, супутникові антени тощо. Це зазвичай обладнання старого зразка. За наявності високочастотних перешкод на них реагують не лише світлодіодні лампочки, а й інші побутові прилади: блимає екран телевізора, шумлять динаміки. Ліквідувати перешкоди можна, встановивши електромережу спеціальні фільтри, які продаються в магазинах електрообладнання.

Що стосується ламп освітлення, то у разі підвищення напруги на 10 % термін експлуатації саме ламп розжарювання знижується в 4 рази. Після зниження напруги на 10 % знижується світловий потік ламп розжарювання на 40 % і люмінесцентних ламп на 15 %. При величині зниження напруги більш ніж на 10 % люмінесцентні лампи мерехтять, а при зниженні більш ніж на 20 % просто не загоряються.

Відхилення хоча б одного з показників якості напруги (відхилення напруги; коливання напруги; несинусоїдальність напруги; несиметрія напруги; провал напруги; імпульс напруги і тимчасове перенапруження) призводить до відповідних змін під час роботи світлодіодних ламп.

Так відхилення напруги призводить до зменшення терміну експлуатації, коливання напруги спричиняє мигання ламп, тобто до різких змін світлового потоку, які після перевищення певного порогу можуть відбиватися на зоровому сприйнятті людей. Одним з ефективних методів стабілізації напруги в мережі є використання блоків живлення (внутрішніх або зовнішніх), що мають високий рівень точності. Для світлодіода визначальним є сила струму, який через нього протікає, ніж напруга, що підводиться до нього. Відповідно, блок живлення, вбудований в світлодіодну лампу або світильник, теоретично повинен стабілізувати саме струм. Але для реалізації цієї задачі необхідно одночасно враховувати велику кількість факторів, причому, більшість з них має випадковий характер.

Енергоефективне світлодіодне освітлення у промисловості поступово замінює старі типи освітлювальних приладів. Однак у деяких випадках компанії стикаються з постійними поломками нових дорогих ламп. Причиною цього є маловідомі проблеми з якістю електроенергії.

Коли підприємство розглядає перехід на світлодіодні (LED) світильники, зазвичай на першому місці розглядаються високі характеристики світлодіодів.

Наприклад, порівняно з газорозрядними лампами високої інтенсивності (HID), світлодіоди не містять токсичних речовин, миттєве включення, можливість регулювання яскравості та термін експлуатації не 1–2 роки, а в середньому п’ять років.

Світлодіодне освітлення дійсно має всі перераховані вище переваги. Але в деяких випадках, замінивши тисячі ламп у виробничих цехах, теплицях, складах тощо, компанія стикається з каскадом поломок світлодіодних світильників, які не відпрацювали навіть третину належного терміну. У низці випадків світлодіоди виходять з ладу в строк від одного місяця до двох років після початку експлуатації. Найгірше, коли лавиноподібне зростання відмов ламп відбувається протягом року після завершення п’ятирічного гарантійного терміну.

У більшості випадків постачальники змінюють драйвери світлодіодів (електронні блоки керування живленням LED-ламп). Але якщо ламп кілька тисяч, то ремонт обійдеться надто дорого і вимагатиме багато часу. Не кажучи вже про зупинення виробництва у разі, коли продовження роботи без освітлення неможливе. У цьому разі збитки можуть сягати мільйони гривень. Очевидно, це не той результат, на який чекали керівники підприємства, переходячи на новий тип надійного та економічно ефективного LED-освітлення.

Необхідно лише вирішити проблему з якістю електроенергії, яка для світлодіодів має бути вищою, як для іншого сучасного обладнання. Яскраві лампи HID використовують електромагнітний баласт із міді та заліза і можуть витримати практично будь-які перешкоди в електромережі. Частіше вони виходять з ладу через неправильний тепловий режим роботи, ніж через погану якість енергії. Тому лініям живлення HID-освітлення часто не приділяють особливу увагу. Зараз більшість виробників LED пропонують потужні промислові світильники, що працюють під напругою до 480 В змінного струму. На жаль, найчастіше виробники не враховують особливості промислового використання світильників, спираючись лише на досвід у сфері вуличного освітлення. У результаті драйвери LED проєктуються для захисту від стрибків напруги і зазвичай оснащуються пристроями захисту від перенапруг, розрахованими на 10 кВ при 10/20 кА.

У разі використання промислового світлодіодного освітлення виникають складніші проблеми з якістю електроенергії. На виробництві існує безліч споживачів з нелінійним навантаженням, наприклад, регульовані приводи (ШІМ / ЧІМ) або асинхронні двигуни, що споживають великі пускові струми. Ці пристрої створюють у мережі живлення значний рівень перешкод у широкому діапазоні частот і здатні значно погіршити якість енергії у всіх рівнях, високих і низьких частотах. Перешкоди створюють також перемикання контакторів.

У загальному діапазоні перешкод є інтергармоніки. В європейські технічні вимоги щодо якості електроенергії поняття інтергармоніки було включено у 1994 р. Це явище призводить до збоїв у роботі детекторів переходу через нуль, наприклад, у пристроях регулювання яскравості світлодіодних світильників. Найчастіше джерелом інтергармонік стають зношені

електродвигуни, зварювальні апарати, індукційні печі та інше обладнання. В окремих випадках поломки світлодіодних світильників спричиняють періодичні короткочасні аномалії електромережі.

Помилкою є використання лише загальних параметрів моніторингу якості електроенергії та відсутність даних про явища, які можуть пояснити причину поломки драйверів світлодіодів. У результаті короткі перехідні процеси, що призводять до стрибків напруги, випадають із поля зору.

Здебільшого правильний моніторинг якості електроенергії дозволяє своєчасно виявити проблему та запобігти поломці дорогого світлодіодного освітлення. Однак для цього іноді необхідно оновити обладнання моніторингу електромережі. Справа в тому, що старі прилади та програмне забезпечення можуть «ігнорувати» важливі сигнали, такі як аномалії гармонік та мерехтіння.

Необхідні аналізатори якості енергії, які відповідають міждержавному стандарту IEC 61000-4-7:2009. Він був прийнятий у 2014 р. та враховує сучасні вимоги до електроніки та якості енергії. Також у ньому рознесено поняття гармоніки, інтергармонік та інших спектральних складових від 2 кГц до 9 кГц.

Отже, необхідно документувати найпотужніші, а також нелінійні навантаження, що працюють на лініях освітлення. Наприклад, більшість ліній з’єднані разом через загальний перемикач на розподільному пристрої. Можливо, на підприємстві є більше однієї лінії 480 В, але в будь-якому випадку необхідно знати, які типи перешкод є на кожній лінії, до якої підключатиметься LED-освітлення. Необхідно виявити можливі помилки у розведенні фаз і нейтралі, правильно спроєктувати та побудувати систему заземлення, яка повинна мати низький імпеданс та витримувати високочастотні перешкоди.

Ретельний моніторинг якості енергії за допомогою сучасних пристроїв є невід’ємною частиною надійного функціонування систем світлодіодного освітлення. Моніторинг повинен проводитися до встановлення безпосередньо після, а також регулярно повторюватися з тривалими місячними періодами вимірів для виявлення всіх можливих аномалій електромережі.

Рисунок 2.3 – Аналізатор Fluke 1738 [5]

На сьогодні такі процедури легко виконуються за допомогою досконалих приладів для тестування якості електроенергії. Наприклад, аналізатор Fluke

1738 (рис. 2.3) здатний реєструвати більше 500 параметрів якості електроенергії, включаючи рівні гармонік, стрибки напруги та спотворення форми струму, які можуть призвести до поломки драйверів світлодіодних світильників.

2.2 Відхилення напруги і частоти

2.2.1 Відхилення напруги

Відхилення напруги – відміна фактичної напруги в усталеному режимі роботи системи електропостачання від його номінального значення [6, 11].

Відхилення напруги в тій чи іншій точці мережі відбувається під впливом зміни навантаження відповідно до її графіка.

Відхилення напруги від номінальних значень відбуваються через добові, сезонні і технологічні зміни електричного навантаження споживачів; зміни потужності компенсуючих пристроїв; регулювання напруги генераторами електростанцій і на підстанціях енергосистем; зміни схеми і параметрів електричних мереж.

Відхилення напруги визначається різницею між чинним значенням напруги U і номінальним значеннями напруги $U_{НОМ}$, В [6, 11]:

$$\delta U = U - U_{\text{НОМ}}, \tag{2.2}$$

або у відсотках:

$$\delta U = \frac{U - U_{\text{НОМ}}}{U_{\text{НОМ}}} \cdot 100\%, \tag{2.3}$$

Усталене відхилення напруги dU_y дорівнює у відсотках:

$$\delta U_y = \frac{U_y - U_{\text{НОМ}}}{U_{\text{НОМ}}} \cdot 100\%, \tag{2.4}$$

де U_y – усталене (діюче) значення напруги за інтервал усереднення.

В електричних мережах однофазного струму діюче значення напруги визначається як значення напруги основної частоти U (1) без урахування вищих гармонійних складових напруги, а в електричних мережах трифазного струму – як діюче значення напруги прямої послідовності основної частоти U1 (1).

Стандартом нормуються відхилення напруги на виводах приймачів електричної енергії. Нормально допустимі і гранично допустимі значення усталеного відхилення напруги дорівнюють відповідно ±5 % і ±10 % від номінального значення напруги і в точках загального приєднання споживачів електричної енергії повинні бути встановлені в договорах енергопостачання для годин мінімуму і максимуму навантажень в енергосистемі з урахуванням необхідності виконання норм стандарту на висновках приймачів електричної енергії відповідно до нормативних документів.

2.2.2 Електропривод, електронна апаратура і комп'ютери

Після зниження напруги на затискачах асинхронного електродвигуна на 15 % момент знижується на 25 %. Двигун може не запуститися або зупинитися. Після зниження напруги збільшується споживаний від мережі струм, що тягне розігрів обмоток і зниження терміну експлуатації двигуна. За тривалої роботи на зниженій на 10 % напрузі термін експлуатаціїелектродвигуна знижується вдвічі. За підвищення напруги на 1 % збільшується споживана двигуном реактивна потужність на 3–7 %. Знижується ефективність роботи приводу і мережі.

При зниженні напруги можуть виникати збої в роботі, що призводять до втрати даних. Нерідкі відмови блоків живлення внаслідок підвищеного струму споживання за зниженої напруги і їхнього перегріву за підвищеної. У сучасній електронній техніці часто встановлюють спеціальні блоки, що вимикають пристрій у разі відхилення напруги для запобігання його виходу з ладу. Тому багато пристроїв втрачають працездатність у разі відхилення напруги від норми.

Після зниження напруги суттєво погіршується технологічний процес, збільшується його тривалість. Отже, збільшується собівартість виробництва. У разі підвищення напруги знижується термін експлуатації обладнання, підвищується ймовірність аварій. За значних відхилень напруги відбувається зрив технологічного процесу.

2.2.3 Відхилення частоти

Відхилення частоти – різниця між дійсним і номінальним значеннями частоти виражене у Гц [6, 11]:

$$\Delta f = f - f_{\text{ном}}, \tag{2.5}$$

або у відсотках:

$$\Delta f = \frac{f - f_{\text{ном}}}{f_{\text{ном}}} \cdot 100\%. \tag{2.6}$$

Стандартом встановлюються нормально і гранично допустимі значення відхилення частоти, що дорівнюють ±0,2 Гц і ±0,4 Гц відповідно.

Зниження частоти відбувається при дефіциті потужності працюючих в системі електростанцій.

Для усунення цих явищ необхідно ремонтувати або модернізувати існуючі та будувати нові електростанції. А поки їх немає, активно застосовується радикальна міра – автоматичне частотне розвантаження (АЧР), тобто вимкнення частини споживачів при зниженні частоти. Це ще називають віяловими вимкненнями.

Для споживача важливо знати, в яку чергу вимикають його обладнання від мережі, за такого розвитку подій (вказується при укладенні договору

електропостачання) аргументовано вимагати зміни черговості або мати власні резервні генерувальні потужності.

Підвищення частоти відбувається за різкого скидання навантаження в системі електропостачання – ситуація аварійна і дія [11] на неї не поширюється, а в сталому режимі роботи мережі така подія дуже рідкісна.

Жорсткі вимоги стандарту до відхилень частоти напруги живлення обумовлені значним впливом частоти на режими роботи електрообладнання, хід технологічних процесів виробництва і, як наслідок, техніко-економічні показники роботи промислових підприємств.

Електромагнітна складова збитку обумовлена збільшенням втрат активної потужності в електричних мережах і зростанням споживання активної та реактивної потужностей. Відомо, що зниження частоти на 1 % збільшує втрати в електричних мережах на 2 %.

Технологічна складова збитків спричинена переважно недовипуском промисловими підприємствами своєї продукції і вартістю додаткового часу роботи підприємства для виконання завдання. Згідно з експертними оцінками, значення технологічного збитку на порядок вище електромагнітного.

Аналіз роботи підприємств з безперервним циклом виробництва показав, що більшість основних технологічних ліній обладнано механізмами з постійним і вентиляторним моментами опорів, а їхніми приводами слугують асинхронні двигуни. Частота обертання роторів двигунів пропорційна зміні частоти мережі, а продуктивність технологічних ліній залежить від частоти обертання двигуна.

2.3 Вплив відхилень напруги на роботу споживачів електроенергії

Відповідно до [11] встановлюються нормально та гранично допустимі відхилення напруги на висновках приймачів електричної енергії, які становлять ±5 % та ±10 % номінального значення напруги. Насамперед на споживачах відображається відхилення напруги, що встановилося. Після зниження напруги до її номінального значення відбувається зменшення світлового потоку від світлодіодних ламп, знижується освітленість у приміщенні, на робочих місцях. Зокрема, зниження напруги на 10 % призводить до зменшення освітленості робочої поверхні в середньому на 40 %, що викликає зниження продуктивності праці, підвищену стомлюваність персоналу. Підвищення напруги для світлодіодних ламп також на 10 % призводить до скорочення їхнього терміну експлуатації та викликає надмірне освітлення робочих поверхонь, що несприятливо позначається на сприйнятті інформації з моніторів та цифрових приладів.

Газорозрядні люмінесцентні лампи при зазначеному діапазоні зміни напруги не настільки істотно змінюють світловіддачу, але збільшення напруги на 10–15 % призводить до різкого зниження їхнього терміну експлуатації, а зниження напруги на 20 % викликає відмови запалювання ламп.

Найефективніше споживачі електричної енергії працюють за номінальної напруги. Проте забезпечити подавання номінальної напруги до всіх споживачів

практично неможливо. Всякий провідник має певний опір, тому проходження струму електричною мережею пов'язане з втратами напруги. Ці втрати напруги не залишаються сталими. Зміна як активного, так і реактивного навантаження, вимикання окремих споживачів та деякі інші фактори спричинюють зміну втрат напруги в мережі, а тим самим – і зміну напруги на затискачах електроспоживачів. Ці зміни можуть бути швидкими і короткочасними (наприклад, у разі пуску короткозамкнених електродвигунів) або повільними і тривалими (у разі поступової зміни навантаження і плавного регулювання напруги) [8, 10]. Різниця між дійсною напругою, що подається до електроспоживача чи певної точки електромережі (за сталої напруги або плавної і повільної її зміни), і номінальною напругою називається відхиленням напруги, яке розглянули в попередньому підрозділі.

За зміни напруги лише на 1 % потужність різних ламп змінюється на 1,5 %, світловий потік – на 3,5 %, світлова віддача – на 1,8 %, строк експлуатації – на 13 %.

Якщо напруга на лампі розжарювання більша від номінальної на 10 %, то строк експлуатації її зменшиться в 4 рази, а за зменшення напруги на 10 % світловий потік зменшиться до 67 % номінального.

Люмінесцентні лампи менш чутливі до зміни напруги. На кожний 1 % зміни напруги їхній світловий потік також змінюється на 1 %, а світлова віддача – на 0,5 %. Проте за напруги, меншої від номінальної на 6–7 %, ці лампи не загоряються, а за напруги, більшої на 6–7 % за номінальну, перегріваються дроселі та інша допоміжна апаратура.

Нагрівальні прилади у разі зміні напруги змінюють свою потужність пропорційно квадрату напруги. Значне відхилення напруги в силових електромережах може призвести до браку у виробництві і прискореного спрацювання двигунів. Крім того, зниження напруги веде до зменшення потужності електродвигунів. Момент обертання електродвигунів пропорційний квадрату напруги. За напруги, наприклад, що дорівнює 90 % номінальної, момент обертання становитиме 81 % номінального. Напруга (відхилення напруги) у споживачів значною мірою залежить від втрати напруги в елементах електричних мереж. Рівень напруги можна дещо збільшити, підвищивши напругу джерела живлення, але за зміни навантажень коливання напруги джерела буде достатньо великим, а отже, будуть значними і відхилення напруги в споживачів. Крім цього, умови роботи електроізоляції допускають відхилення напруги в установках напругою до 220 кВ не більше 15 %, напругою 330 кВ – не більше 10 %, з вищими напругами – не більше 5 %. Зменшити коливання напруги можна також, збільшивши переріз проводів, але це веде до значного збільшення витрат металу. Техніко-економічні розрахунки показують, що підтримувати рівень напруги в заданих межах доцільніше регулюванням напруги. Регулювання напруги поліпшує режим напруги у споживачів, підвищує якість електроенергії, що поставляється споживачам, збільшує допустиму втрату напруги в мережі, дає змогу зменшувати переріз проводів. Проєктуючи електричні мережі, вибирають засоби регулювання, межі і ступені

регулювання, місця встановлення регуляторів, а також систему автоматизації регулювання напруги.

Напругу електричної мережі можна регулювати: а) зміною напруги на затискачах генератора; б) зміною коефіцієнта трансформації трансформаторів і автотрансформаторів; в) застосуванням статичних конденсаторів і синхронних компенсаторів. Найпоширенішим є централізоване регулювання напруги на шинах підстанцій, до яких приєднано декілька ліній. Якщо таке регулювання не забезпечує належного рівня напруги у деяких груп споживачів (за різних графіків навантаження, великій довжині лінії або підвищених вимогах до якості напруги), то його доповнюють місцевим регулюванням напруги.

3 НАЯВНОСТІ СПОТВОРЕНЬ ЯКОСТІ ЕЛЕКТРОЕНЕРГІЇ. АРХІТЕКТУРА І ХАРАКТЕРИСТИКА МІКРОКОНТРОЛЕРІВ

Спотворення ПЯЕ надають пагубні впливи на розподільну мережу та споживача. При цьому вихід показників за межі норм, що визначаються EN 50160-2010, супроводжується зміною параметрів трифазної системи напруги.

Ідеальна система трифазної напруги характеризується частотою 50 Гц, зсувом фаз, рівним 120º, і рівністю величин напруг у кожній фазі. Для підтримки цих параметрів у межах норми існують організаційні та технічні засоби. До переваг організаційних заходів можна віднести відносну дешевизну та універсальність, але такого підходу найчастіше виявляється недостатньо, тому широкого поширення набули технічні засоби, здатні забезпечити високу якість електроенергії.

3.1 Засоби вимірювання та покращення якості електричної енергії

Вимір шумів у мережі здійснюється спеціальними приладами. Але якщо таких приладів немає, то варто застосовувати додаткові конкретні заходи. Прилад, яким необхідно виміряти перешкоди в електромережі, зазвичай буде живитися від того ж джерела, вимірювання якого необхідно провести. Якщо неправильно підключити дроти, то виникнуть похибки при знятті показань. На рисунку 3.1 нижче зображено схема підключення приладу, за допомогою якого здійснюється вимірювання:

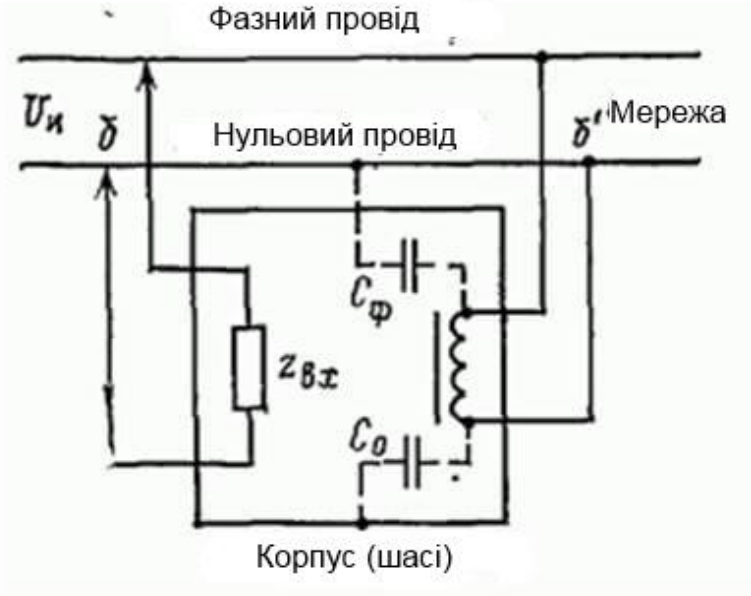


Рисунок 3.1 – Схема підключення вимірювального приладу

Щоб виміряти перешкоди, використовують і осцилограф. За наявності запам’ятовувальної трубки, прилад здатний буде зробити вимір.

3.2 Спеціалізований мікроелектронний програмований прилад

У наш час у домівках, офісних будівлях із кожним днем збільшується кількість електронних пристроїв, отже і споживання енергоресурсів збільшується та їхня вартість. У цьому випадку стає достатньо важливим облік спожитих ресурсів та вживання ефективних заходів для запобігання їхніх втрат. Ефективним рішенням виконання цих завдань є створення системи, яка в режимі реального часу буде контролювати кількість спожитих ресурсів, спроби розкрадання, регулювати навантаження, своєчасно виявляти несправності в лічильнику та контролювати сплату за поживання енергоресурсів споживачів.

Невід’ємною складовою такої системи, для контролю за споживанням та стабільності напруги, є лічильник та стабілізатор напруги. Ці компоненти є одними з найважливіших умов для стабільної роботи будь-якої системи контролю. Головний критерій до системи – це доступність, простота в користуванні, модульність для поступового впровадження. Завдяки розвитку технології Internet of things (IoT – інтернет речей), система контролю за споживанням енергоресурсів має бути бездротовою та керуватися за допомогою мобільного пристрою, щоб людина могла керувати споживанням електроенергії в цілому і не мала доступу до електричного обладнання (електричний щиток, лічильник тощо). Ця система є автономною, але вона може бути реалізована як компонент більшої системи, щоб здійснювати подальші дії без втручання споживача, зберігаючи при цьому можливість контролювати необхідну інформацію. Наприклад, якщо така система буде впроваджена в громадських будівлях (таких як університети, адміністративні будівлі тощо), проблем із підключенням не буде, а заощаджене споживання

ресурсів буде очевидним. Також вона не буде затратною завдяки модульності системи, що дозволить купувати компоненти поступово. Однак найголовніше вирішити проблему, як підключити таку систему до більш глобальної реалізованої системи у вигляді вузла (хабу).

3.3 Історія розвитку мікроконтролерної техніки

Мікроконтролер – це багатофункціональна мікросхема, що містить в собі набір програмно налаштовуваних модулів та інтерфейсів.

Мікроконтролери конструктивно виконані у вигляді однієї мікросхеми і містять всі пристрої, необхідні для реалізації цифрових систем керування мінімальної конфігурації: процесор, запам'ятовувальний пристрій даних, запам'ятовувальний пристрій команд, внутрішній генератор тактових сигналів, а також програмовані модулі для зв'язку з зовнішнім середовищем.

Ось короткий перелік виробів, що будуються на базі мікроконтролерів: автовідповідачі, мобільні телефони, зарядні пристрої, факси, модеми, таймери, системи сигналізації, вимірювальні прилади, електронні лічильники води, газу та електроенергії, дозиметри, прилади автосигналізації, електронні блоки керування авто, промислові контролери, промислові роботи, регулятори температури, вологості, тиску, схеми керування принтерами і плотерами, мережеві контролери, сканери, схеми управління аудіосистемами, системи синтезу мовних повідомлень, відеоігри, системи дистанційного керування, касові апарати тощо.

Термін «мікроконтролер» (МК) витіснив з ужитку раніше використовуваний термін «однокристальна мікроЕОМ». Перший патент на однокристальну мікроЕОМ був виданий в 1971 році інженерам М. Кочрену і Г. Буну, співробітникам компанії «Texas Instruments». Саме вони запропонували на одному кристалі розмістити не тільки мікропроцесор, але й пам'ять, пристрої вводу-виводу. З появою однокристальних мікроЕОМ пов'язують початок ери комп'ютерної автоматизації в галузі керування. Мабуть, ця обставина і визначила термін «мікроконтролер» (control – керування).

У 1980 році фірма «Intel» випускає мікроконтролер i8048. Трохи пізніше в цьому ж році «Intel» випускає наступний мікроконтролер: i8051. Вдалий набір периферійних пристроїв, можливість гнучкого вибору зовнішньої або внутрішньої програмної пам'яті і прийнятна ціна забезпечили цьому мікроконтролеру успіх на ринку. З погляду технології мікроконтролер i8051 був для свого часу дуже складним виробом – у кристалі було використано 128 тисяч транзисторів, що в 4 рази перевищувало кількість транзисторів в 16-розрядному мікропроцесорі i8086.

Популярністю у розробників-початківців користуються 8-бітові мікроконтролери PIC фірми «Microchip Technology» і AVR фірми «Atmel», 16-бітові MSP430 фірми «TI», а також, більш складні та продуктивні – 32-бітові з ARM ядром, архітектуру яких розробляє фірма «ARM» і продає ліцензії іншим фірмам для їхнього виробництва [9].

Мікроконтролери об’єднуються в сімейства. До одного сімейства відносять вироби, які мають однакове ядро – сукупність таких понять, як система команд, циклограма роботи центрального процесора (ЦП), організація пам’яті програм і пам’яті даних, система переривань і базовий набір периферійних пристроїв. Відмінності між різними представниками одного сімейства полягають переважно в складі периферійних пристроїв і обсязі пам’яті програм або даних. Найбільш важлива особливість сімейства центрального процесора – програмна сумісність на рівні двійкового коду всіх вхідних до нього МК.

3.3.1 Microchip

Перші значні зміни відбулися з появою PIC-контролерів фірми «Microchip». Ці чіпи пропонувалися за рекордно низькими цінами, що дозволило їм в короткий термін захопити значну частину ринку мікроконтролерів. До того ж кристали від «Microchip» не поступаються, а часто і перевершують мікроконтролери x51 по продуктивності і не вимагали коштовних засобів програмування.

Разом із контролерами з’явилися дешеві комплекти PICSTART, що містять все, що було потрібно для того, щоб, не маючи ні коштів, ні навичок роботи з PIC-контролерами, швидко створити і налагодити на ньому продукт.

Ці мікроконтролери мали хороші порти, але все інше було зроблено дуже незручно. Архітектура залишала бажати кращого, система команд була вкрай обмежена. Проте PIC – контролери залишаються популярними в тих випадках, коли потрібно створити недорогу електронну систему керування.

3.3.2 Мікроконтролер 8051

Мікроконтролер Intel 8051, що вийшов у 1980 році, став класичним зразком пристроїв цього класу. Цей 8-бітний мікроконтролер поклав початок цілому сімейству мікроконтролерів, які панували на ринку аж до недавнього часу. Більшість фірм виробників мікроконтролерів і сьогодні випускають пристрої, засновані на цій архітектурі. Серед них «Philips», «Atmel», «Dallas», «OKI», «Siemens». На даний момент цей першопроходець давно «залишився в історії».

3.3.3 Atmel

Справжня революція у світі мікроконтролерів сталася у 1996 році, коли корпорація «Atmel» представила своє сімейство чіпів на новому прогресивному ядрі AVR. Більш продумана архітектура AVR, швидкодія, що перевершує контролери Microchip, вдала цінова політика сприяли відтоку симпатій багатьох розробників від недавніх претендентів на звання контролера номер один.

Мікроконтролери AVR мають більш розвинену систему команд, яка налічує до 133 інструкцій, продуктивність, що наближається до 1 мільйону операцій на 1 МГц, Flash ПЗП програм з можливістю внутрішньосхемного перепрограмування. AVR – архітектура оптимізована під мову високого рівня Сі.

Величезну роль зіграла доступність програмного забезпечення і засобів підтримки розробки. У «Atmel» безкоштовно розповсюджуються багато програмних продуктів. Добре відомо, що розвинені засоби підтримки розробок при освоєнні і знайомстві з будь-яким мікроконтролерним сімейством, відіграють не менш значущу роль, ніж самі кристали. Фірма «Atmel» приділяє цьому питанню велику увагу. Надзвичайно вдале й абсолютно безкоштовне середовище розробки AVR Studio, що працює під Windows. Провідні сторонні виробники випускають багатий спектр компіляторів та программаторов, що сприяє ще більщій популяризації.

3.3.4 STMicroelectronics

«STMicroelectronics» – європейська мікроелектронна компанія, одна з найбільших, яка займаються розробкою, виготовленням і продажем різних напівпровідникових електронних і мікроелектронних компонентів. Сьогодні штаб квартира компанії знаходиться в Женеві, у той же час, її холдингова компанія «STMicroelectronics» зареєстрована в Амстердамі, однак компанія історично пов’язана з Італією і Францією. Компанія «ST» одна з перших випустила свої мікроконтролери з ядром Cortex-M0 (2007 р.) і швидко стала домінуючим гравцем на ринку.

4 ПРОГРАМНІ ЗАСОБИ ПІДТРИМКИ ПРОЄКТУВАННЯ ІНТЕЛЕКТУАЛЬНОЇ СИСТЕМИ ОСВІТЛЕННЯ

4.1 Вбудовані системи

4.1.1 Основні поняття вбудованих систем

Отже, що таке вбудовані системи? Можна дати просте визначення: вбудовані системи – це комп’ютерні системи, які зовсім не схожі на комп’ютери. У таких системах складність комп’ютера прихована від користувача [12].

Швидше за все у вас є комп’ютер, наприклад, ноутбук або щось подібне. І швидше за все у вас виникали складності в його використанні. Скажімо, ви хотіли встановити нове програмне забезпечення, але стався конфлікт, і ви не змогли цього зробити. Так іноді буває з комп’ютерними іграми. Ви купили нову гру, а вона не працює, тому що їй потрібна нова версія драйверів відеокарти. Ви встановлюєте нові драйвери, а у вас перестає працювати інша гра, яка працювала до цього, оскільки їй не підходять нові драйвера. Або нові драйвери

можуть взагалі не працювати з вашою відеокартою і треба купити також й нову відеокарту. Всі ці проблеми виникають через універсальність наших «звичайних» комп'ютерів: ми використовуємо їх для вирішення дуже різних завдань, причому ми намагаємося їх вирішувати за допомогою одного і того ж обчислювального пристрою.

З вбудованими системами все не так. Вбудована система зазвичай використовується для якоїсь однієї конкретної задачі. Якщо це камера, то вона повинна знімати зображення або відео. Якщо це автомобіль, то він повинен їздити як автомобіль. Усередині них може перебувати спеціальна комп'ютерна система, але тільки вона вбудована в пристрій і тому її складність прихована від користувача, який знає, як працювати з цими системами через простий інтерфейс.

Наприклад, цифрова камера використовує той самий інтерфейс, що і стара механічна камера – потрібно натиснути кнопку і вона зробить фотографію. Зовні цифрова камера управляється так само, як механічна, але всередині вона влаштована набагато складніше, за рахунок чого користувач отримує додаткові переваги. Йому не потрібно більше проявляти плівку, він може відразу ж побачити результат і навіть більш того – саме тут на камері виконати його попередню обробку, яку раніше можна було зробити тільки на комп'ютері. Ще раз повторимо, що при цьому вся ця складність прихована від користувача за простим і зрозумілим інтерфейсом.

Вбудовані системи не завжди взаємодіють безпосередньо з користувачем: вони можуть робити це через інший пристрій. Що це означає? Давайте розглянемо USB-накопичувач, який зберігає дані. Ця річ, як ми розуміємо, не взаємодіє безпосередньо з людиною, адже у нас немає розніму для підключення такого накопичувача до свого тіла або чогось на кшталт цього! Ми підключаємо його до комп'ютера, і потім взаємодіємо з ним через комп'ютер, щоб отримати доступ до файлів. І це не просто мікросхема пам'яті: у накопичувачі є мікрокомп'ютер, який управляє його роботою. Таким чином, це теж вбудована система, хоча вона безпосередньо не взаємодіє з людиною.

Те ж саме може бути і в інших пристроях, наприклад, антиблокувальна система всередині автомобіля. Людина натискає на педаль гальма, а педаль гальма також пов'язана з антиблокувальною гальмівною системою, таким чином, людина взаємодіє з автомобілем, а в автомобілі взаємодіють один з одним його підсистеми. Важливою особливістю вбудованих систем, що відрізняє їх від звичайних комп'ютерів, є те, що в їхній роботі ключовим фактором є ефективність. Це означає, що не завжди достатньо просто виконувати потрібну функцію, щоб вирішувати завдання. Можна сказати, що спосіб вирішення має бути нестандартним. Наприклад, він має працювати швидко, або він має працювати з низьким енергоспоживанням, або він повинен бути дешевим. І в цьому насправді полягає велика відмінність між розробкою вбудованих систем і, наприклад, традиційним дизайном програмного забезпечення.

Коли вивчають програмування, то як кінцева мета зазвичай ставиться вирішення якогось конкретного завдання. Достатньо написати код, який робить

те, що потрібно, наприклад, сортує список. При цьому найчастіше не потрібно, щоб алгоритм використовував мінімально можливу кількість пам'яті, і щоб список з мільйона елементів сортувався за 0,01 секунди. Іноді такі обмеження важливі і студенти відповідних напрямів підготовки навіть вивчають цілі дисципліни, присвячені ефективним алгоритмам і структурам даних, але на практиці в звичайному програмуванні можуть бути вирішені завдання і не завжди висувають такі підвищені вимоги до ефективності – достатньо, щоб програма просто працювала.

Але для вбудованих систем ефективність дуже важлива. Недостатньо просто змусити таку систему працювати. Вона повинна робити це ефективним способом. Причиною цих обмежень є те, що для більшості цих пристроїв дуже важливо зниження їхньої собівартості, як, наприклад, для пристроїв споживчої електроніки. Або ж від їхньої роботи може залежати життя і здоров'я людей, якщо, наприклад, такі пристрої використовують лікарі або військові.

Обговоримо відмінності в процесі розробки вбудованих пристроїв і звичайних комп'ютерів. Одним із найбільш важливих відмінностей є те, що на відміну від настільних і портативних комп'ютерів, які можуть запускати програми будь-якого типу, вбудовані пристрої зазвичай розробляються під конкретну задачу (або набір пов'язаних завдань). Сучасні телефони є винятком, тому що вони розмивають цю різницю. Телефони починалися як типові вбудовані системи, але сьогодні вони практично перетворилися в повноцінні універсальні комп'ютери. Тим не менш, більшість пристроїв, таких як камери, електроніка автомобіля, побутова техніка та інше зроблені для вирішення конкретних завдань. Наприклад, камера – щоб знімати. І вона робить все, що пов'язано з цією діяльністю камери, не роблячи інших речей. Ця властивість є загальною для більшості вбудованих систем. Це важливо, тому що це змінює спосіб розробки пристрою.

Таким чином, розробка фокусується на одному застосуванні, на відміну від систем загального призначення, таких як ноутбук або настільний комп'ютер. Універсальні ноутбуки та комп'ютери часто виявляються більш потужними, ніж необхідно для вирішення того завдання, для якого вони використовуються. Це відбувається тому, що вони мають бути готові працювати з будь-яким типом обчислювальних задач. Наприклад, візьмемо стандартний комп'ютер, скажімо з чотириядерним процесором. У вас може бути чотириядерний Core i7, який працює на частоті 4 ГГц. Ви зазвичай не використовуєте увесь обчислювальний потенціал цієї машини. Вам зовсім не потрібно чотири процесори для запуску MS Word або PowerPoint, незалежно від того, що ви в них робите. Вам також не потрібно 4 ГГц для запуску цих програм. Комп'ютер встигає зробити мільйони різних операцій до того, як ви надрукуєте чергову букву. Відкрийте, заради інтересу, диспетчер задач свого ноутбука і подивіться на відсоток використання часу центрального процесора. Зазвичай (якщо ви не граєте в гру в цей момент і не рендеруєте 3D-сцену), велика частина додатків спить і тільки одиниці споживають десяті частки відсотка (сам диспетчер задач, наприклад). Отже, 99 % відсотків ресурсів центрального процесора не використовується!

Але іноді ця потужність все-таки потрібна. Наприклад, якщо ви обробляєте фотографії або відео. Такі додатки вимагають всю обчислювальну потужність, яка є в комп'ютері. Інший приклад – це ігри зі складною графікою, що вимагають величезну обчислювальну потужність. Тому, коли ви в них граєте, ви використовуєте всі можливості вашого комп'ютера. Але в решті часу ця потужність не використовується.

У цьому сенсі універсальні комп'ютери дуже неефективні. Якщо ви хочете грати в нові ігри, вам доведеться купити дорогий потужний комп'ютер з гарною відеокартою. Або, ви можете купити ігрову приставку, яка зможе запускати ті ж ігри, може бути навіть швидше, і вона буде дешевше комп'ютера. Приставку можна використовувати тільки для ігор, але вона робить це краще, ніж комп'ютер. Отже, її архітектура більш ефективна для вирішення такого завдання. І більшу частину часу ви використовуватимете її, якщо не на повну потужність, то принаймні на більшу частину потужності, оскільки тільки гратимете на ній.

Таким чином, конкретне застосування дозволяє вплинути на процес розробки системи. Ви можете включити в неї тільки ті блоки, які дійсно необхідні. Завдяки цьому стає можливою більш висока ефективність конструкції.

Ще однією великою відмінністю вбудованих систем порівняно зі звичайними настільними комп'ютерами і ноутбуками є те, що апаратне і програмне забезпечення зазвичай розробляється спільно. Якщо, скажімо, ви хочете купити Microsoft Word або іншу програму, ви, очевидно, можете придбати її окремо від комп'ютера. І їх зробили дві абсолютно різні фірми. Наприклад, ваш ноутбук може бути зроблений фірмою «Asus», а Word – фірмою «Microsoft». Одна фірма робить обладнання, а інша – програми. Для вбудованих систем все навпаки. Саму TV-приставку і програмну оболонку для неї робить одна і та ж компанія. Ми говоримо «зазвичай», тому що, звичайно ж, з усього є винятки. Наприклад, компанія «Apple» робить все сама – і комп'ютери, і ноутбуки, і телевізори, і телефони, і операційні системи, і офісні додатки, іншими словами – весь спектр універсальних і вбудованих систем.

Чому зручно робити вбудовані системи повністю самому? Тому що я можу зробити обладнання саме таким, яке необхідно для запуску саме того програмного забезпечення, що потрібно мені. Я можу зробити їх відповідними один одному. Весь процес проєктування стає більш ефективним. Ми купимо тільки ті компоненти, які потрібні, щоб побудувати систему, яку ми хочемо. Ми не говоритимемо, будь-яка система. Ми візьмемо тільки ті компоненти, які потрібні для нашої конкретної системи. І ми писатимемо код, який використовує тільки їх.

Це відрізняє вбудовані системи від систем загального призначення, у яких потрібно мати обладнання на всі випадки життя і програмне забезпечення для всіх випадків життя, навіть якщо це буде необхідно в цьому житті лише раз.

4.1.2 Компоненти вбудованої системи

У цьому підрозділі ми поговоримо про структуру вбудованих систем. Конкретно, це структура апаратних засобів, різні компоненти, причому основний компонент – мікроконтролер.

На рисунку 4.1 схематично зображена загальна схема. Вбудована система має отримувати дані з зовнішнього світу, обробляти їх і потім виводити дані в зовнішній світ. Так що, насамперед, вона має набір давачів для прийому даних. Ці давачі можуть отримувати інформацію про зовнішній світ по-різному, тому існує багато різних типів подібних пристроїв.

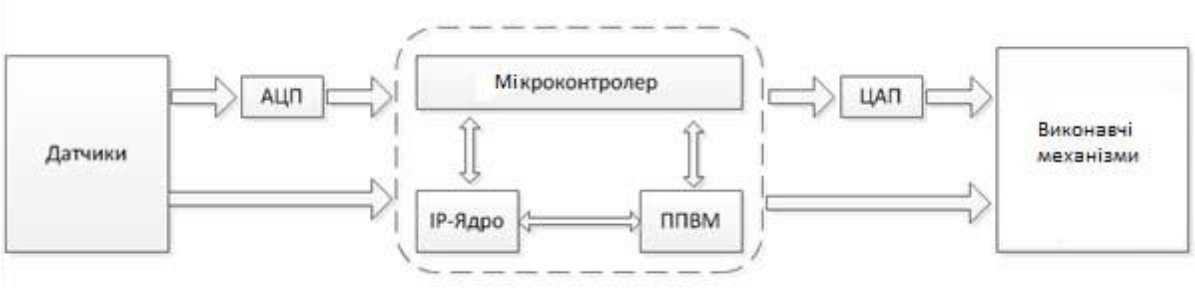


Рисунок 4.1 – Влаштування вбудованих систем [14]

Найпростіший тип давача – це кнопка або щось подібне, яка отримує дані в дуже простому вигляді: натиснута або не натиснута. Система може отримувати і звукову інформацію, використовуючи мікрофон. Відеокамери – це теж давачі, які отримують інформацію у вигляді зображення. Сенсорний екран дозволяє не тільки виводити, а й отримувати інформацію. І так далі – існує велика безліч найрізноманітніших типів давачів, через які вбудована система може отримувати інформацію.

Отже, вхідна інформація надходить у систему і далі вона потрапляє в її ядро, яке займається її обробкою. У кінці цього процесу, коли система вирішила, що робити з інформацією або яке рішення потрібно прийняти на її основі, вона має видати якісь результати. Найчастіше це означає зробити якусь дію в зовнішньому світі. Це робиться за допомогою виконавчих механізмів або приводів, які показані в правій частині схеми на рисунку 4.1.

Виконавчим механізмом може бути світлодіод, який може горіти або блимати. Наприклад, він може сигналізувати, що відеокамера записує або що закінчується батарейка. Але є й інші виконавчі механізми. Наприклад, всередині камери є двигуни, за допомогою яких відбувається управління об’єктивом і наведення різкості. Ці двигуни є виконавчими механізмами, а їхній рух – це вихід системи.

Існує безліч різніх типів виконавчих механізмів: динаміки відтворюють звук, лампи дозволяють системі виводити світло, екран також виводить світло, але в більш детальному і «керованому» вигляді.

Отже, якщо ви зараз подивитесь на вбудовану систему, то побачите, що вона знаходиться посередині. Вона отримує дані від давачів, щось з ними

робить, а потім посилає відповідні сигнали на виконавчі пристрої, щоб змусити їх щось зробити в реальному світі у відповідь на дані, які вона отримала. У цьому сенсі вбудована система забезпечує зв'язок між давачами і виконавчими механізмами.

У центрі вбудованої системи знаходиться мікроконтролер. Більш детально ми обговоримо його в наступних розділах. Крім мікроконтролерів можуть бути й інші компоненти. Два з тих, які зустрічаються доволі часто, показані на рисунку 4.1: це IP-ядра і ПКВМ.

IP – це англійське скорочення від intellectual property, тобто інтелектуальний продукт. IP-ядро становить собою готові блоки для проєктування пристроїв, наприклад, це може бути готова мікросхема, яка виконує одну функцію. Звичайно, «одну функцію» не потрібно розуміти надто буквально – IP-ядро може виконувати кілька функцій, але це тісно пов'язані функції, орієнтовані на якусь конкретну задачу. Отже, цей блок не є універсальною програмованою мікросхемою загального призначення. Навпаки, це мікросхема, яка просто вміє виконувати деякий невеликий набір пов'язаних між собою функцій. Такі пристрої можуть бути дуже корисні, і до того ж вони дешеві в ході виробництва у великих обсягах. Нехай у вас є доволі загальна система, що має велику кількість різних підсистем, як, наприклад, мобільний телефон. Всі мобільні телефони мають виконувати певні алгоритми обробки звуку. Можна зробити спеціальну мікросхему, яка виконуватиме таку обробку і її можна буде продавати виробникам телефонів. Оскільки телефонів виробляють дуже багато, то і мікросхем буде потрібно багато. Щоб зробити одну таку мікросхему, буде потрібно багато часу, праці і грошей і це було б вкрай невигідно, але, якщо необхідно провести сотні тисяч або навіть мільйони подібних мікросхем, – то собівартість кожної з них стає дуже низькою.

IP-ядра роблять для поширених завдань – для тих, які виникають знову і знову і на які є великий попит. Якщо якась задача зустрічається тільки в одній конкретній системі, то для неї не має сенсу робити IP-ядро.

IP-ядра мають взаємодіяти з мікроконтролером, який є центром всієї системи. Він керує IP-ядрами і командує, коли вони повинні починати роботу, передає їм інформацію і отримує результат. Наприклад, якщо IP-ядро виконує стиснення відео, то мікроконтролер каже йому, коли починати стискати дані і які дані стискати. Для цього мікроконтролер передає йому певну послідовність сигналів, а коли мікросхема закінчує свою роботу – вона посилає певні сигнали мікроконтролеру, повідомляючи, що результат готовий.

Інший вид компонентів, який застосовується у вбудованих системах, це вентильні матриці, або ПКВМ. ПКВМ – це апаратні пристрої, а якщо бути точніше – апаратно програмовані мікросхеми. Що це означає? Звичайна мікросхема складається з величезної кількості побудованих на базі транзисторів логічних вентилів. Вентилі також з'єднані між собою дуже складними численними зв'язками, за якими, власне, і подорожують керуючі та інформаційні сигнали. Схема об'єднання цих мікроскопічних пристроїв і обумовлює логіку виконуваних мікросхемою операцій, але тільки ця схема «друкується» на заводі.

4.1.3 Мікроконтролери

У цьому параграфі ми продовжимо говорити про апаратні компоненти вбудованих систем, а саме про мікроконтролери.

Мікроконтролер є центром вбудованої системи, тому в цьому навчальному посібнику ми приділимо багато уваги саме роботі з ним. На рисунку 4.2 ви бачите одну з плат Arduino. Це не мікроконтролер, це друкована плата з безліччю елементів, серед яких, придивившись, ви зможете побачити велику чорну прямокутну мікросхему. Ось це якраз і є мікроконтролер, який виконує програми, що керують пристроєм.

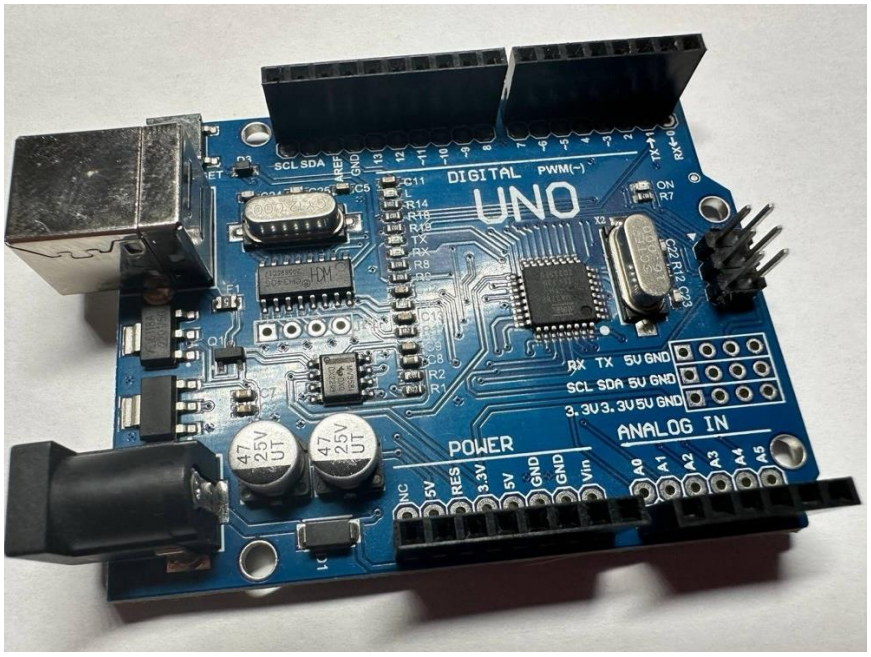


Рисунок 4.2 – Плата Arduino UNO

Отже, робота цієї мікросхеми полягає в тому, щоб виконувати код, і це центр всієї системи. Він зчитує дані з інших компонентів, і він також керує іншими компонентами. Чим відрізняється мікроконтролер від мікропроцесора, який використовується в звичайних комп'ютерах? Мікроконтролер зазвичай менший і слабший, ніж мікропроцесор, тому, коли говорять про останній, то мають на увазі мікросхему від компанії «Intel» або «AMD» , що встановлюється в настільний комп'ютер або ноутбук, і яка працює на дуже великій швидкості.

Мікроконтролер ж зазвичай істотно слабший. При цьому з погляду функціонування він робить приблизно те ж саме, але тільки у вбудованих системах. З цим пов'язані й обмеження. Всі компоненти вбудованих систем мають бути енергоефективними і дешевими і для них, найчастіше, не потрібна продуктивність, як у процесора з 6 ядрами і частотою 4 ГГц. Якщо потрібно реалізувати конкретну взаємодію з користувачем, то така продуктивність швидше за все і не потрібна, тому для вбудованих систем зазвичай потрібен дешевий процесор, що споживає мало енергії.

Яка продуктивність мікроконтролерів в абсолютному вираженні? Зараз доволі часто використовується частота від 16 МГц, хоча може бути навіть і менше, наприклад, 8 МГц або навіть 4 МГц. Це майже в кількасот разів повільніше, ніж процесор звичайного комп'ютера. Звичайно, можуть застосовувати і більш швидкі мікросхеми – до 500 МГц або навіть до 1 ГГц. Це потрібно, наприклад, для обробки відеозображень.

Інша відмінність від звичайних комп'ютерів в тому, що в звичайних комп'ютерах процесор і пам'ять – це окремі мікросхеми, причому зазвичай пам'ять становить кілька мікросхем. Ми можемо змінювати і додавати пам'ять незалежно від процесора. У мікроконтролерах початкового рівня все об'єднано в одну мікросхему, тобто за своєю суттю такий мікроконтролер – це міні-комп'ютер, зроблений на одній мікросхемі, на якій знаходиться і мікропроцесор, і пам'ять, й інші необхідні компоненти. Зазвичай пам'яті у них теж набагато менше, ніж в настільних комп'ютерах, але такий варіант може бути зручний, оскільки потрібно тільки мінімальну кількість зовнішніх компонентів, щоб мікропроцесор міг працювати. У найпростіших випадках потрібно тільки живлення.

Отже, мікроконтролер – це інтегрована мікросхема, яка виконує програму. На рисунку 4.3 мікроконтролер виглядає трохи по-іншому, ніж ті, що ми бачили раніше. На цьому рисунку ми заглянули всередину чорної оболонки і побачили те, як насправді виглядає мікросхема. Це маленький кремнієвий чіп, захований в захисний корпус. На рисунку 4.3 видно мікропроцесор після того, як чіп був встановлений в корпус.

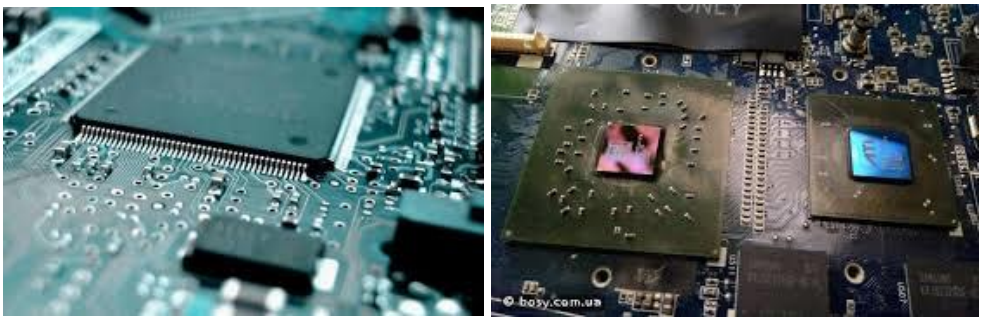


Рисунок 4.3 – Мікроконтролер

Сам корпус є чорним матеріалом, який оточує чіп. Він потрібен насамперед для захисту, а крім того, для охолодження мікросхеми, оскільки допомагає більш ефективно відводити тепло. Це необхідно, тому що всі чіпи сильно нагріваються під час роботи. Настільки сильно, що якщо ви спробуєте торкнутися відкритого чіпа під час його роботи, то отримаєте такі ж відчуття, як і від розпеченої лампочки (за кількістю тепла, що виділяється, сучасні процесори можна порівняти зі звичайними 100-ватними лампочками).

Крім того, ви бачите, що по всьому периметру чіпа знаходяться контактні площадки. Це те, за допомогою чого мікросхема спілкується із зовнішнім світом. Вони з'єднані дротами з чіпом і виведені на маленькі шматочки металу

на зовнішній стороні корпусу, так щоб можна було припаяти мікроконтролер до інших компонентів системи.

Існує багато мов програмування, на яких можна писати подібні програми. У межах цього конспекту лекцій використовується мова Сі для програмування мікроконтролера, встановлена на платі Arduino (див. рис. 4.2). Крім Сі використовують й інші мови. Перш ніж виконати програму, очевидно, потрібно десь зберегти. Отже, всередині мікроконтролера має бути пам’ять для програми. Зараз це зазвичай флеш-пам’ять, така ж, як в USB-накопичувачах.

Флеш-пам’ять – це незалежна пам’ять, тобто така, з якої дані не стираються після відключення живлення. Коли живлення з’являється знову, мікроконтролер починає виконання програми з самого початку.

Де, на чому і як пишуть програми для подібних пристроїв? Зазвичай це роблять на звичайному ноутбуці або настільному комп’ютері. Це пов’язано з тим, що як вже згадано раніше, мікроконтролери порівняно повільні і слабкі.

Рисунок 4.4 – Програматор

Тому програму пишуть і компілюють на потужному комп’ютері, а потім остаточна версія компільованої програми має бути записана в пам’ять мікроконтролера. Робиться це за допомогою спеціального пристрою, що називається програматором (рис. 4.4).

Це невелика плата, яка за допомогою USB приєднується до комп’ютера, а до розташованого на ній слоту підключається мікроконтролер. Є спеціальні програми на комп’ютері, які зв’язуються з цим пристроєм і записують через нього програму. Це один із способів запрограмувати мікроконтролер. З таким програматором мікроконтролер програмують окремо, а потім вже встановлюють в систему, де він працюватиме. Іноді буває так, що в пристрої вже передбачений спеціальний рознім, і тоді мікроконтролер програмують прямо в системі. Але в нашому випадку ми так не робитимемо, тому що ми використовуємо Arduino. Для неї не потрібен окремий програматор, оскільки він вже вбудований в плату. Можна просто взяти Arduino і, підключивши її до USB порту ноутбука або настільного ПК, програмувати її безпосередньо.

4.2 Середовища розробки програмного забезпечення для мікроконтролерів AVR

Перші мікроконтролери AVR були розроблені в дослідницькому центрі Atmel в Норвегії групою інженерів, до складу якої входили Alf Bogen та Vergard Wollan. Ініціали їхніх імен та посилання на тип архітектури (Risc architecture) і сформували назву «AVR». Перший AVR мікроконтролер AT90S1200 був випущений у 1996–1997 р.

Сьогодні AVR-мікроконтролери за показниками «ціна – швидкодія – енергоспоживання» вважають одними з найкращих серед 8-бітових контролерів з RISC архітектурою. Об'єм їхнього продажу подвоюється щороку. По інформації, що наведена в огляді фірми «Atmel» Product Selection Guide, Volume 11, 2015, відзначається, що на сьогоднішній день випущено біля 7 мільярдів AVR-мікроконтролерів.

Сфери застосування AVR-мікроконтролерів надзвичайно широкі – від найпростіших приладів до складних систем збору обробки інформації та керування, що застосовуються у побуті та промисловості.

Перший офіційний каталог мікроконтролерів фірми «Atmel» було випущено в 1997 році. До складу цього каталогу ввійшли мікроконтролери першого сімейства Classic. А вже в 1999 році у другому випуску каталогу були приведені дані мікроконтролерів трьох сімейств: Classic, Mega, Tiny.

З 2008 р. фірма «Atmel» почала серійний випуск нового сімейства XMega 8-бітових AVR-мікроконтролерів.

Сьогодні фірмою «Atmel» виробляються 8, 8/16 та 32-розрядні AVR-мікроконтролери. Всього випускається 294 різних типів AVR-мікроконтролерів.

4.3 Інтегроване середовище розробки програм AVR Studio

Для створення нового проєкту необхідно вибрати у головному меню «**Project (Проект)|New... (Новий) »**. У вікні, що з'явилося, в рядку «**Project name (Ім'я проекту) »** потрібно ввести ім'я нового проєкту, наприклад «new». Далі, натиснувши лівою кнопкою мишки на рядку «**AVR Assembler»,** вибрати тип проєкту (рис. 4.5). Натиснути кнопку «**OK»**.

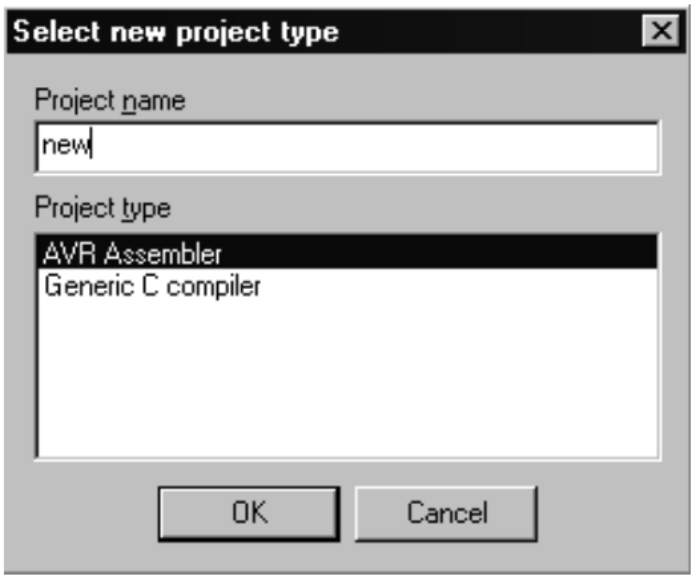


Рисунок 4.5 – Вікно вибору типу проєкту

У вікні, що утворилось, необхідно натиснути правою кнопкою мишки на стрічці «**Assembler Files»**, у меню, що з’явилося (рис. 4.6) натиснути лівою кнопкою мишки на стрічці «**Add File... »** (Додати файл).

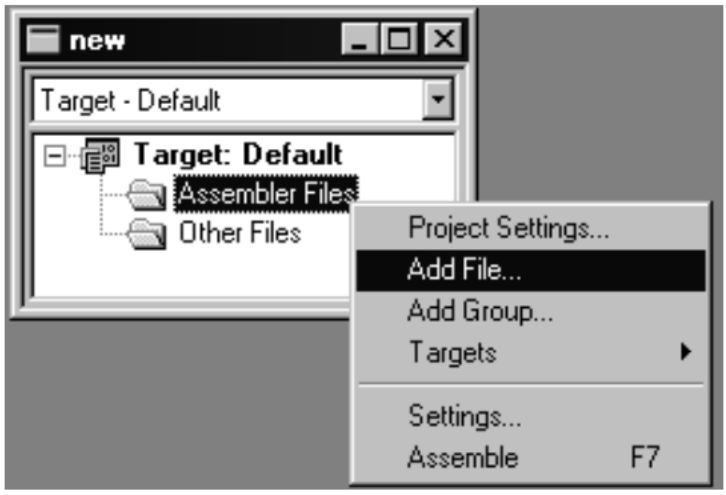


Рисунок 4.6 – Список файлів проєкту

У вікні «**Open file»** необхідно створити порожній текстовий файл, змінити ім’я файлу майбутньої програми та надати йому розширення **.asm**. Таким чином буде створений порожній файл робочої програми з розширенням **.asm**. Якщо у вас вже існує файл програми (з розширенням **.asm**), тоді у вікні «**Open file»** потрібно вибрати цей файл. Для того щоб зробити створений файл основним, необхідно натиснути правою кнопкою мишки на імені створеного файлу в вікні проєкту, а потім у меню, що з’явилося, натиснути лівою кнопкою мишки на рядку «**Assembler entry file»** (рис. 4.7). Тепер необхідно відкрити

створений файл, двічі натиснувши на ньому лівою кнопкою мишки у вікні проєкту. У вікні, що з'явилось, можна набирати текст програми.

Рисунок 4.7 – Вибір основного файлу

Після введення тексту програми необхідно натиснути клавішу **F7,** щоб провести компіляцію програми. Після цього з'являється вікно (рис. 4.8), у якому міститься інформація про правильність написаної програми. У разі виявлення помилок повідомляється про всі рядки, у яких виявлено помилки. Натиснувши на рядок, у якому повідомляється про помилку, потрапимо в те місце програми, де виявлено помилку. У разі завершення компіляції без помилок, тобто в вікні (рис. 4.8), нижній рядок буде мати вигляд **Assembly complete with no errors** (програма відкомпільована без помилок), можна розпочати відлагодження роботи програми. Після натиснення клавіші **F11** почнеться процес відлагодження. Для перегляду результатів роботи програми із меню «**View**» (Вид) треба викликати вікна «**Registers**» **(**Регістри), «**Processor**» (Процесор), «**New Memory View**» (Новий вид пам'яті). Вікно «**Registers**» відображає поточний зміст регістрів загального призначення **R0**-**R31**. Вікно «**Processor**» відображає поточне значення регістрових пар (X, Y, Z), значення програмного лічильника, значення покажчика стеку та всіх восьми прапорців регістру стану.

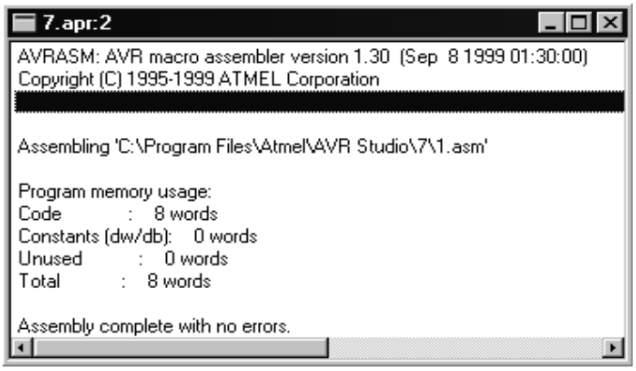


Рисунок 4.8 – Вікно результатів компіляції коду програми

Вікно «**New Memory View»**, залежно від обраного режиму, відображає поточний зміст пам'яті даних, пам'яті програм, пам'яті введення-виведення, пам'яті EEPROM. Робоче вікно програми набуде вигляду (рис. 4.9):

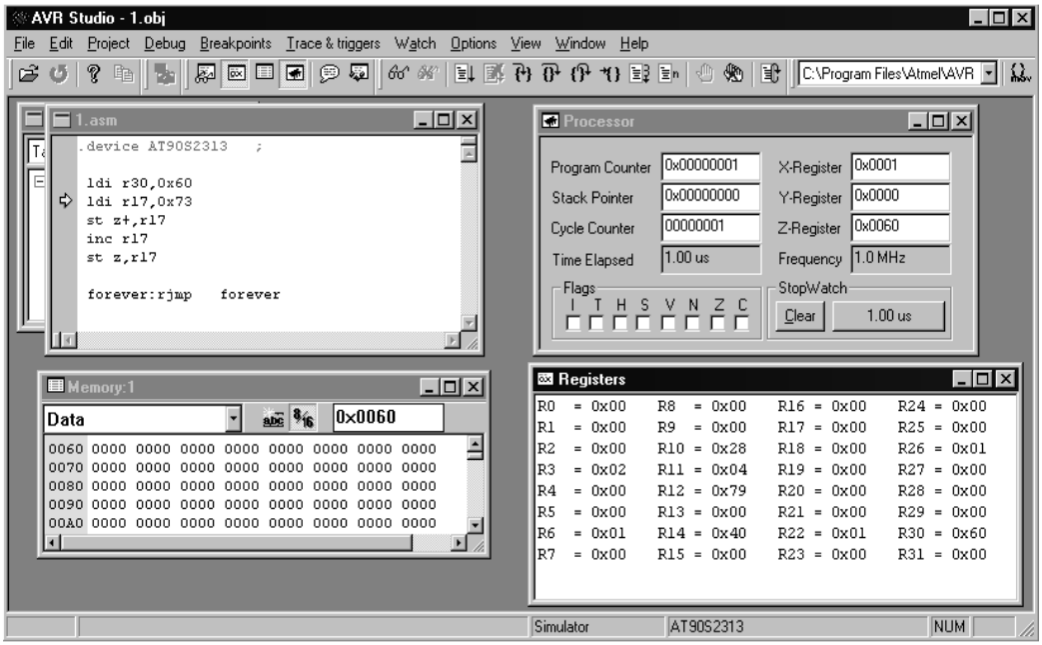


Рисунок 4.9 – Робоче вікно програми

Після натискання клавіші **F11 (Trace into)** відбувається виконання поточних команд, що відображається відповідними змінами у вікнах. Компілятор працює з вихідними файлами, що містять інструкції, мітки і директиви. Інструкції і директиви зазвичай мають один чи декілька операндів.

Примітка. Рядок коду не повинен бути довшим 120 символів. Будь-який рядок може починатися з мітки, що є набором символів та закінчується двокрапкою. Мітки використовуються для вказівки місця, у яке передається керування при переходах, а також для завдання імен змінних.

Вхідний рядок може мати одну з чотирьох форм:

– [мітка:] директива [операнди] [Коментар];

– [мітка:] інструкція [операнди] [Коментар];

– коментар;

– порожній рядок.

Коментар починається після крапки з комою (;) і до кінця рядка ігнорується компілятором.

4.4 Робота з графічним інтерфейсом користувача AVR STUDIO 4

Після завантаження **AVR STUDIO 4** з'явиться діалогове вікно. Для створення нового проєкту потрібно натиснути «[**Create New Project**]» (Створити новий проєкт).

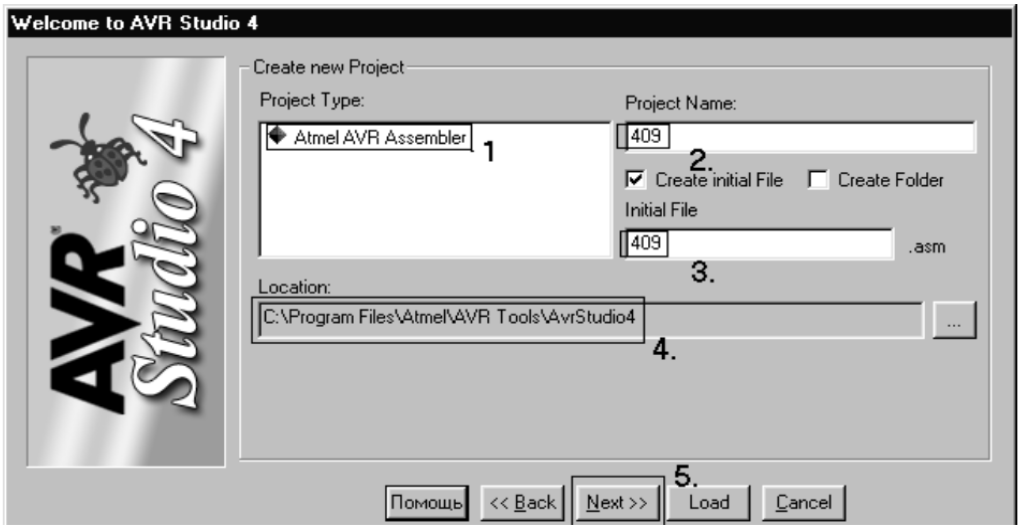


Рисунок 4.10 – Встановлення налаштувань проєкту

Другим етапом є встановлення налаштувань проєкту. На цьому етапі задаються розширення, назва й адреса файлу, що створюється (рис. 4.10). Для цього необхідно виконати такі дії:

– у вікні «**Project type»** необхідно обрати Atmel AVR Assembler;

– у вікні «**Project Name»** необхідно ввести ім'я проєкту та для створення нового файлу встановити прапорець «**Create initial File»**;

– вказати шлях збереження файлів проєкту.92

Третім етапом є вибір системи налагодження. Програмне забезпечення **AVR STUDIO 4** має у своєму складі програми налагодження для широкого спектра задач (рис. 4.11), що передбачає:

– вибір вбудованого симулятора-налагоджувача;

– вибір пристрою, для якого створюється програма.

Четвертим етапом є написання програми. На початку цього етапу необхідно зберегти створений файл кнопкою «**Save»** у файловому меню. Після введення тексту програми необхідно натиснути клавішу **F7,** щоб провести компіляцію програми. Після цього з'являється вікно, у якому міститься інформація про правильність написаної програми. У разі виявлення помилок повідомляється про всі рядки, у яких виявлено помилки. Натиснувши на рядок, у якому повідомляється про помилку, у робочій програмі буде показано місце, де виявлено помилку. У разі завершення компіляції без помилок, тобто у вікні нижній рядок буде мати вигляд **Assembly complete with no errors** (програма відкомпільована без помилок). Тепер можна розпочинати відлагодження роботи програми (аналогічно як для **AVR STUDIO 3**).

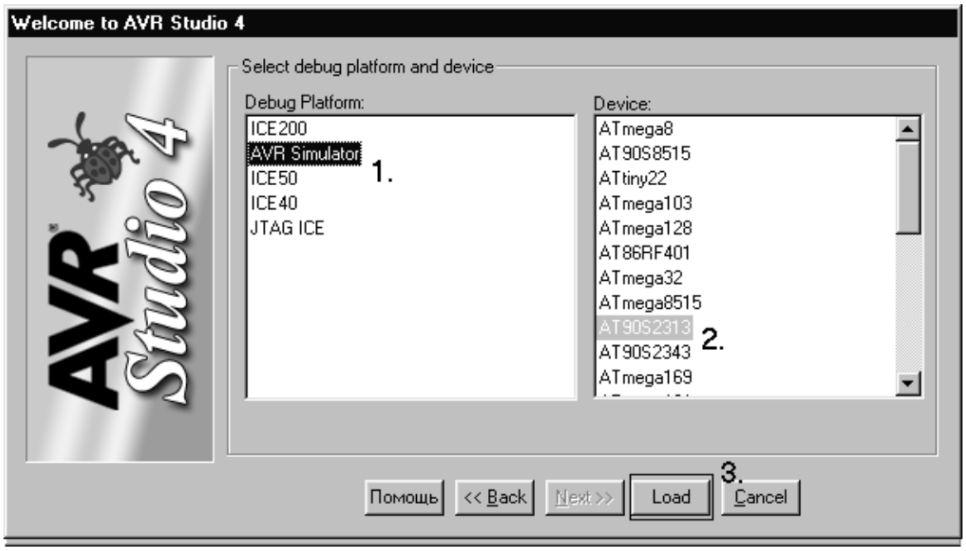


Рисунок 4.11 – Вибір симулятора та мікроконтролера

4.5 Програмне забезпечення для програмування мікроконтролерів

Процес написання програм для МК AVR як і для будь-яких інших МК, складається з декількох етапів:

– підготовка вихідного тексту програми на якій-небудь мові програмування;

– компіляція програми;

– налагодження й тестування програми;

– остаточне програмування й підготовка до серійного виробництва.

Мікропрограма пристрою має бути написана на одній з мов програмування. Зараз для МК AVR існують декілька мов програмування, а також різних засобів підтримки розробки, що використовують одну мову, але різняться за функціональністю. На кожному з етапів необхідне застосування спеціальних програмних й апаратних засобів. Варто відзначити, що базовий набір програмного забезпечення (компілятор асемблера, ПЗ для програмування) поширюється фірмою «Atmel» безкоштовно. Однак за доволі довгий період часу, що пройшов з моменту появи цих МК, з’явилася велика кількість програмного забезпечення сторонніх виробників.

4.5.1 Компілятори асемблера та мови С

Вибір компілятора асемблера є найменш принциповим питанням, оскільки сам процес компіляції (перетворення вихідного тексту програми в машинний код) виконується доволі однозначно. Власне, всі відмінності між асемблерами полягають у можливостях процесора, що обробляє макрокоманди, наявності текстового редактора або середовища розробки й типу операційної системи, що

підтримується **AVR Studio.** Доволі вдалим вибором при розробці програмного забезпечення для МК сімейства AVR може стати інтегроване середовище розробки AVR Studio фірми Atmel [3, 9, 19, 20], що містить у собі текстовий редактор із підсвічуванням синтаксису, компілятор асемблера, симулятор, відладчик й інтерфейс із апаратними емуляторами. Програма розрахована на роботу під керуванням операційних систем Windows 9x, 2000, XP, Vista. До недоліків AVR Studio можна віднести деяку нестабільність роботи налагоджувача, а також неповну симуляцію периферійних пристроїв (зокрема, відсутня симуляція АЦП). До плюсів же відноситься насамперед підтримка практично всіх МК сімейства AVR.

Останнім часом усе популярніше стає використання компіляторів мов високого рівня під час написання програм для МПС. Найбільше поширені компілятори мови С, оскільки в цій мові найбільш просто реалізуються всі необхідні можливості із керування апаратними засобами МК.

5 АПАРАТНІ ЗАСОБИ ПІДТРИМКИ МІКРОКОНТРОЛЕРІВ AVR

Донині поширеним способом створення прототипу майбутнього пристрою був такий: підібравши електронні компоненти, розробник брав у руки паяльник і збирав на макетних платах окремі вузли або пристрій в ціле. Далі починався процес налагодження: виправлення помилок принципової схеми, встановлення режимів роботи, уточнення параметрів компонентів, що застосовуються. Цей варіант не втратив своєї актуальності і нині, але застосовується тільки при розробках простих пристроїв, при виробництві одиничних екземплярів і при обмеженому бюджеті. Але якщо на перший план виходять такі критерії, як швидкість, зручність, надійність розробки, то без професійних засобів не обійтися.

До переваг використання засобів розробки можуть належати:

− зменшення часу виходу готової продукції;

− зменшення матеріальних витрат і ризику під час розроблення;

− використання власних ресурсів для прискорення розробки;

− вільне використання власних розробок надалі.

До спеціалізованих професійних інструментальних засобів налагоджування належать:

1. Оціночні та демонстраційні плати (Evaluation & demonstration board). Дозволяють швидко ознайомитися з тим чи іншим приладом або набором приладів та в короткі терміни розробити на ньому пристрій.

2. Внутрішньосхемні емулятори (In-circuit emulator). Найбільш потужні й універсальні налагоджувальні інструменти, що становлять набір апаратно-програмних засобів, які дозволяють заміщувати собою емульований мікроконтролер у реальній схемі.

3. Програмні симулятори (Simulator). Програмні засоби, здатні імітувати роботу мікроконтролера і його пам’яті.

4. Відлагоджувачі (Debugger). Свого роду міст між розробником і пристроєм налагодження, що дозволяє користувачеві одночасно контролювати перебіг виконання програми і бачити відповідність між вихідним текстом, образом програми в машинних кодах і станом всіх ресурсів емульованого мікроконтролера.

5. Емулятори ПЗП (ROM emulator). Програмно-апаратні засоби, здатні заміщувати ПЗП на платі, що відлагоджується, шляхом підстановки замість нього ОЗП і завантаження програми за допомогою комп'ютера через один із стандартних інтерфейсів.

6. Програматори (Programmer). Пристрої, що дозволяють програмувати мікросхеми пам'яті, МК і програмовані логічні інтегральні схеми.

7. Інтегровані середовища розробки (Integrated Development Environment).

5.1 Внутрішньосхемні відлагоджувачі

Найважливіший інструмент будь-якого розробника програмного забезпечення – внутрішньосхемний відлагоджувач. Він з'єднує мікроконтролер з ПК через спеціальний інтерфейс для налагодження та програмування, так відкривається доступ до внутрішніх алгоритмів функціонування мікроконтролера. Завантаження програми в мікроконтролер займає всього кілька секунд, при цьому не потрібно видаляти або перепаювати будь-які елементи. Запрограмований відлагоджувач може управляти процесором, запускаючи виконання програми, зупиняючи або реалізуючи її в покроковому режимі, а також зчитувати дані в режимі реального часу з усіх областей пам'яті мікроконтролера і регістрів введення / виведення. Для пристроїв AVR існує три внутрішньосхемних відлагоджувача [15, 17].

AVR Dragon. Внутрішньосистемний відлагоджувач для 8- і 32-розрядних мікроконтролерів AVR з підтримкою внутрішньосхемного налагодження (OCD). З його допомогою можна виконувати символьне налагодження на всіх пристроях з підтримкою внутрішньосхемного налагодження (OCD) з використанням режимів SPI, JTAG, PDI (на окремих пристроях), високовольтного послідовного програмування і паралельного програмування, а також стандартне налагодження через інтерфейси SPI, JTAG, PDI. Робоча область дозволяє створювати власні схеми або додавати гнізда для бажаних пристроїв. Відлагоджувач підтримує технологію NanoTrace (залежно від модуля OCD на пристрої AVR) з використанням пам'яті цільового пристрою.

JTAGICE 3. Засіб розробки середнього рівня для 8- і 32-розрядних мікроконтролерів Atmel AVR з підтримкою внутрішньосхемного налагодження для проведення символьного налагодження на рівні вихідного коду, роботи в режимі NanoTrace (якщо підтримується пристроєм) і програмування.

AVR ONE!. Професійний інструмент розробки для всіх 8- та 32-розрядних пристроїв AVR з підтримкою внутрішньосхемного налагодження (OCD). Відлагоджувач Atmel AVR ONE! застосовується для символьного налагодження на рівні вихідного коду, трасування програм та програмування пристроїв. Він забезпечує повний цикл розробки і є найшвидшим

налагоджувальним інструментом, що пропонується компанією «Atmel», дозволяє програмувати в режимах SPI, JTAG, PDA і aWire, а також здійснювати налагодження через інтерфейси debugWIRE, JTAG, PDI і aWire. Також підтримує функцію LiveDebug, і допоміжний інтерфейс Nexus для високошвидкісного трасування програм, даних на частоті до 200 МГц у буферному або потоковому режимі. Заснований на потужній матриці FPGA, що забезпечує високошвидкісну передачу даних між ПК-хостом і пристроєм AVR, сприяючи швидкому завантаженню програм та зменшенню часу відгуку під час налагодження.

5.2 Засоби розробки для мікроконтролерів різних фірм

Сучасні засоби розробки для мікроконтролерів AVR фірми «Atmel» та сторонніх виробників розглянуто в [5, 14, 15, 17]. ICE 200 – внутрішньосхемний емулятор фірми «Atmel» для налагодження пристроїв на МК набору AVR (ATtiny12, AT90S2313, AT90S2333/4433, AT90S4414/8515, AT90S4434/8535). Емулятор підключається до пристрою за допомогою спеціальної емуляційної головки і може використовуватися з платами, на яких встановлені панельки DIP8, DIP20, DIP28 і DIP40.

JTAGICE – внутрішньосхемний JTAG емулятор з фоновим налагодженням програми користувача, який може бути використаний як внутрішньосхемний програматор для мікроконтролерів AVR, що мають JTAG-інтерфейс. STK500 – система відлагодження, яка полегшує роботу з AVR-мікроконтролерами та забезпечує підтримку програмування через паралельний та послідовний інтерфейси. Додатково система може бути використана як ISP-програматор. Використовуючи інтегровану середу розробки AVR Studio ATSTK500, забезпечує режими симуляції та емуляції, а також внутрішньосхемного програмування AVR-мікроконтролерів.

STK501 – модуль розширення до STK500, що дозволяє збільшити список пристроїв, що підтримуються: ATmega64, ATmega103, ATmega128. STK502 – модуль розширення, розроблений для підтримки мікроконтролерів ATmega169. У ATSTK502 входять роз’єми, перемички й апаратні засоби для повної підтримки всіх особливостей mega169, а також управління вбудованим LCD. STK600. Це повноцінний початковий комплект із системою розробки для 8- та 32-розрядних мікроконтролерів AVR. Комплект оснащений функціями моделювання та випробування нових проєктів. Пристрої AVR взаємодіють з комплектом STK600 за допомогою інноваційної сендвіч-системи плат для підключення та маршрутизації, тобто передачі сигналів від пристрою до відповідного обладнання. Сендвіч-система складається з базової плати з гніздами, на якій розміщується пристрій AVR, і плати маршрутизації сигналів (для кожного пристрою своя карта), яка направляє сигнали до різних блоків основної плати STK600 залежно від пристрою. Така система спрощує апаратне налаштування при переході від одного мікроконтролера AVR до іншого, оскільки всі сеанси зв’язку пристрою з материнською платою визначаються платою маршрутизації, специфічною для такого пристрою. Система

маршрутизації повністю пасивна. У перенаправленні сигналів не беруть участь електронні компоненти, тому всі контакти введення / виведення на роз'ємах доступні безпосередньо, без необхідності зміни електричних схем. Комплект забезпечує доступ до всіх виводів пристрою, а також підтримку корисних апаратних функцій таких, як: кнопки, світлодіодні індикатори та флешпам'ять даних Atmel DataFlash, становлять повноцінну систему моделювання та випробування нових рішень. Безкоштовні програмні платформи AVR Studio і AVR32 Studio дозволяють розробникам писати і компілювати внутрішнє програмне забезпечення для мікроконтролерів, використовуючи асемблер або мову С, а також завантажувати готові програми в цільові пристрої AVR. Плата також оснащена регульованим джерелом напруги і регульованим генератором тактових імпульсів.

5.3 Апаратні засоби підтримки проєктування та відлагодження систем

Логічні аналізатори є універсальними приладами для аналізу функціонування цифрових систем [14]. Вони дозволяють контролювати логічний стан декількох десятків точок системи протягом заданого проміжку часу і видати інформацію про їхній стан у візуальному (на екрані монітора) або друкованому вигляді. Форма подання може бути символьна або графічна (часові діаграми сигналів). Вхідні канали аналізатора підключаються до точок контролю за допомогою зондів-кліпсів. Кількість каналів у сучасних аналізаторах зазвичай становить від 16 до 200. Запуск аналізатора проводиться автоматично при надходженні на певні канали заданого коду (адреси, даних або комбінації керувальних сигналів) або послідовності кодів. Після запуску в пам'ять аналізатора записується послідовність значень логічних сигналів у точках контролю. Обсяг цієї пам'яті визначає число контрольованих точок на часовій осі (глибину контролю), яке може становити від декількох тисяч до сотень тисяч. На екран виводяться кілька десятків точок для кожного каналу з можливістю перегляду всієї записаної в пам'яті послідовності станів. Максимальна частота дискретизації часових інтервалів для різних моделей аналізаторів має значення від декількох десятків до сотень мегагерц.

Логічні аналізатори, які випускаються провідними виробниками електронно-вимірювальної апаратури: «Hewlett-Packard», «Tektronix», «John Fluke» і низкою інших, – реалізуються у вигляді автономних вимірювальних приладів з власним дисплеєм або аналізаторів блоків, що підключаються до персонального комп'ютера. Наведемо основні характеристики логічних аналізаторів компанії «Hewlett-Packard», які широко використовуються для налагодження систем на базі мікроконтролерів [5]. Аналізатори серії HP1660 є автономні прилади, які дозволяють контролювати від 34 до 136 каналів з глибиною до 8К точок, забезпечуючи контроль логічних станів з частотою до 100 МГц при роздільній здатності 2 нс (частота дискретизації 500 МГц). Окремі моделі цієї серії містять також двоканальний осцилограф зі смугою пропускання 250 МГц або генератор послідовностей 32-розрядних тестових сигналів із частотою до 200 МГц. Аналізатори серії HP 1670 забезпечують

глибину контролю до 128 К точок при інших аналогічних параметрах. Ці серії логічних аналізаторів найбільш ефективні для налагодження систем на базі 8-розрядних мікроконтролерів набору AVR. Логічні аналізатори є універсальними приладами для аналізу функціонування цифрових систем [14]. Вони дозволяють контролювати логічний стан декількох десятків точок системи протягом заданого проміжку часу і видати інформацію про їхній стан у візуальному (на екрані монітора) або друкованому вигляді. Форма подання може бути символьна або графічна (часові діаграми сигналів). Вхідні канали аналізатора підключаються до точок контролю за допомогою зондів-кліпсів. Кількість каналів у сучасних аналізаторах зазвичай становить від 16 до 200. Запуск аналізатора проводиться автоматично після надходження на певні канали заданого коду (адреси, даних або комбінації керувальних сигналів) або послідовності кодів. Після запуску в пам'ять аналізатора записується послідовність значень логічних сигналів у точках контролю. Обсяг цієї пам'яті визначає число контрольованих точок на часовій осі (глибину контролю), яке може становити від декількох тисяч до сотень тисяч. На екран виводяться кілька десятків точок для кожного каналу з можливістю перегляду всієї записаної в пам'яті послідовності станів. Максимальна частота дискретизації часових інтервалів для різних моделей аналізаторів має значення від декількох десятків до сотень мегагерц.

Особливо корисним у процесі налагодження цифрових систем є використання осцилографів змішаних сигналів, які дозволяють одночасно контролювати логічний стан і значення аналогових сигналів у різних точках схеми. У цьому класі засобів налагодження оптимальним за комплексом технічних і експлуатаційних характеристик є прилад HP54645D компанії «Hewlett-Packard» [16], який містить двоканальний цифровий осцилограф із смугою пропускання 100 МГц (частота дискретизації 200 МГц), чутливістю 10 мВ/поділ, і 16-канальний логічний аналізатор з частотою дискретизації 400 МГц, глибиною контролю до 2М точок. Комбінація синхронного запуску осцилографа і логічного аналізатора з великою глибиною контролю відкриває нові можливості для виявлення причин виникнення збоїв і завад під час роботи мікроконтролерних систем. Такий прилад дозволяє визначати причини виникнення завад на шинах живлення, виявляти імпульсні завади в лініях зв'язку, контролювати спільну роботу аналогових і цифрових блоків, налагоджувати процедури послідовного обміну даними при спотвореннях форми переданих сигналів, а також розв'язувати багато інших проблем, які доволі важко вирішити іншими способами. Схемний емулятор (СЕ) становить програмно-апаратний комплекс, який у процесі налагодження заміщає мікроконтролер в реалізованої системі.

До структури типового СЕ входять такі блоки:

− емулятор мікроконтролера (у голівці);

− пам'ять траси, яка зберігає значення сигналів, що встановлюються на виводах мікроконтролера в процесі виконання програми;

− блок контрольних переривань, який реалізує зупинки в контрольних точках, заданих користувачем із клавіатури комп'ютера;

− емуляційна пам’ять (ОЗП), яка замінює в процесі налагодження внутрішнє ПЗП мікроконтролерів або інші розділи пам’яті, зовнішній доступ до яких під час налагодження обмежений;

− таймер, який використовується для контролю часу виконання фрагментів програми.

СЕ дозволяє вводити в систему тестову або робочу програму і контролювати її виконання, забезпечуючи переривання в контрольних точках. Умовами переривання можуть бути різні комбінації значень адреси, даних і керувальних сигналів, що надходять на виводи мікропроцесора або мікроконтролера, що емулюється. Ці комбінації задаються користувачем з клавіатури комп’ютера, що управляє. Після зупинки користувач може отримати на екрані інформацію про поточний стан будь-яких регістрів і комірок пам’яті системи. За допомогою пам’яті траси можна переглянути стан системної шини для певної кількості попередніх циклів виконання програми. Дизасемблер дозволяє аналізувати виконання програми відповідно до її вихідного тексту мовою асемблера. Пам’ять траси працює аналогічно пам’яті логічного аналізатора, тому СЕ може виконувати також і його функції. Об’єм пам’яті траси в різних СЕ дозволяє контролювати від 4 К до 512 К програмних циклів. Таймер слугує для визначення часу виконання фрагментів програми з урахуванням реальної тактової частоти системи. Програмне забезпечення СЕ складається з монітора – службової програми, що забезпечує роботу всіх блоків під керуванням базового комп’ютера, транслятора, що дозволяє програмувати роботу системи мовою високого рівня або Асемблера, і відлагоджувача. Ці програмні засоби зазвичай функціонують у складі інтегрованого середовища програмування-налагодження. Більшість сучасних СЕ використовують символьні відлагоджувачі і дизасемблери, застосування яких робить процес налагодження більш простим і наочним. Програмне забезпечення СЕ реалізує видачу даних на екран монітора в зручному для користувача форматі. Деякі моделі СЕ надають можливості аналізу ефективності виконуваної програми, забезпечуючи інформацію про частоту звернення до певних її фрагментів, і дозволяють виконувати налагодження мультипроцесорних систем за допомогою організації багатоемуляторних комплексів. СЕ, що реалізують перераховані вище функції, називають налагоджувальними комплексами або системами розвитку (development system).

6 СТАНДАРТИ МОВ ПРОГРАМУВАННЯ C/C++

С – проста й вишукана мова програмування, що була розроблена співробітником фірми «*Bell Labs*» Денісом Рітчі спільно з Кеном Томпсоном у 1972 році [16]. Спочатку її використовували як інструментальну мову операційної системи *UNIX*. Однак вона виявилась настільки вдалою, що доволі швидко стала однією з найпопулярніших мов системного програмування. Швидке зростання популярності привело до розробки в 1983 році стандарту мови С, після

чого її стали використовувати повсюдно. Мова С має низку переваг, які роблять її універсальним інструментом користувача МК техніки:

– простота;

– відносно висока швидкодія та відносно малий розмір програм, написаних на С;

– високий рівень мобільності програм, написаних на С;

– суміщення можливостей мови низького рівня зі зручністю мови програмування високого рівня.

Все перелічене вище робить мову С зручним засобом для розробки програм різноманітного призначення. Однак програмування – це галузь знань, що доволі швидко розвивається, виникають нові концепції та методи програмування. Нові методи вимагають нових можливостей, нових засобів розробки. Так розвиток концепції об'єктно-орієнтованого програмування привів до модифікації мови програмування С та розробки на її основі мови «С з класами», яка в подальшому розвинулась у мову програмування С++ [16]. Мова С++ вдало поєднала в собі переваги мови С з перевагами об'єктно-орієнтованого програмування, завдяки чому на сьогодні вона стала одним з основних засобів розробки сучасного програмного забезпечення.

6.1 Елементи мови *С*. Алфавіт мови. Константи. Ідентифікатори. Ключові слова. Коментарі. Оператори

Під елементами мови програмування розуміють її базові конструкції, які використовують при розробці програм [10], а саме:

– алфавіт;

– константи;

– ідентифікатори;

– ключові слова;

– коментарі.

З елементів мови формуються лексеми. Лексема – це одиниця тексту програми, яка має самостійний зміст для компілятора мови С. Лексема не може містити інших лексем. Алфавіт – це сукупність дозволених в мові символів чи груп символів, що розглядаються як єдине ціле. У програмах на С використовують дві множини символів: множину символів мови С та множину відображуваних символів. Множина символів мови С містить літери, цифри та знаки пунктуації, які мають певний сенс для компілятора мови С. Множина символів мови С є підмножиною відображуваних символів. Множина відображуваних символів складається із всіх літер, цифр і символів, що можуть бути представлені як окремий символ на клавіатурі персонального комп'ютера. Оператори, ідентифікатори, службові слова та інші елементи програми на мові С можуть містити тільки символи із множини символів мови С, однак символьні змінні, символьні константи й коментарі можуть містити будь-який символ із множини відображуваних символів. Множина символів мови С містить великі (A-Z) та малі (a-z) літери латинського алфавіту й арабські цифри (0-9). Літери й цифри використовують при формуванні констант,

ідентифікаторів і ключових слів. Компілятор мови С розрізняє великі й малі літери, вважаючи їх різними символами. Символи «пробіл», «табуляція», «перехід на нову строку», «повернення каретки», «нова сторінка», «вертикальна табуляція», «горизонтальна табуляція» називають пробільними, оскільки вони виконують ті ж функції, що й пробіли між словами в тексті. Розглядається як пробільний і символ кінця файлу. Компілятор мови С ігнорує ці символи, якщо їх використовують не в складі символьних змінних і констант. Коментарі компілятор мови С також розглядає як пробільні символи. Розділові символи з множини символів мови С використовують для різноманітних призначень. Розділові символи наведено у таблиці 6.1. Ці символи мають спеціальний зміст для компілятора мови С. Правила їхнього використання будуть розглянуті пізніше. Константа – це число, символ або строка символів. Константи використовують для визначення постійних величин. В С розрізняють чотири типи констант: цілі, з плаваючою крапкою, символьні константи й символьні рядки.

Таблиця 6.1 – Деякі розділові символи

Символ	Назва	Символ	Назва
1	2	3	4
,	Кома	.	Крапка
;	Крапка з комою	:	Двокрапка
?	Знак питання	`	Одинарні лапки (апостроф)
!	Знак оклику	|	Вертикальна лінія
/	Слеш, знак ділення	\	Обернений слеш
~	Тильда	_	Підкреслювання
(	Ліва кругла скобка	)	Права кругла скобка
{	Ліва фігурна скобка	}	Права фігурна скобка
[	Ліва квадратна скобка	]	Права квадратна скобка
<	Знак «менше»	>	Знак «більше»
%	Відсоток	&	Амперсанд
#	Ґратка	^	Стрілка вгору
-	Знак мінус	=	Знак рівності
+	Знак плюс	*	Знак множення

Цілі константи – це набір цифр, які становлять ціле число. Цифри записують підряд без будь-яких інших символів між ними. Цілі константи можуть бути десятковими, вісімковими або шістнадцятковими.

Десяткові константи складаються з арабських цифр «0»–«9». Десяткова константа починається з ненульової цифри за винятком випадку, коли константа дорівнює 0. Вісімкові константи складаються з арабських цифр «0»–«7». Вісімкова константа завжди починається з цифри 0, що є ознакою вісімкової константи. Шістнадцяткові константи складаються з арабських цифр «0»–«9» та літер «A» – «E» або «a» – «e».

Літери заміняють цифри із значеннями «10»–«15» відповідно. Шістнадцяткова константа завжди починається з 0x або 0X, що є ознакою шістнадцяткової константи.

Цілі константи можуть бути додатними або від'ємними. Перед від'ємними константами ставиться знак «-», перед додатними – «+». Константа без знаку вважається додатною. Наведемо кілька прикладів цілих констант:

123 – десяткове ціле число 123;

–12 – десяткове ціле від'ємне число, модуль якого дорівнює 12;

012 – вісімкове ціле число. В десятковій системі його значення – 10;

0x12 – шістнадцяткове ціле число. В десятковій системі його значення – 18.

Ідентифікатори – це імена змінних, функцій і міток, які використовують в програмі. Ідентифікатор вводять при визначенні змінної або функції, або як мітка оператора. Ідентифікатор може складатися з літер, цифр, символів підкреслення, але починатися він може лише з літери або символу підкреслення. Проте, хоч формально ідентифікатор може починатися із символу підкреслення, бажано їх не використовувати. Це пов'язано з тим, що такі ідентифікатори використовуються як системні.

Регістр має значення. Наприклад, змінні Dog та dog з точки зору компілятора С є різними змінними. Довжина ідентифікатора може бути довільна, проте значущими є перші 32 символи. Якщо перші 32 символи двох ідентифікаторів співпадають, то такі ідентифікатори вважаються однаковими.

Ключові слова – це слова, які в мові програмування С мають спеціальне значення.

Перелік ключових слів:

auto do for return switch
break double goto shot typedef
case else if signed union
char enum int sizeof unsigned
continue extern long static void
default float register struct while.

Ключові слова не можна використовувати як ідентифікатори, вони можуть бути використані лише так, як це передбачено мовою програмування С. У наведеному вище списку подано лише ті ключові слова, які описані в стандарті мови С. Кожна конкретна реалізація компілятора може мати свої власні додаткові ключові слова, які не увійшли до наведеного вище списку. Повну інформацію про ключові слова для конкретного компілятора можна отримати із супровідної документації на конкретний компілятор.

Коментарі – це будь-яка послідовність символів, що міститься між парами символів «/*» та «*/» [10]. Коментарі вводять для документування програми. Компілятором вони сприймаються як окремий пробільний символ та ігноруються під час компіляції. Коментарі можуть займати кілька рядків тексту.

Оператори – це основні елементи програми. Власне програма складається з послідовності операторів. Оператори керують процесом виконання програми. Оператор в мові С – це вираз, що закінчується символом « ; ».

Мова програмування С містить повний набір операторів структурного програмування:

– оператори присвоювання;
– умовний оператор (if else);
– порожній оператор (;);
– складений оператор ({ } блочний оператор);
– оператор покрокового циклу (for);
– оператор циклу з передумовою (while);
– оператор циклу з післяумовою (do while);
– оператор продовження циклу (continue);
– оператор розриву (break);
– оператор вибору (switch);
– оператор безумовного переходу (goto);
– оператор виклику функції;
– оператор повернення із функції (return).

Умовний оператор використовують тоді, коли, залежно від якоїсь умови, необхідно виконати один з двох операторів. Умова – це вираз, що складається з допустимих в С операцій. Порожній оператор – це оператор, який складається лише з символу «крапка з комою» – « ; ». Цей оператор не виконує ніяких дій. Його використовують там, де за синтаксисом повинен бути оператор, але при цьому потрібно, щоб не виконувалось ніяких дій. Складений оператор в мові С – це кілька операторів, розміщених між символами фігурних дужок « { » та « } ». Складений оператор використовують у тих випадках, коли за синтаксисом С у цьому місці може знаходитись лише один оператор, а за алгоритмом програми необхідно виконати кілька операторів.

6.2 Типи даних мови програмування *С*. Функції. Директиви препроцесора

Одним із ключових питань під час програмування практичних задач є правильна організація даних, оскільки від цього залежить ефективність використання пам’яті та процесорного часу. Для спрощення цієї задачі в С реалізовано набір базових типів даних і механізм розробки користувачем нестандартних типів даних на основі базових. Базові типи даних умовно можна розділити на прості та структуровані. До простих типів даних відносяться цілі типи й типи з плаваючою крапкою. Перелік простих типів та їх характеристики наведено у таблиці 6.2.

Таблиця 6.2 – Прості типи

Тип	Розмір, біт	Діапазон значень	
signed char, char	8	– 128…127	
signed int, signed	16	залежить від реалізації	
signed short int, signed short, short	16	– 32768…32767	
signed long int, signed long, long	32	– 2147483648…2147483647	
unsigned char	8	0…255	
unsigned int, unsigned	16	залежить від реалізації	
unsigned short int, unsigned short	16	0…65535	
unsigned long int, unsigned long	32	0…4294967295	
		діапазон модуля	точність (число десяткових цифр)
float	32	$3.4*10^{-38}$ …$3.4*10^{38}$	7
double, long float	64	$3.4*10^{-308}$ …$3.4*10^{308}$	15
Long double	80	$3.4*10^{-4932}$ …$3.4*10^{4932}$	19

Для опису в програмі змінних базових типів достатньо вказати тип і після нього перерахувати через кому ідентифікатори змінних, які повинні мати цей тип. Масив – це набір однотипних елементів. Кожен елемент має свій номер (індекс). Всі елементи масиву впорядковані за своїм індексом. При описі масиву вказується тип його елементів, ім’я масиву й кількість елементів масиву – n.

Структура – це складний тип даних, що об’єднує кілька змінних (однакових або різних типів) в єдине ціле для організації зручного доступу до них. Елементи структури називають полями. Назви полів у межах однієї структури повинні бути унікальними, в той же час у різних структурах можна використовувати поля з однаковими назвами. Опис структури здійснюється за допомогою ключового слова «struct».

Об’єднання – це засіб, який дозволяє звертатись до одних і тих же даних (у різні моменти часу) по-різному, як до кількох різних змінних різних типів. Синтаксис об’єднань аналогічний синтаксису структур за винятком того, що замість ключового слова «struct» використовується ключове слово «union».

Перерахування (enumeration) – це синонім слова «список». У програмах на С перераховний тип – це іменований список цілих констант. Кожна з констант має власне ім'я. Імена констант повинні бути унікальними. Для створення такого списку використовують ключове слово «Enum».

Покажчик – це змінна, яка містить адресу іншої змінної (вказує на цю змінну). Покажчики є ефективним засобом доступу до значень змінних. В С покажчики використовують доволі інтенсивно. Достатньо сказати, що ім'я масиву є не що інше, як покажчик на перший елемент масиву. Задається покажчик за допомогою символу « * ».

Програмування мовою С базується на понятті функції. Що ж таке функція? Функція – це самостійна одиниця програми, спроєктована для реалізації конкретної задачі. Використання функцій дозволяє більш ефективно використовувати результати попередніх робіт при розробці нової програми й створювати бібліотеки готових рішень для задач, які часто зустрічаються в програмуванні. До складу кожної програми на С входить, як мінімум, одна функція з назвою «main», з якої розпочинається виконання програми. Окремі частини визначення можуть бути відсутні. Наприклад, функція може не мати аргументів. Може також бути опущений тип результату, тоді, за замовчуванням, тип результату int.

Визначення функції може міститись у будь-якому з файлів проєкту. Якщо функцію необхідно виконати в іншому файлі, то в ньому повинен знаходитись опис функції. У мові програмування С сама функція не є змінною, але є можливість визначити покажчик на функцію і працювати з ним, як зі звичайною змінною: присвоювати, розміщувати в масиві, передавати як параметр функції, повертати як результат виконання функції тощо. Фактично покажчик на функцію містить адресу першої команди функції, на яку він вказує. Це може бути зручно в тому випадку, якщо наперед невідомо, яку функцію необхідно викликати в певному місці програми і це залежить від результатів деяких попередніх подій, визначених у програмі.

Однією з важливих особливостей мови програмування С є наявність препроцесора. Він становить макропроцесор, який використовується для обробки тексту програми перед компіляцією. Компілятор мови С сам викликає препроцесор, однак препроцесор може бути викликаний і незалежно від компілятора. Обробка програми препроцесором керується директивами препроцесора.

Директиви препроцесора – це інструкції, записані в програмі на мові С. Їх зазвичай використовують для того, щоб полегшити модифікацію програм під час переходу на іншу платформу, при розробці кількох варіантів програми (наприклад повної та демонстраційної версій). Директиви препроцесора дозволяють замінити лексеми програми деякими значеннями, додати до файлу з програмою інший файл, виконати умовну компіляцію (відключення деяких частин програми від проєкту за виконання певних умов) тощо. Директиви препроцесора мови С завжди починаються із символу «#».

Препроцесор C розрізняє такі директиви:

#define #else #if #ifdef #ifndef
#elif #endif #error #include #pragma
#undef.

Умовна компіляція – це зручний засіб для створення кількох версій однієї програми. Наприклад, під час розроблення робочої, навчальної та демонстраційної версій програми замість того, щоб розробляти три різних проєкти, можна задати три різних сценарії компіляції одного проєкту. Зробити це можна за допомогою директив умовної компіляції – #if, #elif та #endif.

Директиви умовної компіляції дозволяють вказати блоки коду, які будуть включені в кінцеву програму залежно від версії програми.

6.3 Структура програми на С

Програма на мові програмування С становить сукупність розглянутих вище елементів: директив препроцесора, описів змінних, функцій, операторів. Програма може складатися з довільної кількості перелічених елементів. Порядок їхнього розміщення має значення. Зокрема, будь-яка змінна повинна бути описана до першого її використання. Порядок розміщення директив препроцесора може суттєво впливати на кінцеву програму.

Будь-яка програма на С повинна містити функцію з назвою «main». Саме з цієї функції починається виконання програми. Вона визначає дії, що виконуються програмою, та здійснює, за необхідності, виклик інших функцій.

Сама по собі бібліотека мови С не є частиною мови, однак закладений в ній набір функцій, типів і макросів складає системне середовище, що підтримує стандарт мови С. Ми розглянемо лише основні бібліотечні засоби стандарту ANSI [16]. Для більш детального знайомства з бібліотеками різних версій мови С пропонуємо звернутись до технічної документації.

6.4 Засоби *С*++, не пов’язані безпосередньо з об’єктно-орієнтовним програмуванням

У С++ можна використовувати два види коментарів: звичайні, оформлені за правилами С, та однорядкові, що починаються з символів «//» і продовжуються до кінця рядка [17]. Мова С вимагає, щоб всі описи локальних змінних усередині блоку поміщалися перед першим виконуваним оператором. У С++ можна описувати змінні в будь-якій точці блоку до першого їхнього використання. Часто це буває дуже зручно.

Змінні, визначені таким чином, є локальними. Область доступності таких змінних від місця їхнього визначення і до кінця блоку, в якому вони визначені. Час життя змінної визначається часом виконання цього блоку від місця визначення змінної. У «класичному» С наявність прототипів функцій не є обов’язковою. Це часто призводить до появи помилок, які важко виявити, оскільки компілятор не може перевірити, чи відповідають визначенню цієї функції типи аргументів і тип

значення, що є результатом дії функції. C++ більш суворий: він вимагає, щоб в кожному модулі, де відбувається звернення до функції, було присутнє або її оголошення (прототип), або її визначення, де вказано типи аргументів і тип результату. Якщо в модулі знаходиться прототип функції, то її визначення може знаходиться в іншому модулі. C++ дозволяє використання аргументів за замовчуванням. Ці аргументи використовують, якщо при виклику функції передано недостатню кількість аргументів. У класичному C локальна змінна перекриває глобальну змінну з тим же ім'ям, і нема засобів для доступу до такої глобальної змінної в цій ситуації. У C++ існує оператор дозволу області доступності « :: », який дає можливість звернутися до такої глобальної змінної. Модифікатор const, як і в звичайному C, забороняє зміну значень змінних. Така константа повинна отримати значення при описі, оскільки надалі їй нічого не можна присвоїти.

У C++ константи, визначені за допомогою модифікатора const, мають таку ж область доступності, як і статичні змінні. Вони не доступні в інших модулях програми, навіть якщо в цих модулях описати їх як extern. Тому, якщо якусь константу необхідно використовувати в різних модулях проєкту, то вона повинна бути визначена в кожному модулі, де потрібно її використовувати.

Здебільшого компілятор трактує константу, описану за допомогою модифікатора const, так само, як і іменовану константу, створену директивою #define, тобто просто підставляє у відповідних місцях величину, що відповідає значенню цієї константи. Проте модифікатор const має перевагу перед #define, оскільки забезпечує можливість контролю типів на етапі компіляції, що дозволяє уникнути багатьох помилок, які важко визначити.

Модифікатор volatile, навпаки, повідомляє компілятор, що значення цієї змінної може бути змінено фоновим процесом, наприклад, при обробці переривань. З погляду компілятора це означає, що при обчисленні виразів, що містять таку змінну, потрібно працювати безпосередньо з тією ділянкою пам'яті, де вона записана, а не з її копією, що знаходиться в регістрі процесора. Операція динамічного розподілу пам'яті – це одна з найбільш важливих і часто використовуваних операцій у будь-якій більш-менш серйозній програмі, тому дуже важливо мати зручні засоби для її виконання. У класичному C для цієї мети існують функції «calloc», «malloc», «realloc» та «free». За допомогою цих операторів можна ефективно керувати розподілом пам'яті під різні динамічні об'єкти. Проте під час їхнього використання програміст повинен дуже чітко орієнтуватися в роботі з покажчиками й чудово уявляти структуру об'єктів, під які виділяється пам'ять, тому динамічний розподіл пам'яті в класичному C викликає священний трепет навіть у бувалих програмістів.

6.5 Об'єктно-орієнтоване програмування (ООП)

Архітектура сучасних операційних систем стає все більше й більше об'єктно-орієнтованою. Працювати в таких системах неможливо без володіння основами ООП, тому розглянемо основні поняття та принципи ООП. Основна ідея об'єктно-орієнтованого програмування полягає в об'єднанні даних і алгоритмів для їхньої обробки в єдине ціле, в деякий абстрактний тип даних. У C++ цей тип,

створений програмістом, прийнято називати класом. Клас специфікує члени-дані, які називають полями, і операції для роботи з цими даними, які називають методами [19].

Як вже наголошувалося, клас – це тип даних. Конкретний екземпляр цього типу називається об'єктом цього класу. Фактично будь-який з раніше розглянутих типів даних можна, у деякому розумінні, вважати класом, а конкретну змінну цього типа – об'єктом. У основі ООП лежать три основні поняття – інкапсуляція, успадкування і поліморфізм.

Інкапсуляція – це об'єднання в єдине ціле даних (полів) і алгоритмів для їхньої обробки (методів). Успадкування – це властивість класів, яка дозволяє розробляти нові (породжені) класи на основі вже існуючих (базових). При цьому породжені класи можуть успадковувати властивості базових.

Поліморфізм – це властивість споріднених класів, породжених від загального базового класу, вирішувати схожі за змістом задачі різними способами. Фактично, об'єктно-орієнтоване програмування – це модульне програмування нового рівня, коли основний акцент переноситься на смисловий зв'язок між даними й процедурами їхньої обробки. Доступ до відкритих полів класу здійснюється так само, як в структурах і об'єднаннях: з використанням операторів прямого «.» та опосередкованого «->» вибору. Кожен створюваний об'єкт класу має свою базову адресу. Таким чином, однойменні члени різних об'єктів цього класу ніяк не пов'язані між собою. А як же бути якщо необхідно, щоб всі об'єкти цього класу мали деякий загальний член? Немає нічого простішого. Достатньо описати його як static. Іноді виникає необхідність, щоб якась функція, що не є членом класу, мала доступ до закритих полів об'єктів цього класу. У C++ є така можливість. Для цього необхідно оголосити цю функцію другом цього класу.

Під час роботи з об'єктом програма виконує низку дій, а саме:

– виділення пам'яті під об'єкт;

– ініціалізація об'єкта;

– знищення об'єкта.

Для виконання цих операцій в компіляторі передбачений стандартний набір дій. Проте часто цей набір не задовольняє розробників. Це буває, наприклад, якщо потрібно задати значення за замовчуванням для тих чи інших полів об'єкта, або якщо механізм ініціалізації і знищення об'єктів цього класу залежить від ситуації тощо. У C++ передбачена можливість заміни стандартних процедур шляхом використання конструкторів і деструкторів.

Конструктор – це функція-метод класу з таким самим ім'ям, як у класу. Цей метод призначений для створення та ініціалізації об'єктів класу, виділення пам'яті під об'єкти й присвоювання початкових значень полям об'єкта.

Деструктор – це функція-метод класу, яка відповідає за коректне вивільнення пам'яті під час знищення об'єкта. Ім'я функції-деструктора завжди таке саме, як і у класу, до якого додається символ «~» на початку. Конструктори й деструктори не можуть повертати значень, і тому у їхньому описі відсутній тип результату. Конструктор може не мати вхідних параметрів. У такому випадку його називають конструктором за замовчуванням. Клас може мати кілька конструкторів і лише один деструктор. Під час використання кількох конструкторів діють ті самі правила, що і для перевантажених функцій. У мові C++, як і в C, відсутні

вбудовані оператори вводу-виводу. У програмах на C++ операції вводу-виводу зазвичай здійснюються за допомогою функцій, що містяться в стандартних бібліотеках. У C++ при організації вводу-виводу ключовим поняттям є потік.

Відповідно до концепції C++ потік визначається, як деякий абстрактний тип даних, що представляє якусь послідовність елементів даних, направлену від джерела до приймача. Кількість елементів потоку називають його довжиною length, а порядковий номер деякого доступного в цей момент елемента називають поточною позицією. Кожен потік має один з трьох можливих режимів доступу:

– тільки для читання;

– тільки для запису;

– для читання і запису.

На основі поняття потоку в C++ для вводу-виводу створена стандартна бібліотека потокового вводу-виводу iostream. У цій бібліотеці визначені стандартні потоки вводу-виводу:

а) cin – надає послідовний доступ на читання стандартного вхідного потоку. Зазвичай він пов’язаний з клавіатурою, але може бути перевизначений;

б) cout – надає послідовний доступ на запис у стандартний вихідний потік. Зазвичай він пов’язаний з дисплеєм, але може бути перевизначений;

в) cerr – надає доступ на запис у стандартний потік помилок. Зазвичай він пов’язаний з дисплеєм. Від потоку cout відрізняється тим, що коли стандартний вихідний потік перевизначений у файл, то цей потік все одно пов’язаний з дисплеєм;

г) clog – повністю аналогічний потоку cerr, але використовує буферизацію.

Крім того, у цій бібліотеці визначені перевантажений оператор форматованого вводу даних із потоку « >> », перевантажений оператор форматованого виведення даних у потік « << », маніпулятори й прапори, що визначають формати даних, а також ряд функцій неформатованого вводу-виводу.

Самі потоки в C++ реалізовані як класи. У основі ієрархії класів лежить клас ios, на основі якого породжені класи ostream (описує вихідний потік) та istream (описує вхідний потік).

7 ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ AVR

7.1 Загальна інформація

МК набору AVR підтримують такі режими програмування: режим послідовного програмування (по інтерфейсу SPI); режим високовольтного паралельного програмування (HVPP); режим програмування через інтерфейс JTAG.

Під високою напругою розуміють напругу керування (12 В), що подається на вивід RESET мікроконтролера для переведення останнього в режим програмування. Які з режимів програмування підтримують деякі МК AVR, можна дізнатися, звернувшись до таблиці 7.1.

До того ж, МК AVR мають можливість самопрограмування, тобто зміна вмісту пам’яті програм під керуванням самого мікроконтролера. У процесі програмування можуть виконуватися такі операції:

- знищення кристала (chip erase);
- читання / запис FLASH-пам’яті програм;
- читання / запис EEPROM-пам’яті даних;
- читання / запис конфігураційних комірок;
- читання / запис комірок захисту;
- читання комірок ідентифікатора;
- читання байта калібрування.

Таблиця 7.1 – Режими програмування мікроконтролерів набору AVR

Режим програмування	ATitiny13	ATitiny25	ATitiny2313	ATmega8515x/8535x	ATmega8x	ATmega16x/32x	ATmega64x/128x	ATmega48x/88x/168x	ATmega162x
1. Послідовний (інтерфейс SPI)	+	+	+	+	+	+	+	+	+
2. Високовольтний паралельний	+	+	+	+	+	+	+	+	+
3. По інтерфейсу JTAG						+	+		+

Усі моделі мікроконтролерів поставляються зі стертою пам’яттю програм та пам’яттю даних (у всіх комірках знаходиться число $FF) і придатні до негайного програмування. Конфігураційні комірки (Fuse Bits) визначають різні параметри конфігурації мікроконтролера. Ці комірки розташовані в окремому адресному просторі, доступному тільки під час програмування. Усі конфігураційні комірки згруповані в кілька байтів, а склад цих комірок залежить від конкретної моделі мікроконтролера.

Однак, на відміну від конфігураційних комірок, комірки ідентифікатора доступні тільки для читання. Значення коду пристрою (комірка $ 02) у різних моделей може збігатися. Тому пристрій варто ідентифікувати тільки за сукупністю значень комірок $ 01 і $ 02, оскільки саме ця пара чисел є унікальною для кожного мікроконтролера. У калібрувальні комірки під час виготовлення мікроконтролера заносяться константи калібрування, які призначені для підстроювання на номінальну частоту внутрішнього RC-генератора. Кількість цих комірок залежить від того, на скількох частотах може працювати внутрішній RC-генератор.

У моделях ATmega8515x/8535x і ATmega8x/16x/32x/64x/128x є чотири 8-бітних комірки, а в інших моделях – одна комірка. Знаходяться вони в старших байтах адресного простору комірок ідентифікатора (одна комірка – за адресою $ 000, чотири комірки – за адресами $ 000, $ 001, $ 002 і $ 003 для частот 1, 2, 4 і 8 МГц відповідно). Завантаження калібрувальної константи в регістр OSCCAL здійснюється апаратно при знаходженні мікроконтролера в стані скидання. Однак у моделях ATmega8515x/8535x і ATmega8x/16x/32x/64x/128x генератор автоматично калібрується тільки на частоту 1 МГц. Тому при використанні іншої частоти RC-генератора його калібрування необхідно здійснювати вручну. Для цього вибору програматор під час програмування має прочитати вміст комірки калібрування і занести його за будь-якою адресою FLASH-пам’яті програм. А програма має після старту зчитати це значення з пам’яті програм і занести його в регістр OSCCAL.

7.2 Програмування по послідовному каналу

У режимі програмування по послідовному каналу програмування пам’яті програм і даних здійснюється по послідовному інтерфейсу SPI. Цей режим зазвичай використовується для програмування (перепрограмування) мікроконтролера безпосередньо в пристрої.

Схему ввімкнення мікросхем у режимі програмування по послідовному каналу наведено на рисунку 7.1. На цьому ж рисунку наведено два варіанти розведення колодки для підключення програматора, рекомендовані компанією «Atmel». Як видно з рисунка 7.1, для обміну даними між програматором і пристроєм використовуються три лінії: SCK (тактовий сигнал), MOSI (вхід даних) і MISO (вихід даних).

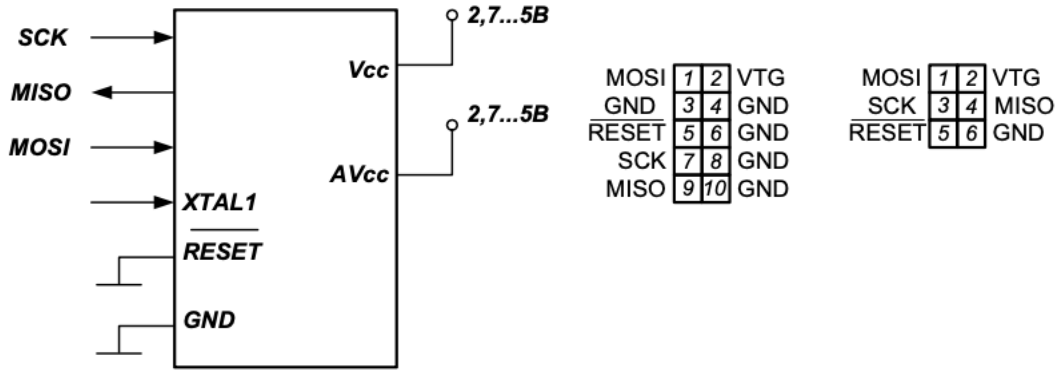


Рисунок 7.1 – Увімкнення мікроконтролерів у режимі програмування по послідовному каналу

Як і в робочому режимі, під час програмування по послідовному каналу мікроконтролеру потрібно джерело тактового сигналу. Як джерела тактового сигналу може використовуватися будь-яке таке джерело. При цьому має виконуватися така умова: тривалість імпульсів як «0», так і «1» рівня сигналу

SCK має бути більше 2 (при $f_{ск}$ <12 МГц) або 3 (при $f_{ск}$ > 12 МГц) періодів тактового сигналу мікроконтролера. Програмування здійснюється шляхом посилання 4-байтних команд на вивід MOSI-мікроконтролера. Результат виконання команд читання знімається з виводу MISO-мікроконтролера. Передача команд і видача результатів їхнього виконання здійснюються від старшого біта до молодшого.

Для переведення мікроконтролера в режим програмування по послідовному каналу необхідно виконати такі дії:

1. Подати на мікроконтролер напругу живлення, при цьому на виводах SCK і RESET має бути присутньою напруга «0» рівня.

2. Зачекати не менше 20 мс.

3. Послати на вивід MOSI команду «Дозвіл програмування».

Для контролю проходження команди при посиланні 3-го байта повертається значення 2-го байта ($53). Якщо значення, що повертається, відмінно від зазначеного, необхідно подати на вивід RESET позитивний імпульс і знову послати команду «Дозвіл програмування». Причому незалежно від значення, що повертається, необхідно передавати всі 4 байти команди. Відсутність повернення числа $53 після кількох спроб вказує на відсутність зв’язку між програматором і мікросхемою або на несправність мікросхеми.

Після завершення програмування на вивід RESET можна подати напругу рівня «1» для переведення мікроконтролера в робочий режим або вимкнути його.

В останньому випадку необхідно виконати таку послідовність дій:

1. Подати на вивід XTAL1 напругу рівня «0», якщо тактування мікроконтролера здійснюється від зовнішньої схеми.

2. Подати на вивід RESET напругу рівня «1».

3. Відключити напругу живлення від мікроконтролера.

Програмування пам’яті програм мікроконтролерів набору Mega здійснюється посторінково. Спочатку вміст сторінки побайтно заноситься в буфер по командах «Завантаження сторінки FLASH-пам’яті». У кожній команді передаються молодші біти адреси змінною комірки (положення комірки всередині сторінки) і записується значення. Вміст кожної комірки має завантажуватися в такій послідовності: спочатку молодший байт, потім старший.

Фактичне програмування сторінки FLASH-пам’яті здійснюється після завантаження буфера сторінки за командою «Запис сторінки FLASH-пам’яті». У команді передаються старші біти адреси комірок (номер сторінки).

Подальше програмування пам’яті можна буде виконувати тільки після завершення запису сторінки. Визначити момент закінчення запису можна трьома способами. Перший і найбільш простий спосіб – витримувати між посиланням команд паузу тривалістю не менше TWD_FLASH.

Другий спосіб полягає в контролюванні вмісту будь-якої із записуваних комірок після посилання команди запису: до завершення запису комірки при її читанні повертається значення $ FF? а після завершення – записане значення.

Третій спосіб – аналіз стану прапорця готовності RDY/BSY за допомогою відповідної команди і відновлення процесу програмування після встановлення прапорця в «1».

У всіх старих моделях програмування EEPROM-пам’яті здійснюється звичайним способом – побайтно. У нових моделях з’явився альтернативний спосіб запису EEPROM-пам’яті – посторінковий. Вміст сторінки побайтно заноситься в буфер по командах «Завантаження сторінки EEPROM-пам’яті», а потім здійснюється фактичне програмування сторінки EEPROM-пам’яті за командою «Запис сторінки EEPROM-пам’яті». Значення адрес, що передаються в цих командах, визначаються так само, як і при програмуванні EEPROM-пам’яті. Для визначення моменту закінчення запису можна використовувати будь-який з описаних вище способів.

7.3 Паралельне програмування

У режимі паралельного програмування, як випливає з його назви, від програматора до мікроконтролера передаються одночасно всі біти коду команди або байта даних. Цей режим задіє велику кількість виводів мікроконтролера і, до того ж, вимагає використання додаткового джерела підвищеної напруги (12 В). Тому програмування в паралельному режимі здійснюється спеціалізованими програматорами. Основне застосування цього режиму – «прошивка» мікроконтролерів перед установкою їх на плату в умовах масового виробництва.

Схема ввімкнення мікросхем у режимі паралельного програмування зображено на рисунку 7.2.

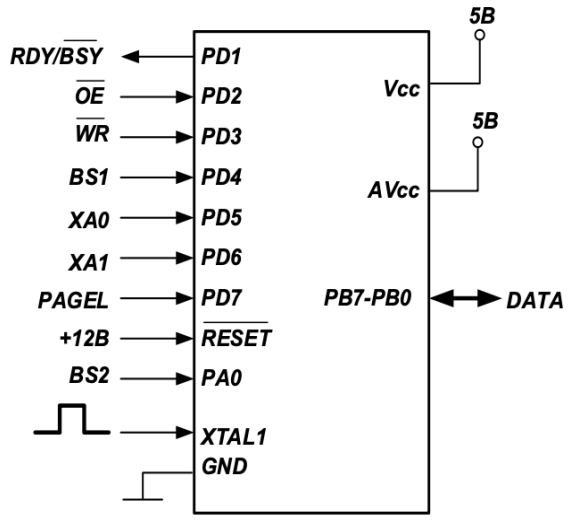


Рисунок 7.2 – Увімкнення мікроконтролерів у режимі паралельного програмування

Першою операцією під час програмування мікроконтролера є його переведення в режим програмування. Для переведення мікроконтролера в режим програмування необхідно виконати таки дії:

1. Подати на мікроконтролер напругу живлення.

2. Подати на вивід RESET напругу рівня «0» і сформувати не менше трьох імпульсів на виводі XTAL1.

3. Подати на виводи PAGEL, XA1, XA0, BS1 напругу рівня «0» на час не менше 100 нс.

4. Подати напругу 11,5...12,5 В на вивід RESET і утримувати напругу рівня «0» на виводах PAGEL, XA1, XA0, BS1 протягом, як мінімум, 10 мкс. Будь-яка активність на зазначених виводах протягом цього часу призведе до того, що мікроконтролер не перейде в режим програмування.

5. Зачекати не менше 300 мкс перед посиланням наступної команди.

Окремо варто сказати про мікроконтролери ATmega8515x/8535x, ATmega8x і ATmega48x/88x/168x, оскільки в цих моделях вивід скидання може бути задіяний під лінію введення / виведення (якщо конфігураційна комірка RSTDISBL запрограмована).

У цьому випадку перед виконанням дій, описаних вище, необхідно:

1. Подати на виводи PAGEL, XA1, XA0, BS1 напругу рівня «0».

2. Подати на мікроконтролер напругу живлення (VCC), а на вивід RESET – напругу 11,5...12,5 В (V_{PP}).

3. Зачекати не менше 100 нс.

4. Перепрограмувати комірку RSTDISBL (якщо встановлений захист, то спочатку необхідно виконати знищення кристала).

5. Вийти з режиму програмування (вимкнути живлення або подати на вивід RESET напругу рівня «1»).

6. Перевести мікроконтролер у режим програмування, як було описано раніше.

Команда «Знищення кристала» має виконуватися перед кожним перепрограмуванням мікроконтролера. Така команда повністю знищує вміст FLASH- і EEPROM-пам’яті, а потім скидає комірки захисту (записує в них логічну 1). Однак на стан конфігураційних комірок ця команда не впливає. До того ж у низці моделей мікроконтролерів набору Mega можна запобігти цьому EEPROM-пам’яті шляхом програмування конфігураційної комірки EESAVE. Програмування байтів конфігурації мікроконтролерів набору Mega здійснюється так.

Молодший конфігураційний байт:

1. Завантажити команду «Запис конфігураційних комірок» (код 0100 0000).

2. Завантажити молодший байт даних. Якщо біт скинутий в 0, виконується програмування відповідної комірки, якщо встановлений в 1 – її скидання.

3. Записати молодший байт конфігурації.

Старший конфігураційний байт:

1. Завантажити команду «Запис конфігураційних комірок» (код 0100 0000).

2. Завантажити молодший байт даних. Якщо біт скинутий в 0, виконується програмування відповідної комірки, якщо встановлений в 1 – її скидання.

3. Записати старший байт конфігурації. Додатковий конфігураційний байт записується аналогічно.

7.4 Програмування по інтерфейсу *JTAG*

Інтерфейс JTAG був розроблений групою провідних фахівців із проблем тестування електронних компонентів (Joint Test Action Group). Вбудований у більшість мікроконтролерів набору Mega, інтерфейс JTAG може бути використаний для:

− тестування друкованих плат;

− конфігурації (програмування) кристала;

− внутрішньосхемного налагодження.

Доступ до модуля JTAG здійснюється через чотири виводи мікроконтролера, що складають так званий «порт тестового доступу» (Test Access Port – TAP): TMS, TCK, TDI і TDO. Стандартний роз'єм для підключення пристроїв наведений на рисунку 7.3.

Роботою модуля JTAG керує TAP-контролер, який є скінченним автоматом зі 16 станами. Перехід між станами здійснюється за наростаючим фронтом сигналу TCK відповідно до сигналу, що присутній на виводі TMS. Після ввімкнення живлення контролер знаходиться в стані Test-Logic-Reset.

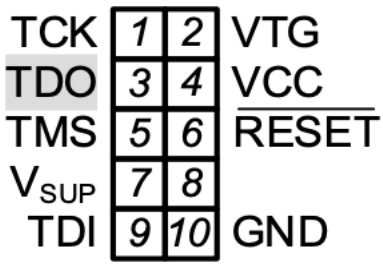


Рисунок 7.3 – Розводка роз'єму JTAG

Дозвіл / заборона інтерфейсу JTAG здійснюється за допомогою конфігураційної комірки JTAGEN. Якщо вона не запрограмована (1), то виводи TAP працюють як звичайні контакти портів введення / виведення, а TAP-контролер знаходиться в стані скидання. Для ввімкнення інтерфейсу комірка JTAGEN має бути запрограмована (стан за замовчуванням). До того ж має бути скинутий біт JTD регістра MCUCSR або MCUCR. Причому, для зміни стану цього біта нове значення необхідно записати в нього двічі протягом чотирьох тактів.

7.5 Самопрограмування мікроконтролерів набору *Mega*

Усі МК набору Mega мають можливість самопрограмування, тобто самостійно змінювати вміст своєї пам'яті програм. У всіх мікроконтролерах набору, за винятком моделі ATmega48x, уся область пам'яті програм логічно розділена на дві секції – секцію прикладної програми (Application Section) і секцію завантажувача (Boot Loader Section). Зміна параметрів пам'яті програм здійснюється програмою-завантажувачем, що розташована в однойменній секції завантаження нового вмісту пам'яті програм. Для вивантаження пам'яті програма-завантажувач може використовувати будь-який інтерфейс передачі даних (USART, SPI, TWI), що наявний у складі конкретного мікроконтролера.

Завантажувач може змінювати вміст обох секцій. Це дозволяє йому модифікувати власний код і навіть видаляти себе з пам'яті, якщо в ньому не буде потреби. Рівень доступу (читання / запис) до кожної з секцій задається користувачем за допомогою комірок захисту BLB02: BLB01 та BLB12: BLB11 (див. табл. 3.3, табл. 3.4). У моделі ж ATmega48x зміна вмісту пам'яті програм може бути здійснена з будь-якого її місця, тобто секцією завантажувача, по суті справи, є вся пам'ять програм. Перехід до програми-завантажувача може здійснюватися різними способами. Зокрема, вона може бути викликана з основної програми командами CALL/JMP. Іншим способом є переміщення вектора скидання в початок секції завантажувача. У цьому випадку запуск програми-завантажувача буде здійснюватися автоматично після кожного скидання мікроконтролера. Положення вектора скидання визначається станом комірки конфігурації BOOTRST. Якщо в ній міститься 1, вектор скидання розташовується на початку пам'яті програм, за адресою $0000. Якщо комірка запрограмована (0), то вектор скидання розташовується на початку секції завантажувача. Розмір секції завантажувача і відповідно розмір секції прикладної програми практично у всіх мікроконтролерах задається за допомогою двох конфігураційних комірок – BOOTSZ1: BOOTSZ0. Виняток становлять лише моделі ATmega48x, у яких поділ на секції відсутній.

8 ВВЕДЕННЯ В СИСТЕМУ ВВОДУ-ВИВОДУ C/C++

У мові C++ дії, що пов'язані з операціями введення і виведення, виконуються за допомогою функцій бібліотек. Функції ведення і виведення бібліотек мови дозволяють читати дані з файлів та пристроїв і писати дані у файли і на пристрої.

Бібліотека мови C++ підтримує три рівня введення-виведення даних:

– введення-виведення потоку;

– введення-виведення нижнього рівня;

– введення-виведення для консолі і порту.

При введенні-виведенні потоку всі дані розглядаються як потік окремих байтів. Для користувача потік – це файл на диску або фізичний пристрій, наприклад, дисплей чи клавіатура, або пристрій для друку, з якого чи на який

направляється потік даних. Операції введення-виведення для потоку дозволяють обробляти дані різних розмірів і форматів від одиночного символу до великих структур даних. Програміст може використовувати функції бібліотеки, розробляти власні і включати їх у бібліотеку. Для доступу до бібліотеки цих класів потрібно включити в програму відповідні заголовні файли.

За замовчуванням стандартні введення і виведення повідомлень про помилки відносяться до консолі користувача (клавіатури та екрана). Це означає, що завжди, коли програма очікує введення зі стандартного потоку, дані повинні надходити з клавіатури, а якщо програма виводить дані – то на екран.

У мові С++ існує декілька бібліотек, які містять засоби введення-виведення, наприклад: stdio.h, iostream.h. Найчастіше застосовують потокове введення-виведення даних, операції якого включені до складу класів istream або iostream. Доступ до бібліотеки цих класів здійснюється за допомогою використання у програмі директиви компілятора #include <iostream.h>.

Для потокового введення даних вказується операція «>>» («читати з»). Це перевантажена операція, визначена для всіх простих типів і покажчика на char. Стандартним потоком введення є cin.

8.1 Файлові структури даних

Є три основні класи файлового вводу / виводу в мові С++:
- ifstream (є дочірнім класу istream);
- ofstream (є дочірнім класу ostream);
- fstream (є дочірнім класу iostream).

За допомогою цих класів можна виконувати однонаправлений файловий ввід, однонаправлений файловий вивід і двонаправлений файловий ввід / вивід. Для їхнього використання потрібно всього лише підключити заголовок fstream.

На відміну від потоків cout, cin, cerr і clog, які відразу ж можна використовувати, файлові потоки мають бути явно встановлені програмістом. Тобто, щоб відкрити файл для читання і/або запису, потрібно створити об’єкт відповідного класу файлового вводу / виводу, вказавши ім’я файлу як параметр. Потім, за допомогою оператора вставки (<<) або оператора вилучення (>>), можна записувати дані в файл або зчитувати вміст файлу. Після виконання даних дій потрібно закрити файл – явно викликати метод close() або просто дозволити файловій змінній вводу / виводу вийти з області видимості (деструктор файлового класу вводу / виводу закриє цей файл автоматично замість нас).

8.2 Базові положення системи вводу-виводу *C/C++*

Файловий вивід

Для запису в файл використовується клас ofstream. Наприклад:

```
#include <iostream>
#include <fstream>
#include <cstdlib> // для використання функції «exit()»

int main()
{
	using namespace std;

	// Клас ofstream використовується для запису даних в файл.
	// Створюємо файл SomeText.txt
	ofstream outf("SomeText.txt");

	// Якщо ми не можемо відкрити цей файл для запису даних,
	if (!outf)
	{
		// то виводимо повідомлення про помилку і виконуємо функцію «exit()»
		cerr << "Uh oh, SomeText.txt could not be opened for writing!" << endl;
		exit(1);
	}

	// Записуємо в файл наступні два рядки
	outf << "See line #1!" << endl;
	outf << "See line #2!" << endl;

	return 0;

	// Коли outf вийде з області видимості, то деструктор класу ofstream автоматично закриє наш файл
}
```

Якщо дивитись в каталог вашого проєкту (*ПКМ по вкладці з назвою вашого .cpp-файлу в Visual Studio > «Відкрити папку, що містить»*), то побачите файл з ім'ям *SomeText.txt*, у якому знаходяться такі рядки:

See line #1!
See line #2!

Зверніть увагу, можна використати **метод put()** для запису одного символу в файл.

Файловий ввід

Тепер ми спробуємо прочитати вміст файлу, який створили в попередньому прикладі. Зверніть увагу, ifstream поверне 0, якщо ми досягли кінця файлу (це зручно для визначення «довжини» вмісту файлу). Наприклад:

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib> // для використання функції «exit()»

int main()
{
	using namespace std;

	// ifstream використовується для читання вмісту файла.
	// Спробуємо прочитати вміст файлу SomeText.txt
	ifstream inf("SomeText.txt");

	// Якщо ми не можемо відкрити цей файл для читання його вмісту,
	if (!inf)
	{
		// то виводимо наступне повідомлення про помилку і виконуємо функцію «exit()»
		cerr << "Uh oh, SomeText.txt could not be opened for reading!" << endl;
		exit(1);
	}

	// Поки є дані, які ми можемо прочитати,
	while (inf)
	{
		// то переміщуємо ці дані в рядок, який потім виводимо на екран
		string strInput;
		inf >> strInput;
		cout << strInput << endl;
	}

	return 0;

	// Коли inf вийде з області видимості, то деструктор класу ifstream автоматично закриє наш файл
}
```

Результат виконання програми:

```
See line #1!
See line #2!
```

Результат не відповідає запиту. Як вже відомо, оператор вилучення працює з «відформатованими даними», тобто він ігнорує всі пробіли, символи табуляції і символ нового рядка. Щоб прочитати весь вміст як є, без його ділення на частини, нам потрібно використати **метод getline()**:

```
#include <iostream>
#include <fstream>
#include <string>
#include <cstdlib> // для використання функції «exit()»

int main()
{
	using namespace std;

	// ifstream використовується для читання вмісту файлів.
	// Ми спробуємо прочитати вміст файлу SomeText.txt
	ifstream inf("SomeText.txt");

	// Якщо ми не можемо відкрити файл для читання його вмісту,
	if (!inf)
	{
		// то виводимо наступне повідомлення про помилку і виконуємо функцію «exit()»
		cerr << "Uh oh, SomeText.txt could not be opened for reading!" << endl;
		exit(1);
	}

	// Поки є, що читати,
	while (inf)
	{
		// то переміщуємо те, що можемо прочитати, в рядок, а потім виводимо цей рядок на екран
		string strInput;
		getline(inf, strInput);
		cout << strInput << endl;
	}

	return 0;

	// Коли inf вийде з області видимості, то деструктор класу ifstream автоматично закриє наш файл
}
```

Буферизований вивід

Вивід у мові C++ може бути буферизований. Це означає, що все, що виводиться в файловий потік, не може відразу ж бути записано на диск (у конкретний файл). Це зроблено насамперед з міркувань продуктивності. Коли дані буфера записуються на диск, то це називається очищенням буфера. Одним із способів очищення буфера є закриття файлу. У такому випадку весь вміст буфера буде переміщено на диск, а потім файл буде закрито.

Буферизація виводу зазвичай не є проблемою, але при певних обставинах вона може викликати проблеми у необережних новачків. Наприклад, коли в буфері зберігаються дані, а програма передчасно завершує своє виконання (або в результаті збою, або шляхом виклику функції «exit())». У таких випадках деструктори класів файлового вводу / виводу не виконуються, файли ніколи не закриваються, буфери не очищаються і наші дані губляться назавжди. Ось чому гарною ідеєю є явне закриття всіх відкритих файлів перед викликом функції «exit()».

Також буфер можна очистити вручну, використовуючи **метод ostream::flush()** або відправивши **std::flush** у вихідний потік. Будь-який з цих способів може бути корисний для забезпечення негайного запису вмісту буфера на диск в разі збою програми.

***Цікавий нюанс*:** оскільки std::endl; також очищає вихідний потік, то його надмірне використання (яке призводить до непотрібних очищень буфера) може вплинути на продуктивність програми (тому що очищення буфера в деяких випадках може бути витратною операцією). З цієї причини програмісти, які турбуються про продуктивність свого коду, часто використовують \n замість std::endl для вставки символу нового рядка у вихідний потік, щоб уникнути непотрібного очищення буфера.

Режими відкриття файлів

Що відбудеться, якщо ми спробуємо записати дані в уже існуючий файл? Повторний запуск наведеної вище програми (найперша) показує, що вихідний файл повністю перезаписується при повторному запуску програми. А що, якщо нам потрібно додати дані в кінець файлу? Виявляється, конструктори файлового потоку приймають необов'язковий другий параметр, який дозволяє вказати програмісту спосіб відкриття файлу. Як цей параметр можна передавати такі прапорці (які знаходяться в класі ios):

app – відкриває файл у режимі додавання;

ate – переходить у кінець файлу перед читанням / записом;

binary – відкриває файл у бінарному режимі (замість текстового режиму);

in – відкриває файл у режимі читання (за замовчуванням для ifstream);

out – відкриває файл у режимі запису (за замовчуванням для ofstream);

trunc – видаляє файл, якщо він вже існує.

Можна вказати відразу кілька прапорців шляхом використання побітового АБО (|).

ifstream за замовчуванням працює в режимі ios::in;

ofstream за замовчуванням працює в режимі ios::out;

fstream за замовчуванням працює в режимі ios::in АБО ios::out, що означає, що ви можете виконувати як читання вмісту файлу, так і запис даних у файл.

Далі напишемо програму, яка додасть два рядки в раніше створений файл *SomeText.txt*:

```
#include <iostream>
#include <cstdlib> // для використання функції «exit()»
#include <fstream>

int main()
{
	using namespace std;

	// Передаємо флаг ios:app, щоб повідомити fstream, що ми збираємося додати свої дані до вже існуючих даних файлу.
	// Ми не збираємося перезаписувати файл.
	// Нам не потрібно передавати флаг ios::out, оскільки ofstream за замовчуванням працює в режимі ios::out
	ofstream outf("SomeText.txt", ios::app);

	// Якщо ми не можемо відкрити файл для запису даних,
	if (!outf)
	{
		// то виводимо наступне повідомлення про помилку і виконуємо функцію «exit()»
		cerr << "Uh oh, SomeText.txt could not be opened for writing!" << endl;
		exit(1);
	}

	outf << "See line #3!" << endl;
	outf << "See line #4!" << endl;

	return 0;

	// Коли outf вийде з області видимості, то деструктор класу ofstream автоматично закриє наш файл
}
```

8.3 Стандартні об’єкти вводу-виводу

Бібліотека iostream

Після підключення заголовку «iostream» ми отримуємо доступ до всієї ієрархії класів бібліотеки iostream, які відповідають за функціонал вводу / виводу даних (включаючи клас, який називається «iostream»). Ієрархія цих класів має вигляд, наведений на рисунку 8.1.

Перше, на що варто звернути увагу в цій ієрархії, – множинне спадкування (те, що насправді не рекомендується використовувати). Проте бібліотека iostream була розроблена і ретельно протестована у відповідний спосіб, щоб уникнути типових помилок, які виникають під час роботи з множинним спадкуванням, тому ви можете спокійно використовувати цю бібліотеку.

Потоки в C++

Друге, – це часте використання слова *«stream»* (тобто *«потік»*). По суті, ввід / вивід у мові C++ реалізований за допомогою потоків. Абстрактно, **потік** – це послідовність символів, до яких можна отримати доступ. З часом потік може надавати або споживати потенційно необмежені обсяги даних.

Ми матимемо справу з двома типами потоків:

Потік вводу (або ***«вхідний потік»***) використовується для зберігання даних, отриманих від джерела даних: клавіатури, файлу, мережі тощо. Наприклад, користувач може натиснути клавішу на клавіатурі в той час, коли програма не очікує вводу. Замість ігнорування натискання клавіші, дані поміщаються у вхідний потік, де потім очікують відповіді від програми.

Потік виводу (або ***«вихідний потік»***) використовується для зберігання даних, наданих конкретному споживачеві даних: монітору, файлу, принтеру тощо. Під час запису даних на пристрій виводу цей пристрій може бути не готовим прийняти дані негайно. Наприклад, принтер все ще може прогріватися, коли програма вже записує дані в вихідний потік. Таким чином, дані перебуватимуть у потоці виводу до тих пір, поки принтер не почне їх використовувати.

Деякі пристрої, такі як файли і мережі, можуть бути джерелами як вводу, так і виводу даних.

Гарна новина полягає в тому, що програмісту не потрібно знати деталі взаємодії потоків із різними пристроями і джерелами даних, йому потрібно тільки навчитися взаємодіяти з цими потоками для читання і запису даних.

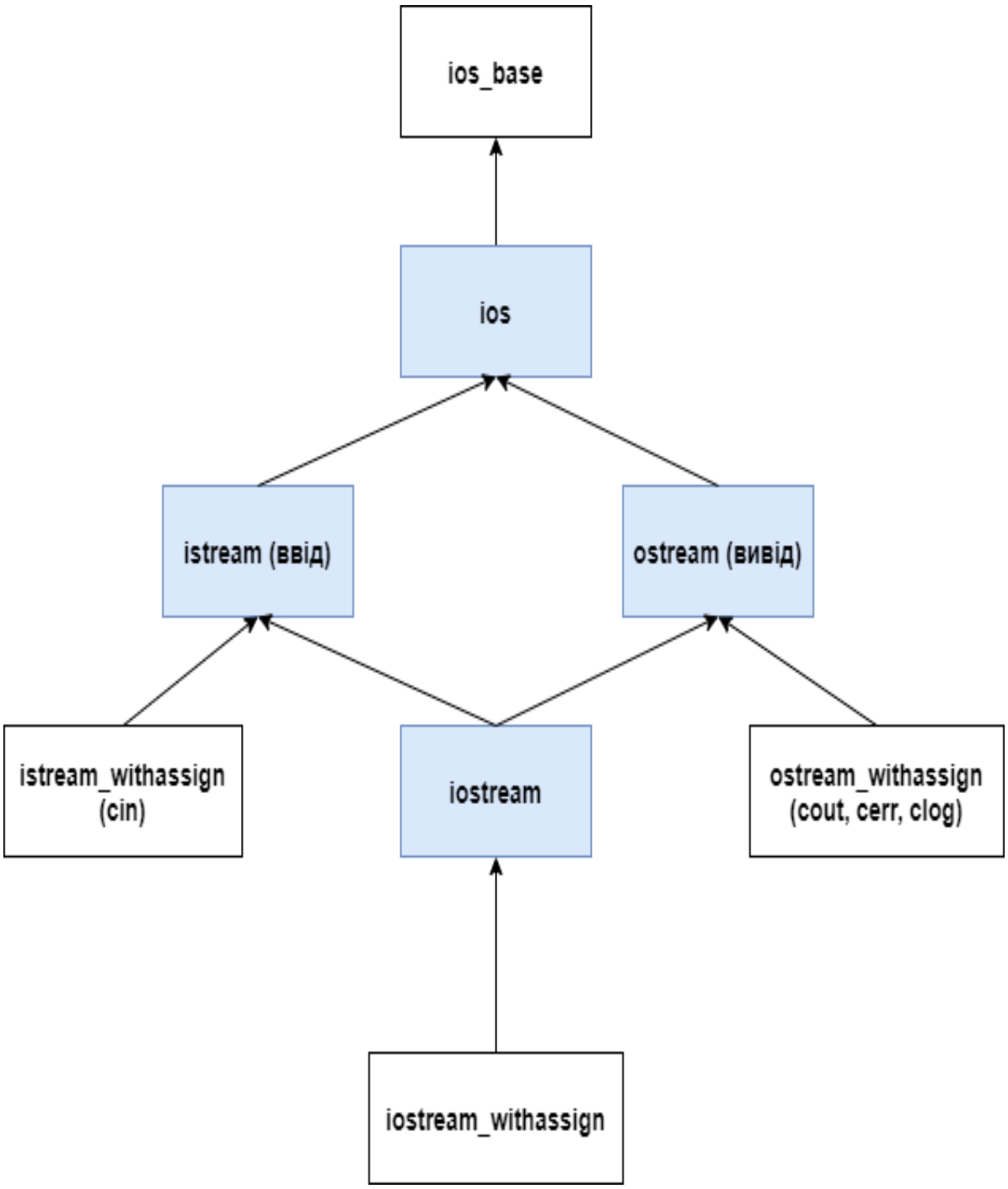


Рисунок 8.1 – Ієрархія класів

Ввід / вивід в C++

Хоча клас ios є дочірнім класу ios_base, дуже часто саме цей клас буде *найбільш батьківським* класом, з яким ви працюватимете / взаємодіятимете напряму. Клас ios визначає купу різних речей, які є загальними для потоків вводу / виводу.

Клас istream використовується для роботи з вхідними потоками. Оператор вилучення >> використовується для вилучення значень з потоку. У цьому є сенс: коли користувач натискає на клавішу клавіатури, код цієї

клавіші поміщається у вхідний потік. Потім програма витягує це значення з потоку і використовує його.

Клас ostream використовується для роботи з вихідними потоками. Оператор вставки << використовується для вставки значень у потік. У цьому також є сенс: ви вставляєте свої значення в потік, а потім споживач даних (наприклад, монітор) використовує їх.

Клас iostream може обробляти як ввід, так і вивід даних, що дозволяє йому здійснювати двонаправлений ввід / вивід. Нарешті, залишилися 3 класи, які закінчуються на _withassign. Ці потокові класи є дочірніми класами istream, ostream і iostream (відповідно). Ви здебільшого не працюватимете з ними напряму.

Стандартні потоки в C++

Стандартний потік – це попередньо підключений потік, який надається програмі її оточенням. Мова C++ поставляється з чотирма попередньо визначеними стандартними об'єктами потоків, які ви можете використовувати (перші три ви вже зустрічали):

cin – клас istream_withassign, пов'язаний зі стандартним вводом (зазвичай це клавіатура);

cout – клас ostream_withassign, пов'язаний зі стандартним виводом (зазвичай це монітор);

cerr – клас ostream_withassign, пов'язаний зі стандартною помилкою (зазвичай це монітор), який забезпечує небуферизований вивід;

clog – клас ostream_withassign, пов'язаний зі стандартною помилкою (зазвичай це монітор), який забезпечує буферизований вивід.

Небуферизований вивід зазвичай обробляється відразу ж, тоді як буферизований вивід зазвичай зберігається і виводиться як блок. Оскільки clog використовується рідко, то його зазвичай ігнорують.

8.4 Ієрархія класів вводу-виводу *C++*

Найбільш яскравим утіленням принципів об'єктно-орієнтованого програмування є система вводу-виводу мови C++. У мові C++ співіснують дві системи вводу-виводу: процедурно-орієнтована, заснована на сімействах функцій «printf()» і «scanf()», і об'єктно-орієнтована, в основі якої лежить ієрархія класів.

В усіх попередніх програмах ми використовували зазвичай систему вводу-виводу мови C, щоб не забігати наперед. Це – застаріла система. Утім, ніяких особливих недоліків вона не має, за винятком декількох обмежень, з якими ми зштовхнулися при описі стандартної бібліотеки шаблонів. Просто вона не є об'єктно-орієнтованою, і цього достатньо, щоб відмовитися від неї на користь нової системи.

Опис засобів об'єктно-орієнтованого вводу-виводу розподілено по двох заголовках: <iostream.h> і <iostream>. Хоча вони майже однакові, заголовок «iostream» містить більш сучасні механізми реалізації вводу-виводу.

В основі системи вводу-виводу мови C++ лежить поняття потоку, що ріднить її з процедурно-орієнтованою системою мови C. Нагадаємо, що потік становить абстракцію фізичного пристрою вводу-виводу і забезпечує однаковий інтерфейс роботи з ним. Основні операції вводу-виводу визначені в мудрій ієрархії класів, що міститься в заголовку <iostream>. Відзначимо одну тонкість: оскільки система вводу-виводу, власне кажучи, є символьною, а в мові C++ існують два види символів – звичайні (8 біт) і розширені (26 біт), класи вводу- виводу повинні враховувати цю обставину. Для того щоб не створювати дві рівносильні ієрархії для окремих видів символів, класи вводу-виводу оголошені шаблонними. Поняття шаблонного класу відноситься до парадигми узагальненого програмування, що ми розглянули в третій частині книги.

Бібліотека вводу-виводу розподілена по таких заголовках у таблиці 8.1.

Таблиця 8.1 – Бібліотека вводу-виводу розподілена по таких заголовках

Поняття	Визначення
<iosfwd>	Неповні оголошення класів
<iostream>	Основні засоби вводу-виводу
<ios>	Базові класи потоків вводу-виводу
<streambuf>	Засоби буферизації потоків
<istream> <ostream> <iomanip>	Засоби форматування і маніпулятори
<sstream> <cstdlib>	Строкові потоки
<fstream> <cstdio> <cwchar>	Файлові потоки

Система вводу-виводу мови C++ складається з двох ієрархій класів. В основі першої ієрархії лежить клас ios_base. Його єдиним спадкоємцем є шаблонний клас basic_ios, що містить визначення операцій вводу-виводу високого рівня. Клас basic_ios є базовим стосовно класів basic_iostream, basic_ostream і basic_iostream, що становлять класи потоків вводу, виводу і вводу-виводу відповідно. Шаблонні класи, що входять у цю ієрархію, мають дві спеціалізації: для звичайних і для розширених символів.

Як відомо, бібліотека вводу-виводу створює дві спеціалізації шаблонних класів, що входять в ієрархію: одну – для восьмибітівих символів, а іншу – для розширених. Нижче наводиться список імен шаблонних класів, призначених для вводу-виводу звичайних і розширених символів. Це створює дві рівносильні ієрархії спеціалізованих класів: 1) ios → istream, ostream → ifstream, iostream, ofstream (для звичайних символів), 2) wios → wistream, wostream → wifstream, wiostream, wfstream (для розширених символів).

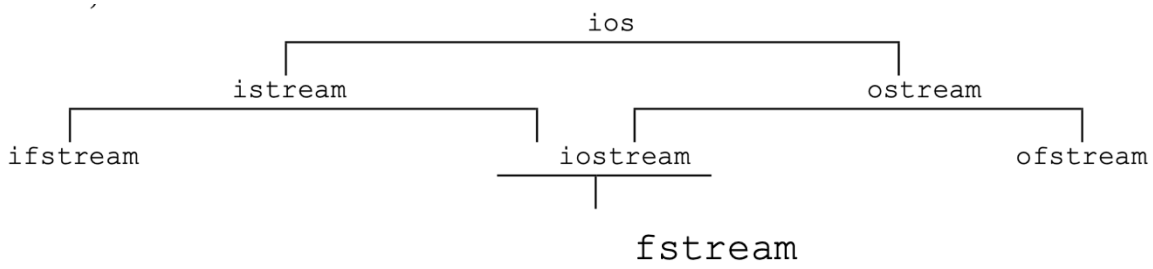


Рисунок 8.2 – Ієрархія шаблонних класів вводу-виводу звичайних класів

У мові C++ передбачено вісім стандартних потоків, що автоматично відкриваються при запуску будь-якої програми. Перші чотири потоки призначені для вводу-виводу звичайних символів: cin – потік вводу, cout – потік виводу, cerr – потік помилок, clog – буферизований потік помилок. Друга четвірка потоків дозволяє вводити і виводити розширені символи. Вони називаються win, wout, werr і wlog.

За замовчуванням потік cin зв’язаний із клавіатурою, а інші потоки – з екраном. При бажанні потоки вводу-виводу можна зв’язати з іншими пристроями (наприклад, принтером чи портом). У мові C++ існує два способи форматування вводу-виводу: 1) за допомогою членів класу ios; 2) за допомогою маніпуляторів.

9 ОСОБЛИВОСТІ МІКРОКОНТРОЛЕРА ARDUINO

Arduino Uno – це пристрій на основі мікроконтролера ATmega328 (datasheet). До його складу входить все необхідне для зручної роботи з мікроконтролером: 14 цифрових входів / виходів (з них 6 можуть використовуватися як ШИМ-виходи), 6 аналогових входів, кварцовий резонатор на 16 МГц, роз’єм USB, роз’єм живлення, роз’єм для внутрішньосхемного програмування (ICSP) та кнопка скидання. Для початку роботи з пристроєм достатньо просто подати живлення від AC/DC-адаптера або батарейки, або підключити його до комп’ютера за допомогою кабелю USB.

На відміну від усіх попередніх плат Arduino, Uno як перетворювач інтерфейсів USB-UART використовує мікроконтролер ATmega16U2 (ATmega8U2 до версії R2) замість мікросхеми FTDI.

На платі Arduino Uno версії R2 для спрощення процесу оновлення прошивки доданий резистор, що підтягує до землі лінію HWB мікроконтролера 8U2. Зміни на платі версії R3 наведені нижче.

Розпинка 1.0: додано контакти SDA та SCL (біля виводу AREF), а також два нових виводи, розташовані біля виводу RESET. Перший – IOREF – дозволяє платам розширення підлаштовуватися під робочу напругу Arduino. Цей контакт призначений для сумісності плат розширення як з 5В-Ардуїно на базі мікроконтролерів AVR, так і з 3.3В-платами Arduino Due. Другий контакт ні до чого не приєднаний та зарезервований для майбутніх цілей. Поліпшено завадостійкість ланцюга скидання.

Мікроконтролер ATmega8U2 замінено на ATmega16U2. "Uno" (у перекладі з італійської – «один») названий з нагоди майбутнього випуску Arduino 1.0. Спільно з Arduino 1.0 ці пристрої є базовими версіями мікроконтролера Arduino. Uno – еталонна модель платформи Arduino і є останньою у серії USB-плат; для порівняння з попередніми версіями варто звернутись на офіційний сайт виробника плат Arduino.

9.1 Arduino – контролер системи освітлення

Світлодіодне освітлення приміщень може включати використання LED-ламп, стрічок або потужних світлодіодів, для керування якими необхідний мікроконтролер. Це дає змогу проєктувати освітлювальні системи будь-якої складності з використанням як стандартних компонентів, так і спеціалізованих контролерів і драйверів.

Перехід від традиційних ламп розжарювання до світлодіодного освітлення створив проблему регулювання яскравості. Звичайні димери, що застосовувалися для ламп розжарювання, мають обмежену сумісність із LED-лампами, і лише деякі моделі з маркуванням *Dimmable* підтримують роботу з ними. У разі використання світлодіодних стрічок їхнє живлення зазвичай здійснюється через блоки живлення на 12 або 24 В, які підключаються до мережі 220 В через механічний вимикач, проте цей спосіб не дозволяє регулювати яскравість і може спричиняти електромагнітні перешкоди та перевантаження вимикачів.

Для керування яскравістю світлодіодів у складніших системах застосовуються сучасні технології, що базуються на широтно-імпульсній модуляції (ШІМ, або PWM). Серед найпоширеніших методів:

- безпосереднє регулювання PWM-сигналом;
- керування через протокол DMX-512;
- керування через протокол DALI.

Ці технології дозволяють централізовано контролювати яскравість світлодіодного освітлення, що особливо актуально для офісів, магазинів, ресторанів та інших комерційних приміщень, а також інтегрувати управління через смартфон, планшет чи комп’ютер.

9.2 Основні характеристики

Arduino Uno може живиться від USB або від зовнішнього джерела живлення – тип джерела вибирається автоматично. Як зовнішнє джерело живлення (не USB) може використовуватися мережевий AC / DC-адаптер або акумулятор / батарея. Штекер адаптера (діаметр – 2.1 мм, центральний контакт – позитивний) необхідно вставити у відповідний роз’єм живлення на платі. У разі живлення від акумулятора / батареї, її провід необхідно під’єднати до виводів Gnd і Vin роз’єму POWER. Напруга зовнішнього джерела живлення може бути в межах від 6 В до 20 В. Однак зменшення напруги живлення нижче 7 В призводить до зменшення напруги на виводі 5 В, що може стати

причиною нестабільної роботи пристрою. Використання напруги більше 12 В може призводити до перегріву стабілізатора напруги і виходу плати з ладу. З огляду на це рекомендується використовувати джерело живлення з напругою в діапазоні від 7 В до 12 В.

Одна з основних характеристик – це розрядність мікроконтролера. Що це означає? Як і в більшості сучасних обчислювальних систем пам'ять, з якою працює мікроконтролер, логічно і схематично відділена від нього (хоч і може перебувати з ним на одному чіпі), і тому мікроконтролер не може безпосередньо виконувати операції з даними, які в ній записані (наприклад, складати). Для цього йому потрібно скопіювати ці дані в спеціальні осередки пам'яті, які як раз є безпосередньою частиною мікросхеми і називаються регістрами. Після того як операція буде виконана, дані з регістрів зазвичай копіюються назад в пам'ять. Так зроблено тому, що реєстрова пам'ять, яка працює на тих самих швидкостях, що і мікроконтролер, є дуже дорогою, тому її не може бути багато. З іншого боку, основна пам'ять, винесена в свою власну мікросхему, робиться за дещо іншою технологією, що дозволяє її істотно здешевити і, відповідно, зробити її більше, правда, за все треба платити, тому подібна пам'ять, хоч її і багато, працює значно повільніше, ніж реєстрова – у десятки, а іноді і в сотні разів! Тому основний цикл роботи сучасного мікропроцесора виглядає так: скопіювати потрібні дані з основної пам'яті в регістри, виконати операцію над регістрами, скопіювати результат назад у зовнішню пам'ять або продовжити обчислення над значеннями. Дані, які знаходяться в регістрах, як, власне, і всюди в комп'ютерах, подані у двійковому вигляді, тобто у вигляді нулів і одиниць, а регістри також мають конкретний розмір, який визначає, скільки двійкових розрядів або скільки біт можна в нього записати.

VIN. Напруга, що надходить в Arduino безпосередньо від зовнішнього джерела живлення (не пов'язане з 5 В від USB або іншою стабілізованою напругою). Через цей вивід можна як подавати зовнішнє живлення, так і споживати струм, коли пристрій живиться від зовнішнього адаптера.

5V. На вивід надходить напруга 5 В від стабілізатора напруги на платі, незалежно від того, як живиться пристрій: від адаптера (7 – 12 В), від USB (5 В) або через вивід VIN (7 – 12 В). Живити пристрій через виводи 5 V або 3V3 не рекомендується, оскільки в цьому випадку не використовується стабілізатор напруги, що може привести до виходу плати з ладу.

3V3. 3.3 В, що надходять від стабілізатора напруги на платі. Максимальний струм, споживаний від цього виводу, становить 50 мА.

GND. Загальний мінусовий вивід.

IOREF. Цей вивід надає платам розширення інформації про робочу напругу мікроконтролера Arduino. Залежно від напруги, зчитуваної з виводу IOREF, плата розширення може переключитися на відповідне джерело живлення або задіювати перетворювачі рівнів, що дозволить їй працювати як з 5В, так і з 3.3В-пристроями.

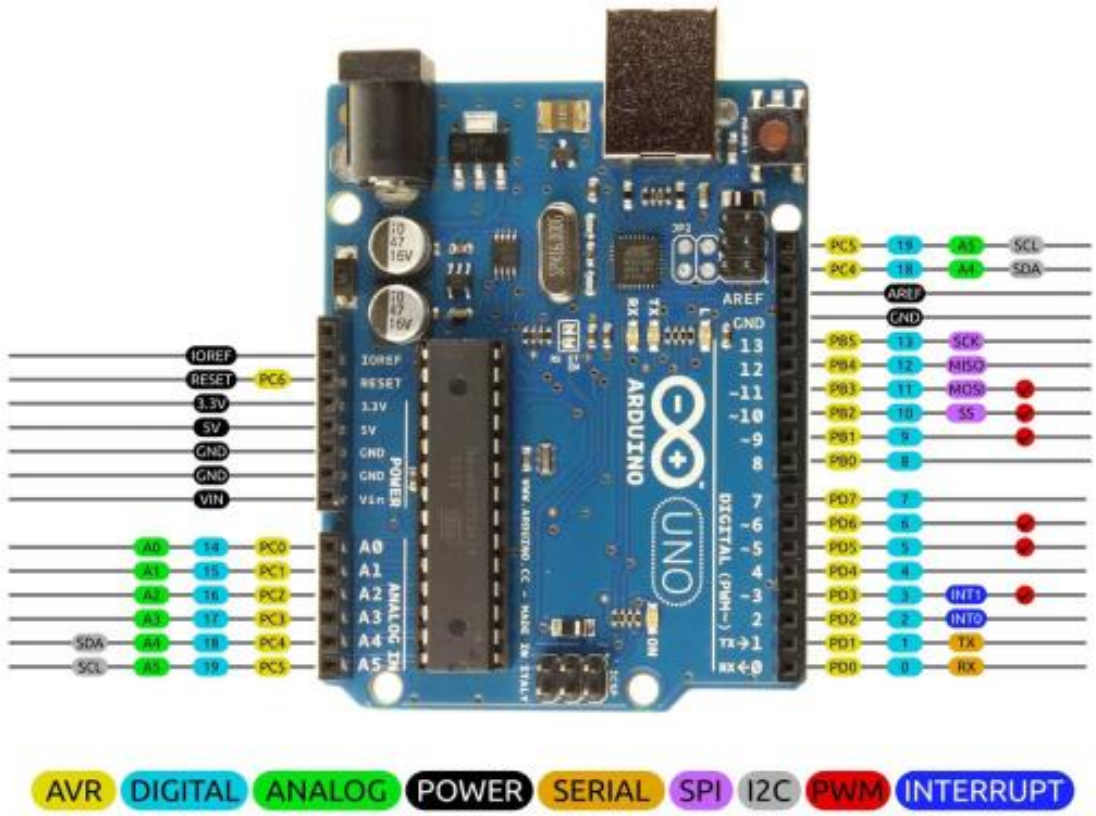


Рисунок 9.1 – Вигляд плати Arduino UNO

Пам'ять. Обсяг флешпам'яті ATmega328 становить 32 КБ (з яких 0.5 КБ використовуються завантажувачем). Мікроконтролер також має 2 КБ пам'яті SRAM і 1 КБ EEPROM (з якої можна зчитувати або записувати інформацію за допомогою бібліотеки EEPROM).

9.3 Периферійні пристрої

Arduino Uno надає низку можливостей для здійснення зв'язку з комп'ютером, ще одним Arduino або іншими мікроконтролерами. У ATmega328 є універсальний асинхронний приймач-передавач (UART), що дозволяє здійснювати послідовну передачу даних за допомогою цифрових виводів 0 (RX) і 1 (TX) та перетворювача USB-UART. Мікроконтролер ATmega16U2 на платі виконує функції такого перетворювача, і при підключенні до ПК, дозволяє Arduino включитися, як віртуальний COMпорт. Прошивка мікросхеми 16U2 використовує стандартні драйвера USBCOM, тому установка зовнішніх драйверів не потрібна. На платформі Windows необхідний тільки відповідний .inf-файл. У пакет програмного забезпечення Arduino входить спеціальна програма, що дозволяє зчитувати і відправляти на Arduino прості текстові дані. При передачі даних через мікросхему-перетворювач USB-UART під час USB-з'єднання з комп'ютером, на платі будуть блимати світлодіоди RX і TX. Бібліотека SoftwareSerial дозволяє реалізувати послідовний зв'язок на будь-яких цифрових виводах Arduino Uno. Практично жоден МПП не

обходиться без кнопок і простих давачів. За допомогою цих периферійних елементів у МПП поступає різна інформація, яка використовується для зміни алгоритму роботи програми.

У деяких пристроях інколи потрібне застосування дискретних оптичних давачів: світлодіода і фототранзистора. Вони розраховані на роботу в інфрачервоному діапазоні, як і ті, що застосовуються в щілинних і відбивних оптронах. Дискретні оптопари використовуються в системах, де великий об'єкт перешкоджає шлях світловому променю між світлодіодом і фототранзистором, або там, де відстань дуже велика для використання оптрона. У деяких пристроях інколи потрібне застосування дискретних оптичних давачів: світлодіода і фототранзистора. Вони розраховані на роботу в інфрачервоному діапазоні, як і ті, що застосовуються в щілинних і відбивних оптронах. Дискретні оптопари використовуються в системах, де великий об'єкт перешкоджає шлях світловому променю між світлодіодом і фототранзистором, або там, де відстань дуже велика для використання оптрона.

9.4 Програмування мікроконтролера

Отже, вхідна інформація надходить у систему і далі вона потрапляє в її ядро, яке займається її обробкою. Наприкінці цього процесу, коли система вирішила, що робити з інформацією або яке рішення потрібно ухвалити на її основі, вона має видати якісь результати. Найчастіше це означає зробити якусь дію в зовнішньому світі. Це робиться за допомогою виконавчих механізмів або приводів, які показані в правій частині схеми на рисунку 9.1. Виконавчим механізмом може бути світлодіод, який може горіти або блимати. Наприклад, він може сигналізувати, що відеокамера записує або що закінчується батарейка. Але є й інші виконавчі механізми. Наприклад, всередині камери є двигуни, за допомогою яких відбувається управління об'єктивом і наведення різкості. Ці двигуни є виконавчими механізмами, а їхній рух – це вихід системи. Існує безліч різних типів виконавчих механізмів: динаміки відтворюють звук, лампи дозволяють системі виводити світло, екран також виводить світло, але в більш детальному і «керованому» вигляді. Щоб компоненти в проєкті виконували необхідні нам дії, нам потрібно написати програму, яка надасть Arduino потрібні інструкції. Розробка програм для Arduino ведеться в інструменті, що називається інтегрованим середовищем розробки (ПВЄ). Це середовище розробки можна завантажити безкоштовно з сайту www.arduino.cc/en/Main/Software, вибравши дистрибутив для операційної системи Windows, MacOS або Linux. Після встановлення та запуску середовища ви зможете писати комп'ютерні програми (набори покрокових інструкцій, відомих у світі Arduino під терміном «скетчі»), які потім передаються на пам'ять Arduino через USB-інтерфейс. Мікрокомп'ютер Arduino виконуватиме інструкції зі скетчу, попутно взаємодіючи з навколишнім середовищем.

9.5 Цифровий ввід / вивід за допомогою плати Arduino

За замовчуванням всі порти Arduino визначаються як входи, і немає потреби описувати це в коді. Порти зазвичай прописуються в функції ініціалізації змінних. Ініціалізація порту вводу-виводу Arduino:

1. *pinMode* (pin, mode) Параметри: pin: номер виводу, режим роботи якого задається, mode: приймає значення: INPUT – вхід, у цьому режимі відбувається зчитування даних із давачів, стану кнопок, аналогового і цифрового сигналу. Порт знаходиться в так званому високоімпедансному стані, тобто на вході високий опір. OUTPUT – вихід, залежно від команди прописаної в коді, порт приймає значення одиниці або нуля. Вихід стає свого роду керованим
джерелом живлення і видає максимальний струм (20 мА та 40 мА в піковому значенні) в навантаження, що до нього підключене. INPUT_PULLUP – порт працює як вхід, але до нього підключається та званий підтягувальний резистор з номіналом 20 – 50 кОм. Значення, що повертаються – немає.

2. digitalWrite (pin, value) Параметри: pin: номер виводу, value: значення HIGH або LOW. Значення, що повертаються – немає.

3. digitalRead (pin) Параметри: pin: номер цифрового виводу, з якого необхідно вважати значення (int). Значення, що повертаються HIGH або LOW.

4. Цифрові виходи. Робота зі світлодіодом. Одне з найпростіших дій, яке може виконати мікроконтролер, це перемикання стану контактів цифрових входів-виходів. Будь-який з цифрових виходів мікроконтролера можна програмним способом переключити між високим рівнем (+5 V) і низьким (0 V). За допомогою такого перемикання відбувається управління простими пристроями. Перемикання здійснюється функцією «digitalWrite», виклик якої має вигляд: digitalWrite (номер_контакту, рівень_сигналу). Параметр рівень_сигналу може приймати два значення: HIGH (високий, +5 V) або LOW (низький). Параметр номер_контакту – це число, що відповідає номеру контакту на платі Arduino. Зверніть увагу, що одні й ті ж виводи мікроконтролера можуть виступати як в ролі цифрових виходів, так і в ролі цифрових входів. Тому перед їхнім використанням необхідно вказати, в якій ролі використовуватиметься контакт: входу або виходу. Це робиться за допомогою функції «pinMode»: pinMode (номер_контакту, режим_контакту). Її аргумент режим_контакту може приймати значення: OUTPUT (вихід) і INPUT (вхід). Таким чином, щоб встановити на виході № 2 високий рівень сигналу, достатньо виконати таку програму: const int pin = 2; void setup () { pinMode (pin, OUTPUT); digitalWrite (pin, HIGH);}

Один з найбільш простих для підключення виконавчих пристроїв, яким можна керувати за допомогою цифрового виходу, – це світлодіод. Цей пристрій є напівпровідниковим приладом, здатним випромінювати світло під час проходження через нього електричного струму в прямому напрямку (від анода до катода). На рисунку 9.2 показано зовнішній вигляд світлодіода і його позначення на електричній схемі.

Для того щоб правильно включити світлодіод в електричний ланцюг, необхідно відрізняти катод від анода. Зробити це можна за трьома ознаками: 1) анод світлодіода має довший провідник; 2) з боку катода корпус світлодіода трохи зрізаний; 3) всередині світлодіода катод ширше, ніж анод.

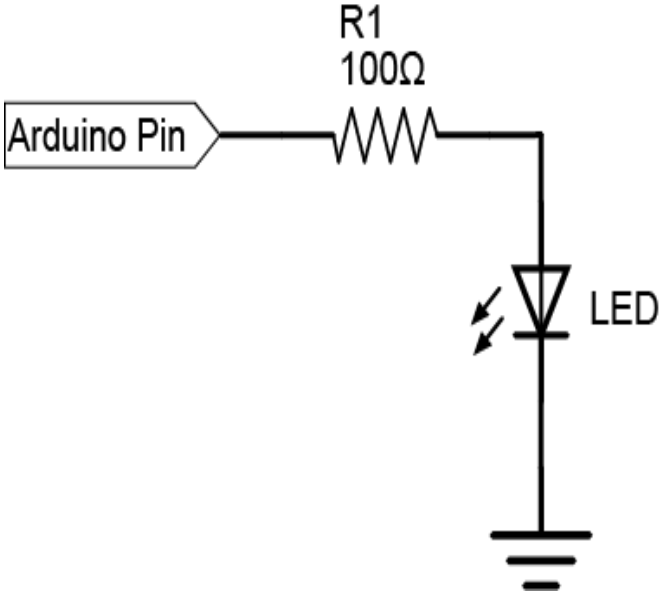


Рисунок 9.2 – Електрична схема

У сучасній мікроелектроніці застосовуються мініатюрні світлодіоди для поверхневого монтажу. Такі індикатори, наприклад, є на Arduino Uno для інформування користувача про стан системи. Важливо відзначити, що робоча напруга живлення світлодіода варіюється від 1,85 Вт до 2,5 Вт при рекомендованій силі струму не більше 20 мА. Щоб сила струму не перевищила це значення, при підключенні світлодіода до виходу мікроконтролера, який видає напругу 5 В, у ланцюг потрібно додати послідовний обмежуючий резистор опором від 200 Ом до 500 Ом. В іншому випадку струм через світлодіод буде занадто великим, що може пошкодити як діод, так і вивід мікроконтролера. На рисунку 9.2 наведена електрична схема.

9.6 Алгоритм роботи пристрою

Arduino – це електронний конструктор і зручна платформа швидкої розробки електронних пристроїв для новачків і професіоналів. Платформа користується величезною популярністю в усьому світі завдяки зручності і простоті мови програмування, а також відкритої архітектури і програмного коду. Пристрій програмується через USB без використання програматорів. Arduino – це електронний конструктор і зручна платформа швидкої розробки електронних пристроїв для новачків і професіоналів. Платформа користується величезною популярністю в усьому світі завдяки зручності і простоті мови програмування, а також відкритої архітектури і програмного коду. Пристрій програмується через USB без використання програматорів.

У цьому підрозділі ми ознайомимося з середовищем розробки Arduino IDE (Integrated Development Environment – інтегроване середовище розробки). Це середовище потрібно використовувати, якщо у вас є реальна плата Arduino. Рядок меню редактора (рис. 9.3) включає такі основні елементи: файл, правка, скетч, сервіс і довідка. Розглянемо докладніше кожен з них.

У пункті «Файл» можна знайти команди, що відповідають за створення нової програми, завантаження з диска вже існуючої, збереження її змін, а також команди для завантаження програми на мікроконтролер: – «Створити» – створити нову програму (у цьому середовищі програми називаються скетчами); – «Відкрити» – відкрити існуючу програму; – «Папка зі скетчами» – відкрити програму із заданої папки; – «Приклади» – відкрити приклад програми. Пункт меню «Правка» містить команди, пов'язані з редагуванням тексту програми, що містить копіювання, вставку, налаштування відступів і пошук.

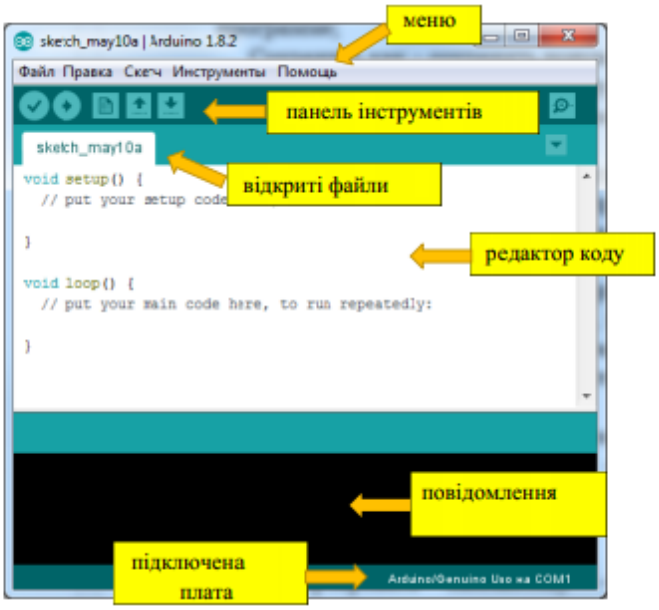


Рисунок 9.3 – Області вікна середовища розробки Arduino

У розділі «Скетч» розміщуються команди для контролю за процесом компіляції програми:

– «Перевірити / Компілювати» – компілювати програму;

– «Показати папку скетчів» – відкрити системну папку з програмами;

– «Додати файл» – додати до проєкту файл із даними або програмою;

– «Імпортувати бібліотеку» – підключити до програми бібліотеку зі списку встановлених;

– пункт меню «Сервіс» містить допоміжні функції для роботи з самим мікроконтролером;

– «Автоформатування» – автоматична розстановка відступів, переносів рядків тощо;

– «Архівувати скетч» – архівація папки з програмою, і збереження архіву у вказане місце;

– «Монітор порту» – відкрити вікно для обміну даними з мікроконтролером;

– «Плата» – вибір поточної плати (у нашому випадку Arduino Uno);

– «Послідовний порт» – вибір порту, до якого підключено пристрій;

– «Програматор» – вибір програм;

– «Записати завантажувач» – запис програми-завантажувача в мікроконтролер (також нами не використовується). Нарешті, меню «Довідка» містить докладний опис всіх функцій самого редактора Arduino IDE, а також всілякі команди і прийоми роботи з платформою. На панель інструментів винесені у вигляді кнопок найчастіше використовувані функції середовища. Їхнє призначення описано на рисунку 9.4. Безпосередньо текст програми створюється і редагується у вікні редактора коду. Це вікно є типовим текстовим редактором з підсвічуванням синтаксису програми. У нижній частині вікна Arduino IDE є область, яка слугує для виведення повідомлень і повідомлень про помилки, що виникають у процесі компіляції програми або під час завантаження програми в мікроконтролер.

Кнопка	Опис
	Перевірити / Компілювати програму
	Завантажити програму в Arduino
	Створити нову програму
	Відкрити існуючу програму
	Зберегти програму
	Монітор послідовного порту

Рисунок 9.4 – Позначення основних кнопок

Онлайн-емулятор Arduino. Для того щоб виконувати вправи безпосередньо з МК не обов’язково купувати плату (хоча це і бажано, якщо у вас є така можливість). Ми використовуватимемо вебдодаток «Autodesk CIRCUITS», який дозволяє моделювати Arduino і різні електронні компоненти прямо у веббраузері без необхідності завантаження і установки чогось собі на комп’ютер. Ця система абсолютно безкоштовна і все, що вам потрібно для роботи з нею, – вихід в інтернет. Заходимо на сайт http://circuits.io і натискаємо у верхньому правому куті кнопку «Sign up for free». Можна зареєструватися безпосередньо в системі або використовувати можливість входу за допомогою вже наявних облікових записів від Google, Facebook і інших сервісів.

У попередньому розділі ми говорили про мову програмування, яка використовується в Arduino. Ця мова дуже схожа на мову Сі, але є і деякі відмінності. По-перше, коли пишуть програму для Arduino, вона обов’язково має бути написана в одному файлі, який називають скетч. Ви можете часто зустріти таке поєднання слів «скетч для Arduino». Це означає програма для Arduino. Є відмінність у структурі скетчу від структури звичайної програми на Сі, і викликана вона відмінністю між роботою програми на звичайному

комп'ютері і у вбудованих системах. Коли виконують програму на звичайному комп'ютері, то вона запускається, виконує якісь дії, а потім завершується. Якщо програму пишуть мовою Сі, то за її виконання відповідає функція «main ()». Під час запуску програми запускається саме ця функція. З неї програміст вже може викликати інші функції програми в тому порядку, в якому вони мають виконуватися. Коли функція «main ()» повертає управління за допомогою return, програма завершується. Але для програми, що управляє мікроконтролером, така поведінка не має сенсу. Поки на пристрій подано живлення, мікроконтролер має керувати пристроєм, а значить, він повинен виконувати свою програму. Ця програма ніколи не завершується сама, вона перестає працювати, тільки якщо вимкнути мікроконтролер. Тому структура програми для Arduino відрізняється від структури звичайної програми мовою Сі. У програмі для Arduino не має бути функції «main ()», замість неї мають бути дві функції: «setup ()» і «loop ()».

9.7 Сучасні проєкти

Наведемо кілька базових прикладів використання платформи Arduino:

- система «Розумний будинок»;
- робототехніка;
- автоматичні вентилятори;
- світлофори;
- охоронні системи;
- мініметеостанції;
- мультитестери;
- квадрокоптери.

Отже, Arduino – це зручна платформа для реалізації проєктів різної складності. Вона прийнятна як початківцям, які ще не мають навичок у сфері робототехніки, так і досвідченим користувачам. Платформа Arduino за технічним оснащенням максимально підходить для навчального процесу з проєктування різноманітних автоматизованих технічних систем та роботів, завдяки сприйнятливому середовищу програмування, можливості спостереження фізичних процесів у реальному часі. Для платформи Arduino наявна велика кількість матеріалів для розробки, починаючи від бібліотек, які можна використовувати для спрощення програмування, закінчуючи використанням уже готових проєктів, що можуть надихнути на створення нових. Більш потужні плати Arduino (TRE, YUN, DUE) можуть бути застосовані для розв'язання більш складних задач, зв'язаних з розробкою великих проєктів.

Наприклад, регулятор температури в будинку. Реалізувати такий проєкт можна з використанням декількох плат Arduino Nano і однієї Arduino Uno / Mega, яка буде виступати в ролі бази. Зв'язок між модулями можна реалізувати за допомогою NRF24L01 – модуля радіозв'язку, який дає можливість об'єднувати до 6 плат.

Smart-теплиця. Використовуючи всього одну плату Arduino Mega і контролер DHT22, ви зможете фіксувати і виводити на екран інформацію про температуру в теплиці, а також передавати команди на запуск поливу, управління моторами для відкриття і закриття дверей.

Роботи різного типу та складності. Наприклад, за допомогою ультразвукового далекоміра HC-SR04 ваш робот зможе фіксувати відстань до перешкод і огинати їх під час руху. Застосувавши драйвер двигунів L293D, ви отримаєте в своє розпорядження 3 сервоприводи і 4 двигуни. За допомогою модуля HC-06 у вас з'явиться можливість управляти роботом по Bluetooth через смартфон.

3D-принтер. Крім самої плати, необхідні будуть драйвери двигунів L298N, а також самі двигуни. Інша частина роботи – це рама і розробка програмного коду.

10 АЛГОРИТМ СИСТЕМИ ВІДДАЛЕНОГО УПРАВЛІННЯ ОСВІТЛЕННЯМ

Як і будь-яка програмована система, Arduino складається з апаратної частини – друкованої плати з електронними компонентами, роз'ємами для під'єднання, зокрема і типу USB, а також іншими ланцюгами, таку плату у комп'ютерників на професійному сленгу прийнято називати «залізом»; програмної частини – вона серед «ардуїнщиків» зветься «скетч» чи готової програми.

У комплектацію набору заліза для розумного освітлення на Arduino входять готові стандартні деталі та компоненти (рис. 10.1):

– плата для макетування системи розумного освітлення;
– кілька світлодіодів, постійних резисторів та конденсаторів;
– акумуляторна батарея на напругу 9;
– клавіатура матричного типу;
– плата розширення;
– кнопки увімкнення / вимкнення;
– різнокольорові провідні перемички.

У програмному пакеті серед готових програм управління освітленням Arduino, вони ж скетчі, передбачено такий цикл управління, як «імітація присутності» людини у приміщенні. По ньому світло в різних приміщеннях будинку автоматично вмикається і через деякий час вимикається. Для спостерігача з вулиці це вигладить як присутність принаймні однієї людини в будівлі. Він активується включенням із пульта ДУ:

– пожежо-охоронні, що спрацьовують на розбитті скла на найважливіших вікнах, їхнього несанкціонованого відчинення, давачі відкриття дверей, воріт та хвіртки;

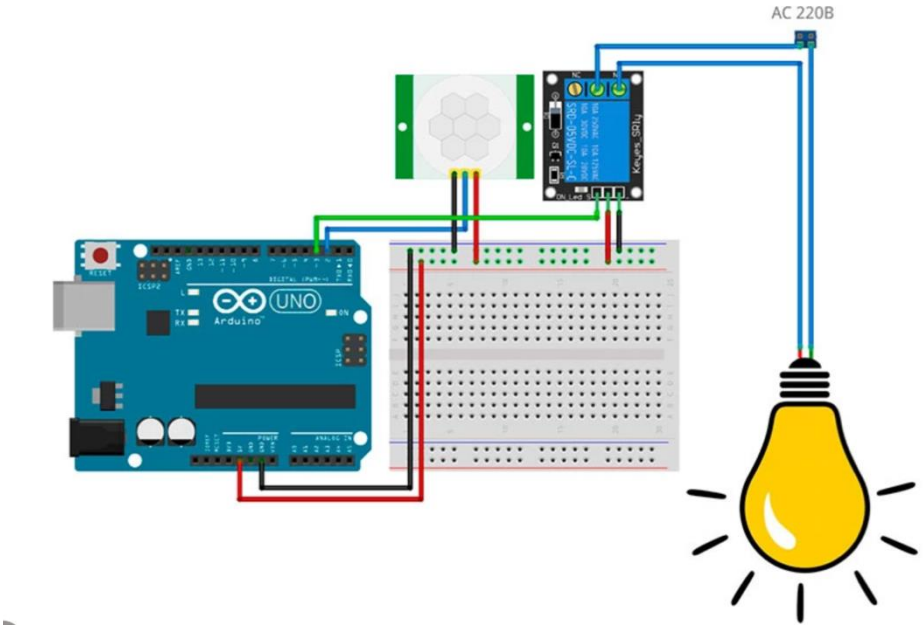


Рисунок 10.1 – Приклад застосування для освітлення

– протікання води з водопроводу та/або каналізації;
– температури повітря в будинку у відповідальних кімнатах та на північній стіні будинку;
– рухи та присутності;
– освітлення на відкритому просторі – для дозволу роботи всіх вуличних світильників вашого будинку тощо.

Всі ці пристрої, давачі, підсистеми керування світильниками легко вивчити самостійно та зібрати своїми руками схему на макетній платі.

10.1 Периферія, яка може бути присутня у мікроконтролерах

Основна периферія, яка може бути присутня в мікроконтролерах:

1. Порти вводу / виводу – вони використовуються для зв’язку із зовнішнім світом, тому до портів можна підключити різні пристрої. Вони можуть працювати двонаправлено, тобто порти можуть виводити інформацію, введену в них мікроконтролером, або зчитувати дані, що з’являються на лінії порту ззовні. Порти можуть бути цифровими або аналоговими. Цифровий ввід / вивід дає можливість безпосереднього моніторингу та управління апаратним забезпеченням, є основною характеристикою мікроконтролерів. Мікроконтролер за своєю суттю цифровий, тому нам потрібні відповідні способи перетворення аналогових сигналів у цифровий і назад. Цю проблему вирішує аналоговий модуль мікроконтролера.

2. Аналогово-цифровий перетворювач (АЦП) – використовується для перетворення аналогового сигналу в цифровий. Процес полягає у спрощенні аналогового сигналу до дискретної форми.

3. Цифрово-аналоговий перетворювач (ЦАП) – електронний пристрій, який перетворює цифровий сигнал в аналоговий сигнал у вигляді електричного

струму або напруги, пропорційних цьому числу. Іншими словами, це система, яка перетворює дискретний цифровий сигнал в еквівалентний аналоговий сигнал.

4. Інтерфейси зв'язку – мікроконтролери зазвичай містять кілька інтерфейсів зв'язку, а іноді навіть кілька екземплярів певного інтерфейсу, наприклад два модулі UART. Основна мета будь-якого такого інтерфейсу полягає в тому, щоб дозволити мікроконтролеру взаємодіяти з іншими блоками, чи то інші мікроконтролери, периферія або хост-ПК. Реалізація таких інтерфейсів може приймати різні форми, але здебільшого інтерфейси можна класифікувати за кількістю властивостей: вони можуть бути як послідовними, так і паралельними, синхронними або асинхронними, використовувати шину або зв'язок point-point, бути повнодуплексний або напівдуплексний, і може бути заснований на принципі master-slave (ведучий-ведений).

Основні типи інтерфейсів:

– SCI (англ. Serial Communication Interface) – інтерфейс послідовного зв'язку забезпечує асинхронний інтерфейс зв'язку (англ. Universal Asynchronous Receiver Transmitter, UART). Модуль UART використовує два дроти, передавальну (TXD) і приймальну (RXD) лінію для повного або напівдуплексного зв'язку;

– SPI (англ. Serial Peripheral Interface) – це простий синхронний інтерфейс point-point, заснований на принципі master-slave. Він забезпечує повнодуплексний зв'язок між ведучим (зазвичай контролером) та одним (або більше) веденими пристроями (зазвичай периферійними пристроями);

– IIC (англ. Inter Integrated Circuit) – це синхронна шина, яка працює за принципом master-slave. Він використовує два дроти SCL (послідовна лінія годинника, англ. Serial Clock Line) та SDA (послідовна лінія передачі даних, англ. Serial Data Line) для напівдуплексного зв'язку. Протокол широко використовується для зв'язку на короткі відстані між одним або кількома контролерами та периферійними пристроями [23];

– USB (англ. Universal Serial Bus) – універсальна послідовна шина, що призначена для з'єднання мікроконтролерів та периферійних пристроїв.

10.2 Технічні рішення та переваги

Схема управління освітленням з будь-якого ІЧ-пульта реалізована у такий спосіб.

Після натискання на кнопку ІЧ-пульта він відправляє пакет закодованих і промодульованих даних на ІЧ-світлодіод, а у разі утримання кнопки ще й пакети повтору. Невидиме інфрачервоне світло від ІЧ-світлодіода потрапляє на Trima-модуль ІЧ-приймач, де перетворюються на демодульовані електричні імпульси.

Бібліотека iarduino_IR постійно зчитує та розкодує імпульси з ІЧ-приймача (використовуючи другий апаратний таймер Arduino).

У коді loop ми звіряємо розкодовані бібліотекою дані з призначеними пристрою (лампі) і якщо вони збіглися, то змінюємо стан («1»/«0») на вході

Trema-модуля Твердотільне реле, отже, вмикаємо або вимикаємо пристрій (лампу).

Пристрій може бути корисним, якщо призначити кнопки телевізійного пульта, що не використовуються (наприклад кольорові кнопки телетексту) для управління освітленням в кімнаті. Замість ламп розжарювання можна підключити будь-який пристрій мережі ~220 В, зі струмом споживання до 2 А.

Для проєкту знадобиться:

– Arduino плата – 1 шт.;
– ІЧ-пульт дистанційного керування – 1 шт. (підійде будь-який телевізійний ІЧ-пульт);
– Trema-модуль ІЧ-приймач – 1 шт.;
– Trema-модуль Твердотільне реле – 3 шт.;
– Trema Shield – 1 шт.;
– пристрої (лампи), якими ми керуватимемо – 3 шт.

Для реалізації проєкту нам необхідно встановити бібліотеку: «Бібліотека iarduino_IR для роботи з ІЧ-приймачами».

ВАЖЛИВО: бібліотека використовує другий апаратний таймер. **НЕ ВИВОДІТЬ СИГНАЛИ ШИМ НА 3 АБО 11 ВИВОДУ!**

У цій схемі використовуються тільки цифрові модулі, їх можна підключати до будь-яких (як цифрових, так і аналогових) виводів Arduino. Наприклад, ми підключили всі модулі до аналогових виводів. Не всі знають, що аналогові виводів Arduino можуть працювати як звичайні цифрові виводи, що дозволяють отримувати (від ІЧ-приймача) і передавати цифрові сигнали (на твердотільні реле) у вигляді логічних «0» і «1».

Якщо Ви підключатимете пристрої до інших виводів, то їхні номери потрібно вказати в другому (оголошення об’єкта IR) і третіх (оголошення масиву pinRelay) рядках скетчу. Код програми написаний так, що ви можете підключити стільки реле, скільки є вільних виводів у Arduino, просто перерахувавши номери виводів у третьому рядку скетчу (оголошення масиву pinRelay).

Не встановлюйте Trema-модуль ІЧ-приймач поряд із джерелами яскравого світла. Він може перешкоджати прийому слабкого інфрачервоного світла від ІЧ-пульта.

Алгоритм роботи

Під час старту скетч послідовно призначає код кнопки ІЧ-пульта кожному твердотільному реле. Спочатку блимає світлодіод і замикається ланцюг твердотільного реле, вивід якого вказаний першим. Якщо натиснути на будь-яку кнопку ІЧ-пульта, то код цієї кнопки присвоюється цьому реле і почне блимати наступне, і так, доки всім реле не будуть призначені кнопки ІЧ-пульта. Після призначення кнопок всі реле знаходяться у вимкненому стані. Якщо натиснути кнопку ІЧ-пульта, то ввімкнеться реле, якому присвоєно код цієї кнопки. Якщо повторно натиснути на вказану кнопку, реле вимкнеться. Пристрій не відреагує на натискання клавіш, код яких не присвоювався жодному реле. Якщо Ви не хочете надавати коди кнопок при кожному старті, то явно вкажіть їх при оголошенні масиву varRelay, а з коду setup видаліть

цикли for і while (рис.10.2). Дізнатися код кожної кнопки можна, написавши такий рядок: if(IR.check()){Serial.println(IR.data);}

Рисунок 10.2 – Прописаний скетч для проєкту

11 ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРА ДЛЯ ПІДКЛЮЧЕННЯ СИСТЕМИ ОСВІТЛЕННЯ ТА ЇЇ ТЕСТУВАННЯ

Давач руху (рис. 11.1) з урахуванням піроелектричного ефекту (PIR, passive infrared motion sensor). Такі давачі часто використовуються в охоронних системах та в побуті для виявлення руху в приміщенні. Наприклад, на принципі детектування руху засноване автоматичне включення світла у під'їзді або у ванній. Піроелектричні давачі достатньо просто влаштовані, недорогі та невибагливі в установці та обслуговуванні. До речі, існують інші способи детектування руху. Сьогодні все частіше використовують системи комп'ютерного зору для розпізнавання об'єктів та траєкторії їхнього переміщення.

У тих самих охоронних системах застосовуються лазерні детектори, які дають тривожний сигнал під час перетину променя. Також використовуються тепловізійні давачі, здатні визначити рух живих істот.

Для виконання простого прикладу з давачем руху, крім самого модуля давача, знадобиться Arduino-сумісний контролер, релейний модуль, макетна плата та трохи проводів вилка-розетка та вилка-вилка.

Рисунок 11.1 – Давач руху. Приклад

Піроелектрики – це діелектрики, які створюють електричне поле при зміні їхньої температури. На основі піроелектриків роблять давачі вимірювання температури, наприклад LHI778 або IRA-E700 (рис. 11.2). Кожен такий давач містить два чутливі елементи розміром 1 мм × 2 мм, що підключені з протилежною полярністю. І як ми побачимо далі, наявність двох елементів допоможе нам детектувати рух.

Рисунок 11.2 – Давач IRA-E700 компанії Murata

Працюватимемо з давачем руху HC-SR501, у якому встановлений один такий піроелектричний давач. Зверху піроелектрик оточений напівсферою, розбитою на кілька сегментів. Кожен сегмент цієї сфери є лінзою, яка фокусує теплове випромінювання на різні ділянки ПІР-давача. Часто як лінзу використовують лінзу Френеля (рис. 11.3).

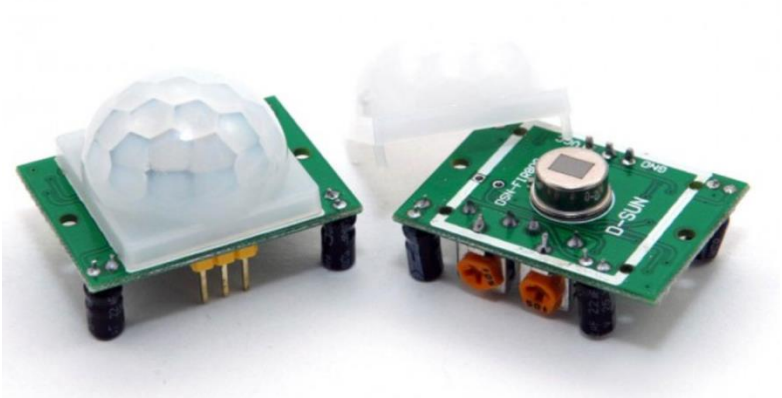
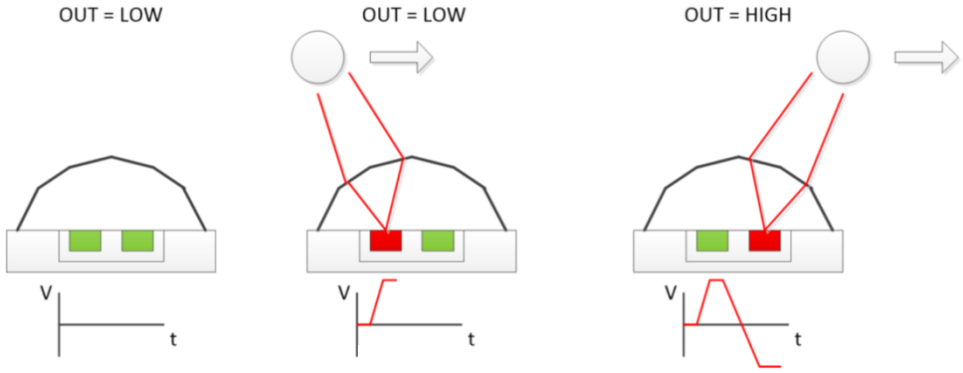


Рисунок 11.3 – Давач руху з лінзою Френеля

Принцип роботи давача руху такий. Припустимо, що давач встановлено в порожній кімнаті. Кожен чутливий елемент отримує постійну дозу випромінювання, а отже, і напруга на них має постійне значення (рис. 11.4, а).



а б в

Рисунок 11.4 – Принцип роботи давача руху

Як тільки в кімнату заходить людина, вона потрапляє спочатку до зони огляду першого елемента, що призводить до появи позитивного електричного імпульсу на ньому (рис. 11.4, б). Людина рухається, і її теплове випромінювання через лінзи потрапляє вже другий PIR-елемент, який генерує негативний імпульс. Електронна схема давача руху реєструє ці різноспрямовані імпульси та робить висновки про те, що в поле зору давача потрапила людина. На виході давача генерується позитивний імпульс (рис. 11.4, в).

Далі необхідно налаштувати давач руху HC-SR501 (рис. 11.5). Цей модуль дуже поширений і застосовується в багатьох DIY проєктів через свою дешевизну. У давача є два змінних резистора та перемичка для налаштування режиму. Один із потенціометрів регулює чутливість приладу. Чим вона більша, тим далі «бачить» давач. Також чутливість впливає розмір детектированого об’єкта. Наприклад, можна виключити зі спрацьовування собаку чи кішку.

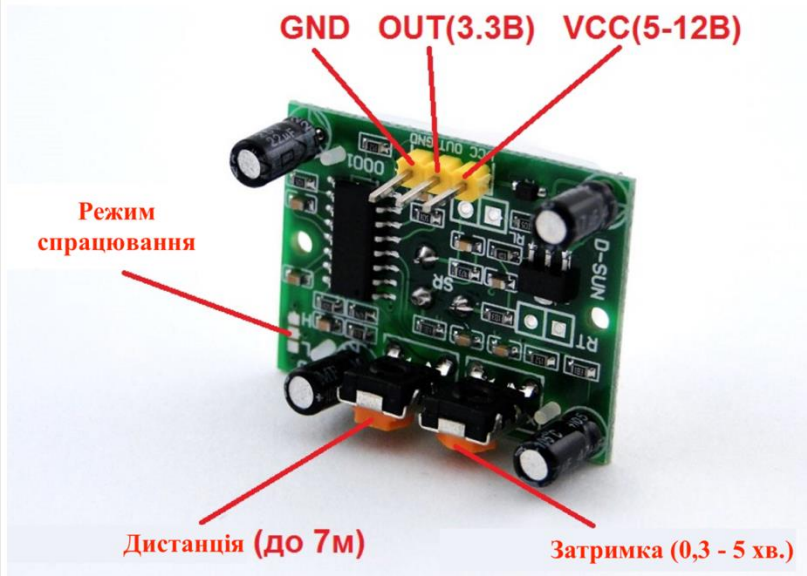


Рисунок 11.5 – Регулювання давача руху

Другий потенціометр регулює час спрацьовування T. Якщо давач виявив рух, він на виході генерує позитивний імпульс довжиною T. Нарешті, третій елемент керування – перемичка, яка перемикає режим давача. У положенні L давач веде відлік T від першого спрацьовування. Допустимо, ми хочемо керувати світлом у ванній кімнаті. Зайшовши в кімнату, людина викличе спрацювання давача, і світло увімкнеться рівно на час T. Після закінчення періоду, сигнал на виході повернеться у вихідний стан, і давач буде очікувати наступне спрацьовування. У положенні давач H починає відлік часу T кожного разу після виявлення руху. Інакше кажучи, будь-яке ворушіння людини викликає обнулення таймера відліку T. За замовчуванням, перемичка перебуває у стані H.

Для з’єднання з мікроконтролером або безпосередньо з реле HC-SR501 має три виходи. Підключаємо їх до Arduino за такою схемою (рис. 11.6):

HC-SR501	GND	VCC	OUT
Arduino UNO	GND	+5V	2

Принципова схема

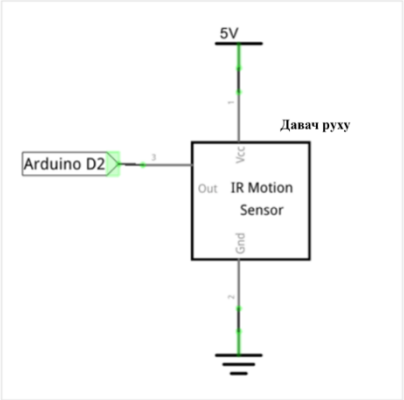


Рисунок 11.6 – Схема підключення давача руху

Зовнішній вигляд макета наведено на рисунку 11.7.

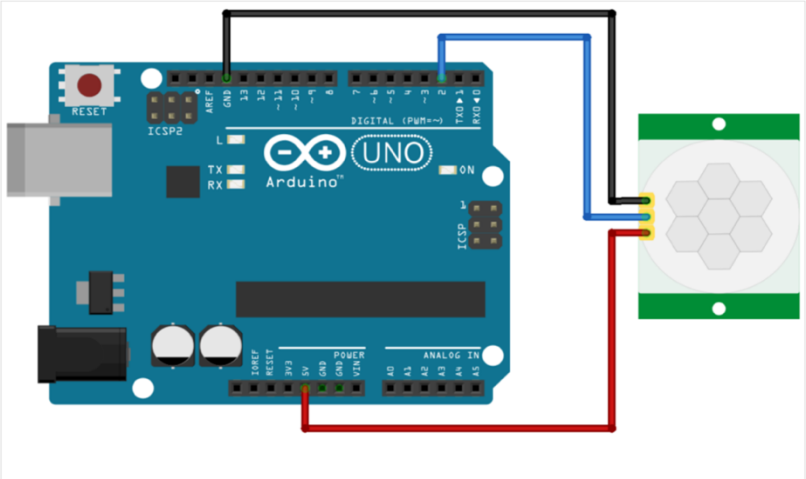


Рисунок 11.7 – Схема підключення давача руху та плати Arduino UNO

Як було зазначено, цифровий вихід давача HC-SR501 генерує високий рівень сигналу при спрацьовуванні. Далі напишемо просту програму, яка відправлятиме в послідовний порт «1», якщо давач побачив рух, і «0» в іншому випадку (рис. 11.8).

Рисунок 11.8 – Скетч програми до спрацювання давача руху

Завантажуємо програму на Arduino та перевіряємо роботу давача. Можна покрутити налаштування давача і подивитися, як це позначиться на його роботі. Наступний крок – система автоматичного включення світла. Для того щоб керувати освітленням у приміщенні, нам потрібно додати до ланцюга реле. Будемо використовувати модуль реле із захистом на основі опторозв'язки.

Увага! Ця схема запалює лампу від мережі 220 Вольт. Рекомендується сім разів перевірити всі з'єднання перед тим, як з'єднувати схему з побутовою електромережею. На рисунку 11.9 наведено принципову схему.

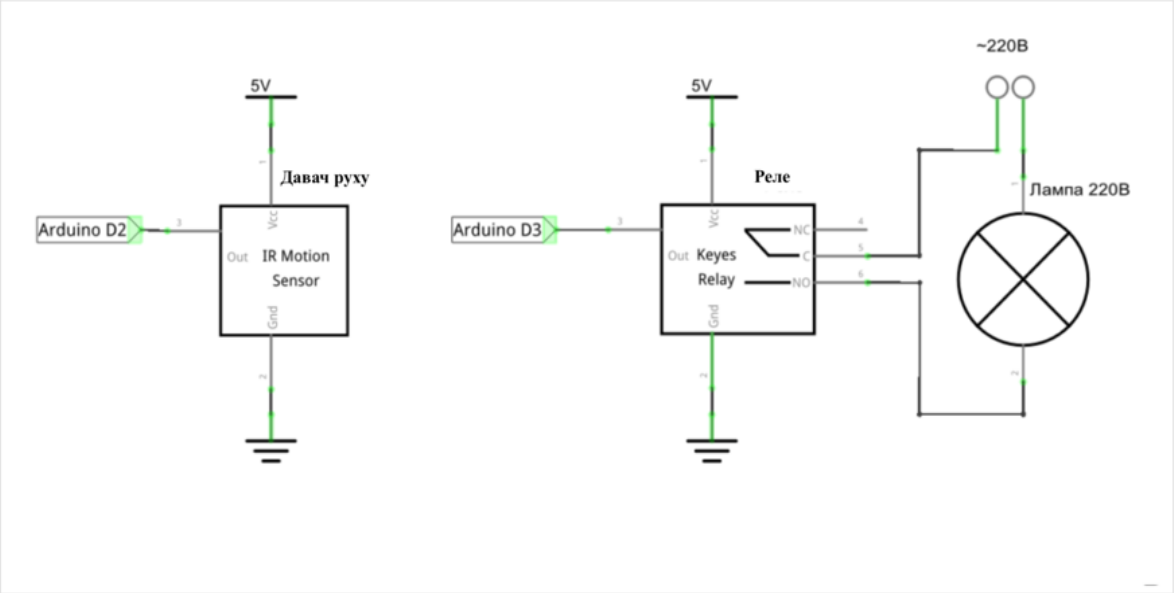


Рисунок 11.9 – Схема підключення лампи

Вигляд макета наведено на 11.10.

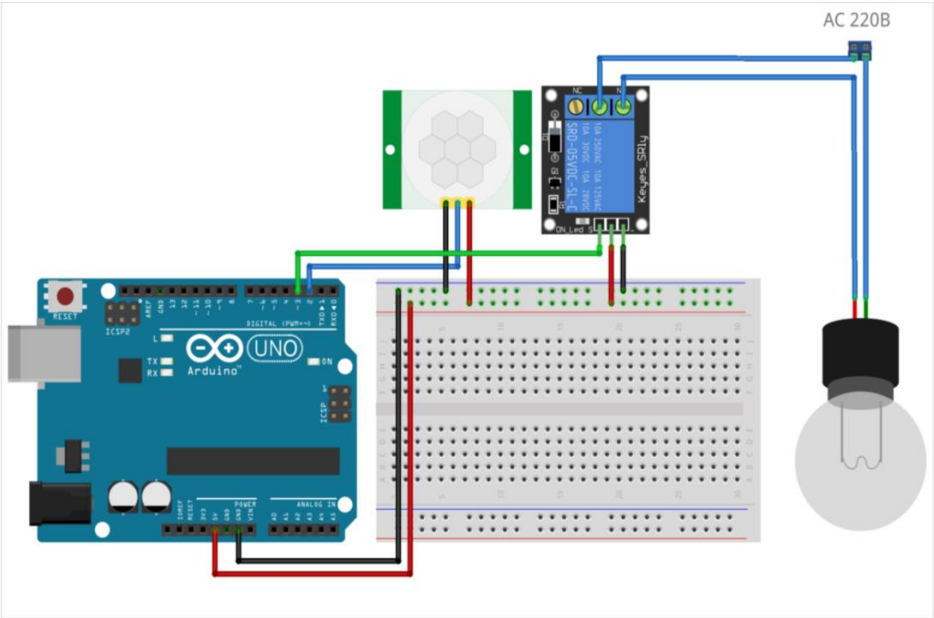


Рисунок 11.10 – Зібраний макет із лампою

Тепер напишемо програму, яка при спрацюванні давача включатиме реле, а отже й освітлення в кімнаті (рис. 11.11).

Рисунок 11.11 – Програма-скетч для спрацювання освітлення в кімнаті

Завантажуємо програму на Arduino, акуратно підключаємо схему до побутової мережі та перевіряємо роботу давача.

Давачі руху оточують нас усюди. Завдяки охоронним системам їх можна зустріти практично в кожному приміщенні. Як ми з'ясували, вони дуже прості у використанні та можуть бути легко інтегровані у будь-який проєкт на Arduino або Raspberry Pi.

Ось кілька ситуацій і місць, де може стати в нагоді давач руху:

– автоматичне включення світла у під'їзді будинку, у ванній кімнаті та туалеті, перед вхідними дверима до приміщення;
– сигналізація у приміщенні та у дворі;
– автоматичне відчинення дверей;
– автоматичне увімкнення охоронної відеокамери.

СПИСОК РЕКОМЕНДОВАНИХ ДЖЕРЕЛ

1. Дистанційний курс ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ [Електрон. ресурс]. – Електрон. текст. дані. – Режим доступу : https://dl.kname.edu.ua/course/index.php?categoryid=51, вільний (дата звернення: 12.01.2025). – Назва з екрана.
2. Методичні рекомендації до виконання лабораторних та контрольних робіт, самостійного вивчення курсу з навчальної дисципліни «Програмування мікроконтролерів» (для бакалаврів денної і заочної форм навчання спеціальності 141 – Електроенергетика, електротехніка та електромеханіка освітньо-професійної програми «Світлотехніка та дизайн світлового середовища») [Електрон. ресурс] / Харків. нац. ун-т міськ. госп-ва ім. О. М. Бекетова ; уклад. : А. І. Колесник, В. М. Поліщук. – Електрон. текст. дані. – Харків: ХНУМГ ім. О. М. Бекетова, 2023. – 26 с. – Режим доступу: https://eprints.kname.edu.ua/65508/1/2023%20204M%20репоз.%20%20Колесник %20фініш.pdf, вільний (дата звернення: 08.03.2025). – Назва з екрана.
3. Релейне обладнання [Електрон. ресурс]. – Електрон. текст. дані. – Режим доступу : https://axiomplus.com.ua/ua/relejnoe-oborudovanie/, вільний (дата звернення: 15.01.2025). – Назва з екрана.
4. Фільтри стримання завад [Електрон. ресурс]. – Електрон. текст. дані. – Режим доступу: https://www.phoenixcontact.com/uk-ua/produkcija/emc-filters, вільний (дата звернення: 02.03.2024). – Назва з екрана.
5. Аналізатор Fluke 1738 [Електрон. ресурс]. – Електрон. текст. дані. – Режим доступу: https://iron-harry.ua/tovar/70004/, вільний (дата звернення: 18.03.2024). – Назва з екрана.
6. Blum J. Exploring Arduino: Tools and Techniques for Engineering Wizardry / J. Blum. – John Wiley & Sons, Inc., 2019. – 512 p.
7. Програмування мікроконтролерів AVR : навч. посіб. / С. М. Цирульник, О. Д. Азаров, Л. В. Крупельницький, Т. І. Трояновська. – Вінниця : ВНТУ, 2018. – 111 с.
8. Методичні вказівки до виконання лабораторних робіт з дисципліни «Мікроелектронні та мікропроцесорні пристрої та системи» освітніх програм «Електричні та електронні апарати, Електромеханічне обладнання енергоємних виробництв» для студентів спеціальності 141 – Електроенергетика, електротехніка та електромеханіка всіх форм навчання. / Уклад. : М. О. Поляков. – Запоріжжя : Національний університет «Запорізька політехніка», 2022. – 35 с.
9. Татарчук Д. Д. Програмування мовами С та С++ : навч. посіб. / Д. Д. Татарчук, Ю. В. Діденко. – Київ : 2012. – 112 с.
10. Arduino IDE 1.8.13 [Електрон. ресурс] : сайт. – Електрон. текст. дані. – Режим доступу: https://www.arduino.cc/en/Main/Software /, вільний (дата звернення: 13.01.2025). – Назва з екрана.

11. ГОСТ 13109-97 Електрична енергія. Сумісність технічних засобів електромагнітна. Норми якості електричної енергії в системах електропостачання загального призначення [Електрон. ресурс]. – Чинний від 2000–01–01 – Електрон. текст. дані. – – Режим доступу: https://dnaop.com/html/42313/doc-ГОСТ_13109-97, вільний (дата звернення: 12.01.2025). – Назва з екрана.

12. Мікропроцесорні пристрої : навч. посіб. для студентів зі спец-ті «Електроніка» / Т. О. Терещенко, В. А. Тодоренко, Л. М. Батрак, Ю. С. Ямненко. – Київ : Кафедра, 2017. – 244 с.

13. Глухов О. В. Вивчення властивостей мікроконтролерів і електронних систем на базі платформи Ардуіно : навч. посіб. для студентів ВНЗ / О. В. Глухов, О. О. Кравчук, Є. В. Левченко. – Харків : ХНУРЕ, 2019. – 192 с.

14. Цирульник С. М. Проектування мікропроцесорних систем : навч. посіб. / С. М. Цирульник, Г. Л. Лисенко. – Вінниця : ВНТУ, 2010. – 201 с.

15. Мікропроцесорні та мікроконтролерні системи: Частина 2. Проектування мікропроцесорних систем: Лабораторний практикум [Електрон. ресурс] : навч. посіб. для студ. освітньої програми «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології» / А. О. Новацький ; КПІ ім. Ігоря Сікорського. – Електрон. текст. дані. – (1 файл: 22,38 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 268 с. – Режим доступу: https://www.google.ru/url?sa=t&source=web&rct=j&opi=89978449&url=https://ela.kpi.ua/bitstream/123456789/43054/1/MP_ta_MKS_2_LabPrakt.pdf&ved=2ahUKEwj96oLGrNyNAxVoU1UIHRdqPVkQFnoECA0QAQ&usg=AOvVaw3IFMvrUpCtmgRFvWfphA4F, вільний (дата звернення: 12.02.2025). – Назва з екрана.

16. Основи програмування на C/C++ в прикладах. Частина 2 : навч.-метод. посіб. / М. О. Соболь, Н. Ю. Любченко, А. В. Івашко, Ю. В. Паржин, Р. В. Пугачов. – Харків : НТУ «ХПІ», 2022. – 200 с.

17. Мікроконтролери AVR. [Електрон. ресурс] – Електрон. текст. дані. – Режим доступу: http://www.studmed.info/docs/document3940/content., вільний (дата звернення: 06.01.2025). – Назва з екрана.

18. Hadwan H. H. Smart Home Control by using Raspberry Pi & Arduino UNO / H. H. Hadwan, Y. P. Reddy // International Journal of Advanced Research in Computer and Communication Engineering. – 2016. – Vol. 5, № 4. – P. 283–288.

19. Smart Home Automation and Security System using Arduino and IOT / S. Wadhwani, U. Singh, P. Singh, S. Dwivedi // International Research Journal of Engineering and Technology. – 2018. – Vol.5, № 2. – P. 1357–1359.

20. Jabbar Z. A. Implementation of Smart Home Control by Using Low Cost Arduino & Android Design / Z. A. Jabbar, R. S. Kawitkar // International Journal of Advanced Research in Computer and Communication Engineering. – 2016. – Vol. 5, № 2. – P. 248–256.

21. Atmel AVR 8-bit and 32-bit Microcontrollers. [Electronic resource]. – Electronic text data. – Regime of access: http://www.atmel.com/products/microcontrollers/avr/, free (date of the application: 24.11.2024). – Header from the screen.

22. Onsemi specializes in delivering industry-leading intelligent power and intelligent sensing solutions. [Electronic resource]. – Electronic text data. – Regime of access: https://www.fairchildsemi.com/datasheets/FQ/FQP2N60C.pdf, free (date of the application: 27.12.2024). – Header from the screen.

23. Greenchip.com.ua [Електрон. ресурс] – Електрон. текст. дані. – Режим доступу: http://articles.greenchip.com.ua/, вільний (дата звернення: 12.04.2020). – Назва з екрана.

24. Tutorials [Electronic resource]. – Electronic text data. – Regime of access: https://docs.arduino.cc/tutorials/, free (date of the application: 217.10.2024). – Header from the screen.

25. Cornel A. Arduino Development Cookbook / A. Cornel. – UK : Packt Publishing, 2015. – 246 p.

26. Ришкова К. О. Мікроконтролерні модулі як складові Web-систем / К. О. Ришкова, В. С. Лобода // Фізика, електроніка, електротехніка ФЕЕ-2020 : матеріали Міжнародної науково-технічної конференції студентів та молодих вчених. – Суми : СумДУ, 2020. – С. 85.

27. Мікропроцесорні пристрої : навч. посіб. для студ. зі спец. «Електроніка» / Т. О. Терещенко, В. А. Тодоренко, Л. М. Батрак, Ю. С. Ямненко. – Київ : Кафедра, 2017. – 244 с.

Електронне навчальне видання

КОЛЕСНИК Анастасія Ігорівна

ПРОГРАМУВАННЯ МІКРОКОНТРОЛЕРІВ

КОНСПЕКТ ЛЕКЦІЙ

(для здобувачів першого (бакалаврського) рівня вищої освіти денної та заочної форм навчання зі спеціальності 141 – Електроенергетика, електротехніка та електромеханіка)

Відповідальний за випуск *В. А. Герасименко*
Редактор *О. В. Михаленко*
Комп’ютерне верстання *А. І. Колесник*

План 2024, поз. 108Л

Підп. до друку 23.05.2025. Формат 60 × 84/16.
Ум. друк. арк. 5,6.

Видавець і виготовлювач:
Харківський національний університет
міського господарства імені О. М. Бекетова,
вул. Чорноглазівська (Маршала Бажанова), 17, Харків, 61002.
Електронна адреса: office@kname.edu.ua
Свідоцтво суб’єкта видавничої справи:
ДК № 5328 від 11.04.2017.