Проектування вбудованих систем

Лабораторний практикум

Рекомендовано Методичною радою КПІ ім.Ігоря Сікорського
як навчальний посібник для студентів, які навчаються за освітньою програмою «Інтегровані інформаційні системи» за спеціальністю
126 «Інформаційні системи та технології»

Київ
КПІ ім. Ігоря Сікорського
2022

Проектування вбудованих систем: Лабораторний практикум [Електронний ресурс] : навч. посіб. для студ. освітньої програми «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології» / А.О. Новацький, В.М. Шимкович; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 40,85 Кбайт). – Київ : КПІ ім. Ігоря Сікорського, 2022. – 463 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 5 від 26. 05. 2022 р.) за поданням Вченої ради факультету Інформатики та обчислювальної техніки (протокол № 5 від 28. 12. 2021 р.)

Електронне мережне навчальне видання

Новацький Анатолій Олександрович, к.т.н., доцент, Шимкович Володимир Михайлович, к.т.н., старший викладач

ПРОЕКТУВАННЯ ВБУДОВАНИХ СИСТЕМ

Лабораторний практикум

Відповідальний редактор: Полторак В. П., к.т.н., доцент, КПІ ім. Ігоря Сікорського, факультет інформатики та обчислювальної техніки, кафедра інформаційних систем та технологій

Рецензент: Селіванов Віктор Львович, к.т.н., доцент, КПІ ім. Ігоря Сікорського, факультет інформатики та обчислювальної техніки, кафедра обчислювальної техніки

Навчальний посібник охоплює теоретичний матеріал та практичні завдання, які необхідні для виконання лабораторного практикуму з дисципліни «Проектування вбудованих систем»». Практикум виконується у комп'ютерному класі кафедри з використанням симулятора «AVR-studio» та моделюючого пакету PROTEUS. В посібнику наводяться рекомендації по використанню цих програмних продуктів та виконанню робіт з наступної тематики: дослідження команд AVR-мікроконтролерів, моделювання периферійних модулів мікроконтролера: універсального асинхронного приймача-передавача; цифро-аналогового перетворювача, інтерфейсів RS-485 та RS-232; аналого-цифрового перетворювача та цифрового вольтметра; пристроїв керування двигуном постійного струму, кроковим двигуном та LCD-дисплеєм, мережі 1-WIRE та годинника реального часу. В роботі наводяться приклади окремих команд мікроконтролера, схеми моделювання периферійних модулів, алгоритми та керуючі програми мовою С та Асемблер для мікроконтролерів сім`ї AVR.

Робота може бути корисною студентам відповідних спеціальностей при вивченні дисциплін, пов'язаних із використанням мікропроцесорних та мікроконтролерних пристроїв та систем, а також при виконанні бакалаврських робіт, курсових проектів, магістерських робіт, в яких використовуються відповідні пристрої. Останнє було враховано при оформленні роботи, яке виконано згідно вимог до конструкторської документації.

ЗМІСТ

ВСТУП

При виконанні лабораторних робіт використовуються симулятор «AVR-studio» та моделюючий пакет PROTEUS. У якості мікроконтролерів розглядається сім`я AVR, яка є представником одного з цікавих напрямів, що розвиваються корпораціями Atmel та Microchip. Об’єми продажів AVR у світі щорічно збільшуються, є багато сторонніх фірм, що випускають програмні й апаратні засоби підтримки розробок для цих мікроконтролерів. Можна вважати, що AVR поступово став одним з індустріальних стандартів серед 8-розрядних мікроконтролерів загального призначення. В даний час у виробництві знаходяться три сімейства AVR: "Tiny", "Mega" та "XMega", що розрізняються об’ємами масивів Flash-, EEPROM- і SRAM-пам’яті, набором периферійних вузлів і побудовою схеми тактування.

AVR являє собою 8-розрядний RISC-мікроконтролер, що має швидке процесорне ядро, Flash-пам’ять програм-ROM, пам’ять даних-SRAM, порти введення/виведення і велику кількість інтерфейсних схем. Гарвардська архітектура AVR реалізує повний логічний і фізичний поділ не тільки адресних просторів, але й інформаційних шин для звернення до ROM і SRAM. Така побудова вже ближче до структури цифрових сигнальних процесорів і забезпечує істотне підвищення продуктивності. Використання однодорівнюєвого конвеєра в AVR також помітно скоротило цикл "вибірка – виконання" команди. Наприклад, у стандартних мікроконтролерів сімейства MCS-51 коротка команда виконується за 12 тактів генератора (1 машинний цикл), протягом якого процесор послідовно зчитує код операції і виконує її. У AVR-мікроконтролерах коротка команда в загальному потоці теж виконується за один машинний цикл, але він складає всього один період тактової частоти. Відміною рисою архітектури AVR є регістровий файл швидкого доступу, що містить 32-байтових регістри загального призначення. Шість регістрів файлу можуть використовуватися як три 16-розрядних покажчики адреси при непрямій адресації даних

(X, Y і Z Pointers), що істотно підвищує швидкість пересилання даних при роботі прикладної програми.

Flash-пам’ять програм AVR може бути завантажена як за допомогою звичайного програматора, так і за допомогою SPI-інтерфейсу, у тому числі безпосередньо на робочій платі – функція ISP. Останні версії кристалів "Mega" та "XMega" мають можливість самопрограмування (функція SPM). Всі AVR мають також блок енергонезалежної пам’яті даних – EEPROM, доступний програмі мікроконтролера безпосередньо в ході її виконання. EEPROM звичайно використовується для збереження проміжних даних, констант, таблиць перекодувань, каліброваних коефіцієнтів і т. ін. Ця пам’ять може бути завантажена ззовні як через SPI-інтерфейс, так і за допомогою звичайного програматора. Два програмованих біти дозволяють захистити ROM і енергонезалежну пам’ять даних EEPROM від несанкціонованого зчитування. У більшість мікроконтролерів входить внутрішня оперативна пам’ять – SRAM. Для деяких мікроконтролерів можливе підключення зовнішньої пам’яті даних об’ємом до 64К.

Внутрішній тактовий генератор AVR може запускатися від зовнішнього генератора або кварцового резонатора, а також від внутрішнього або зовнішнього RC-ланцюга. Всі AVR цілком статичні, їх мінімальна робоча частота нічим не обмежена (аж до покрокового режиму). Деякі мікроконтролери мають додатковий блок – PLL для апаратного збільшення основної тактової частоти в 16 разів. Ця частота може служити джерелом для одного з таймерів/лічильників мікроконтролера, значно підвищуючи точність його роботи.

Мікроконтролери сім`ї AVR мають від 1 до 6 таймерів/лічильників загального призначення з розрядністю 8 або 16 біт.

Загальні риси всіх таймерів/лічильників наступні:

– наявність програмованого попереднього дільника вхідної частоти з різними градаціями ділення. Відміною рисою є можливість роботи таймерів/лічильників на основній тактовій частоті мікроконтролера без попереднього її зниження, що помітно підвищує точність генерації часових інтервалів системи;

– незалежне функціонування від режиму роботи процесорного ядра мікроконтролера (тобто вони можуть бути як зчитані, так і завантажені новим значенням у будь-який час);

– можливість роботи або від зовнішнього джерела опорної частоти, або як лічильник зовнішніх подій. Верхній частотний поріг визначений у цьому випадку як половина основної тактової частоти мікроконтролера. Вибір перепаду зовнішнього джерела (фронт або зріз) програмується користувачем;

– наявність різних векторів переривань для подій "переповнення вмісту", "захоплення", "порівняння";

– вартовий таймер у AVR має свій власний RC-генератор, частота якого залежить від величини напруги споживання мікроконтролера і від температури. Вартовий таймер містить окремий програмований попередній дільник вхідної частоти, що дозволяє підлаштовувати часовий інтервал переповнення таймера і скидання мікроконтролера. Даний таймер можна програмно відключати під час роботи мікросхеми, як в активному режимі, так і в кожному з режимів зниженого енергоспоживання. В останньому випадку це приводить до значного зниження споживаного струму;

– в AVR-мікроконтролер вбудовано систему реального часу – RTC. Таймер/лічильник RTC має окремий попередній дільник, що може бути програмним способом підключений або до джерела основної тактової частоти, або до додаткового асинхронного джерела опорної частоти (кварцовий резонатор або зовнішній синхросигнал). Для цієї мети зарезервовано два виводи мікросхеми. Внутрішній генератор, навантажений на лічильний вхід таймера/лічильника RTC, оптимізовано для роботи з зовнішнім "годинниковим" кварцовим резонатором 32,768 кГц;

– порти введення/виведення AVR мають значне число незалежних ліній "Вхід/Вихід". Вихідні драйвери портів забезпечують струменеву навантажувальну здатність 20 мА на лінію порту (втікаючий струм) при максимальному значенні 40 мА, що дозволяє безпосередньо підключати до

мікроконтролера світлодіоди і біполярні транзистори. Архітектура побудови портів введення/виведення AVR із трьома бітами контролю/керування (замість двох, як це зроблено в більшості 8-розрядних мікроконтролерів) дозволяє розроблювачеві цілком контролювати процес введення/виведення, усуває необхідність мати копію вмісту порту в пам'яті для безпеки і підвищує швидкість роботи мікроконтролера при роботі з зовнішніми пристроями. Особливу значимість здобуває дана можливість AVR при реалізації систем, що працюють в умовах зовнішніх електричних завад;

– AVR-мікроконтролери містять послідовні інтерфейси: USART/UART; SPI; I^2C (TWI) та CAN, що також розширює можливості їх застосування для обміну з сучасною периферією, іншими мікроконтролерами та персональними комп'ютерами;

– до складу більшості AVR входить аналоговий компаратор. Він має окремий вектор переривання в загальній системі переривань мікроконтролера. Тип перепаду, що викликає запит на переривання при спрацьовуванні компаратора, може бути запрограмований як фронт, зріз або переключення. Важливою апаратною особливістю є те, що логічний вихід компаратора може бути програмним чином підключений до входу одного з 16-розрядних таймерів/лічильників, що працює в режимі захоплення. Це дає можливість вимірювати тривалості аналогових сигналів, а також реалізовувати АЦП двотактного інтегрування;

– аналого-цифровий перетворювач побудовано за схемою АЦП послідовного наближення з пристроєм вибірки-збереження. Число незалежних каналів перетворення визначається типом мікроконтролера, розрядність АЦП складає 10 біт. Час перетворення вибирається програмно за допомогою встановлення коефіцієнта дільника частоти, що входить до складу блоку АЦП. Важливою особливістю аналого-цифрового перетворювача є функція придушення шуму при перетворенні, коли на точність не впливають завади, що виникають при роботі процесорного ядра;

– мікроконтролери XMega мають декілька каналів цифро-аналогового перетворення, що значно розширює можливості їх застосування;

– AVR-мікроконтролери можуть бути переведені програмним шляхом в один із шести режимів зниженого енергоспоживання. Для різних сімейств AVR і різних мікроконтролерів у межах кожного сімейства змінюються кількість доступних режимів зниженого енергоспоживання;

– система команд AVR досить розвинута і нараховує до 118 (а в останніх розробках і до 141) різних інструкцій. Майже всі команди мають фіксовану довжину в одне слово (16 біт), що дозволяє в більшості випадків поєднувати в одній команді код операції і операнд(и). В останніх версіях кристалів "XMega" реалізована функція множення. По різноманітності і кількості інструкцій AVR більше схожі на CISC-, чим на RISC-процесори. Наприклад, у PIC-контролерів система команд нараховує до 75 різних інструкцій, а в MCS-51 вона складає 111.

AVR функціонують у широкому діапазоні напруг живлення: від 1,8 до 6,0 Вольт. Температурні діапазони роботи – комерційний і індустріальний.

Програмні й апаратні засоби підтримки для AVR завжди розроблялися і розробляються паралельно із самими кристалами і містять у собі компілятори, внутрішньосхемні емулятори, налагоджувачі, програматори і найпростіші налагоджувані плати – конструктори практично на будь-який смак. Активно йде процес співробітництва зі сторонніми фірмами, що випускають програмні засоби проектування і налагодження, операційні системи, різноманітні налагоджувані комплекси і внутрішньосхемні емулятори для AVR.

AVR-мікроконтролери використовуються у платах Arduino (Ардуіно), апаратної обчислювальної платформи для аматорського конструювання, основними компонентами якої є плата мікроконтролера з елементами введення/виведення та середовищем розробки Processing/Wiring на мові програмування, що є спрощеною підмножиною C/C++. Arduino може використовуватися як для створення автономних інтерактивних об’єктів, так і підключатися до програмного забезпечення, яке виконується на комп’ютері.

Все вище перераховане свідчить про сучасність AVR-мікроконтролерів і можливість їх широкого застосування при проектуванні мікроконтролерних систем різного призначення.

МЕТА ТА ОСНОВНІ ЗАВДАННЯ ДИСЦИПЛІНИ

Згідно робочої програми дисципліни «Проектування вбудованих систем» передбачається проведення зі студентами лабораторних занять, які проводяться у комп'ютерному класі кафедри:

- на персональних комп'ютерах із використанням спеціалізованих програм: емуляторів-налагоджувачів AVR Studio та PROTEUS.

Виконання лабораторних робіт дозволить більш поглиблено вивчити особливості архітектури типових мікроконтролерів, дослідити роботу мікропроцесорної системи (МПС) під час виконання окремих команд та програм, придбати навички по написанню та налагодженню програм на мовах Асемблер та С, навчитися вирішувати питання, пов'язані з обміном інформацією у МПС через паралельні та послідовні порти, програмуванням системи переривань, формуванням інтервалів часу та підрахунком подій у МПС і т. ін.

В результаті вивчення дисципліни ЗНАТИ:

- склад та основні характеристики типових 8-розрядних мікроконтролерів;
- склад, призначення окремих вузлів та роботу типового мікроконтролера за структурною схемою;
- програмну модель, формати команд та даних, способи адресації операндів та характеристику окремих команд типового мікроконтролера;
- особливості архітектури окремих функціональних модулів мікроконтролера: пам'яті; паралельних та послідовних інтерфейсів; таймерів/лічильників зовнішніх подій; переривань;
- організацію взаємодії мікроконтролера із типовими об'єктами керування.

ВМІТИ:

- програмувати окремі модулі мікропроцесорних систем мовами Асемблер та С;
- моделювати окремі частини мікропроцесорних систем на персональному комп’ютері;
- проектувати мікропроцесорні пристрої та системи на базі 8-розрядних мікроконтролерів.

1 ЗАГАЛЬНА ХАРАКТЕРИСТИКА AVR-МІКРОКОНТРОЛЕРА

1.1 Організація пам’яті

1.1.1 Загальна характеристика

В AVR-мікроконтролерах реалізовано Гарвардську архітектуру, у відповідності з якою розділено не лише адресні простори пам’яті програм та пам’яті даних, але також і шини доступу до них [1; 2; 10]. Способи адресації та доступу до цих областей пам’яті також є різними. Така структура дає змогу центральному процесору одночасно працювати як з пам’яттю програм так і з пам’яттю даних, що суттєво збільшує продуктивність. Кожна з областей пам’яті даних – статична пам’ять даних (СПД) та EEPROM-пам’ять також розташовані у своєму адресному просторі.

На рисунку 1.1 наведено карту пам’яті деяких AVR-мікроконтролерів сімейства Mega.

Оскільки AVR-мікроконтролери мають 16-бітну систему команд, об’єм пам’яті програм на рисунку вказано не у байтах, а в 16-бітових словах. Символ «$» є ознакою шістнадцяткової системи числення.

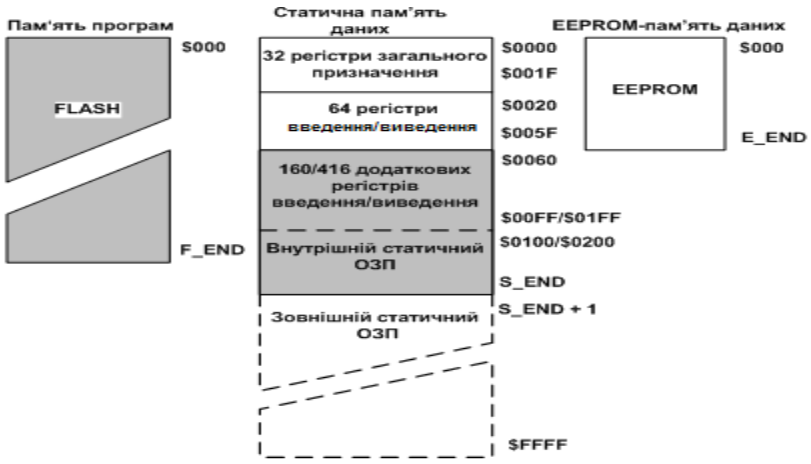
1.1.2 Організація пам’яті програм

В мікропроцесорних системах в пам’яті програм зберігаються команди, що керують роботою мікроконтролера, та константи, що не змінюються під час роботи програми.

Пам’ять програм має шістнадцятибітну організацію, для якої довжина кожної команди кратна одному слову – 16 біт. Наприклад, об’єм пам’яті програм деяких мікроконтролерів сімейства Mega складає від 2К (2*1024) до більш ніж 128К (128*1024) шістнадцятибітних слів (рисунок 1.1). У частини моделей мікроконтролерів сімейства Mega пам’ять програм логічно поділено на 2 рівні частини: область прикладної програми і область завантажувача [1; 2; 10].

В останній може розташовуватися спеціальна програма – завантажувач, що дозволяє мікроконтролеру самостійно керувати завантаженням та

вивантаженням прикладних програм під час роботи програми. Використання цієї області та реалізація програми-завантажувача розглянуто у [1; 2; 10]. Якщо самопрограмування мікроконтролера не використовується, прикладна програма може розташовуватися і в області завантажувача.



Модель	Пам'ять програм (FLASH)		Статична пам'ять даних (СПД)				Пам'ять даних (EEPROM)	
	Верхня границя [F_END]	Об'єм [слів]	Верхня Границя [S_END]	Об'єм СОЗП [байт]	Кількість дод. регістрів введ./вивед.	Зовнішній СОЗП	Верхня границя [F_END]	Об'єм [байт]
ATmega48x	$007FF	2 K	$02FF	512	160		$0FF	256
ATmega8515x	$00FFF	4 K	$025F	512	0	•	$1FF	512
ATmega8535x	$00FFF	4 K	$025F	512	0		$1FF	512
ATmega8x	$00FFF	4 K	$045F	1 K	0		$1FF	512
ATmega88x	$00FFF	4 K	$04FF	1 K	160		$1FF	512
ATmega16x	$01FFF	8 K	$045F	1 K	0		$1FF	512
ATmega162x	$01FFF	8 K	$04FF/$045F	1 K	160/0	•	$1FF	512
ATmega164x	$01FFF	8 K	$04FF	1 K	160		$1FF	512
ATmega165x	$01FFF	8 K	$04FF	1 K	160		$1FF	512
ATmega168x	$01FFF	8 K	$04FF	1 K	160		$1FF	512
ATmega32x	$03FFF	16 K	$085F	2 K	0		$3FF	1 K
ATmega324x	$03FFF	16 K	$08FF	2 K	160		$3FF	1 K
ATmega325x	$03FFF	16 K	$08FF	2 K	160		$3FF	1 K
ATmega3250x	$03FFF	16 K	$08FF	2 K	160		$3FF	1 K
ATmega64x	$07FFF	32 K	$10FF	4 K	160	•	$7FF	2 K
ATmega640x	$07FFF	32 K	$21FF	8 K	416	•	$FFF	4 K
ATmega644x	$07FFF	32 K	$10FF	4 K	160		$7FF	2 K
ATmega645x	$07FFF	32 K	$10FF	4 K	160		$7FF	2 K
ATmega6450x	$07FFF	32 K	$10FF	4 K	160		$7FF	2 K
ATmega128x	$0FFFF	64 K	$10FF	4 K	160	•	$FFF	4 K
ATmega1280x	$0FFFF	64 K	$21FF	8 K	416	•	$FFF	4 K
ATmega1281x	$0FFFF	64 K	$21FF	8 K	416	•	$FFF	4 K
ATmega2560x	$1FFFF	128 K	$21FF	8 K	416	•	$FFF	4 K
ATmega2561x	$1FFFF	128 K	$21FF	8 K	416	•	$FFF	4 K

Примітка: позначкою «•» відмічені моделі, до яких можна підключити зовнішній СОЗП.

Рисунок 1.1 – Карта пам’яті деяких мікроконтролерів сімейства Mega

Для адресації пам’яті програм використовується лічильник команд: PC (Program Counter). Розмір лічильника команд залежить від об’єму пам’яті, що адресується.

За адресою $0000 пам’яті програм знаходиться вектор скидання.

Після ініціалізації (скидання) мікроконтролера виконання програми починається з цієї адреси.

За цією адресою повинна розміщуватися команда переходу до основної частини ініціалізації програми. Починаючи з адреси $0001 пам'яті програм (моделі з пам'яттю програм 8 Кбайт і менше) або $0002 (останні моделі) розташовується таблиця векторів переривань. Розмір цієї області залежить від моделі мікроконтролера (розподіл областей векторів переривань розглянуто у [1; 2; 10]. При виникненні переривання після збереження у стеку поточного значення лічильника команд відбувається виконання команди, розташованої за адресою відповідного вектора. За цими адресами знаходяться команди переходу до підпрограм обробки переривань. В моделях з пам'яттю програм невеликого об'єму (8 Кбайт і менше) в таблицях векторів переривань використовуються команди відносного переходу – RJMP, а в останніх моделях – команди абсолютного переходу – JMP.

У більшості мікроконтролерів сімейства Mega положення вектора скидання і таблиці векторів переривань може бути змінено. Вони можуть розташовуватися не лише на початку пам'яті програм, як описано вище, але і на початку області завантажувача.

В якості пам'яті програм AVR-мікроконтролерів використовується FLASH-ПЗП, який розрахований, як мінімум, на 10000 циклів очищення/запису.

1.1.3 Організація пам'яті даних

Загальна характеристика

Пам'ять даних AVR-мікроконтролерів розділено на три частини: регістрова пам'ять, статичний оперативний запам'ятовуючий пристрій (СОЗП) – SRAM і енергонезалежна EEPROM-пам'ять.

Регістрова пам'ять включає 32 регістри загального призначення (РЗП), об'єднаних у файл, і 64 основних службових регістри введення/виведення (РВВ). В моделях з розвиненою периферією є також область додаткових (extended) регістрів введення/виведення (ДРВВ). Під перші РВВ в пам'яті даних мікроконтролера відводиться 64 байти, а під ДРВВ – 160 або 416 байт (залежно від моделі).

В обох областях регістрів введення/виведення розташовуються різні службові регістри: регістри керування мікроконтролером, регістри стану і т. ін., а також регістри керування периферійними пристроями, що входять до складу мікроконтролера. Загальна кількість РВВ і ДРВВ залежить від конкретної моделі мікроконтролера.

Для зберігання даних, окрім регістрів загального призначення, використовується також СОЗП. Ряд мікроконтролерів сімейства, крім того, мають можливість підключення зовнішнього статичного ОЗП об'ємом до 64 Кбайт.

Регістрову пам'ять разом із СОЗП називають статичною пам'яттю даних (СПД). Фізично таку пам'ять виконано на основі тригерів, тому вона є енергозалежною – зберігає інформацію, доки є живлення.

Для довготривалого зберігання різної інформації, яка може змінюватися в процесі функціонування готової системи (калібрувальні константи, серійні номери, ключі і т. ін.), в мікроконтролерах сімейства може використовуватися вбудована енергонезалежна EEPROM-пам'ять. Її об'єм для деяких моделей складає від 256 до більш ніж 4 Кбайт. EEPROM-пам'ять розташовано в окремому адресному просторі, а доступ до неї здійснюється за допомогою відповідних РВВ.

Статична пам'ять даних

В AVR-мікроконтролерах використовується лінійна організація статичної пам'яті даних (рисунок 1.2).

До складу статичної пам'яті даних входять регістри загального призначення (РЗП), регістри введення/виведення (РВВ), а також статичний ОЗП (СОЗП). Максимальний об'єм СПД складає 65536 байт та залежить від конкретної моделі сімейства.

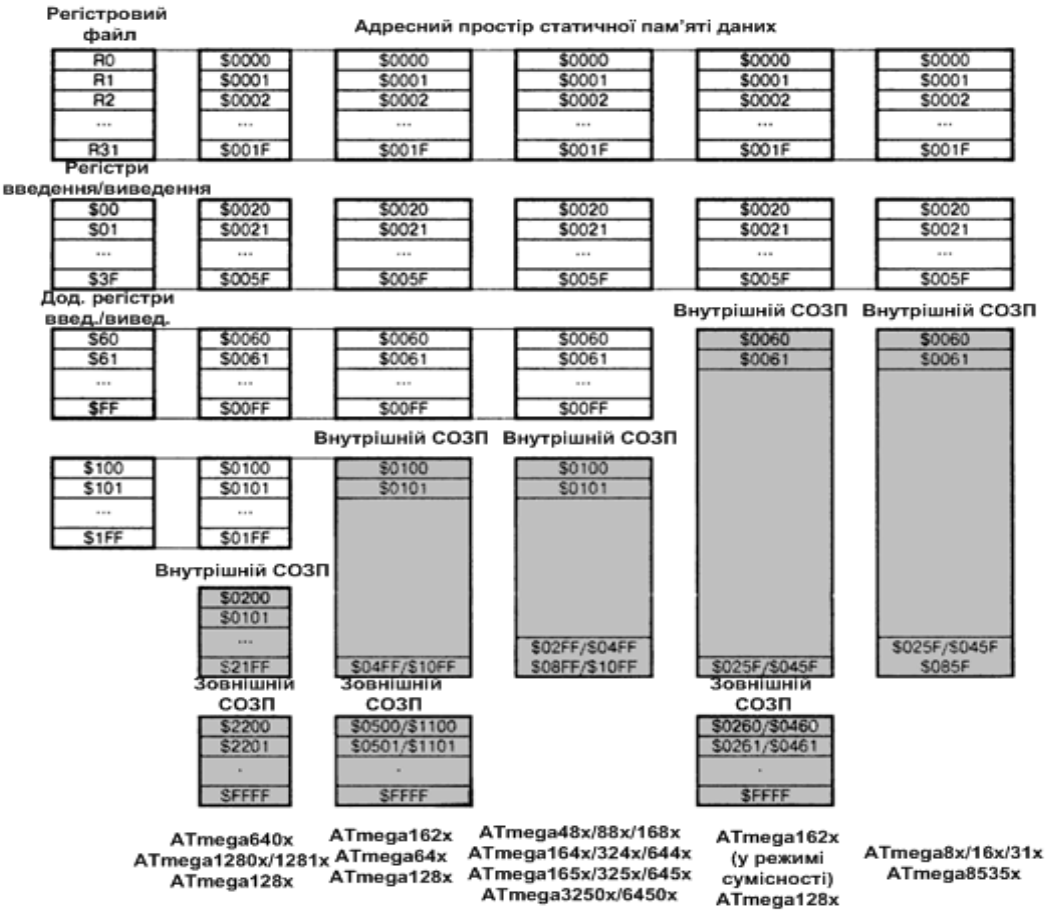


Рисунок 1.2 – Організація статичної пам’яті даних

Регістри загального призначення

Мікроконтролери мають 32 регістри загального призначення, які об’єднано в регістровий файл швидкого доступу (рисунок 1.3).

Всі РЗП безпосередньо доступні арифметико-логічному пристрою (АЛП), на відміну від восьмибітних мікроконтролерів деяких інших фірм, в яких є лише один такий регістр – робочий регістр A/W (акумулятор). Завдяки цьому будь-який РЗП може використовуватися практично у всіх командах і як операнд-джерело, і як операнд-приймач. Таке рішення (у поєднанні з конвеєрною обробкою) дозволяє АЛП виконувати за один такт одну операцію: читання операндів з регістрового файлу, виконання команди і запис результату назад у регістровий файл.

Останні 6 регістрів файлу (R26...R31) можуть об’єднуватися у три 16-бітних регістри: X, Y та Z (рисунок 1.4), що використовуються як покажчики при непрямій адресації статичної пам’яті даних.

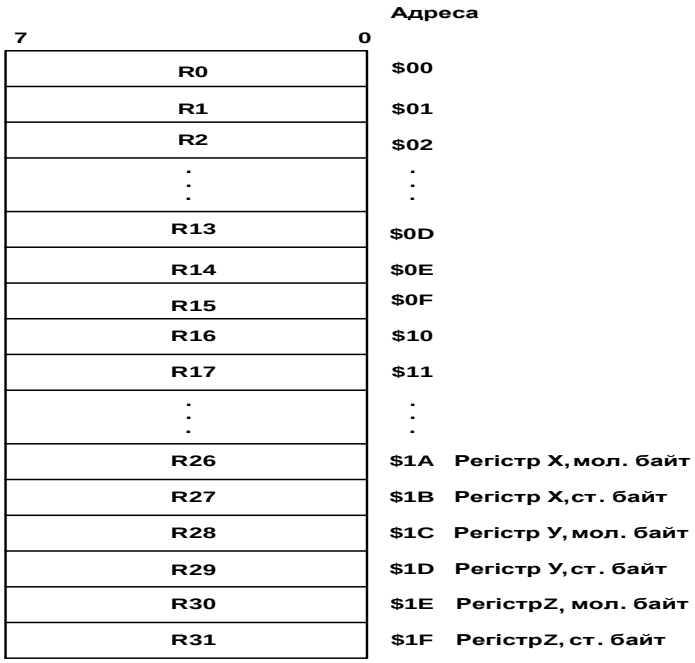


Рисунок 1.3 – Структура регістрів загального призначення

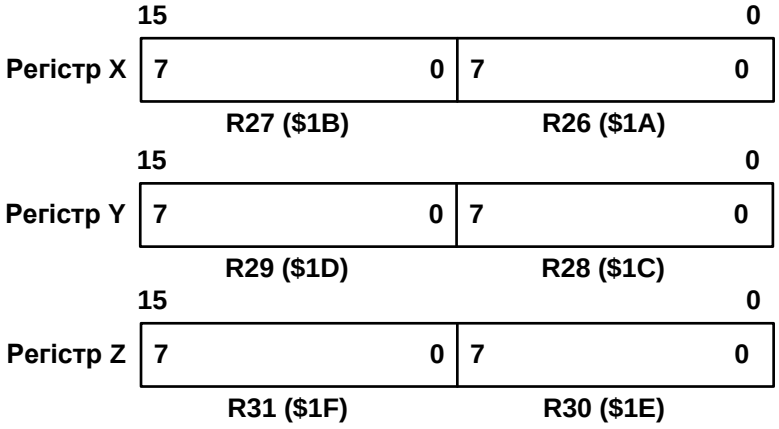


Рисунок 1.4 – Регістри-покажчики X,Y та Z

Як показано на рисунку 1.4, кожен регістр файлу має свою власну адресу в просторі статичної пам’яті даних. Тому до них можна звертатися двома способами – як до регістрів та як до статичної пам’яті, не дивлячись на те, що фізично ці регістри не є комірками СОЗП.

Таке рішення є ще однією відмінною особливістю архітектури AVR, що підвищує ефективність роботи мікроконтролера і його продуктивність.

Стек

В мікропроцесорних системах існують два різновиди стека: перший, який умовно можна назвати «апаратним» та другий, який будемо називати – «програмним». В «апаратному» стеку зберігаються адреси повернення із підпрограм, а «програмний» стек призначено для тимчасового зберігігання частин СПД, які використовуються підпрограмами. Для роботи з «програмним» стеком використовуються команди: PUSH та POP. AVR-мікроконтролери мають обидва різновиди стека (в залежності від моделі). У цьому разі стек розміщується у статичному ОЗП, а його глибина визначається тільки розміром вільної області цієї пам'яті.

В залежності від ємності СОЗП, як покажчик стека використовується або один регістр введення/виведення SPL (SP), розташований за адресою $3D ($5D), або пара регістрів SPH:SPL, розташованих за адресами $3E ($5E) і $3D ($5D) відповідно.

Регістри-покажчики стека є звичайними регістрами введення/виведення і, відповідно, програмно доступні. В наборі команд AVR-мікроконтролерів є команди занесення в стек – PUSH і зчитування зі стека – POP, що дозволяє програмі використовувати стек для своїх потреб. Оскільки після подачі напруги живлення або після скидання, покажчик стека дорівнює нулю, на початку програми його необхідно проініціалізувати, записавши в нього значення верхньої вільної адреси стекової пам'яті даних (зазвичай, це максимальна адреса СОЗП). Стек заповнюється у напряму зменшення значення регістра SP. Початкове значення SP адресує вершину стека – першу вільну комірку стекової пам'яті.

При виклику підпрограм адреса команди, розташованої за командою виклику, зберігається у стеку. Значення покажчика стека при цьому зменшується на 2, тому що для збереження лічильника команд потрібно 2 байти.

При поверненні з підпрограми ця адреса відновлюється зі стека і завантажується в лічильник команд. Значення покажчика стека відповідно збільшується на 2. Те ж відбувається і під час переривання. При виникненні переривання адреса наступної команди також зберігається у стеку, а при поверненні з підпрограми обробки переривання, вона відновлюється зі стека.

Регістри введення/виведення

Всі регістри введення/виведення умовно можна розділити на дві групи: службові регістри мікроконтролера і регістри, що відносяться до конкретних периферійних пристроїв (у тому числі регістри портів введення/виведення).

В AVR-мікроконтролерах частина регістрів введення/виведення розташовуються в так званому основному просторі введення/виведення, розміром 64 байти. У більшості моделей сімейства є також простір додаткових регістрів введення/виведення розміром 160 або 416 байт. Введення додаткових РВВ пов'язане з тим, що для підтримки всіх периферійних пристроїв, наявних в цих моделях, адрес перших 64-х РВВ недостатньо.

Кількість та розподіл адрес простору введення/виведення (як основного, так і додаткового) залежить від конкретної моделі мікроконтролера або, якщо точніше, від складу і можливостей периферійних пристроїв даної моделі [1; 2; 10].

При вказівці адрес деяких РВВ відповідні їм адреси комірок СПД вказуються в дужках. Відповідно, якщо адреса регістра вказується лише в дужках, цей регістр розташовано в просторі додаткових РВВ. Якщо адреса в таблиці РВВ не вказана, це значить, що для даної моделі він зарезервований, і запис за цією адресою заборонено (для сумісності з майбутніми моделями). Якщо адреса, наприклад регістра SREG вказана як $3F ($5F), то це означає що за адресою $3F можна відповідними командами звернутися до РВВ основного простору регістрів введення/виведення, розміром 64 байти, а за адресою $5F можна звернутися до РВВ, як частині простору СПД. Різниця між цими адресами складає $20, тому що у просторі СПД перед РВВ знаходяться 32 РЗП.

Приклади регістрів введення/виведення деяких моделей Mega-AVR-мікроконтролерів наведено у [1; 2; 10].

До регістрів введення/виведення, розташованих в основному просторі введення/виведення, можна безпосередньо звернутися за допомогою команд IN і OUT, що виконують пересилання даних між одним з 32-х РЗП і регістром основного простору введення/виведення. В системі команд є також чотири команди побітового доступу, що використовують як операнди частину регістрів введення/виведення: команди встановлення/скидання окремого біта – SBI і CBI і команди перевірки стану окремого біта – SBIS і SBIC. Ці команди можуть звертатися лише до першої половини основного простору введення/виведення – адреси $00...$1F.

Окрім адресації до регістрів основного простору введення/виведення за допомогою команд IN і OUT, до РВВ можна звертатися як до комірок СПД за допомогою команд ST/SD/SDD і LD/LDS/LDD (для додаткових РВВ тільки цей спосіб є можливим).

Регістр стану SREG

Серед РВВ є один регістр, що часто використовується в процесі виконання програм – регістр стану SREG. Він входить до складу регістрів основного простору введення/виведення та розташовується за адресою $3F ($5F) і містить набір прапорців, що показують поточний стан мікроконтролера (програми). Більшість прапорців при настанні певних подій відповідно до результату виконання команд автоматично встановлюються в одиницю або скидаються в нуль. Всі біти цього регістра доступні як для читання, так і для запису. Після скидання мікроконтролера вони скидаються в нуль. Формат регістра SREG показано на рисунку 1.5, а опис його його окремих розрядів – в таблиці 1.1.

В останніх моделях мікроконтролерів сімейства, з’явилися 3 регістри введення/виведення загального призначення – GPIOR0, GPIOR1 і GPIOR2. У цих регістрах можна зберігати будь-яку інформацію, проте основне їх призначення – збереження глобальних змінних.

	7	6	5	4	3	2	1	0
	I	T	H	S	V	N	Z	C
Читання(R)\Запис(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Початкове значення	0	0	0	0	0	0	0	0

Рисунок 1.5 – Формат регістра стану SREG

Таблиця 1.1 – Регістр стану SREG

Розряд	Назва	Опис
7	I	**Загальний дозвіл переривань.** Щоб дозволити переривання цей прапорець необхідно встановити в одиницю. Дозволити/заборонити окремі переривання можна встановленням або скиданням відповідних розрядів регістрів масок переривань. Якщо прапорець скинуто в нуль, то переривання заборонено незалежно від стану цих розрядів. Прапорець скидається апаратно після входу у підпрограму обробки переривання і відновлюється командою RETI, дозволяючи обробку наступних переривань
6	T	**Зберігання копійованого біта.** Цей розряд регістра використовується як джерело або приймач командами копіювання біта BLD (Bit Load) та BST (Bit STore). Вказаний розряд будь-якого РЗП можна скопіювати у цей розряд командою BST або змінити відповідно до вмісту даного розряду командою BLD
5	H	**Прапорець половинного перенесення**. Цей прапорець встановлюється в одиницю при виконанні деяких арифметичних операцій, якщо було або перенесення з молодшої тетради байта (з 3-го розряду в 4-й), або позика зі старшої тетради
4	S	**Прапорець знака.** Цей прапорець дорівнює результату операції «виключне АБО» – XOR між прапорцями N (від'ємний результат) і V (переповнення числа у додатковому коді). Відповідно цей прапорець встановлюється в одиницю, якщо результат виконання арифметичної операції менший від нуля. Прапорець дозволяє збільшити розрядність результату арифметичних операцій у додатковому коді з семи значущих біт до восьми із прапорцем S у якості дев'ятого знакового біта
3	V	**Прапорець переповнення додаткового коду.** Цей прапорець встановлюється в одиницю при переповненні розрядної сітки знакового результату. Використовується при роботі зі знаковими числами, які подано у додатковому коді
2	N	**Прапорець від'ємного значення**. Цей прапорець встановлюється в одиницю, якщо старший – 7-й розряд результату операції дорівнює одиниці. Інакше прапорець дорівнює нулю
1	Z	**Прапорець нуля.** Цей прапорець встановлюється в одиницю, якщо результат виконання операції дорівнює нулю
0	C	**Прапорець перенесення.** Цей прапорець встановлюється в одиницю, якщо в результаті виконання операції відбувся вихід за межі байта

Регістри GPIOR0, GPIOR1 і GPIOR2 розташовані в молодшій половині основного простору введення/виведення і відповідно можуть використовуватися в командах побітового доступу SBI/CBI і SBIS/SBIC. Інші РВВ описано у відповідних розділах посібника.

Енергонезалежна пам’ять даних EEPROM

Частина мікроконтролерів сімейства Tiny та всі мікроконтролери сімейств Mega та XMega мають у своєму складі енергонезалежну пам’ять даних (EEPROM-пам’ять). Цю пам’ять розташовано у власному адресному просторі, а її об’єм становить від шестидесяти байт (для сімейства Tiny) та від 256 байт до більш ніж 4 Кбайт для сімейств Mega та XMega.

Для запису та читання EEPROM-пам’яті в готовому пристрої використовуються три регістри введення/виведення: регістр адреси – EEAR, регістр даних – EEDR та регістр керування – EECR. Регістр адреси EEAR фізично розміщується у двох PBB: EEARH:EEARL. Адреси, формати та опис окремих розрядів інших регістрів керування EEPROM-пам’яті наведено у [1; 2 10].

Процедура запису одного байта в EEPROM-пам’ять складається з наступних етапів:

1. Визначити готовность пам’яті до запису. Для цього треба дочекатися, коли скинеться прапорець EEPE (EEWE) регістра EECR.
2. Визначити завершення запису у FLASH-пам’ять програм, якщо він відбувається. Для цього треба дочекатися коли скинеться прапорець SPMEN регістра SPMCR.
3. Завантажити байт даних у регістр EEDR, а необхідну адресу – у регістр EEAR.
4. Встановити в одиницю прапорець EEMPE (EEMWE) регістра EECR.
5. Протягом 4-х машинних циклів після цього записати в розряд EEPE (EEWE) регістра EECR одиницю. Після встановлення цього розряду процесор пропускає 2 такти перед виконанням наступної інструкції.

Другий пункт введено через те, що запис в EEPROM-пам’ять не може виконуватися одночасно із записом у FLASH-пам’ять. Тому перед виконанням запису в EEPROM-пам’ять варто переконатися, що програмування FLASH-пам’яті завершено.

Якщо в програмі відсутній завантажувач, тобто мікроконтролер ніколи не змінює вміст пам’яті програм, другий крок може бути пропущений.

На процес звернення до EEPROM-пам’яті впливає внутрішній RC-генератор, що калібрується. Відповідно, тривалість циклу запису залежить від частоти цього генератора, напруги живлення та температури [1; 10]. За закінченням циклу запису розряд EEPE (EEWE) апаратно скидається, після чого програма може почати запис наступного байта.

При запису в EEPROM можуть виникнути деякі проблеми, викликані перериваннями:

1. При виникненні переривання між 4-м та 5-м етапами описаної вище послідовності, запис в EEPROM буде перервано, тому що за час обробки переривання прапорець EEMPE (EEMWE) скинеться в нуль.

2. Якщо в підпрограмі обробки переривання, що виникло під час запису в EEPROM-пам’ять, також відбувається звернення до неї, то буде змінено вміст регістрів адреси та даних EEPROM. В результаті перший перерваний запис буде зірвано.

Для запобігання описаних проблем рекомендується забороняти всі переривання (скидати біт I регістра SREG) на час виконання пунктів 2...5 описаної вище послідовності.

Нижче наведено два приклади реалізації функції запису в EEPROM-пам’ять (керування перериваннями не виконується).

Приклад на асемблері

```
EEPROM_write:
sbic EECR, EEWE; очікування, коли скинеться прапорець EEWE
rjmp  EEPROM_write; відносний безумовний перехід на мітку
; EEPROM_write
out    EEARH, r18; запис старшого байта адреси з регістра r18 в EEARH
out    EEARL, r17; запис молодшого байта адреси з регістра r17 в EEARL
out    EEDR, r16; запис даних з регістра r16 у регістр даних EEDR
```

```
sbi     EECR, EEMWE; встановлення прапорця EEMWE регістра EECR
sbi     EECR, EEWE; встановлення прапорця EEWE регістра EECR
ret; вихід з підпрограми
```

Приклад на С

```
void EEPROM_write (unsigned int uiAddress, unsigned char ucData)
{
While (EECR & (1<<EEWE)); /* Очікування завершення попереднього
                                    запису*/
        EEAR = uiAddress;               /*Ініціалізація регістрів*/
        EEDR = ucData;
        EECR | = (1<<EEMWE);        /*Встановлення прапорця EEMWE*/
        EECR | = (1<<EEWE);   /*Початок запису в EEPROM*/
}
```

Перед виконанням операції зчитування також необхідно проконтролювати стан прапорця EEPE (EEWE), тому що поки виконується операція запису в EEPROM-пам’ять (прапорець EEWE встановлено), не можна виконувати ні зчитування EEPROM-пам’яті, ні зміни регістра адреси.

Після завантаження необхідної адреси в регістр EEAR необхідно встановити в одиницю розряд EERE регістра EECR. Коли зчитані дані буде поміщено в регістр даних EEDR, відбудеться апаратне скидання цього розряду. Однак стежити за станом розряду EERE для визначення моменту завершення операції зчитування не потрібно, тому що операція зчитування з EEPROM завжди виконується за один такт. Крім того, після встановлення розряду EERE в одиницю процесор пропускає 4 такти перед початком виконання наступної інструкції.

Нижче наведено два приклади реалізації функції зчитування з EEPROM-пам’яті (керування перериваннями не виконується).

Приклад на асемблері

```
EEPROM_read:
sbic   EECR, EEWE; очікування, коли скинеться прапорець EEWE
rjmp   EEPROM_read; відносний безумовний перехід на мітку
       ;EEPROM_read
out    EEARH, r18; запис старшого байта адреси з регістра r18 в EEARH
out    EEARL, r17; запис молодшого байта адреси з регістра r17 в EEARL
sbi    EECR, EERE; встановлення прапорця EERE регістра EECR
in     r16, EEDR; пересилання даних з регістра EEDR в регістр r16
ret; вихід з підпрограми
```

Приклад на С

```
void EEPROM_write (unsigned int uiAddress)
{
while(EECR & (1<<EEWE));  /* Очікування завершення попереднього*/
                          /*запису */
      EEAR = uiAddress;           /* Ініціалізація регістра адреси */
      EECR | = (1<<EERE);   /*Виконання зчитування */
Return EEDR;
}
```

При використанні EEPROM-пам’яті необхідно дотримуватися деяких запобіжних заходів, щоб уникнути ушкодження даних, що в ній зберігаються. При зниженні напруги живлення нижче деякої величини дані, що зберігаються в пам’яті, можуть бути ушкоджені. Якщо напруга живлення буде нижче норми, то сам мікроконтролер може виконувати команди некоректно.

Щоб уникнути ушкодження даних, що зберігаються в EEPROM-пам’яті, треба скористатися однієї із трьох наступних рекомендацій:

1. Утримувати мікроконтролер у стані скидання увесь час, поки напруга живлення перебуває нижче норми. Для цього варто використовувати

вбудований детектор зниженої напруги живлення – Brown-out Detector (BOD).

2. Утримувати мікроконтролер у «сплячому» режимі – Power Down поки напруга живлення перебуває нижче норми. Оскільки в цьому режимі мікроконтролер не може виконувати ніяких команд, таке рішення ефективно захищає службові регістри EEPROM від ненавмисного запису.

3. Зберігати константи у FLASH-пам’яті програм, якщо вони не повинні мінятися під час роботи програми. Мікроконтролер не може самостійно робити запис у FLASH-пам’ять, тому при зниженні напруги живлення її вміст не буде ушкоджено.

1.2 Програмна модель AVR-мікроконтролера

1.2.1 Загальні відомості

Однією з основних характеристик архітектури будь-якого мікроконтролера є програмна модель (модель програміста), яка включає частину структури мікроконтролера, що доступна програмісту за допомогою системи команд. На рисунку 1.6 в якості приклада наведено програмну модель одного з AVR-мікроконтролерів. Ця модель має регістри загального призначення (РЗП), регістри введення/виведення (РВВ), статичний оперативний запам’ятовуючий пристрій (СОЗП), який в технічній літературі англійською мовою називають SRAM, ємністю 128 байт, EEPROM-пам’ять даних, ємністю 512 Кбайт та постійний запам’ятовуючий пристрій FLASH-типу, ємністю 2К слів (4 Кбайт). РЗП, РВВ та СОЗП називають статичною пам’яттю даних, тому що вони є енергозалежними. Відповідно EEPROM-пам’ять даних є енергонезалежною.

1.2.2 Регістри загального призначення

32 регістри загального призначення об’єднано у файл, структуру якого показано на рисунку 1.7. В AVR-мікроконтролерах усі РЗП безпосередньо доступні АЛП на відміну від мікроконтролерів деяких фірм, в яких є тільки один такий регістр, наприклад, робочий регістр A/W – акумулятор.

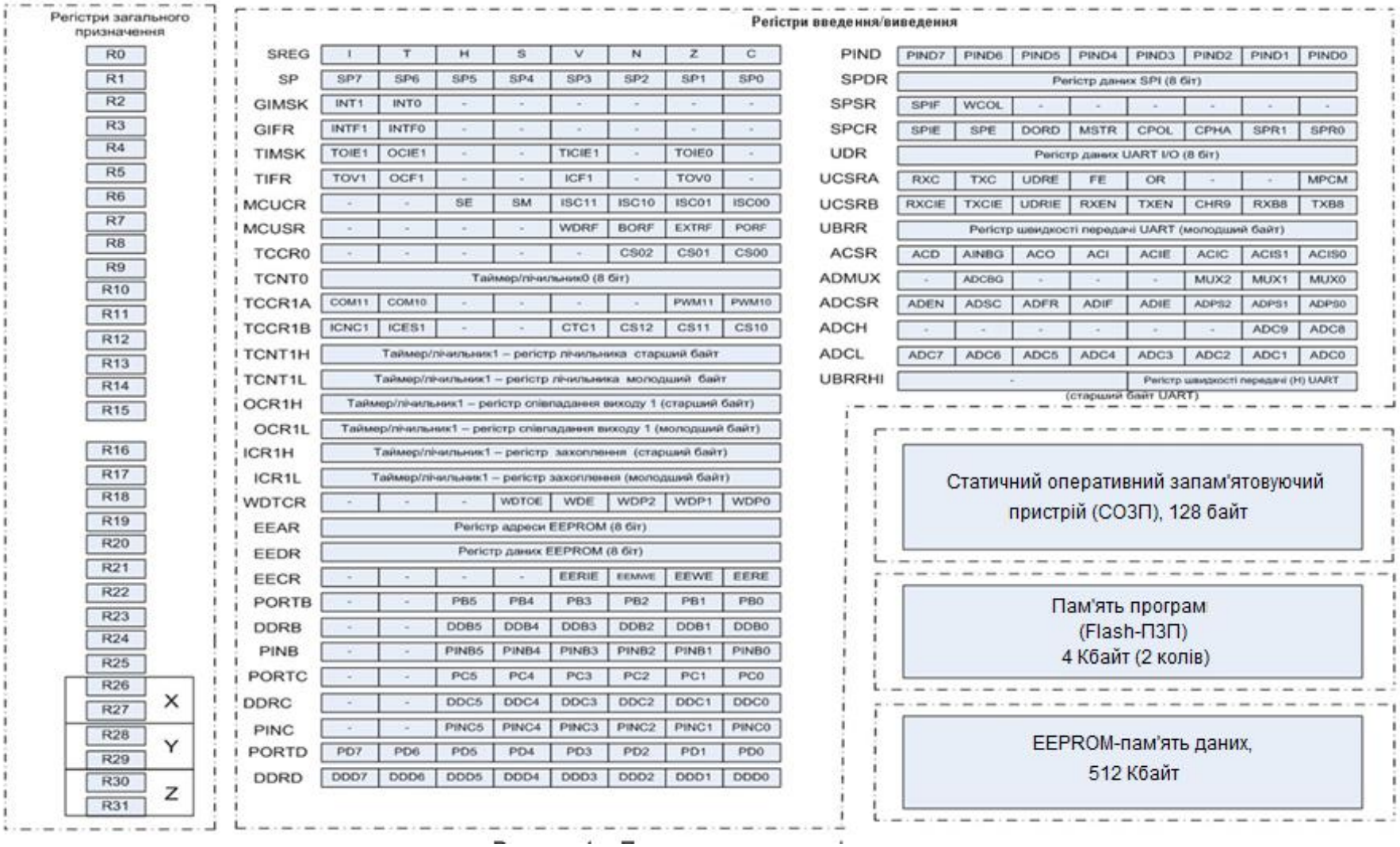


Рисунок 1.6 – Програмна модель мікроконтролера

7р 0р		7р 0р		Адреса
R0		R0		$00
R1		R1		$01
R2		R2		$02
...		...		
R13		R13		$0D
R14		R14		$0E
R15		R15		$0F
R16		R16		$10
R17		R17		$11
...		...		
R26		R26		$1A Регістр X, мол. байт
R27		R27		$1B Регістр X, ст. байт
R28		R28		$1C Регістр Y, мол. байт
R29		R29		$1D Регістр Y, ст. байт
R30 (Регістр Z)		R30		$1E Регістр Z, мол. байт
R31		R31		$1F Регістр Z, ст. байт
AT90S1200		Інші моделі		

Рисунок 1.7 – Структура файлу регістрів загального призначення

Завдяки цьому будь-який РЗП може використовуватись командами як операнд-джерело, або як операнд-приймач.

Винятком є лише п’ять арифметичних і логічних команд, що виконують дії між константою і регістром – SBCI, SUBI, CPI, ANDI, ORI, а також команда завантаження константи в регістр – LDI. Ці команди можуть звертатися тільки до другої половини РЗП – R16...R31. Декілька регістрів загального призначення використовуються як покажчики при непрямій адресації статичної пам’яті даних.

Як показано на рисунку 1.7, кожен регістр файлу, має свою власну адресу у просторі статичної пам’яті даних (СПД). Тому до них можна звертатися також як до комірок СПД.

На рисунку 1.8 наведено регістри-покажчики X,Y і Z, які використовуються при непрямій адресації операндів.

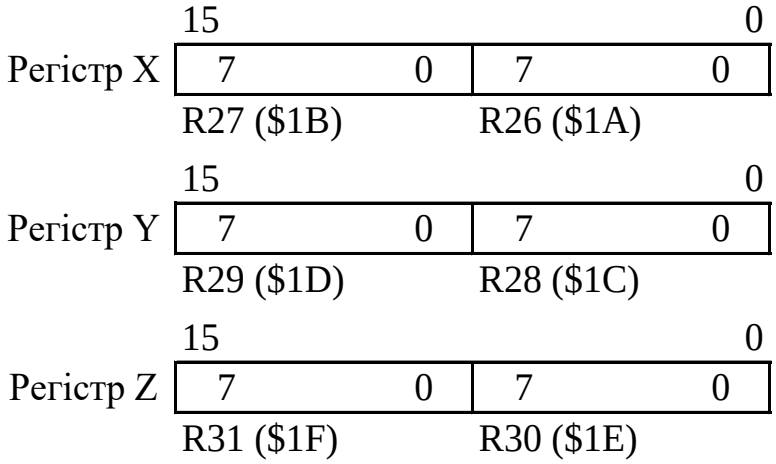


Рисунок 1.8 – Регістри-покажчики X,Y і Z

1.2.3 Регістри введення/виведення

Регістри введення/виведення (РВВ) розташовуються в основному просторі введення/виведення розміром 64 байти, або у додатковому просторі РВВ. Усі РВВ можна розділити на дві групи: службові регістри мікроконтролера і регістри, що відносяться до периферійних пристроїв, у тому числі порти введення/виведення. Розмір кожного регістра – 8 розрядів.

Розподіл адрес простору введення/виведення залежить від конкретної моделі мікроконтролера, тому що різні моделі мають різний склад периферійних пристроїв і, відповідно, різну кількість регістрів [1; 10]. У таблиці 1.2, як приклад, наведено розміщення в адресному просторі введення/виведення деяких РВВ, спільних для більшості мікроконтролерів сімейства. Інші регістри введення/виведення розглянуто у [1…2; 10]. Якщо адресу в таблиці не зазначено, це означає, що її зарезервовано, і запис за цією адресою не рекомендується.

Таблиця 1.2 – Приклад регістрів введення/виведення

Назва	**Функція**	**Адреса**
ACSR	Регістр керування і стану аналогового компаратора	$08 ($28)*
ADCH	Регістр даних АЦП (старший байт)	$05 ($25)
ADCL	Регістр даних АЦП (молодший байт)	$04 ($24)
ADCSR	Регістр керування і стану АЦП	$06 ($26)
ADMUX	Регістр керування мультиплексором АЦП	$07 ($27)

Закінчення таблиці 1.2

Назва	Функція	Адреса
DDRB	Регістр напряму даних порту B	$17 ($37)
DDRC	Регістр напряму даних порту C	$14 ($34)
DDRD	Регістр напряму даних порту D	$11 ($31)
EEAR	Регістр адреси EEPROM	$1E($3E)
EECR	Регістр керування EEPROM	$1C($3C)
EEDR	Регістр даних EEPROM	$10 ($30)
GIFR	Загальний регістр прапорців переривань	$3A ($5A)
GIMSK	Загальний регістр маски переривань	$3B ($5B)
ICR1H	Регістр захоплення таймера/лічильника 1 (старший байт)	$27 ($47)
ICR1L	Регістр захоплення таймера/лічильника 1 (молодший байт)	$26 ($46)
MCUCR	Загальний регістр керування мікроконтролером	$35 ($55)
OCR1H	Регістр збігу таймера/лічильника 1 (старший байт)	$2B ($4B)
OCR1L	Регістр збігу таймера/лічильника 1 (молодший байт)	$2A ($4A)
PINB	Виводи порту B	$16 ($36)
PINC	Виводи порту C	$13 ($33)
FIND	Виводи порту D	$10 ($30)
PORTC	Регістр даних порту C	$15 ($35)
PORTD	Регістр даних порту D	$12 ($32)
SP	Покажчик стека	$3D ($5D)
SPCR	Регістр керування SPI	$00 ($20)
SPDR	Регістр даних SPI	$0F ($2F)
SPSR	Регістр стану SPI	$0E ($2E)
SREG	Регістр стану мікроконтролера	$3F ($5F)
TCCR0	Регістр керування таймером/лічильником 0	$33 ($53)
TCCR1A	Регістр керування A таймером/лічильником 1	$2F ($4F)
TCCR1B	Регістр керування B таймером/лічильником 1	$2E($4E)
TCNT0	Лічильний регістр таймера/лічильника 0 (8-розрядний)	$32($52)
TCNT1H	Лічильний регістр таймера/лічильника 1 (старший байт)	$2D ($4D)
TCNT1L	Лічильний регістр таймера/лічильника 1 (молодший байт)	$2C ($4C)
TIFR	Регістр прапорців переривань від таймерів/лічильників	$38 ($58)
TIMSK	Регістр маски переривань від таймерів/лічильників	$39($59)
UBRR	Регістр швидкості передачі UART (молодший байт)	$09 ($29)
UBRRHI	Регістр швидкості передачі UART (старший байт)	$03 ($23)
UCR	Регістр керування UART	$0A ($2A)
UDR	Регістр даних UART	$0C ($2C)
USR	Регістр стану UART	$0B ($2B)
WDTCR	Регістр керування вартовим таймером	$21 ($41)

* У дужках указано адреси РВВ, якщо вони адресуються як комірки статичної пам’яті даних.

До будь-якого регістра введення/виведення можна звернутися за допомогою команд IN і OUT, що виконують пересилання даних між одним з тридцятидвох РЗП та основного простору РВВ. В чотирьох командах порозрядного доступу, операндами є регістри введення/виведення: команди встановлення/скидання окремого розряду РВВ – SBI та CBI і команди перевірки стану окремого розряду – SBIS і SBIC. Останні чотири команди можуть звертатися тільки до першої половини основного простору регістрів введення/виведення – адреси $00...$1F.

До регістрів введення/виведення можна звертатися двома способами:

- командами IN і OUT;
- командами роботи з СПД.

У першому випадку використовуються адреси РВВ, що належать основному простору введення/виведення – $00...$3F. У другому випадку адресу простору введення/виведення РВВ необхідно збільшити на $20. Далі при наведенні адрес РВВ після адреси простору введення/виведення в дужках вказуються відповідні їм адреси комірок СПД, яка включає: РЗП, РВВ та СОЗП.

Нижче розглянуто деякі службові регістри мікроконтролера, адреси яких, як правило, не змінюються від моделі до моделі. Наприклад, регістр SREG завжди розташовано за адресою $3F ($5F), регістр GIMSK – за адресою $3B ($5B) і т. ін.

Регістр стану SREG

Регістр стану SREG являє собою набір прапорців, які показують поточний стан програми, яку виконує мікроконтролер. Ці прапорці автоматично встановлюються в одиницю, або скидаються в нуль відповідно до результату виконання команд, які впливають на стан прапорців. Усі розряди цього регістра доступні як для читання, так і для запису в будь-який момент часу. Після скидання мікроконтролера всі розряди регістра скидаються в нуль. Вміст цього регістра та його опис наведено вище у 1.1.3.

Регістр-покажчик стека SP

У мікроконтролерах, які мають ємність СОЗП до 256 байт, покажчик стека реалізовано на одному регістрі SPL (SP), який розташовано за адресою $3D ($5D).

В іншому випадку покажчик стека виконано на парі регістрів SPH:SPL, які відповідно мають адреси: $3E ($5E) і $3D ($5D). Усі розряди цих регістрів доступні як для читання, так і для запису в будь-який момент часу. Після скидання мікроконтролера вміст регістрів дорівнює нулю. Тому на початку програми покажчик стека треба завантажити відповідним значенням – як правило, це найбільша адреса СПД для конкретного мікроконтролера.

Регістр стану мікроконтролера MCUSR

Регістр стану мікроконтролера містить прапорці, стан яких дозволяє визначити причину, через яку відбулося скидання мікроконтролера [1; 2; 10]. Регістр розташовано за адресою $34 ($54).

1.2.4 Функціонування конвеєра

Для підвищення швидкодії AVR-мікроконтролерів при виконанні програми використовується дворівневий конвеєр (рисунок 1.9).

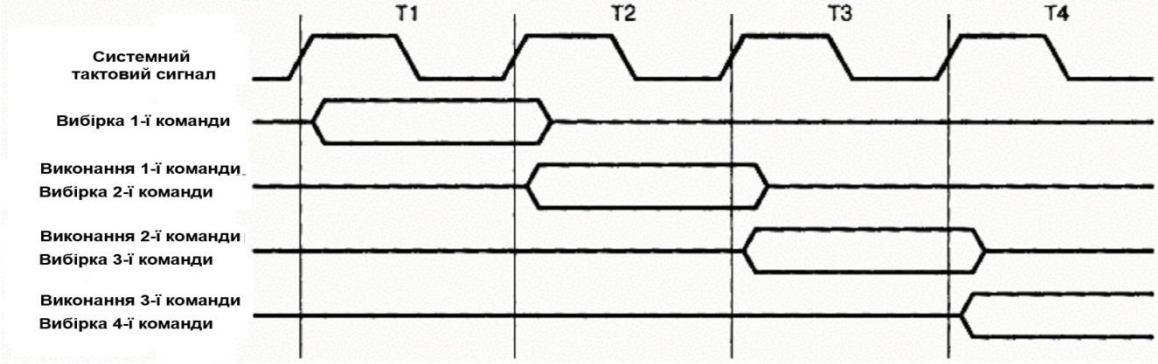


Рисунок 1.9 – Послідовність виконання команд у конвеєрі

Під час першого машинного циклу відбувається вибірка команди з пам’яті програм та її декодування.

У наступному циклі ця команда виконується і паралельно відбувається вибірка і декодування другої команди і так далі. Фактичний час виконання більшості команд дорівнює одному машинному циклу, що дозволяє досягати продуктивності до 1 MIPS (Million Instructions Per Second) на один мегагерц тактової частоти.

Завдяки підключенню АЛП безпосередньо до регістрового файлу (РЗП) він виконує одну команду – читання вмісту двох регістрів, виконання операції і запис результату в регістр-приймач за один такт (рисунок 1.10).

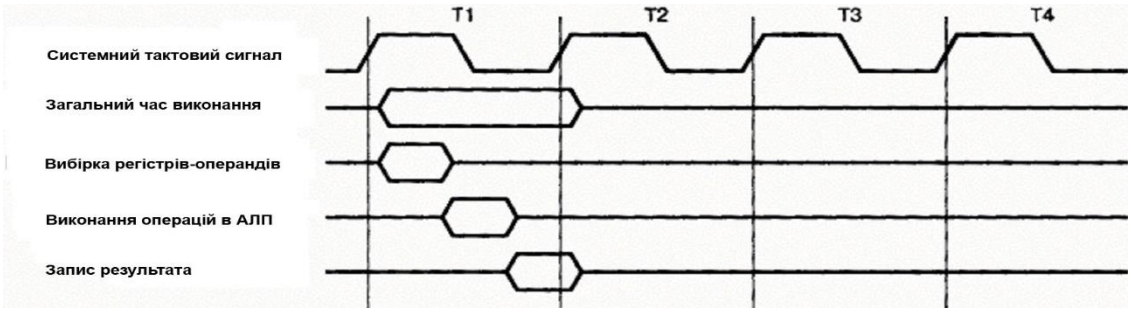


Рисунок 1.10 – Функціонування АЛП

Звернення до внутрішнього СОЗП виконується за два такти [1; 2; 10].

Вище було описано послідовність виконання команд програми в ідеальному випадку. Однак в дійсності дуже часто відбувається порушення нормального порядку функціонування конвеєра. Прикладом команд, що викликають таке порушення, є команди умовного переходу, а також команди типу «перевірка/пропуск» – пропускають наступну команду, якщо результат перевірки позитивний. Якщо умова, що перевіряється командою умовного переходу, істинна, виконання програми повинно продовжуватись з деякої іншої адреси. Оскільки в конвеєрі уже відбулася вибірка команди, розташованої після команди переходу, час виконання команди переходу збільшується на один машинний цикл, під час якого відбувається вибірка команди, розташованої за адресою переходу.

При виконанні команд типу «перевірка/пропуск», наступна команда не виконується у разі істинності умови, яка перевіряється. Однак вибірка команди, що пропускається, уже відбулася. Внаслідок того, що команда не виконується, у конвеєрі утвориться «дірка», що полягає в пропуску одного або двох (в залежності від команди, що пропускається) машинних циклів. Відповідно, команди типу «перевірка/пропуск» виконуються за один машинний цикл, якщо результат перевірки умови негативний, і за два або три цикли, якщо він позитивний.

Команди безумовного переходу – JMP, RJMP і IJMP, команди виклику підпрограм – CALL, RCALL і ICALL і команди повернення з підпрограм – RET і RETI також змінюють вміст лічильника команд – PC, що викликає перехід у пам’яті програм. У результаті виконання цих команд відбувається «розрив» у роботі конвеєра, у наслідок чого виникає затримка виконання програми на декілька машинних циклів. В залежності від команди тривалість затримки становить від двох до чотирьох машинних циклів.

З тієї ж причини порушення нормального функціонування конвеєра відбувається і при виникненні переривання. Максимальна затримка при цьому становить 4 машинних цикли.

1.2.5 Лічильник команд

Одним з важливих регістрів мікроконтролера є лічильник команд PC – program counter. Цей регістр при виконанні програми використовується для адресації комірок пам’яті програм.

Після увімкнення живлення, а також після скидання мікроконтролера в лічильник програм автоматично завантажується значення $000. За цією адресою розташовується команда відносного переходу до частини програми, яка виконує ініціалізацію, – RJMP.

Розмір лічильника команд залежить від ємності адресованої пам’яті. При нормальному виконанні програми вміст лічильника команд автоматично збільшується на 1, або на 2 в залежності від довжини виконуваної команди. Цей порядок порушується при виконанні команд переходу, виклику і повернення з підпрограм, а також при виникненні переривань. При виникненні переривання в лічильник команд завантажується адреса відповідного вектора переривання – $001...$010. Якщо переривання в програмі використовуються, за цими адресами повинні розміщатися команди відносного переходу до підпрограм обробки переривань.

Доступ із програми безпосередньо до лічильника команд не можливий.

1.3 Способи адресації операндів

Загальні відомості

AVR-мікроконтролери підтримують наведені нижче способи адресації операндів – даних, що беруть участь в операціях:

- неявна;
- безпосередня;
- пряма;
- непряма адресації.

Деякі способи адресації мають кілька різновидів в залежності від того, до якої області пам’яті виконується звернення – при прямій адресації, або які додаткові дії виконуються над індексним регістром – при непрямій адресації.

Неявна адресація

При неявній адресації в команді відсутня адреса операнда в явному вигляді. Інформацію про потрібну адресу операнда мікроконтролер отримує з коду операції команди, тобто кажуть, що операнд адресується неявно. Так, наприклад, можуть адресуватися окремі прапорці регістра стану в командах типу BRTS k; BRVS k; BRPL k, пари регістрів X/Y/Z при непрямій адресації, наприклад, команда LD R7, -Y і т. ін.

Безпосередня адресація

При безпосередній адресації операнд входить в саму команду та читається із пам’яті програм у складі машинного коду команди. Це, наприклад, команди ADIW Rd, K6 (K6 = 6 біт); SBCI Rd*, K (K = 8біт) і т. ін.

Пряма адресація

При прямій адресації адреси операндів містяться у машинному коді команди. Ця адресація використовується для звернення до комірок СПД. Існують наступні різновиди прямої адресації: пряма адресація одного РЗП, пряма адресація двох РЗП, пряма адресація РВВ, пряма адресація всієї СПД – РЗП, РВВ та СОЗП.

Пряма адресація одного регістра загального призначення

Цей спосіб адресації використовується в командах, що оперують з одним з регістрів загального призначення. При цьому адреса регістра-операнда (його номер) міститься в розрядах 8...4 (5 біт) машинного коду команди (рисунок 1.11).

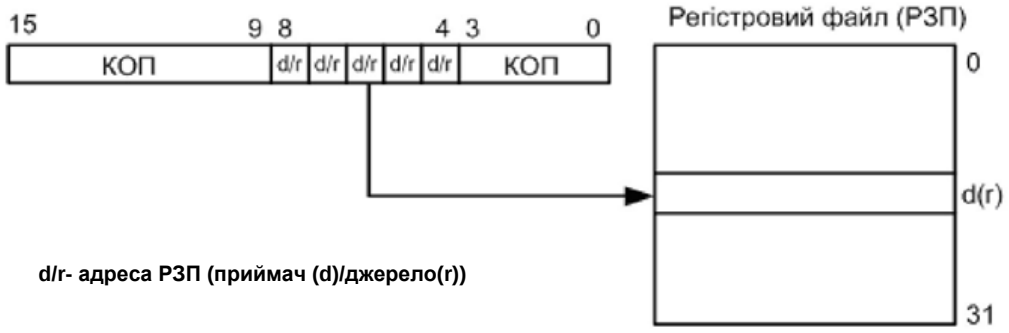


Рисунок 1.11 – Пряма адресація одного регістра загального призначення

Прикладом команд, що використовують цей спосіб адресації, є команди роботи зі стеком – PUSH, POP, команди інкременту – INC, декременту – DEC, а також деякі команди арифметичних операцій.

Пряма адресація двох регістрів загального призначення

Цей спосіб адресації використовується в командах, що оперують одночасно двома регістрами загального призначення. При цьому адреса регістра-джерела міститься в розрядах 9, 3...0 (5 біт), а адреса регістра-приймача в розрядах 8... 4 (5 біт) машинного коду команди (рисунок 1.12).

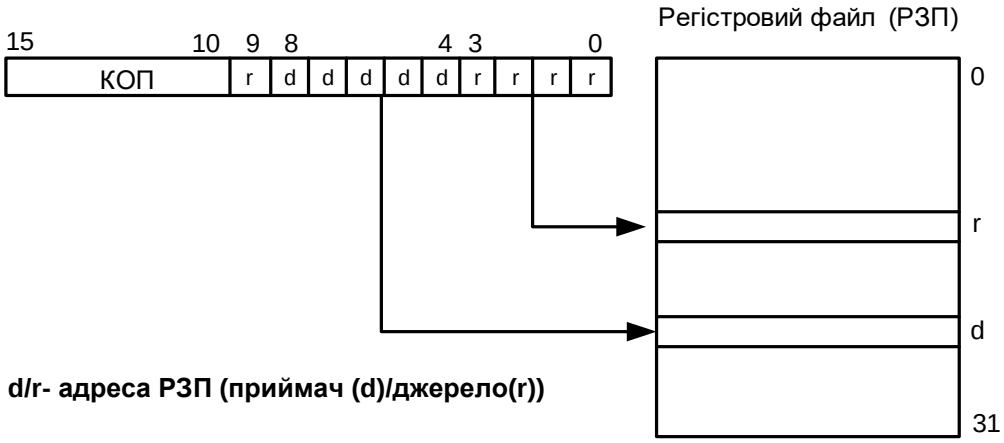


Рисунок 1.12 – Пряма адресація двох регістрів загального призначення

До команд, що використовують цей спосіб адресації, належать команди пересилання даних з одного РЗП у другий – MOV, а також більшість команд арифметичних операцій.

Деякі команди мають тільки один регістр-операнд, але використовують цей спосіб адресації. У цьому випадку той самий регістр є джерелом і приймачем. Як приклад можна навести команду очищення регістра – CLR Rd, яка виконує операцію «виключне АБО» регістра із самим собою – EOR Rd, Rd.

Пряма адресація регістра введення/виведення

Цей спосіб адресації використовується командами пересилання даних між регістром введення/виведення основного простору РВВ і РЗП (регістровим файлом) – IN і OUT. У цьому разі адреса регістра введення/виведення міститься в розрядах 10, 9, 3...0 – 6 біт, а адреса РЗП – у розрядах 8...4 – 5 біт машинного коду команди (рисунок 1.13).

Рисунок 1.13 – Пряма адресація регістрів введення/виведення

Пряма адресація статичної пам’яті даних

Цей спосіб використовується при зверненні до всього адресного простору статичної пам’яті даних, яка включає: РЗП, РВВ та СОЗП.

Є тільки дві команди, що використовують цей спосіб адресації. Це команди пересилання байта між одним з РЗП і коміркою СПД – LDS і STS. Кожна з цих команд займає в пам’яті програм два слова (32 біти). У першому слові міститься код операції та адреса регістра загального призначення (у розрядах з 8-го по 4-й).

У другому слові міститься адреса комірки пам’яті, до якої відбувається звернення (рисунок 1.14).

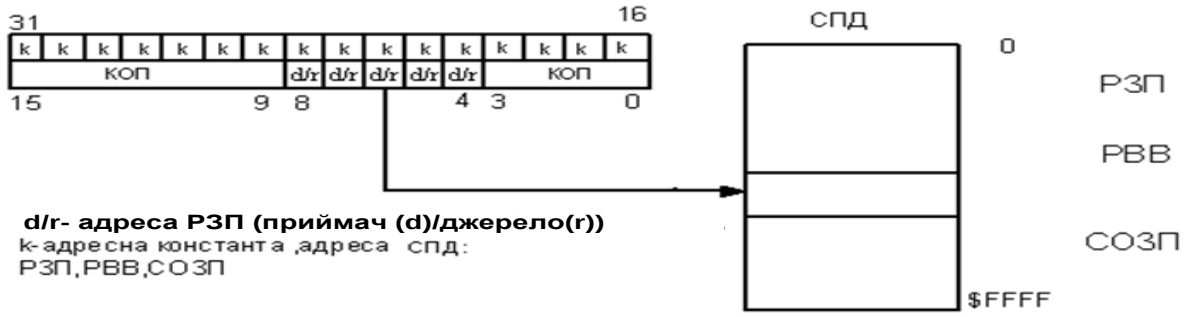


Рисунок 1.14 – Пряма адресація статичної пам’яті даних

В СПД за адресами $00...$1F розташовано регістри загального призначення, за адресами $20...$5F – регістри введення/виведення, а за адресами $60...$FF – СОЗП.

Непряма адресація

При непрямій адресації адреса комірки статичної пам’яті даних міститься в одному з індексних регістрів X, Y і Z.

Є наступні різновиди непрямої адресації:

- проста непряма адресація;
- відносна непряма адресація;
- непряма адресація з переддекрементом;
- непряма адресація з постінкрементом;
- непряма адресація пам’яті програм;
- непряма адресація констант в пам’яті програм.

Проста непряма адресація

При використанні простої непрямої адресації звернення виконується до СПД за адресою, що міститься в одному з індексних регістрів: X, Y або Z (рисунок 1.15). Жодних дій із вмістом індексного регістра при цьому не виконується.

Мікроконтролери підтримують 6 команд (по 2 для кожного індексного регістра) простої непрямої адресації: LD Rd, X/Y/Z (пересилання байта з СПД в

РЗП) і ST X/Y/Z, Rr (пересилання байта з РЗП в СПД). Адреса регістра загального призначення міститься в розрядах 8...4 машинного коду команди.

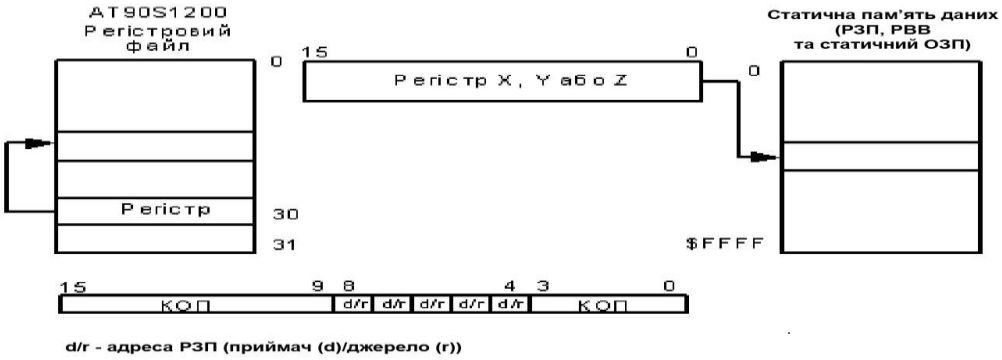


Рисунок 1.15 – Проста непряма адресація

Відносна непряма адресація

В командах відносної непрямої адресації адреса комірки статичної пам'яті даних, до якої виконується звернення, обчислюється додаванням вмісту індексного регістра – Y або Z і константи, що міститься в машинному коді команді (рисунок 1.16).

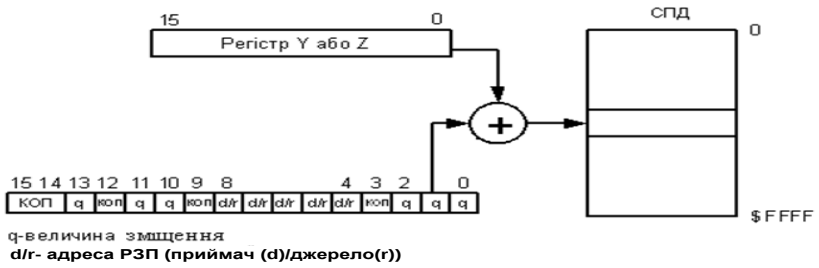


Рисунок 1.16 – Відносна непряма адресація

Мікроконтролери підтримують 4 команди відносної непрямої адресації (дві для регістра Y і дві для регістра Z): LDD Rd, Y+q/Z+q (пересилання байта із СПД у РЗП) і STD Y+q/Z+q, Rr (пересилання байта із РЗП у СПД). Адреса регістра загального призначення міститься в розрядах 8...4 машинного коду команди, а величина зміщення (q) – у розрядах 13, 11, 10, 2...0. Оскільки під значення зміщення виділяється тільки 6 біт, воно не може перевищувати 63 ($0 \leq q \leq 63$).

Непряма адресація з попереднім декрементом (переддекрементом)

При виконанні команд непрямої адресації з переддекрементом вміст індексного регістра спочатку зменшується на одиницю, а потім виконується звернення за отриманою адресою (рисунок 1.17).

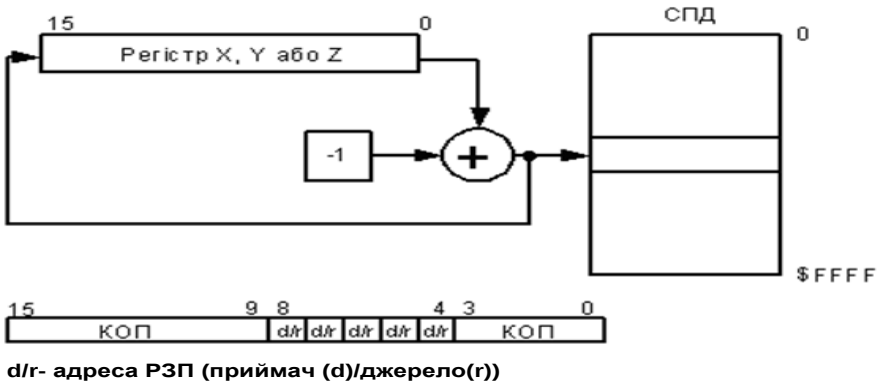


Рисунок 1.17 – Непряма адресація з переддекрементом

Мікроконтролери сімейства підтримують 6 команд (по 2 для кожного індексного регістра) непрямої адресації з переддекрементом: LD Rd, -X/-Y/-Z (пересилання байта з СПД в РЗП) і ST -X/-Y/-Z, Rr (пересилання байта з РЗП в СПД). Адреса регістра загального призначення міститься в розрядах 8...4 машинного коду команди, а регістри X/Y/Z адресуються неявно.

Непряма адресація з постінкрементом

При виконанні команд непрямої адресації з постінкрементом (наступним інкрементом) після звернення за адресою, що міститься в індексному регістрі, вміст індексного регістра збільшується на одиницю (рисунок 1.18).

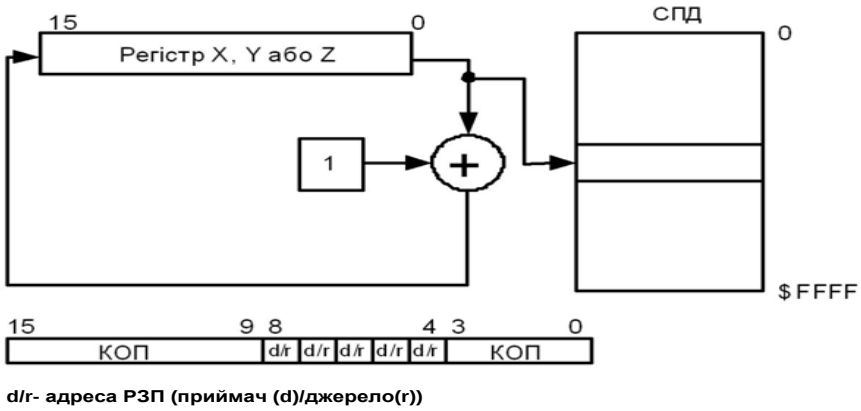


Рисунок 1.18 – Непряма адресація з постінкрементом

Мікроконтролери сімейства підтримують 6 команд (по 2 для кожного індексного регістра) непрямої адресації з постінкрементом: LD Rd, X+/Y+/Z+ (пересилання байта з СПД в РЗП) і ST X+/Y+/Z+, Rr (пересилання байта з РЗП в СПД). Адреса регістра загального призначення міститься в розрядах 8...4 машинного коду команди, а регістри X/Y/Z адресуються неявно.

Непряма адресація пам’яті програм

Такий спосіб адресації використовується в командах IJMP, ICALL (рисунок 1.19).

В результаті виконання такої команди програма продовжує виконуватися з адреси, що міститься в індексному регістрі Z. Таким чином, дія команди зводиться до завантаження вмісту індексного регістра в лічильник команд.

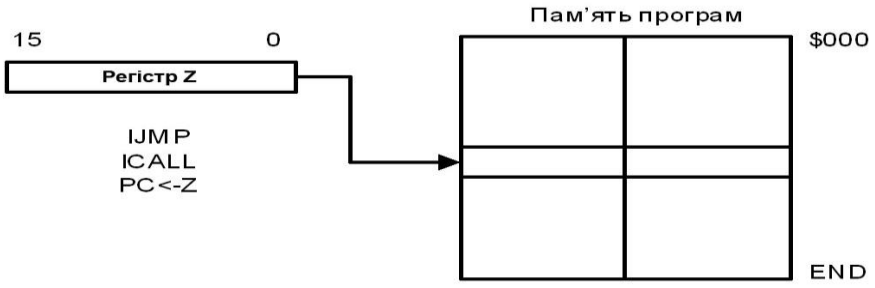


Рисунок 1.19 – Непряма адресація пам’яті програм

Оскільки індексний регістр – 16-розрядний, то максимально можлива величина переходу становить 64 Кслів (128 Кбайт).

Команда виконується за 2 машинних цикли.

Непряма адресація констант в пам’яті програм

Цей спосіб адресації використовується, наприклад, в команді LPM, яка завантажує один байт із пам’яті програм у регістр загального призначення R0 (рисунок 1.20).

Адреса комірки пам’яті, до якої відбувається звернення, міститься в індексному регістрі Z.

При цьому старші п’ятнадцять розрядів регістра – Z1…Z15 адресують слово в пам’яті програм, а молодший біт – Z0 адресує байт в обраному слові. Якщо Z0 = 0, то адресується молодший байт (МБ) слова, а якщо Z0 = 1 – старший байт (СБ).

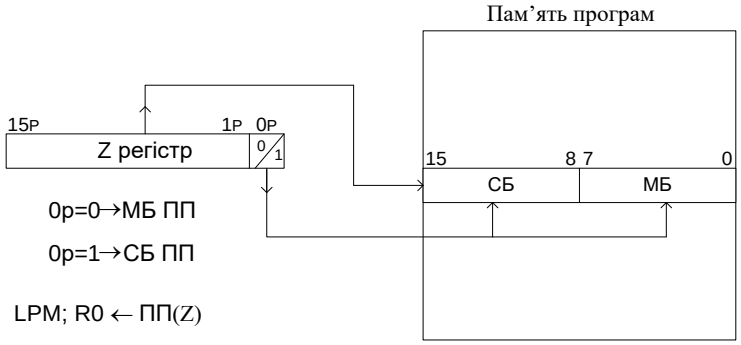


Рисунок 1.20 – Непряма адресація констант в пам’яті програм

Доступний об’єм пам’яті програм для цієї команди не може перевищувати 2^{15} = 32768 слів. Для звернення до розширеної пам’яті програм може використовуватися команда ELPM.

Відносна адресація пам’яті програм

Цей спосіб адресації використовується в командах RJMP, RCALL (рисунок 1.21).

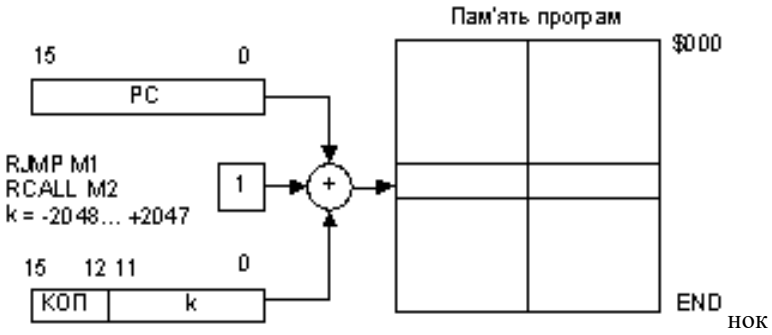


Рисунок 1.21 – Відносна адресація пам’яті програм

При виконанні команди до поточного вмісту лічильника команд (адреси наступної команди) додається дванадцятирозрядне число зі знаком – k, яке представлено у додатковому двійковому коді.

Ця команда має обмеження за областю дії. Через те, що операнд являє собою 12-розрядне число зі знаком, максимальна величина переходу відносно адреси наступної команди становить від -2048 до +2047 слів (приблизно, ±4 Кбайт).

В програмах в якості операндів цієї команди замість адрес використовуються мітки. Компілятор обчислює величину зміщення відносно адреси наступної команди – k. Для цього він віднімає від адреси мітки адресу наступної команди і підставляє це значення у машинний код команди.

1.4 Характеристика команд AVR-мікроконтролерів

1.4.1 Загальні відомості

Мнемоніка команди та мнемокод

Для написання програм мовою «Асемблер» використовують мнемоніки команд та мнемокоди. Мнемоніки введені для полегшення написання програм, і складають основу мови *Асемблера.*

Мнемоніка команди – це представлення коду операції у вигляді сполучення латинських літер, що мають визначений зміст (використовуються англійські слова або скорочення, наприклад, MOV, PUSH, POP, JMP, CLR, NOP і т. ін.*)*.

Мнемокод включає в себе мнемоніку команди та символічний опис операндів – даних, які беруть участь в операції.

Код операції команди

Код операції команди (КОП) – це комбінація бітів (не більше восьми для 8-розрядних МК), що знаходяться на початку машинного коду команди і визначають тип операції, що підлягає виконанню у конкретний момент часу. КОП читається з пам’яті і розміщується в програмно недоступний регістр команд мікроконтролера (РК). Потім він декодується і визначається тип команди, яка повинна бути виконана. Крім КОП в машинний код команди можуть входити адреси та операнди.

Машинний код команди

Машинний код команди представляє собою двійковий код команди, який складається з одного або декількох байтів в залежності від типу команд конкретного МК.

Операнди

Операндами у мікропроцесорній техніці називають дані, які приймають участь у виконанні тієї чи іншої команди (операції). В залежності від способу адресації операндів, дані можуть знаходитися у регістрах та пам'яті. Існують одноопераднідні або двоопераднідні команди.

1.4.2 Формати команд

Більшість мікроконтролерів сімейства AVR мають 14 форматів (типів) базового набору команд, наведених на рисунку 1.22. На рисунку використано наступні скорочення: КОП – код операції; K – константа даних; k – адресна константа; b – номер біта в регістрі введення/виведення або регістрового файлу (РЗП) – 3 біти; s – номер біта в регістрі стану – 3 біти; A – адреса регістра в основному просторі введення/виведення; q – задає зміщення для непрямої адресації – 6 біт; d(r) – регістр-приймач (джерело) з області регістрового файлу.

До першого типу належать команди, код операції яких займає всю довжину команди – 16 біт. До цієї групи належить, наприклад, команда LPM – завантаження регістра загального призначення R0 з пам'яті програм за адресою, яка міститься у регістровій парі Z = R31:R30. Такий формат також мають команди IJMP, ICALL, RET, RETI, NOP, SLEEP та WDT.

До другого типу належать команди, які окрім коду операції містять п'ятирозрядну адресу одного з регістрів загального призначення. Наприклад, це команди DEC Rd; LD Rd, -X; ST Y, Rr і т. ін.

Третій тип мають команди, в яких адресуються два операнди – регістри загального призначення: Rd – приймач, Rr – джерело. Це, наприклад, команди ADD Rd, Rr; CPSE Rd, Rr; AND Rd, Rr і т. ін.

До четвертого типу належать двоопераднідні команди, в яких один операнд: К6 = 6 біт (К = 0...63) входить в саму команду – безпосередня адресація, а другим є пара регістрів: R24, R25; R26, R27; R28, R29; R30, R31, які кодуються двома бітами команди – dd: 00 – R24, R25; 01 – R26, R27; 10 – R28, R29; 11 – R30, R31.

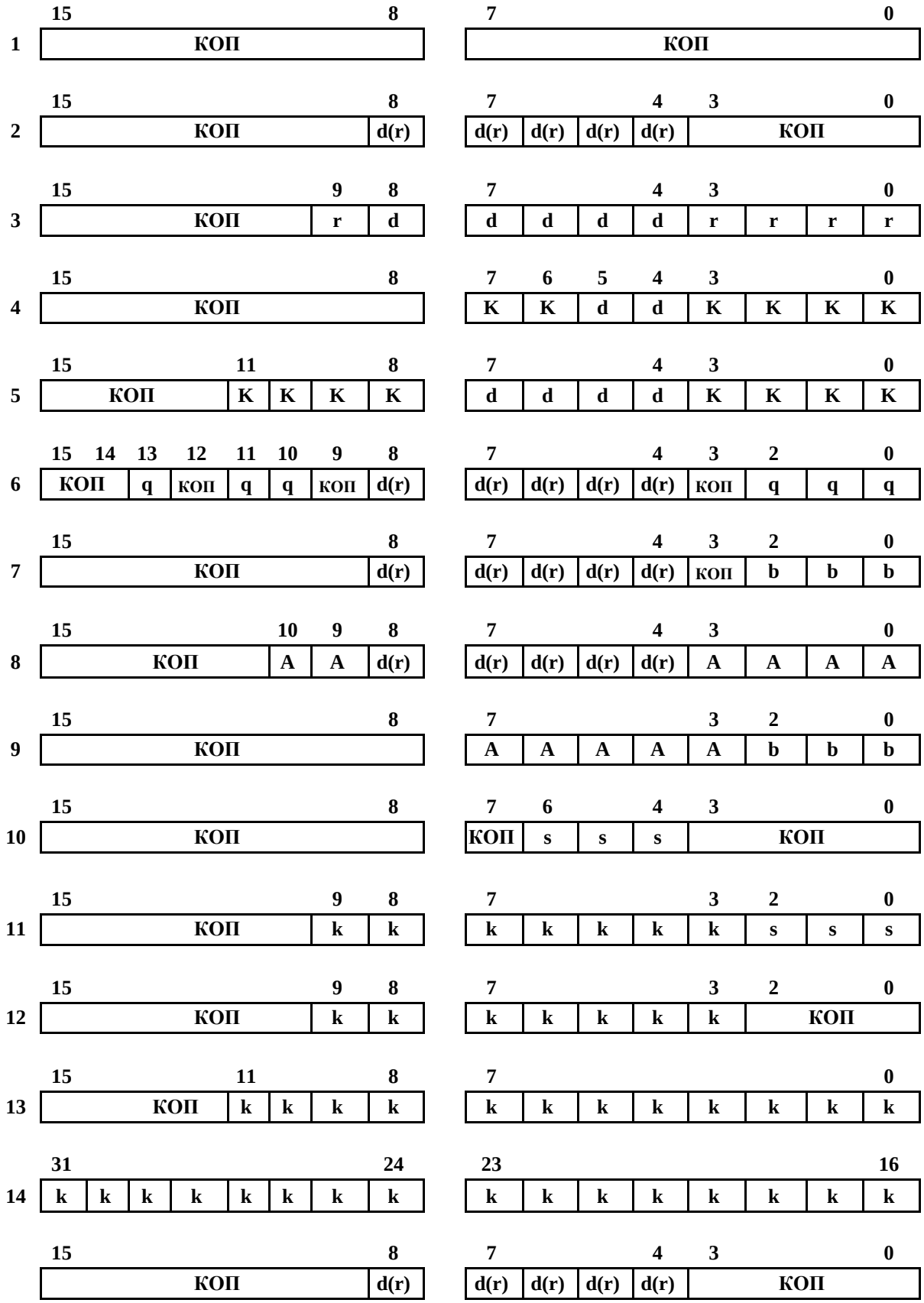


Рисунок 1.22 – Формати базового набору команд

Наприклад, команди ADIW Rdl, K6; SBIW Rdl, K6, де Rdl = R24/R26/R28/R30.

П'ятий тип мають двооперандні команди, наприклад, SBCI Rd*, K; ORI Rd*, K, в яких Rd* – один із шістнадцяти РЗП (R16...R31), а другий К – восьмибітна константа (безпосередній операнд): $0 \le K \le 255$.

До шостого типу належать команди, в яких один з двох операндів може бути РЗП (Rd – приймач, Rr – джерело), а другий міститься у СПД і адресується за допомогою непрямої відносної адресації, коли до вмісту індексного регістра Y або Z додається зміщення q ($0 \le q \le 63$). Це, наприклад, команди LDD Rd, z+q; STD Y+q, Rr і т. ін.

Сьомий тип мають команди, в яких адресується один із восьми бітів РЗП. Наприклад, команди BLD Rd, b; BST Rr, b; SBRC Rr, b і т. ін., де Rd – приймач, Rr – джерело (один з 32-х РЗП), а b = 0...7 – номер біта РЗП.

До восьмого типу належать команди, в яких одним операндом є РЗП (Rd – приймач/Rr – джерело), а другий міститься в одному з 64-х РВВ основного простору. Наприклад, команди IN Rd, P; OUT P, Rr, де P – адреса РВВ (P = 0…63).

Дев'ятий тип мають команди, в яких адресуються окремі біти молодшої половини РВВ (P* = 0...31). Наприклад, команди SBI P*, b, де b = 0...7 – номер біта РВВ.

До десятого типу належать команди, в яких адресується один із 8-ми бітів регістра прапорців SREG. Наприклад, команди BSET s; BCLR s, де s = 0...7 – номер біта регістра стану.

Одинадцятий тип мають команди умовного переходу в залежності від значення вказаних бітів регістра стану. Наприклад, команди BRBS s, k; BRBC s, k, де s = 0...7 – номер біта регістра стану SREG, а k = 7 біт ($-64 \le k \le 63$) при переході додається до адреси наступної команди.

Дванадцятий тип мають команди умовного відносного переходу в залежності від значень окремих прапорців регістра стану (прапорці адресуються неявно). Наприклад, команди BREQ k; BRCC k і т. ін., де k = 7 біт ($-64 \le k \le 63$) при переході додається до адреси наступної команди.

До тринадцятого типу належать команди відносного безумовного переходу: RJMP k та відносного безумовного виклику підпрограм: RCALL k, де k = 12 біт. Діапазон переходів та виклику підпрограм: -2048…+ 2047.

Чотирнадцятий тип мають команди: LDS Rd, k – пряме завантаження та STS k, Rr – пряме збереження, в яких одним операндом є РЗП (Rd – приймач/Rr – джерело), а другий міститься у СПД (РЗП, РВВ та СОЗП). Оскільки k = 16 біт, то кількість комірок СПД = 2^{16} = 65536.

1.4.3 Формати даних

При роботі з мікроконтролером необхідно знати не тільки формати (типи) команд (рисунок 1.22), які керують роботою мікроконтролера, але й формати (типи) даних (операндів), що беруть участь у виконанні тієї або іншої команди (операції) (рисунок 1.23).

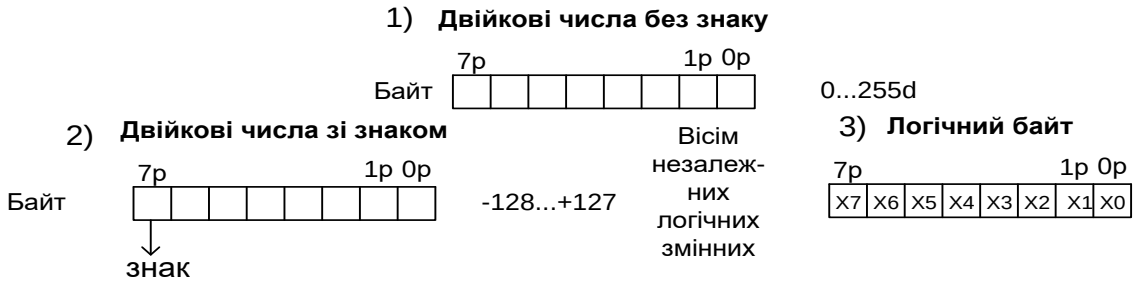


Рисунок 1.23 – Типи (формати) даних мікроконтролера

1.4.4 Довжина команд у байтах та їх розміщення у пам’яті програм

Відповідно до рисунку 1.22 більшість команд мікроконтролерів сімейства AVR мають довжину – одне слово (2 байти), за виключенням команди чотирнадцятого типу, яка має довжину 2 слова (4 байти).

Програма, яка керує роботою МК, послідовно, команда за командою, розміщується в сусідніх комірках пам’яті в порядку зростання їх адрес. Адреса команди, яка повинна виконуватись, знаходиться у програмному лічильнику PC. Пристрій керування мікроконтролера на основі прочитаного коду операції, що міститься в першому байті команди, визначає, скільки ще байтів міститься в команді, та керує їх читанням з пам’яті шляхом збільшення на одиницю адреси, яка

видається при кожному зверненні до пам’яті. Після читання з пам’яті чергової команди МК формує адресу КОП наступної команди.

1.4.5 Вплив команд на прапорці

Як відзначалося раніше, одним з основних регістрів МК-ра є регістр прапорців (ознак), який було описано у підрозділі 1.1.3. Прапорці регістра ознак встановлюються під час виконання ряду команд МК-ра. Під час опису системи команд, яка, як правило, оформлюється у вигляді таблиці, в одній з колонок показується вплив окремих команд на ті або інші прапорці (таблиця 1.3).

Як правило, команди пересилання не змінюють прапорці, окрім команд, які пересилають дані у регістр прапорців. Частіше прапорці змінюють арифметичні, логічні команди і команди порівняння.

Окремі прапорці (ознаки) аналізуються під час виконання команд умовних переходів, виклику і повернення з підпрограм, що дозволяє передавати керування в програмі в залежності від поточного значення прапорців.

1.4.6 Час виконання команд

В AVR-мікроконтролерах окремі команди виконуються за 1, 2, 3 або 4 такти. Тривалість одного такту дорівнює одному періоду тактової частоти f_{BQ}.

1.4.7 Базовий набір команд мікроконтролера

Загальні відомості

AVR-мікроконтролери мають RISC-архітектуру, основною перевагою якої є збільшення швидкодії за рахунок скорочення кількості операцій обміну з пам’яттю програм. Практично всі команди займають одну комірку пам’яті. Виняток становлять команди, в яких одним з операндів є 16-розрядна адреса СПД (РЗП, РВВ та СОЗП).

Підвищення швидкодії досягнуто не за рахунок скорочення кількості команд процесора, а за рахунок збільшення розрядності комірки пам’яті програм до 16. Більшість команд виконується за один машинний цикл (1 такт).

Всі команди AVR-мікроконтролерів можна розділити на декілька груп:

– логічні операції;

– арифметичні операцій та команди зсуву;

– операції з бітами;

– команди пересилання даних;

– команди передачі керування;

– команди керування системою.

Нижче скорочено описано ці команди. Детальнішу інформацію наведено в [1; 2; 10].

У таблиці 1.3 наведено базовий набір команд, якими можна користуватись в типовому AVR-мікроконтролері, а також основні відомості про команди, такі як мнемонічне позначення команди, її опис, тип, кількість тактів, необхідних для її виконання, а також прапорці регістра SREG, на які впливає ця команда.

В таблиці 1.3 використано наступні позначення:

- Rd – регістр-приймач результату, $0 \le d \le 31$;
- Rd* – регістр-приймач результату, $16 \le d \le 31$;
- Rdl = R24/R26/R28/R30 для команд ADIW і SBIW;
- Rr – регістр-джерело;
- P – адреса регістра введення/виведення;
- P* – адреса регістра введення/виведення, який адресується побітово (адреси $00…$1F);
- K – символьна або числова константа (8 біт);
- K6 – символьна або числова константа (6 біт);
- k – адресна константа ;
- b – номер біта в регістрі (3 біти);
- q – константа (6 біт), може бути константний вираз;
- s – номер біта в регістрі стану (3 біти);
- X, Y, Z – індексні регістри непрямої адресації (X = R27:R26, Y = R29:R28, Z = R31:R30).

Таблиця 1.3 – Базовий набір команд AVR-мікроконтролера

№	Мнемонічне позначення	Операнди	Опис	Операція	Прапорці	Тип	Кільк. тактів
АРИФМЕТИЧНІ, ЛОГІЧНІ КОМАНДИ ТА КОМАНДИ ЗСУВУ							
1	ADD	Rd, Rr	Додавання без перенесення	Rd ← Rd + Rr	Z,C,N, V,H,S	3	1
2	ADC	Rd, Rr	Додавання з перенесенням	Rd ← Rd + Rr + C	Z,C,N, V,H,S	3	1
3	ADIW	Rdl, K6	Додавання двох байт із константою	Rdh:Rdl ← Rdh:Rdl+K6	Z,C,N,V,S	4	2
4	SUB	Rd, Rr	Віднімання без перенесення	Rd ← Rd – Rr	Z,C,N, V,H,S	3	1
5	SUBI	Rd*, K	Віднімання константи	Rd* ← Rd* – K	Z,C,N, V,H,S	5	1
6	SBC	Rd, Rr	Віднімання з перенесенням	Rd ← Rd – Rr – C	Z,C,N, V,H,S	3	1
7	SBCI	Rd*, K	Віднімання константи з перенесенням	Rd* ← Rd* – K – C	Z,C,N, V,H,S	5	1
8	SBIW	Rdl, K6	Віднімання з двох байт константи	Rdh:Rdl ← Rdh:Rdl – K6	Z,C,N,V,S	4	2
9	AND	Rd, Rr	Логічне І	Rd ← Rd • Rr	Z,N,V,S	3	1
10	ANDI	Rd*, K	Логічне І з константою	Rd* ← Rd* • K	Z,N,V,S	5	1
11	OR	Rd, Rr	Логічне АБО	Rd ← Rd v Rr	Z,N,V,S	3	1
12	ORI	Rd*, K	Логічне АБО з константою	Rd* ← Rd* v K	Z,N,V,S	5	1
13	EOR	Rd, Rr	Логічне виключне АБО	Rd ← Rd ⊕ Rr	Z,N,V,S	3	1
14	COM	Rd	Побітова інверсія	Rd ← $FF – Rd	Z,C,N,V,S	2	1
15	NEG	Rd	Зміна знака (формування додаткового коду)	Rd ← $00 – Rd	Z,C,N, V,H,S	2	1
16	INC	Rd	Інкремент значення регістра	Rd ← Rd + 1	Z,N,V,S	2	1
17	DEC	Rd	Декремент значення регістра	Rd ← Rd – 1	Z,N,V,S	2	1
18	ASR	Rd	Арифметичний зсув вправо	Rd(n) ← Rd(n+1), n = 0…6, C←Rd(0), Rd(7)←Rd(7)	Z,C,N,V	2	1
19	LSL	Rd	Логічний/арифметичний зсув вліво	Rd(n+1) ← Rd(n), Rd(0) ← 0, C←Rd(7)	Z,C,N,V	2	1
20	LSR	Rd	Логічний зсув вправо	Rd(n) ← Rd(n+1), Rd(7) ← 0, C←Rd(0)	Z,C,N,V,S	2	1
21	ROL	Rd	Циклічний зсув вліво через C	Rd(0)←C,Rd(n+1)← Rd(n), C←Rd(7)	Z,C,N, V,H,S	2	1
22	ROR	Rd	Циклічний зсув вправо через C	Rd(7)←C,Rd(n)← Rd(n+1),C←Rd(0)	Z,C,N,V,S	2	1
23	TST	Rd	Перевірка на нуль чи від’ємне значення	Rd ← Rd • Rd	Z,N,V,S	2	1
24	CLR	Rd	Очищення регістра	Rd ← Rd ⊕ Rd	Z,N,V,S	2	1
25	SWAP	Rd	Обмін тетрадами	Rd(3…0)←Rd(7…4), Rd(7…4)←Rd(3…0)	-	2	1
26	SER	Rd	Встановлення регістра	Rd ← $FF	-	2	1
КОМАНДИ ПЕРЕДАЧІ КЕРУВАННЯ							
1	RJMP	k	Відносний безумовний перехід	PC ← PC + k + 1	-	13	2
2	IJMP		Непрямий безумовний перехід	PC ← Z	-	1	2
3	RCALL	k	Відносний безумовний виклик підпрограми	STACK ← PC+1, PC ← PC + k + 1, SP ← SP-2	-	13	3
4	ICALL		Непрямий безумовний виклик підпрограми	STACK ← PC+1, PC ← Z, SP ← SP-2	-	1	3

Продовження таблиці 1.3

№	Мнемо-нічне позначення	Операнди	Опис	Операція	Прапорці	Тип	Кільк. тактів
5	RET		Повернення з підпрограми	PC ← STACK, SP ← SP+2	-	1	4
6	RETI		Повернення з підпрограми обробки переривання	PC ← STACK, SP ← SP+2, I ← 1	I	1	4
7	CPSE	Rd, Rr	Порівняти, пропустити, якщо рівні	if (Rd = Rr), то PC ← PC + 2/3	-	3	1/2/3
8	CP	Rd, Rr	Порівняти	Rd - Rr	Z,N,V, C,H,S	3	1
9	CPC	Rd, Rr	Порівняти з перенесенням	Rd - Rr - C	Z,N,V, C,H,S	3	1
10	CPI	Rd*, K	Порівняти з константою	Rd* - K	Z,N,V, C,H,S	5	1
11	SBRC	Rr, b	Пропустити, якщо біт у регістрі скинутий	if (Rr(b) = 0), то PC ← PC + 2/3	-	7	1/2/3
12	SBRS	Rr, b	Пропустити, якщо біт у регістрі встановлений	if (Rr(b) = 1), то PC ← PC + 2/3	-	7	1/2/3
13	SBIC	P*, b	Пропустити, якщо біт у порту скинутий	if (P*(b) = 0), то PC ← PC + 2/3	-	9	1/2/3
14	SBIS	P*, b	Пропустити, якщо біт у порту встановлений	if (P*(b) = 1), то PC ← PC + 2/3	-	9	1/2/3
15	BRBS	s, k	Перейти, якщо прапорець у SREG встановлений	if (SREG(s) = 1) then C←PC+k+1	-	11	1/2
16	BRBC	s, k	Перейти, якщо прапорець у SREG скинутий	if(SREG(s) = 0) then PC←PC+k+1	-	11	1/2
17	BREQ	k	Перейти, якщо дорівнює	if (Z = 1) then PC ← PC + k + 1	-	12	1/2
18	BRCS	k	Перейти, якщо прапорець перенесення встановлений	if (C = 1) then PC ← PC + k + 1	-	12	1/2
19	BRNE	k	Перейти, якщо не дорівнює	if (Z = 0) then PC ← PC + k + 1	-	12	1/2
20	BRCC	k	Перейти, якщо прапорець перенесення скинутий	if (C = 0) then PC ← PC + k + 1	-	12	1/2
21	BRSH	k	Перейти, якщо дорівнює або більше	if (C = 0) then PC ← PC + k + 1	-	12	1/2
22	BRLO	k	Перейти, якщо менше	if (C = 1) then PC ← PC + k + 1	-	12	1/2
23	BRMI	k	Перейти, якщо мінус	if (N = 1) then PC ← PC + k + 1	-	12	1/2
24	BRPL	k	Перейти, якщо плюс	if (N = 0) then PC ← PC + k + 1	-	12	1/2
25	BRGE	k	Перейти, якщо більше або дорівнює (зі знаком)	if (N ⊕ V = 0) then PC ← PC + k+1	-	12	1/2
26	BRLT	k	Перейти, якщо менше (зі знаком)	if (N ⊕ V = 1) then PC← PC+ k+1	-	12	1/2
27	BRHS	k	Перейти, якщо прапорець половинного перенесення встановлений	if (H = 1) then PC ← PC + k + 1	-	12	1/2
28	BRHC	k	Перейти, якщо прапорець половинного перенесення скинутий	if (H = 0) then PC ← PC + k + 1	-	12	1/2
29	BRTS	k	Перейти, якщо прапорець T встановлений	if (T = 1) then PC ← PC + k + 1	-	12	1/2
30	BRTC	k	Перейти, якщо прапорець T скинутий	if (T = 0) then PC ← PC + k + 1	-	12	1/2
31	BRVS	k	Перейти, якщо прапорець переповнення встановлений	if (V = 1) then PC ← PC + k + 1	-	12	1/2
32	BRVC	k	Перейти, якщо прапорець переповнення скинутий	if (V = 0) then PC ← PC + k + 1	-	12	1/2
33	BRIE	k	Перейти, якщо переривання дозволені	if (I = 1) then PC ← PC + k + 1	-	12	1/2
34	BRID	k	Перейти, якщо переривання заборонені	if (I = 0) then PC ← PC + k + 1	-	12	1/2

Продовження таблиці 1.3

№	Мнемо-нічне позначення	Операнди	Опис	Операція	Прапорці	Тип	Кіл. тактів
КОМАНДИ ПЕРЕСИЛАННЯ ДАНИХ							
1	MOV	Rd, Rr	Копіювання регістра	Rd ← Rr	-	3	1
2	LDI	Rd*, K	Завантаження константи	Rd* ← K	-	5	1
3	LD	Rd, X	Непряме завантаження	Rd ← (X)	-	2	2
4	LD	Rd, X+	Непряме завантаження з постінкрементом	Rd ← (X), X ← X + 1	-	2	2
5	LD	Rd, -X	Непряме завантаження з переддекрементом	X ← X - 1, Rd ← (X)	-	2	2
6	LD	Rd, Y	Непряме завантаження	Rd ← (Y)	-	2	2
7	LD	Rd, Y+	Непряме завантаження з постінкрементом	Rd ← (Y), Y ← Y + 1	-	2	2
8	LD	Rd, -Y	Непряме завантаження з переддекрементом	Y ← Y - 1, Rd ← (Y)	-	2	2
9	LDD	Rd, Y+q	Непряме завантаження зі зміщенням	Rd ← (Y + q)	-	6	2
10	LD	Rd, Z	Непряме завантаження	Rd ← (Z)	-	2	2
11	LD	Rd, Z+	Непряме завантаження з постінкрементом	Rd ← (Z), Z ← Z+1	-	2	2
12	LD	Rd, -Z	Непряме завантаження з переддекрементом	Z ← Z - 1, Rd ← (Z)	-	2	2
13	LDD	Rd, Z+q	Непряме завантаження зі зміщенням	Rd ← (Z + q)	-	6	2
14	LDS	Rd, k	Пряме завантаження	Rd ← (k)	-	14	2
15	ST	X, Rr	Непряме збереження	(X) ← Rr	-	2	2
16	ST	X+, Rr	Непряме збереження з постінкрементом	(X) ← Rr, X ← X + 1	-	2	2
17	ST	- X, Rr	Непряме збереження з переддекрементом	X ← X - 1, (X) ← Rr	-	2	2
18	ST	Y, Rr	Непряме збереження	(Y) ← Rr	-	2	2
19	ST	Y+, Rr	Непряме збереження з постінкрементом	(Y) ← Rr, Y ← Y + 1	-	2	2
20	ST	- Y, Rr	Непряме збереження з переддекрементом	Y ← Y - 1, (Y) ← Rr	-	2	2
21	STD	Y+q, Rr	Непряме збереження зі зміщенням	(Y + q) ← Rr	-	6	2
22	ST	Z, Rr	Непряме збереження	(Z) ← Rr	-	2	2
23	ST	Z+, Rr	Непряме збереження з постінкрементом	(Z) ← Rr, Z ← Z + 1	-	2	2
24	ST	-Z, Rr	Непряме збереження з переддекрементом	Z ← Z - 1, (Z) ← Rr	-	2	2
25	STD	Z+q, Rr	Непряме збереження зі зміщенням	(Z + q) ← Rr	-	6	2
26	STS	k, Rr	Пряме збереження	(k) ← Rr	-	14	2
27	LPM*		Завантаження байта з програмної пам’яті	R0 ← (Z)	-	1	3
28	IN	Rd, P	Читання порту	Rd ← P	-	8	1
29	OUT	P, Rr	Запис у порт	P ← Rr	-	8	1
30	PUSH	Rr	Занесення регістра в стек	STACK ← Rr; SP← SP - 1	-	2	2
31	POP	Rd	Витягнення регістра зі стека	SP← SP + 1, Rd ← STACK	-	2	2
КОМАНДИ РОБОТИ З БІТАМИ							
1	SBR	Rd*, K	Встановити біт (біти) у регістрі	Rd* ← Rd* v K	Z,N,V,S	5	1
2	CBR	Rd* , K	Скинути біт (біти) у регістрі	Rd* ← Rd* • ($FF - K)	Z,N,V,S	5	1
3	SBI	P*, b	Встановити біт у РВВ	I/O(P*, b) ← 1	-	9	2
4	CBI	P*, b	Скинути біт у РВВ	I/O(P*, b) ← 0	-	9	2
5	BSET	S	Встановити вказаний розряд регістра SREG	SREG(s) ← 1	SREG(s)	10	1
6	BCLR	S	Скинути заданий розряд регістра SREG	SREG(s) ← 0	SREG(s)	10	1
7	BLD	Rd, b	Завантажити біт з T у регістр	Rd(b) ← T	-	7	1
8	BST	Rr, b	Зберегти біт з регістра в T	T ← Rr(b)	T	7	1
9	SEC		Встановити прапорець перенесення	C ← 1	C	1	1
10	CLC		Скинути прапорець перенесення	C ← 0	C	1	1
11	SEN		Встановити прапорець від’ємного числа	N ← 1	N	1	1
12	CLN		Скинути прапорець від’ємного числа	N ← 0	N	1	1

Закінчення таблиці 1.3

№	Мнемонічне позначення	Операнди	Опис	Операція	Прапорці	Тип	Кільк. тактів
13	SEZ		Встановити прапорець нуля	Z←1	Z	1	1
14	CLZ		Скинути прапорець нуля	Z←0	Z	1	1
15	SEI		Встановити прапорець переривань	I←1	I	1	1
16	CLI		Скинути прапорець переривань	I←0	I	1	1
17	SES		Встановити прапорець знака	S←1	S	1	1
18	CLS		Скинути прапорець знака	S←0	S	1	1
19	SEV		Встановити прапорець переповнення	V←1	V	1	1
20	CLV		Скинути прапорець переповнення	V←0	V	1	1
21	SET		Встановити прапорець T	T←1	T	1	1
22	CLT		Скинути прапорець T	T←0	T	1	1
23	SEH		Встановити прапорець половинного перенесення	H←1	H	1	1
24	CLH		Очистити прапорець половинного перенесення	H←0	H	1	1
КОМАНДИ КЕРУВАННЯ МІКРОКОНТРОЛЕРОМ							
1	NOP		Пуста операція	—	—	1	1
2	SLEEP		Переведення у режим сну	—	—	1	3
3	WDR		Скидання вартового таймера	—	—	1	1

Нижче наведено короткий опис окремих команд базового набору.

Арифметичні операції і команди зсуву

До даної групи належать команди, що виконують такі операції, як додавання, віднімання, зсув – вправо/вліво та інкремент/декремент. Усі операції виконуються тільки над регістрами загального призначення. Операнди можуть бути знаковими/беззнаковими числами, а також подані у додатковому коді.

Усі команди цієї групи виконуються за один машинний цикл, за винятком команд, що оперують двобайтовими значеннями, що виконуються за два цикли.

Логічні операції

Ці команди дозволяють виконувати стандартні логічні операції над байтами: «логічне множення» – І, «логічне додавання» – АБО, операцію «виключне АБО», а також обчислення зворотного і додаткового кодів числа. До цієї групи можна віднести також команди очищення/встановлення регістрів і команду перестановки

тетрад. Всі логічні операції виконуються за один машинний цикл над регістрами загального призначення, а результат зберігається в одному з РЗП.

Команди операцій з бітами

До даної групи належать команди, що виконують встановлення або скидання заданого розряду РЗП або РВВ. Є також команди для зміни розрядів регістра стану SREG, тому що перевірка стану розрядів саме цього регістра виконується найчастіше. Умовно до цієї групи можна віднести також дві команди передачі керування типу «перевірка/пропуск», що пропускають наступну команду в залежності від стану розряду РЗП або РВВ.

Усі задіяні розряди РВВ мають свої символічні імена. Визначення цих імен описано у тому ж файлі, що й визначення символічних імен для адрес регістрів [1; 2; 10]. Таким чином, після включення у програму зазначеного файлу в командах замість числових значень номерів розрядів можна буде вказувати їхні символічні імена.

Всі команди цієї групи виконуються за один машинний цикл, за винятком випадків, коли в результаті перевірки відбувається пропуск наступної команди. У цьому разі команда виконується за два або три машинних цикли залежно від довжини команди, що пропускається.

Команди пересилання даних

Команди цієї групи призначено для пересилання вмісту комірок, що лежать в просторі адрес статичної пам’яті даних – РЗП, РВВ та СОЗП. Поділ простору адрес на три частини обумовлює різноманітність команд даної групи. Пересилання даних може виконуватись в наступних напрямках:

- РЗП <=> РЗП;
- РЗП <=> РВВ;
- РЗП <=> статична пам’ять даних.

До даної групи також віднесяться команди PUSH і POP, які дозволяють зберігати у стеку і відновлювати зі стека вміст РЗП.

На виконання команд пересилання потрібно від одного до трьох машинних циклів в залежності від типу команди.

Команди передачі керування

У цю групу входять команди переходу, виклику підпрограм, повернення з них і команди типу «перевірка/пропуск», що пропускають наступну за ними команду при виконанні деякої умови. Також до цієї групи належать команди порівняння, що формують прапорці регістра SREG, які призначено переважно для роботи разом з командами умовного переходу. У командах цього типу виконується перевірка умови, результат якої впливає на виконання наступної команди. Якщо умова істинна, наступна команда ігнорується. Наприклад, команда SBRS Rd, b перевіряє розряд b регістра Rd і ігнорує (пропускає) наступну команду, якщо цей розряд дорівнює одиниці. Перехід до наступної інструкції виконується збільшенням лічильника команд на одиницю, а пропуск команди викликає завантаження нового значення в лічильник команд. Отже, коли умова, що перевіряється, істинна, у конвеєрі виникає затримка. Тривалість затримки залежить від довжини команди, що пропускається, і становить від одного до двох машинних циклів.

Команди передачі керування порушують нормальне (лінійне) виконання основної програми. Щоразу, коли виконується команда з цієї групи (окрім команд порівняння), нормальне функціонування конвеєра порушується. Перед завантаженням у конвеєр нової адреси він зупиняється й очищується послідовність виконуваних команд. Реініціалізація конвеєра призводить до того, що такі команди виконуються протягом декількох машинних циклів [1; 10].

В системі команд мікроконтролерів сімейства є команди як безумовного, так і умовного переходів. Команди непрямого – JMP і відносного – RJMP безумовного переходу є найпростішими в цій групі. Їх функція полягає тільки у записі нової адреси в лічильник команд.

При виконанні команди RJMP змінюється вміст лічильника команд шляхом додавання до нього або віднімання з нього деякого значення, що є операндом команди. Ця команда має обмеження за областю дії.

Через те, що операнд є 12-розрядним числом зі знаком, максимальна величина переходу відносно адреси наступної команди становить від -2048 до +2047 слів (приблизно ±4 Кбайт).

В програмах в якості операндів цієї команди замість констант використовуються мітки. Компілятор (асемблер) віднімає від адреси мітки адресу наступної команди і підставляє отримане значення у відповідне місце машинного коду команди. Нижче наведено приклад, який ілюструє сказане:

```
        cpi r16, $42; порівння регістра r16 з числом $42;
        brne error; перехід на мітку error, якщо r16 ≠ $42;
        rjmp ok ; безумовний перехід на мітку ok, якщо r16 = $42;
error:
        . . .
ok:     nop; місце переходу за командою RJMP.
```

Оскільки команда відносного переходу змінює вміст лічильника команд, вона виконується за 2 машинних цикли.

В результаті виконання команди непрямого переходу IJMP програма продовжує виконуватися з адреси, що міститься в індексному регістрі Z. Таким чином, дія команди зводиться до завантаження вмісту індексного регістра в лічильник команд.

На відміну від команди відносного переходу ця команда має менше обмежень за областю дії. Насправді, оскільки індексний регістр Z – 16-розрядний, максимально можлива величина переходу становить: 64 Кслів (128 Кбайт). Як і команда відносного переходу, команда непрямого переходу виконується за 2 машинних цикли.

Команди умовного переходу

В цих командах виконується перевірка умови, результат якої впливає на стан лічильника команд. Якщо умова істинна, відбувається перехід за вказаною адресою, якщо ж умова хибна, виконується наступна команда.

Команди умовного переходу мають обмеження за областю дії. Нове значення лічильника команд обчислюється додаванням до нього або відніманням від нього деякого зміщення. Оскільки під значення зміщення (число зі знаком) в слові

команди виділяється лише 7 біт, максимальна величина переходу відносно адреси наступної команди становить від -64 до +63 слів.

Оскільки перехід за вказаною адресою здійснюється завантаженням нового значення в лічильник команд, то у випадку істинності умови, що перевіряється, у конвеєрі виникає затримка тривалістю в один машинний цикл.

Всі команди умовного переходу можна розділити на дві підгрупи. Перша підгрупа – команди умовного переходу загального призначення. У цю підгрупу входять дві команди BRBS s, k і BRBC s, k, в яких явно вказується номер прапорця регістра SREG, який перевіряється. Відповідно, перехід здійснюється при SREG.s = 0 (BRBC) або SREG.s = 1 (BRBS). Іншу підгрупу складають 18 команд, кожна з яких виконує перехід за якою-небудь конкретною умовою: «дорівнює», «більше або дорівнює», «було перенесення» і т. ін. Одні з цих команд використовуються після порівняння беззнакових чисел, інші – після порівняння чисел зі знаком. Можливі умови, що перевіряються, а також відповідні їм команди умовного переходу наведено в таблиці 1.4.

Команди, які наведено в таблиці 1.4, є тільки еквівалентними мнемонічними позначеннями команд BRBS s, k та BRBC s, k з визначеними значеннями операнда – s. Наприклад, команда BREQ k має, такий же код операції, що і команда BRBS 1, k, а команда BRGE k – такий же, що і BRBC 4, k. За останньою командою відбувається перехід за значенням k, якщо прапорець S = 1. Значення S = (N+V), де N – значення прапорця від'ємного значення, а V – прапорця переповнення додаткового коду.

Команди виклику підпрограм

Для виклику підпрограм є дві команди: команда відносного виклику – RCALL і команда непрямого виклику – ICALL.

Враховуючі деякі відмінності, які описано нижче, команда RCALL працює так само, як і команда відносного безумовного переходу RJMP.

Команда RCALL зберігає у стеку значення лічильника команд (адресу наступної команди). Потім вміст лічильника команд збільшується або зменшується на деяке значення, що є операндом команди. Оскільки останній є 12-розрядним

числом зі знаком, максимальна величина переходу відносно адреси наступної команди становить від -2048 до +2047 слів (приблизно ±4 Кбайт).

Таблиця 1.4 – Зведена таблиця команд умовного переходу

Перевірка	**Логічна умова**	**Команда**	**Зворотна перевірка**	**Логічна умова**	**Команда**	**Тип даних**
Rd>Rr	(N⊕V) = 0	BRLT*	Rd≤Rr	(N⊕V)=1	BRGE*	Зі знаком
Rd≥Rr	(N⊕V) = 0	BRGE	Rd<Rr	(N⊕V)=1	BRLT	Зі знаком
Rd=Rr	Z = 1	BREQ	Rd≠Rr	Z=0	BRNE	Зі знаком
Rd≤Rr	(N⊕V) = 1	BRGE*	Rd>Rr	(N⊕V)=0	BRLT*	Зі знаком
Rd<Rr	(N⊕V) = 1	BRLT	Rd≥Rr	(N⊕V)=0	BRGE	Зі знаком
Rd>Rr	C = 0	BRLO*	Rd≤Rr	C=1	BRSH*	Без знака
Rd≥Rr	C = 0	BRSH/BRCC	Rd<Rr	C=1	BRLO/BRCS	Без знака
Rd=Rr	Z = 1	BREQ	Rd≠Rr	Z=0	BRNE	Без знака
Rd≤Rr	C = 1	BRSH*	Rd>Rr	C=0	BRLO*	Без знака
Rd<Rr	C = 1	BRLO/BRCS	Rd≥Rr	C=0	BRSH/BRCC	Без знака
«Перенесення»	C = 1	BRCS	«Немає перенесення»	C=0	BRCC	—
«Менше за нуль»	N = 1	BRMI	«Більше за нуль»	N=0	BRPL	—
«Переповнення»	V = 1	BRVS	«Немає переповнення»	V=0	BRVC	—
«Нуль»	Z = 1	BREQ	«Не нуль»	Z=0	BRNE	—
«Половинне перенесення»	H = 1	BRHS	«Немає половинного перенесення»	H=0	BRHC	—

* Оскільки у мікроконтролері відсутні команди переходів за умовами: більше та менше, або дорівнює, то для переходу за цими умовами операнди попередньої команди порівняння повинні бути записані у зворотному порядку, тобто замість CP Rd, Rr –> CP Rr, Rd.

У програмах в якості операндів команди RCALL, як і у випадку команди RJMP, використовуються мітки. Асемблер сам обчислює величину зміщення відносно адреси наступної команди шляхом віднімання від адреси мітки адреси наступної команди і підставляє це значення в машинний код команди.

Команда відносного виклику підпрограм виконується за 3 машинних цикли, два з яких витрачаються на збереження у стеку двох байт лічильника команд (адреси наступної команди).

Враховуючі деякі відмінності, які описано нижче, команда ICALL працює так само, як і команда непрямого безумовного переходу IJMP.

Команда ICALL зберігає у стеку значення лічильника команд (адресу наступної команди). Потім у лічильник команд завантажується вміст індексного

регістра. Оскільки індексний регістр – 16-розрядний, максимально можлива величина переходу становить 64 Кслів (128 Кбайт). Як і команда RCALL, команда непрямого виклику підпрограм виконується за 3 машинних цикли.

Підпрограма повинна закінчуватися командою повернення RET, як показано в наступному прикладі:

```
rcall   sp_test    ;  виклик підпрограми  sp_test
...                ;  текст основної програми
sp_test            ;  мітка підпрограми
push    r2         ;  збереження r2 у стеку
...                ;  виконання підпрограми
pop     r2         ;  відновити r2 зі стека
ret                ;  повернення з підпрограми
```

У наведеному вище прикладі команда ret замінює адресу, що міститься в лічильнику команд, адресою команди, наступної за командою rcall.

Команди повернення з підпрограм

В кінці кожної підпрограми обов’язково повинна міститися команда повернення з неї. В системі команд AVR-мікроконтролерів таких команд є дві. Для повернення з підпрограми, що викликається командами RCALL і ICALL, використовується команда RET. Для повернення з підпрограми обробки переривання використовується команда RETI.

Обидві команди відновлюють зі стека вміст лічильника команд, який зберігається там перед переходом до підпрограми. Команда повернення з підпрограми RETI додатково встановлює в одиницю прапорець глобального дозволу переривань I регістра SREG, що скидається апаратно при виникненні переривання.

На виконання кожної з команд повернення з підпрограми потрібно 4 машинних цикли.

Команди керування мікроконтролером

У цю групу входять 3 команди:

– NOP – пуста операція;

– SLEEP – переведення мікроконтролера в режим зниженого енергоспоживання;

– WDR – скидання вартового таймера.

Команди NOP і WDR виконуються за один машинний цикл, а команда SLEEP – за три машинних цикли.

Вище скорочено було розглянуто 118 базових команд. Реальна кількість команд більша, тому що за деякими номерами наведено опис команд, які виконуються за схожими алгоритмами, але відрізняються способами адресації операндів. Детальний опис базових команд наведено у [1; 2; 10].

1.4.8 Нові команди AVR-мікроконтролерів

Архітектура AVR-мікроконтролерів постійно оновлялася. Це оновлення стосується як структури мікроконтролера, так і його системи команд. Кожне сімейство, послідовно успадковує набір команд попереднього сімейства. Але є деякі виключення.

Нижче розглядаються нові команди, які на даний час з'явилися у нових AVR-мікроконтролерах: Mega та XMega (таблиця 1.5) [1; 3].

JMP

Код операції:

15														1	0
1	0	0	1	0	1	0	k	k	k	k	k	1	1	0	k
31														17	16
k	k	k	k	k	k	k	k	k	k	k	k	k	k	k	k

Алгоритм виконання:

$$PC \leftarrow k.$$

Опис: для мікроконтролерів з 22-розрядним програмним лічильником (PC), максимальна ємність пам'яті програм дорівнює 2^{22} = 4 Мслів (8 Мбайт), наприклад у контролері AT90SC6464C-USB, а для контролерів з 16-розрядним PC, відповідно 2^{16} = 64 Кслів (128 Кбайт).

Довжина команди: 2 слова (4 байти).

Таблиця 1.5 – Нові команди мікроконтролерів AVR

Команда	Операція	Опис	Дія	Прапорці	Такти
Classic Core AVR (до 128К програмного простору)					
JMP	k	Безумовний прямий перехід у пам'яті програм ємністю 64 К/4 Мслів	PC ← k (16/22 біти)	–	3/4
CALL	k	Безумовний прямий виклик підпрограми із пам'яті програм ємністю 64 К/4 Мслів	STACK ← PC + 2 SP ← SP - 2/SP - 3 PC ← k (16/22 біти)	–	4/5
Enhanced Core AVR (до 8К програмного простору)					
MUL	Rd, Rr	Множення беззнакових чисел	R1:R0 ← RdxRr	Z, C	2
MULS	Rd, Rr	Множення знакових чисел	R1:R0 ← RdxRr	Z, C	2
MULSU	Rd, Rr	Множення знакового числа на беззнакове	R1:R0 ← RdxRr	Z, C	2
FMUL	Rd, Rr	Множення дробових беззнакових чисел	R1:R0←RdxRr	Z, C	2
FMULS	Rd, Rr	Множення дробових знакових чисел	R1:R0 ← RdxRr	Z, C	2
FMULSU	Rd, Rr	Множення дробового знакового числа на беззнакове	R1:R0 ← RdxRr	Z, C	2
MOVW	Rd, Rr	Копіювання слова	Rd+1:Rd ← Rr+1:Rr	–	1
LPM Rd, Z	Rd	Завантаження регістра Rd із пам'яті програм з адресою у індексному регістрі Z	Rd ←(Z)	–	3
LPM Rd, Z+	Rd	Завантаження регістра Rd із пам'яті програм з адресою у індексному регістрі Z та наступним інкрементом вмісту Z	Rd ← (Z) Z ← Z + 1	–	3

Продовження таблиці 1.5

Команда	Операція	Опис	Дія	Прапорці	Такти
SPM		Використовується для самопрограмування мікроконтролера, який має відповідний блок	Дивись опис команди, наведений нижче	–	Виконується із пам'яті завантажувача
Enhanced Core AVR (до 128 Кслів програмного простору)					
BREAK		Зупинка процесора після виконання команди	PC ← PC + 1 зупинка виконання програми	–	1
Enhanced Core AVR (до 4 Мслів програмного простору)					
EIJMP		Безумовний непрямий перехід у пам'яті програм ємністю 4 Мслів	PC← (EIND:Z)	–	2
EICALL		Безумовний непрямий виклик підпрограми із пам'яті програм ємністю 4 Мслів	STACK← PC + 1 SP ← SP-3 PC ← (EIND:Z)	–	4
ELPM		Завантаження регістра R0 із розширеної пам'яті програм, з адресою у регістрах RAMPZ та Z	R0 ← (RAMPZ:Z)	–	3
ELPM Rd, Z	Rd	Завантаження регістра Rd із розширеної пам'яті програм, з адресою у регістрах RAMPZ та Z	Rd ← (RAMPZ:Z)	–	3
ELPM Rd, Z+	Rd	Завантаження регістра Rd із розширеної пам'яті програм, з адресою у регістрах RAMPZ та Z, та наступним інкрементом вмісту RAMPZ:Z	Rd ← (RAMPZ:Z) RAMPZ:Z ← RAMPZ:Z+1	–	3

Закінчення таблиці. 1.5

Команда	Операція	Опис	Дія	Прапорці	Такти
XMEGA CoreAVR					
DES	K	Шифрування даних	якщо H = 0: R15:R0←ENCRYPT (R15:R0, K) якщо H = 1: R15:R0←DECRYPT (R15:R0, K)	–	1/2
LAC Z, Rd	Rd	Запис (Z) у Rd та очищення бітів (Z) за маскою, яку вказано у Rd	Temp ← Rd; Rd ← (Z); (Z) ← ($FF – Temp)*(Z); PC ← PC + 1	–	1
LAS Z, Rd	Rd	Запис (Z) у Rd та встановлення бітів (Z) за маскою, яку вказано в Rd	Temp ← Rd; Rd ← (Z); (Z) ← Temp v (Z); PC ← PC + 1	–	1
LAT Z, Rd	Rd	Запис (Z) у Rd та підсумовування за модулем 2 бітів (Z) з маскою, яку вказано в Rd	Temp ← Rd; Rd ← (Z); (Z) ← Temp xor (Z); PC ← PC + 1	–	1
XCH Z, Rd	Rd	Обмін даними між Rd та (Z)	Temp ← Rd; Rd ← (Z); (Z) ← Temp; PC ←PC+1	–	1

CALL

Код операції:

15														1	0
1	0	0	1	0	1	0	k	k	k	k	k	1	1	1	k
31															16
k	k	k	k	k	k	k	k	k	k	k	k	k	k	k	k

Алгоритм виконання:

STACK ← PC + 2
SP ← SP - 2
PC ← k, якщо k = 16 біт

} Безумовний прямий виклик підпрограми із пам'яті програм ємністю 2^{16} = 64К слів (128 Кбайт)

STACK ← PC + 2
SP ← SP - 3
PC ← k, якщо k = 22 біт

} Безумовний прямий виклик підпрограми із пам'яті програм ємністю 2^{22} = 4М слів (8 Мбайт)

Довжина команди: 2 слова (4 байти).

MUL

Код операції:

15							8	7							0
1	0	0	1	1	1	r	d	d	d	d	d	r	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr

PC ← PC + 1.

Операнди: $0 \le d \le 31$; $0 \le r \le 31$ – беззнакові величини.

Довжина команди: 1 слово (2 байти).

MULS

Код операції:

15							8	7							0
0	0	0	0	0	0	1	0	d	d	d	d	r	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr

PC ← PC + 1.

Операнди: $16 \le d \le 31$; $16 \le r \le 31$ – знакові величини.

Довжина команди: 1 слово (2 байти).

MULSU

Код операції:

15							8	7							0
1	0	0	1	0	0	1	1	0	d	d	d	0	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr

PC ← PC + 1.

Операнди: 16 ≤ d ≤ 23; 16 ≤ r ≤ 23 (у Rr – беззнакове число, Rd – знакове число)

Довжина команди: 1 слово (2 байти).

FMUL

Код операції:

15							8	7							0
0	0	0	0	0	0	1	1	0	d	d	d	0	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr

PC ← PC + 1.

Опис: числа подаються у вигляді N.Q, де N – значення числа до коми у двійковому вигляді, Q – значення числа після коми у двійковому вигляді. Таким чином операнди подаються в вигляді (N1.Q1) і (N2.Q2). Беззнакові 8-розрядні дробові числа використовують формат, в якому числа можуть лежати у діапазоні: [0, 2]. Біти 6…0 являють собою дробову частину, а 7-й біт – цілу частину (0 або 1), тобто 1.7 формат. Результат множення (формат результату): 2.14 зсувається вліво на один розряд для приведення до формату: 1.15 та записується у пару регістрів R1:R0.

Операнди: 16 ≤ d ≤ 23; 16 ≤ r ≤ 23 – беззнакові дробові величини.

Довжина команди: 1 слово (2 байти).

FMULS

Код операції:

15							8	7							0
0	0	0	0	0	0	1	1	1	d	d	d	0	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr

PC ← PC + 1.

Опис: числа подаються в вигляді N.Q, де N – значення числа до коми у двійковому вигляді, Q – значення числа після коми у двійковому вигляді. Таким чином операнди подаються в вигляді (N1.Q1) і (N2.Q2). Дробові числа зі знаком подібно до цілих чисел зі знаком використовують формат двійкового доповнення, що дозволяє подавати перші у діапазоні: [-1, 1]. Якщо, наприклад, ціле число без знака має двійковий код: 10110010, то його десятковий еквівалент дорівнює: 128 + 32 + 16 + 2 = 178. Відповідно, десятковий еквівалент цього числа як знакового цілого дорівнює: 178 - 256 = -78. Якщо наведене вище двійкове число подати у форматі беззнакового дробового, то отримаємо:

1 + 0,25 + 0,125 + 0,015625 = 1,390625.

Аналогічно наведеному вище правилу обчислення цілих чисел зі знаком можна отримати десятковий еквівалент двійкового числа 10110010, якщо вважати що воно відповідає дробовому числу зі знаком: 1,390625 - 2 = -0,609375.

Результат множення (формат результату): 2.14 зсувається вліво на один розряд для приведення до формату: 1.15 та записується у пару регістрів R1:R0.

Операнди: $16 \le d \le 23$; $16 \le r \le 23$ – дробові величини зі знаком.

Довжина команди: 1 слово (2 байти).

FMULSU

Код операції:

15							8	7							0
0	0	0	0	0	0	1	1	1	d	d	d	1	r	r	r

Алгоритм виконання:

R1:R0 ← RdxRr

PC ← PC + 1.

Опис: числа подаються в вигляді N.Q, де N – значення числа до коми у двійковому вигляді, Q – значення числа після коми у двійковому вигляді. Таким чином операнди подаються в вигляді: N1.Q1 і N2.Q2 (дивись опис команди FMULS).

Операнди: $16 \le d \le 23$; $16 \le r \le 23$ (обидві величини дробові, одна з яких, у Rr – беззнакова, друга величина, у Rd – знакова).

Довжина команди: 1 слово (2 байти).

MOVW

Код операції:

15							8	7							0
0	0	0	0	0	0	0	1	d	d	d	d	r	r	r	r

Алгоритм виконання:

Rd+1:Rd ← Rr+1:Rr

PC ← PC + 1.

Опис: команда копіює вміст однієї пари регістрів загального призначення в іншу пару регістрів загального призначення. Пара регістрів джерела (Rr+1:Rr) залишається незмінною. Пара регістрів приймача (Rd+1:Rd) завантажується копією регістрів Rr+1:Rr. Номери регістрів Rr, Rd повинні бути парними.

Операнди: $0 \le d \le 30$; $0 \le r \le 30$.

Довжина команди: 1 слово (2 байти).

LPM Rd, Z

Код операції:

15							8	7							0
1	0	0	1	0	0	0	d	d	d	d	d	0	1	0	0

Алгоритм виконання:

Rd ← ПП (Z)

PC ← PC + 1.

Операнди: 0 ≤ d ≤ 31, ПП (Z) – комірка пам’яті програм, адреса якої міститься в індексному регістрі Z.

Довжина команди: 1 слово (2 байти).

LPM Rd, Z+

Код операції:

15							8	7							0
1	0	0	1	0	0	0	d	d	d	d	d	0	1	0	1

Алгоритм виконання:

Rd ← ПП (Z), Z ← Z + 1

PC ← PC + 1.

Операнди: 0 ≤ d ≤ 31, ПП (Z) – комірка пам’яті програм, адреса якої міститься в індексному регістрі Z.

Довжина команди: 1 слово (2 байти).

SPM

Код операції:

15							8	7							0
1	0	0	1	0	1	0	1	1	1	1	0	1	0	0	0

Опис: ця команда використовується для самопрограмування мікроконтролера, який має відповідний блок. Команда зберігається, і відповідно виконується, у спеціальній області пам’яті, яка називається завантажувачем. Існує п’ять режимів використання команди, які програмуються за допомогою додаткових регістрів керування:

(RAMPZ : Z) ← $FFFF, очищення сторінки пам’яті програм;

(RAMPZ : Z)← R1:R0, запис слова у тимчасовий буфер пам’яті програм;

(RAMPZ : Z) ← R1:R0, запис сторінки у тимчасовий буфер пам’яті програм;

(RAMPZ : Z)← TEMP, запис тимчасового буфера у пам’ять програм;

BLBITS ← R1:R0, встановлення бітів блокування завантаження програм.

Більш детальний опис цієї команди наведено у [1; 2].

BREAK

Код операції:

15 8 7 0

1	0	0	1	0	1	0	1	1	0	0	1	1	0	0	0

Алгоритм виконання:

PC ← PC+1.

Опис: ця інструкція використовується налагоджувачем, який вбудовано у мікроконтролер, та зазвичай в робочих програмах не використовується. Після виконання команди BREAK процесор зупиняється, що дає можливість налагоджувачеві працювати із внутрішніми ресурсами.

Операнди: немає.

Довжина команди: 1 слово (2 байти).

EIJMP

Код операції:

15 8 7 0

1	0	0	1	0	1	0	0	0	0	0	1	1	0	0	1

Алгоритм виконання:

PC (15:0) ← Z (15:0)

PC (21:16) ← EIND (6 біт),

де EIND – ім’я додаткового регістра керування.

Довжина команди: 1 слово (2 байти).

EICALL

Код операції:

15 8 7 0

1	0	0	1	0	1	0	1	0	0	0	1	1	0	0	1

Алгоритм виконання:

STACK ← PC + 1

SP ← SP - 3

PC (15:0) ← Z(15:0)

PC (21:16) ← EIND (6 біт),

де EIND – ім’я додаткового регістра керування.

Довжина команди: 1 слово (2 байти).

ELPM

Код операції:

15 8 7 0

1	0	0	1	0	1	0	1	1	1	0	1	1	0	0	0

Алгоритм виконання:

R0 ← ПП (RAMPZ:Z),

де RAMPZ – ім’я додаткового регістра керування;

PC ← PC + 1.

Операнди: R0 – регістр загального призначення; (RAMPZ:Z) – комірка пам’яті програм, адреса якої міститься у двох регістрах: RAMPZ (6 біт) та Z – 16 біт.

Довжина команди: 1 слово (2 байти).

ELPM Rd, Z

Код операції:

15 8 7 0

1	0	0	1	0	0	0	d	d	d	d	d	0	1	1	0

Алгоритм виконання:

Rd ← ПП (RAMPZ:Z),

де RAMPZ – ім’я додаткового регістра керування;

PC ← PC + 1.

Операнди: Rd ($0 \le d \le 31$) – регістр загального призначення;

ПП (RAMPZ:Z) – комірка пам’яті програм, адреса якої міститься у двох регістрах: RAMPZ (6 біт) та Z – 16 біт.

Довжина команди: 1 слово (2 байти).

ELPM Rd, Z+

Код операції:

15 8 7 0

1	0	0	1	0	0	0	d	d	d	d	d	0	1	1	1

Алгоритм виконання:

Rd ← ПП (RAMPZ:Z), RAMPZ:Z ← RAMPZ:Z+1,

де RAMPZ – ім’я додаткового регістра керування;

PC ← PC + 1.

Операнди: Rd (0≤d≤31) – регістр загального призначення;

ПП (RAMPZ:Z) – комірка пам’яті програм, адреса якої міститься у двох регістрах: RAMPZ (6 біт) та Z – 16 біт.

Довжина команди: 1 слово (2 байти).

DES

Алгоритм виконання:

R15:R0 ← ENCRYPT(R15:R0, K), якщо прапорець H = 0,

R15:R0 ← DECRYPT(R15:R0, K), якщо прапорець H = 1.

Опис: команда DES використовується в сучасних мікроконтролерах сімейства XMega для шифрування інформації. Більш детальну інформацію про команду та метод кодування, наведено у додатковій літературі [1; 19].

LAC

Код операції:

15 8 7 0

1	0	0	1	0	0	1	d	d	d	d	d	0	1	1	0

Алгоритм виконання:

Temp ← Rd;

Rd ← (Z);

(Z) ← ($FF – Temp) * (Z);

PC ← PC + 1.

Опис: команда LAC записує Rd у тимчасовий регістр Temp, пересилає (Z) в Rd, інвертоване значення Temp множить на (Z) та результат записує в (Z).

Таким чином до виконання команди в Rd знаходиться маска, за якою необхідно скинути у нуль відповідні біти (Z), а після виконання команди результат записується у (Z) з обнуленими бітами на тих місцях, де у масці знаходилась одиниця. При виконанні команди початкове значення (Z) записується у Rd.

Операнди: $0 \le d \le 31$.

Довжина команди: 1 слово (2 байти).

LAS

Код операції:

15 8 7 0

1	0	0	1	0	0	1	d	d	d	d	d	0	1	0	1

Алгоритм виконання:

Temp ← Rd;

Rd ← (Z);

(Z) ← Temp v (Z);

PC ← PC + 1.

Опис: команда LAS записує Rd у тимчасовий регістр Temp, пересилає (Z) в Rd, логічно підсумовує Temp і (Z) та результат записує в (Z). Таким чином до виконання команди в Rd знаходиться маска, за якою необхідно встановити в одиницю відповідні біти (Z), а після виконання команди результат записується у (Z) з встановленими бітами на тих місцях, де у масці знаходилась одиниця. При виконання команди початкове значення (Z) записується у Rd.

Операнди: $0 \le d \le 31$.

Довжина команди: 1 слово (2 байти).

LAT

Код операції:

15 8 7 0

1	0	0	1	0	0	1	d	d	d	d	d	0	1	1	1

Алгоритм виконання:

Temp ← Rd;

Rd ← (Z);

(Z) ← Temp xor (Z);

PC ← PC + 1.

Опис: команда LAT записує Rd у тимчасовий регістр Temp, пересилає (Z) в Rd, виконує логічне «Виключне АБО» (XOR) між Temp і (Z) та результат записує в (Z). Таким чином до виконання команди в Rd знаходиться маска, за якою необхідно проінвертувати відповідні біти (Z), а після виконання команди результат записується у (Z) з проінвертованими бітами на тих місцях, де у масці знаходилась одиниця. При виконання команди початкове значення (Z) записується у Rd.

Операнди: $0 \leq d \leq 31$.

Довжина команди: 1 слово (2 байти).

XCH

Код операції:

15 8 7 0

1	0	0	1	0	0	0	d	d	d	d	d	0	1	0	0

Алгоритм виконання:

Temp ← Rd;

Rd ← (Z);

(Z) ← Temp;

PC ←PC+1.

Опис: команда XCH міняє місцями значення, що знаходиться за адресою, вказаною в Z, зі значенням Rd.

Операнди: $0 \leq d \leq 31$.

Довжина команди: 1 слово (2 байти).

Контрольні запитання та завдання

1. Назвіть розрядність однієї комірки пам’яті програм мікроконтролера AVR.
2. Опишіть організацію та призначення пам’яті програм.
3. На які три частини розділено пам’ять даних мікроконтролерів сімейства? Опишіть призначення та структуру кожного з них.
4. Які види стека ви знаєте? Який вид стека використовується у мікроконтролерах і де він розміщується?

5. За якою архітектурою виконано організацію пам’яті AVR-мікроконтролерів сімейства Mega?
6. Наведіть структурну схему карти пам’яті AVR-мікроконтролерів сімейства Mega. Опишіть структуру і призначення кожного виду пам’яті.
7. Опишіть структуру регістрового файлу.
8. Де в AVR-мікроконтролерах розташовуються регістри введення/виведення? Який з них використовується найчастіше?
9. Що таке енергонезалежна пам’ять даних (EEPROM)? Які регістри використовуються для керування EEPROM?
10. Назвіть команди, які можна використовувати для звернення до регістрів введення/виведення СПД.
11. Назвіть команди, які можна використовувати для звернення до тридцятидвох регістрів загального призначення СПД.
12. Опишіть формат та призначення окремих розрядів рогістра SREG.
13. Що означає символ «$» при наведенні адрес пам’яті?
14. Для чого використовується лічильник команд (Program Counter – PC) та від чого залежить його розмір?
15. Що знаходиться за адресою $0000 в пам’яті програм?
16. Які регістри розташовуюься в областях регістрів введення/виведення?
17. Скільки байт виділяється для додаткових регістрів введення-виведення в просторі статичної пам’яті даних?
18. Поясніть призначення 16-бітних регістрів: X, Y та Z.
19. Від чого залежить розмір регістра-покажчик стека?
20. Чому при вказівці адрес РВВ деякі з них вказуються в дужках?
21. Назвіть основні етапи створення керуючої програми для МК.
22. Назвіть складові програмної моделі типового AVR-мікроконтролера.
23. Скільки РЗП має типовий AVR-мікроконтролер?
24. Назвіть призначення лічильника команд.
25. Надайте характеристику командам типу «перевірка/пропуск».
26. Назвіть та надайте характеристику командам виклику та повернення з підпрограм.
27. Назвіть способи адресації операндів.
28. Які РЗП використовуються для непрямої адресації?
29. Дайте визначення мнемоніки команди та мнемокоду.
30. Дайте визначення коду операції команди та машинного коду команди.
31. Скільки типів команд має більшість мікроконтролерів сімейства AVR? Наведіть коротку характеристику цих типів.
32. Наведіть типи (формати) даних мікроконтролерів AVR.

33. Назвіть довжину команд у байтах та опишіть їх розміщення у пам’яті програм.
34. Опишіть особливості RISC-архітектури мікроконтролерів AVR.
35. Назвіть основні групи команд з базового набору мікроконтролерів AVR. Дайте коротку характеристику кожній з груп.
36. Назвіть та дайте характеристику новим командам.
37. Назвіть команди, які можуть використовуватись для виконання «булевих» операцій. Наведіть приклади їх використання.
38. Чим відрізняється алгоритми виконання команд арифметичних та логічних зсувів? Наведіть приклади їх використання.
39. Який діапазон десяткових чисел відображає восьмирозрядне число зі знаком?
40. Чим відрізняються мнемоніка та мнемокод команд?
41. Для чого використовується регістр PC?
42. Опишіть роботу конвеєра при виконанні програми.

2 ОПИС НАЛАГОДЖУВАЧА AVR-STUDIO 4

2.1 Інсталяція

Завантажити AVR Studio 4 можна на сайті atmel.com на відповідній сторінці (http://www.atmel.com/tools/atmelstudio.aspx). Нижче розглянуто повну інсталяцію всіх необхідних продуктів, яку зображено на рисунках 2.1…2.6.

Після запуску інсталятора з’явиться наведене нижче вікно (рисунок 2.1).

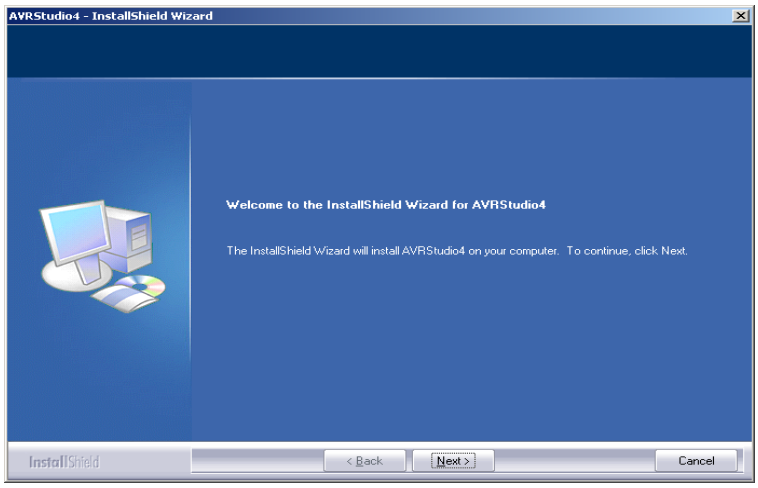


Рисунок 2.1 – Вікно привітання

Далі натисніть Next та прийміть умови лінцензійної угоди (рисунок 2.2).

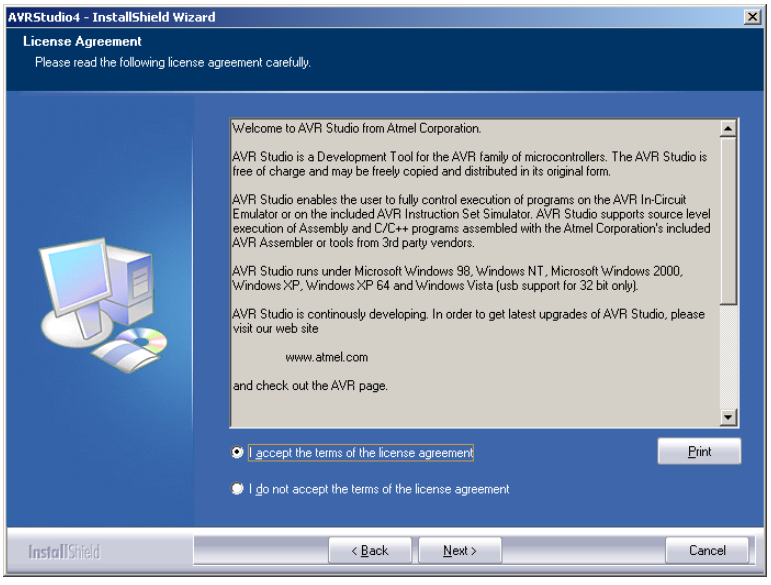


Рисунок 2.2 – Умови ліцензійної угоди

Після цього оберіть місце встановлення (рисунок 2.3).

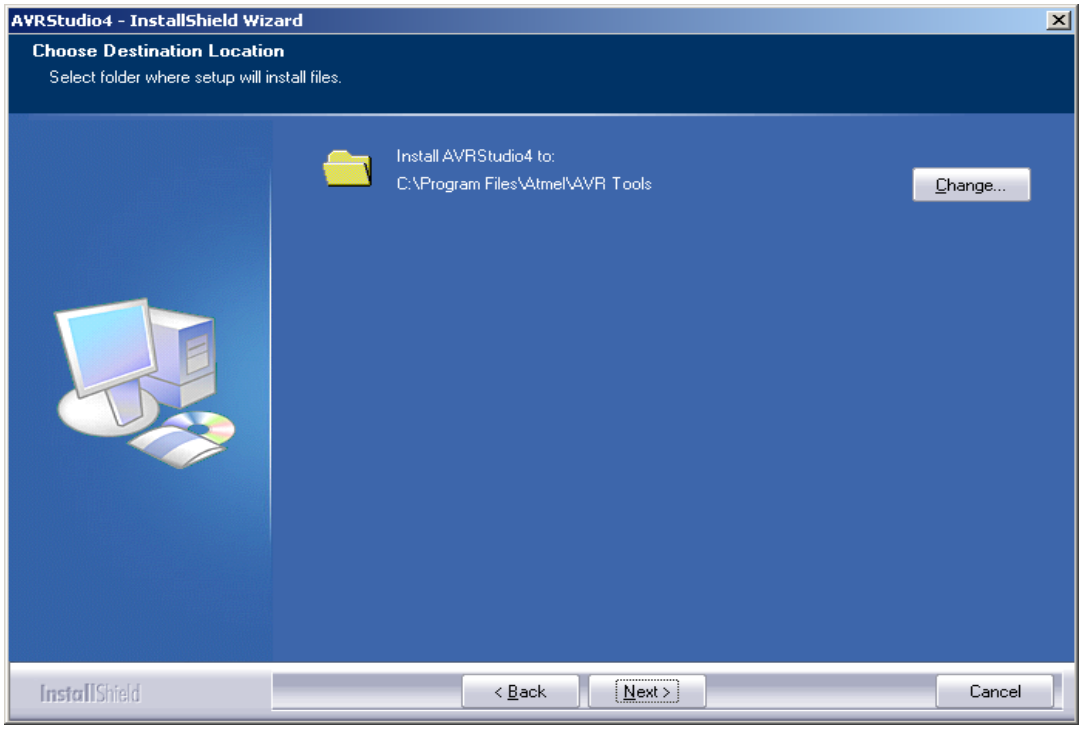


Рисунок 2.3 – Вибір місця інсталяції

Далі оберіть додаткові компоненти для встановлення (рисунок 2.4).

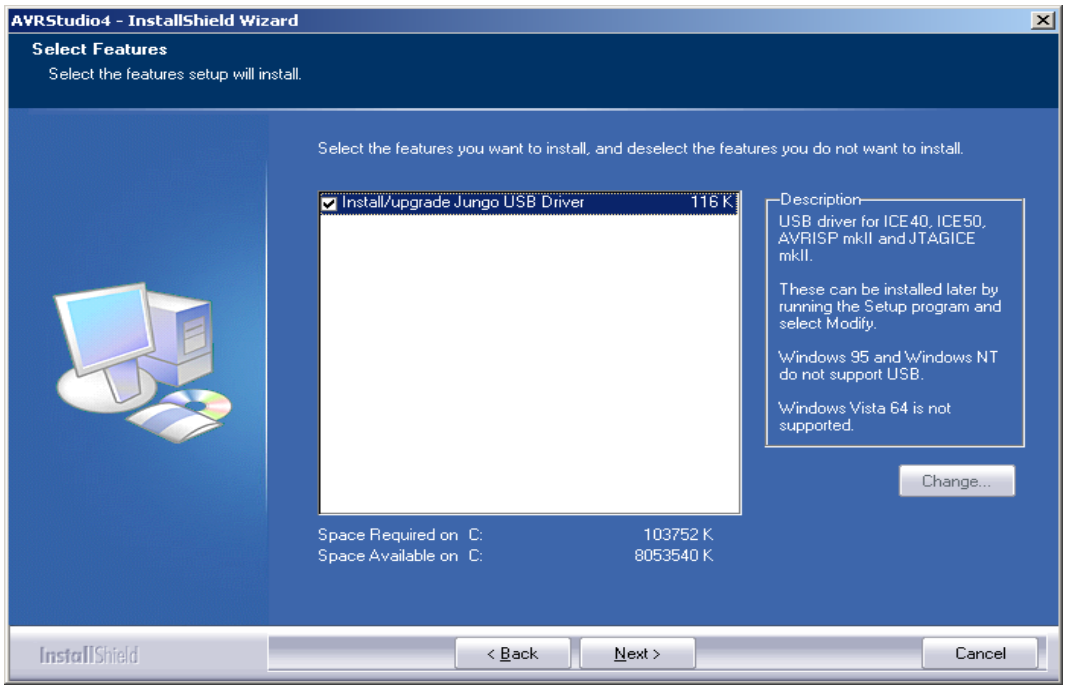


Рисунок 2.4 – Вибір додаткових компонентів

Натиснувши кнопку Install розпочніть установку (рисунок 2.5).

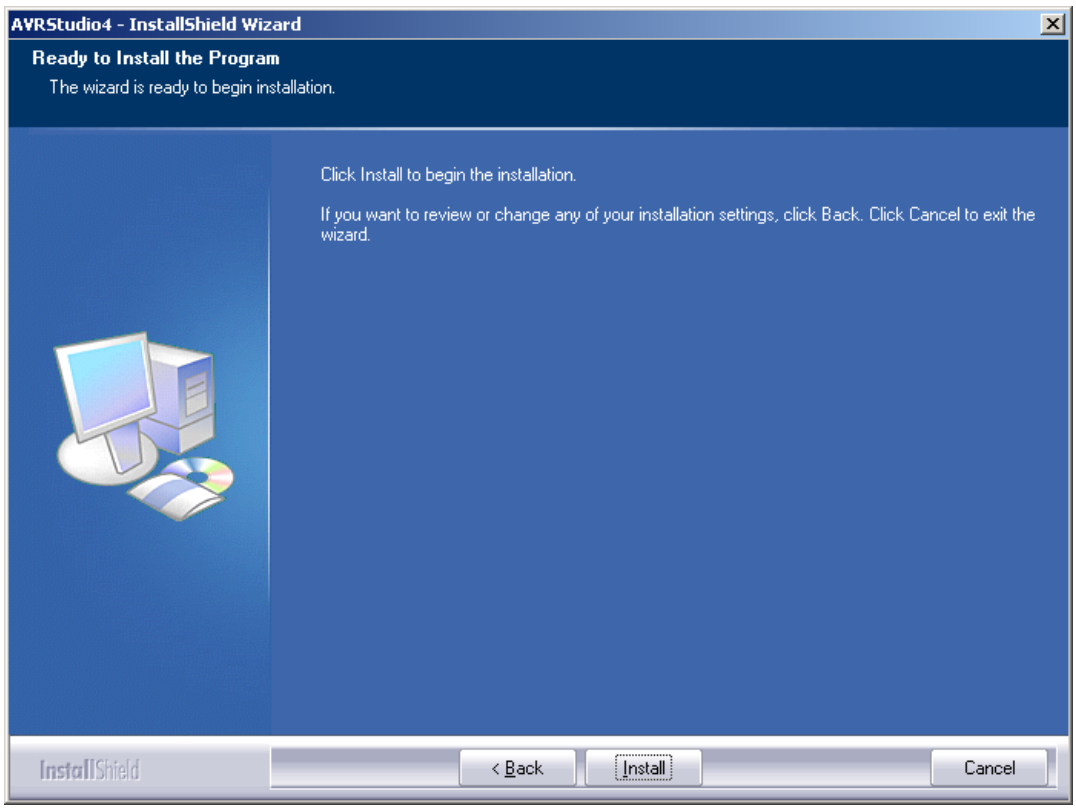


Рисунок 2.5 – Початок інсталяції

AVR Studio 4 встановлено. Натисніть Finish (рисунок 2.6).

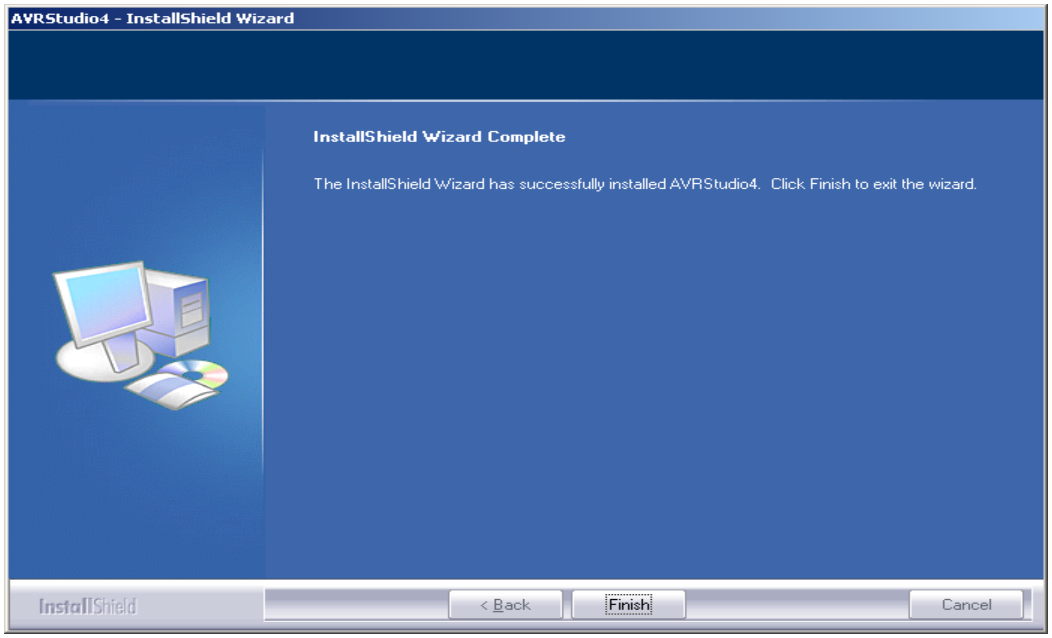


Рисунок 2.6 – Успішне закінчення інсталяції

2.2 Створення та завантаження проекту

Створити новий проект можна за допомогою меню Project → New Project.

У вікні, що з’явилось (рисунок 2.7), оберіть Atmel AVR Assembler, введіть назву та розташування проекту.

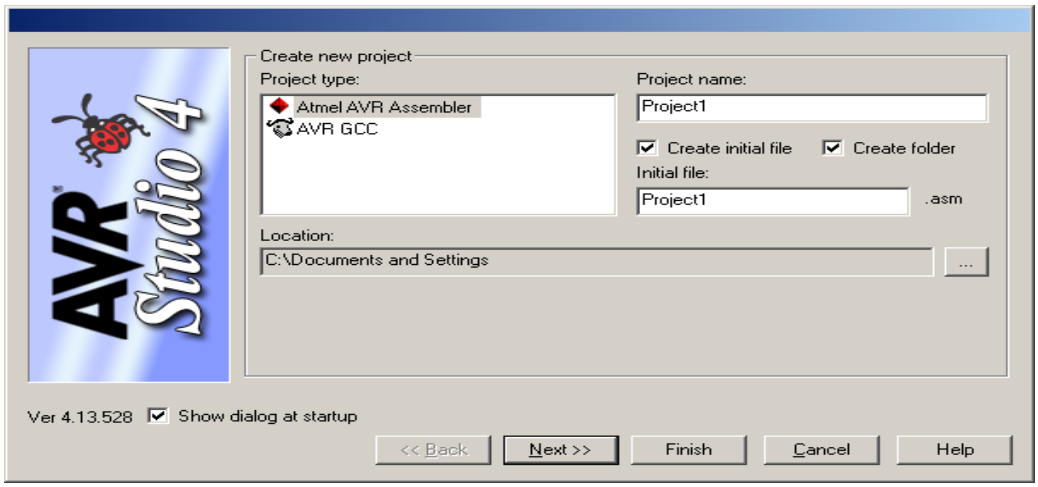


Рисунок 2.7 – Створення проекту

Натисніть Next.

У вікні «Select debug platform and device» (рисунок 2.8) оберіть Debug platform: AVR Simulator та Device, наприклад, ATmega8515.

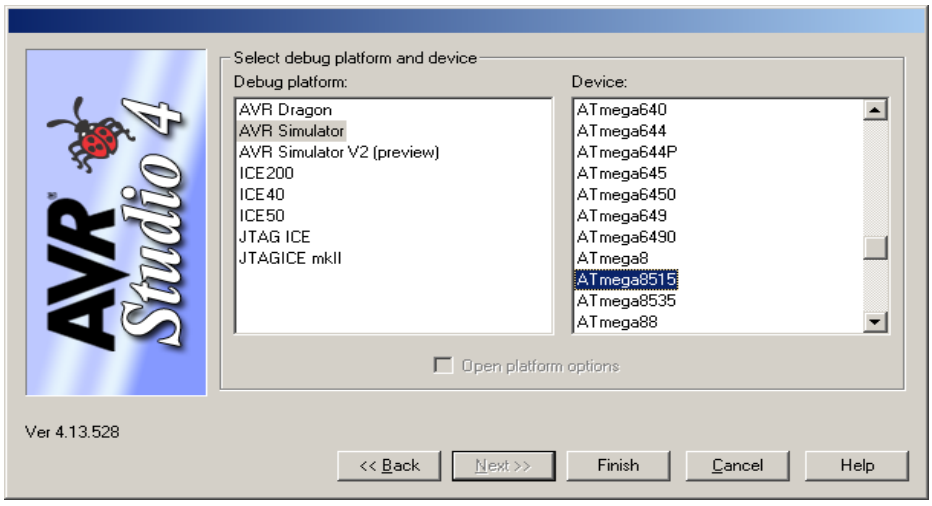


Рисунок 2.8 – Вибір платформи та пристрою

Натисніть Finish.

Для відкриття вікна для запису нового проекту скористуйтеся меню Project → Open Project (рисунок 2.9).

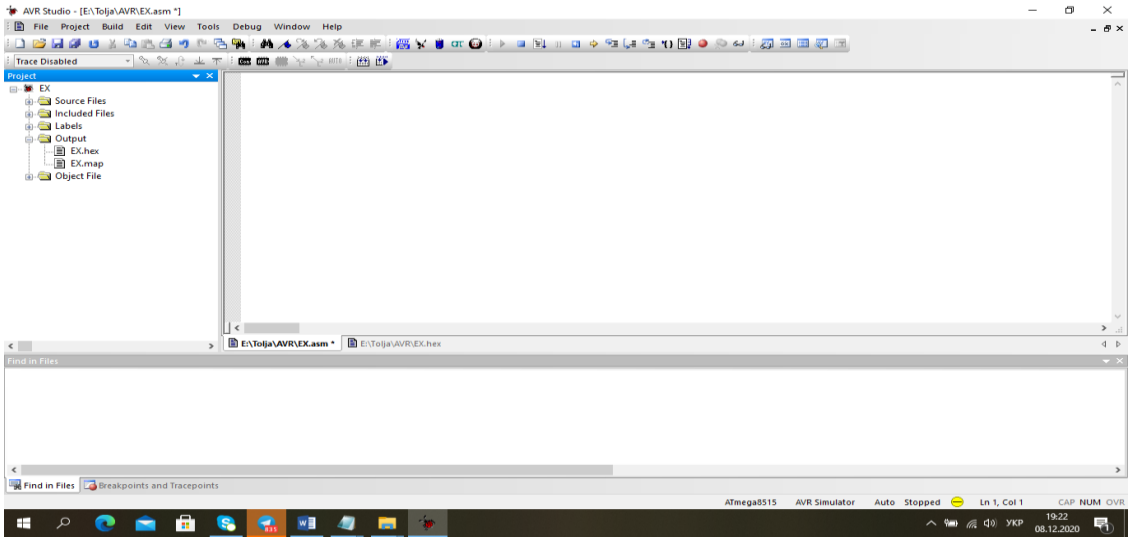


Рисунок 2.9 – Вікно для запису нового проекту

Для запису програми на асемблері треба відкрити папку Source Files. Вікривається пусте вікно для файлу *.asm (в нашему випадку ex.asm) (рисунок 2.10).

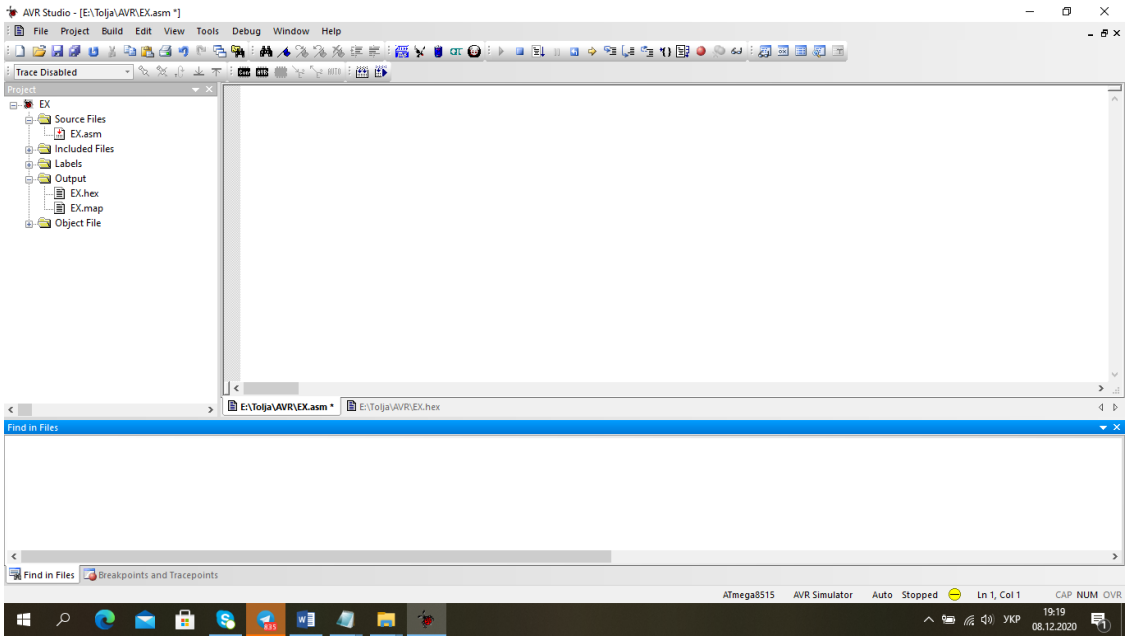


Рисунок 2.10 – Пусте вікно для запису файла *.asm

В це вікно записуємо програму на асемблері (рисунок 2.11).

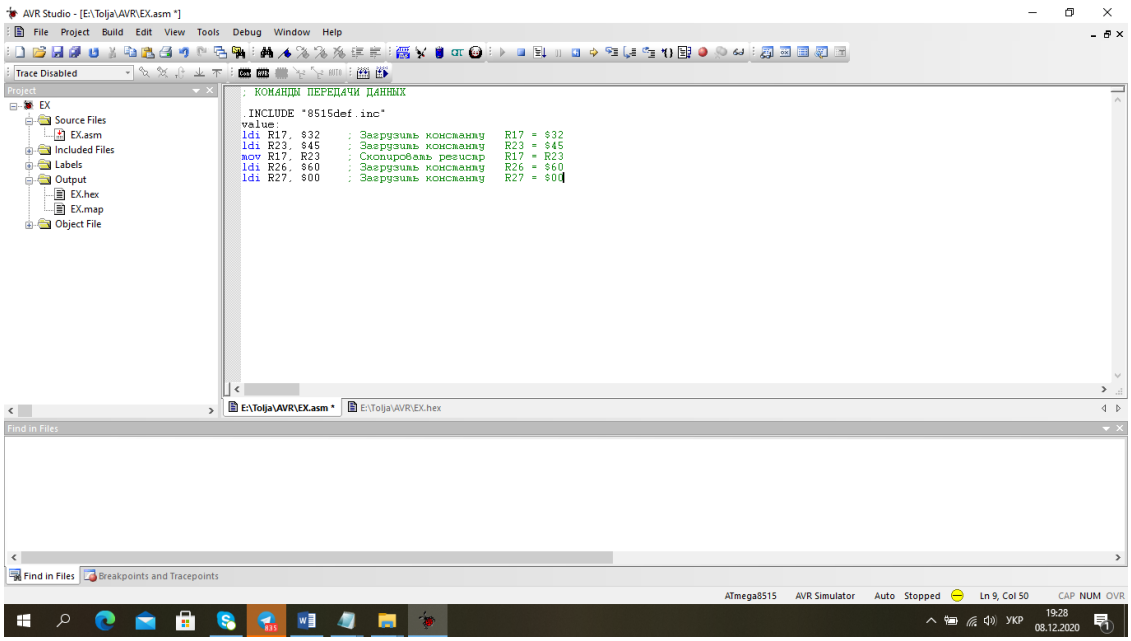


Рисунок 2.11 – Вікно с текстом файла *.asm

2.3 Запуск проекту

Для запуску проекту існують дві команди: Assemble та Assemble and run 2. Команди доступі з панелі інструментів або з відповідного меню (Build → Build та Build → Build and run) (рисунок 2.12):

Assemble – асемблерування створеного файлу;

Assemble and run – асемблерування файлу та запуск налагоджування, якщо операція асемблерування була успішною.

Рисунок 2.12 – Панель запуску проекту

Якщо операція була успішною, в нижній частині екрану у вікні Build з’явиться повідомлення про завершення асемблерування, відсутність помилок та попереджень (рисунок 2.13). В іншому випадку буде вказана кількість помилок та їх місце знаходження.

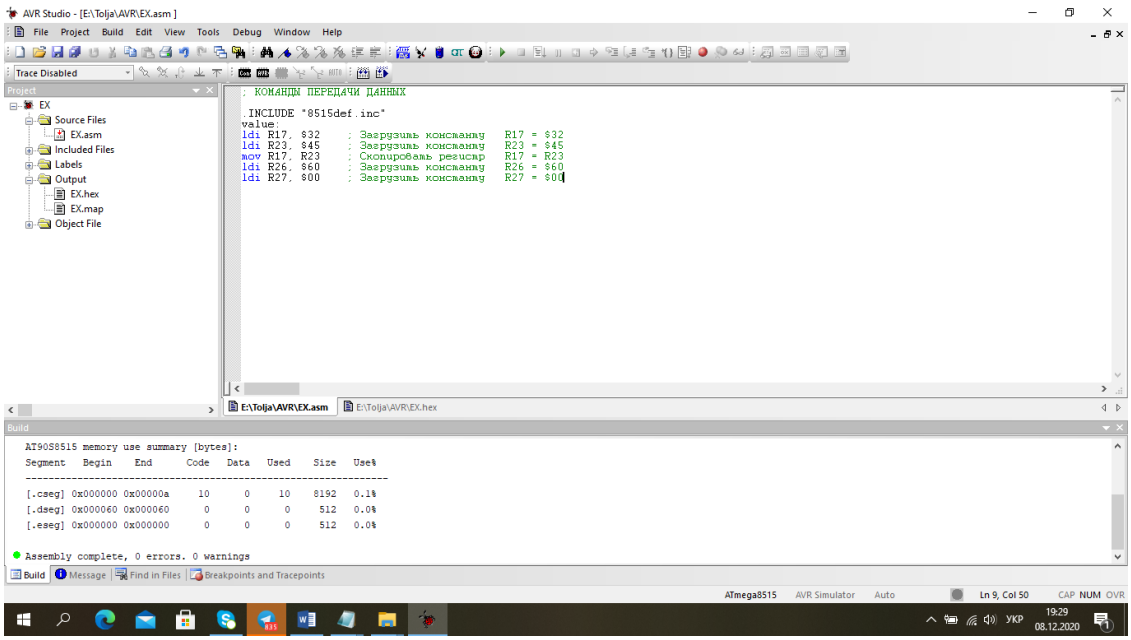


Рисунок 2.13 – Вікно повідомлення про завершення асемблерування та відсутність помилок та попереджень

2.4 Перегляд створеного hex-файлу

Увійдіть до меню Project → Assembler Options. Переконайтесь, що у вас обрано вікно налаштування параметрів асемблерування за замовчуванням (рисунок 2.14).

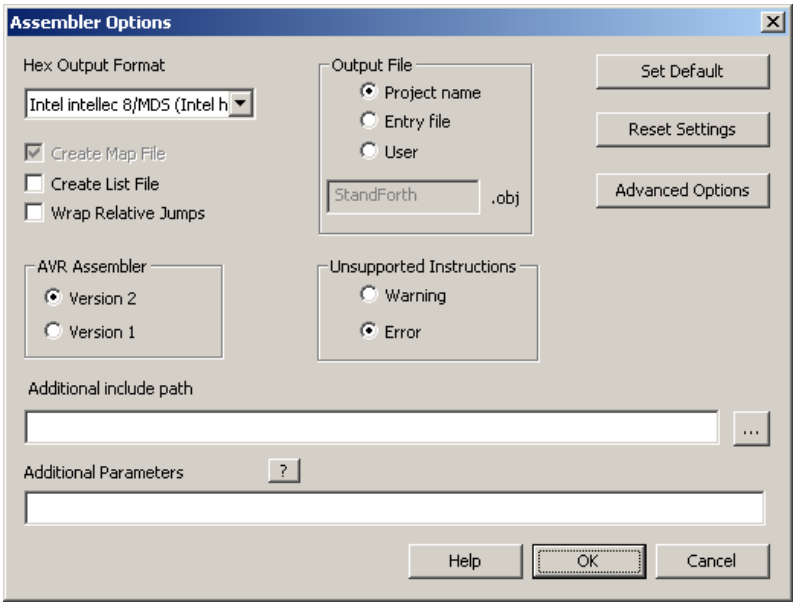


Рисунок 2.14 – Вікно параметрів асемблерування

Після запуску проекту створений hex-файл можна переглянути у вікні Project, в теці Output (рисунок 2.15).

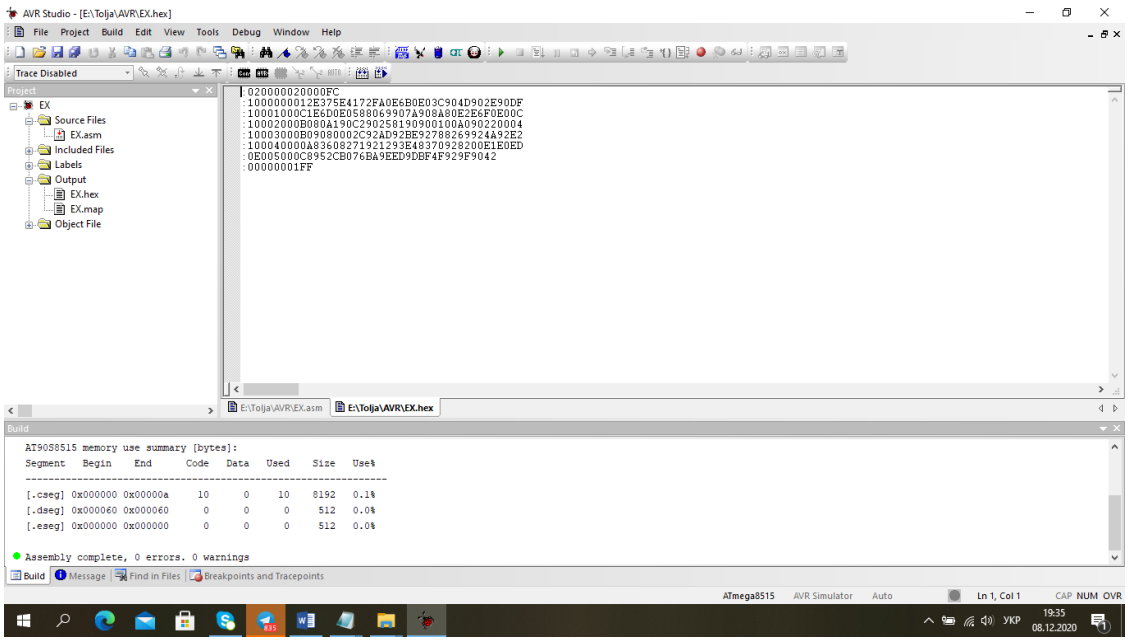


Рисунок 2.15 – Вікно з створеним hex-файлом: *.hex

2.5 Налагоджування програми

2.5.1 Запуск та зупинка програми

Для запуску та зупинки налагоджування скористайтесь відповідними кнопками панелі (рисунок 2.16) або за допомогою меню Debug.

Рисунок 2.16 – Панель налагоджування

Start Debugging розпочинає процес налагоджування, при якому доступні всі можливі команди керування. Зазвичай під час налагоджування зміна коду неможлива. Відбувається підключення платформи налагоджування, завантаження object-файлу.

Stop Debugging зупиняє процес налагоджування, відбувається відключення платформи налагоджування, стає доступним редагування коду.

Run розпочинає чи продовжує виконання програми. Виконання буде проводитись доки воно не буде зупинене користувачем або не зустрінеться точка зупину.

Break зупиняє виконання програми. В цей час інформація у всіх вікнах оновлюється. Кнопка Break доступна тільки під час виконання програми.

- Reset виконує скидання процесу виконання поточної операції.
- Show next statement використовується для встановлення жовтого маркера в місці поточного значення покажчика програми.
- Single step або Trace Into виконує одну операцію.
- Step Over виконує одну операцію. Якщо операція містить виклик функції чи процедури, вони також виконуються. При зустрічі з точкою зупину, виконання припиняється.
- Step Out виконується поки поточна функція не закінчилась. При зустрічі з точкою зупину, виконання припиняється.
- Run to Cursor виконується поки програма не досягне операції, поміченої курсором у вікні редагування програми. Якщо така операція недосяжна, програма буде виконуватись до зупинки користувачем. Команда доступна тільки якщо вікно Source Window активне.
- Auto Step багаторазово виконує операцію Trace Into.

2.5.2 Точки зупинки

В програмному коді може бути задана необмежена кількість точок зупинки. Всі точки зупинки зберігаються між сесіями, навіть якщо код програми було змінено.

Для створення програмної точки зупинки (breakpoint) натисніть відповідну кнопку меню (рисунок 2.17) або клавішу F9.

Рисунок 2.17 – Панель створення точки зупину

- Включити/виключити програмну точку зупину у поточному місці знаходження.
- Видалити всі точки зупину.

Для створення точки зупинки даних (data breakpoints) у вікні I/O window виберіть необхідний регістр або натисніть View → Memory/Register та викличте контекстне меню. Виберіть Add data breakpoint для необхідної величини.

Для контролю обох видів точок зупинки використовується вкладка Breakpoints and tracepoints, яка стандартно розміщена в нижній частині вікна проекту (рисунок 2.18).

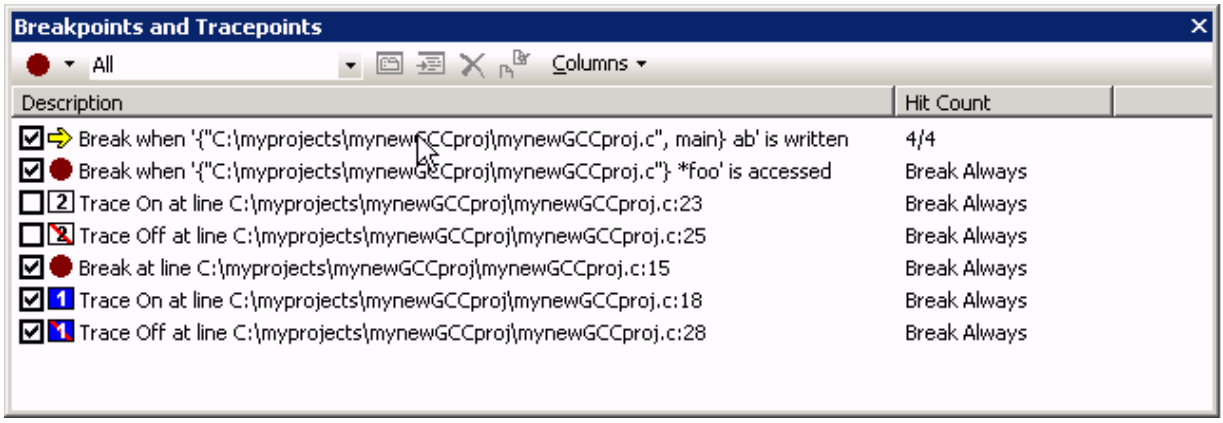


Рисунок 2.18 – Вікно перегляду контрольних точок

В цьому вікні можливе додавання нових точок зупинки, змінення та видалення існуючих. Для зміни параметрів обраної точки двічі клацніть мишкою на необхідній позиції.

Для задання точки зупинки даних просто перетягніть вибрану змінну з вікна редагування програми у вікно Breakpoints and tracepoints.

2.5.3 Перегляд стану процесора та регістрів

Для відображення вікна Processor зайдіть в меню View → Toolbars → Processor.

Перегляд стану процесора дозволяє слідкувати за такими величинами, як поточний лічильник програми, значення покажчика стека, X/Y/Z-покажчик, покажчик циклу, стан регістрів від R0 до R31 (рисунок 2.19).

2.5.4 Перегляд стану пам’яті

AVR Studio 4 дозволяє перегляд та редагування всіх видів пам’яті обраної платформи. Доступні види пам’яті: Program, SRAM (Data), EEPROM, external SRAM та I/O memory. Для виклику вікна натисніть меню View → Memory. Вікно перегляду стану пам’яті програм зображено на рисунку 2.20.

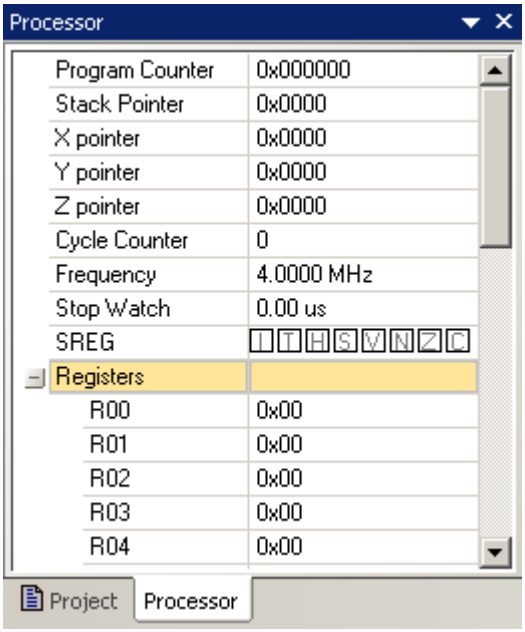


Рисунок 2.19 – Вікно перегляду стану процесора та регістрів

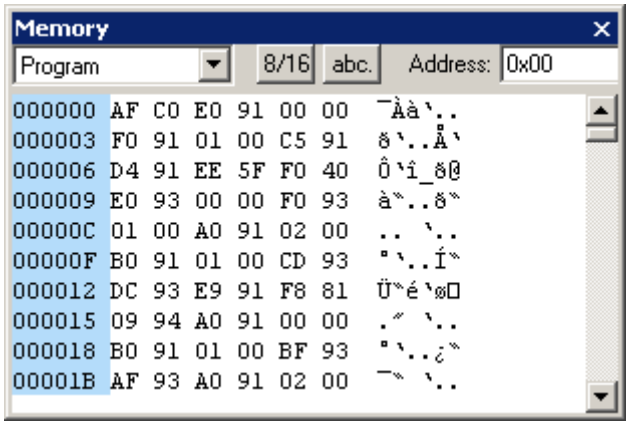


Рисунок 2.20 – Вікно перегляду стану пам’яті програм

Вікно для редагування комірки пам’яті викликається подвійним кліком.

Тип памяті, що редагується, можна обрати з випадаючого списку в лівій частині вікна.

Контрольні запитання та завдання

1) Опишіть створення нового проекту в AVR Studio
2) Як виконати запуск проекту?
3) Як створити hex-файл та виконати його перегляд?
4) Як виконати налогодження програми?

3 МОДЕЛЮВАННЯ ОКРЕМИХ МОДУЛІВ МІКРОКОНТРОЛЕРІВ СІМ'Ї AVR У ПАКЕТІ PROTEUS

3.1 Загальна характеристика пакету

Багато радіоаматорів-початківців стикалися з ситуацією коли, вирішивши зібрати вподобаний і, безсумнівно, потрібний пристрій, через недосвідченість, а може через помилки в схемі або ж за іншими обставинами, спалювали насилу придбані дорогі радіодеталі. І скоріше за все більшість, обпікшись на перших невдачах, закидали заняття радіоелектронікою назавжди.

В наш час широкої комп'ютеризації, знайшовся вихід з цього глухого кута. З'явилася величезна кількість програм симуляторів, які заміняють реальні радіодеталі і прилади віртуальними моделями. Симулятори дозволяють, без складання реального пристрою, налагодити роботу схеми, знайти помилки, отримані на стадії проектування, зняти необхідні характеристики і багато іншого.

Одна з таких програм PROTEUS VSM. Але симуляція радіоелементів це не єдина здатність програми. Proteus VSM, яку створено фірмою Labcenter Electronics на основі ядра SPICE3F5 університету Berkeley, є так званим середовищем наскрізного проектування Це означає створення пристрою, починаючи з його графічного зображення (принципової схеми) і закінчуючи виготовленням друкованої плати пристрою, з можливістю контролю на кожному етапі виробництва.

Але, не дивлячись на уявну складність програми, користуватися нею зможуть не тільки професіонали в світі радіоелектроніки, а й новачки, які навчилися, а може поки що і ні, відрізняти резистор від транзистора.

PROTEUS VSM складається з двох самостійних програм ISIS і ARES. ARES це програма для трасування друкованих плат з можливістю створення своїх бібліотек корпусів і в даній роботі розглядатися не буде. Основною програмою є ISIS, в якій передбачено зв'язок з ARES для передачі проекту для розведення друкованої плати.

У «сферу впливів» PROTEUS VSM входять як найпростіші аналогові пристрої, так і складні системи, які створено на популярних нині мікроконтролерах. Доступна величезна бібліотека моделей елементів, поповнювати яку може сам користувач. Природно для цього потрібно досконально знати роботу елемента і вміти програмувати. Можливість анімації схем дозволяє програмі стати прекрасним навчальним посібником на уроках в школі і ВНЗ. Достатній набір інструментів і функцій, серед яких вольтметр, амперметр, осцилограф, всілякі генератори, здатність налагоджувати програмне забезпечення мікроконтролерів, роблять PROTEUS VSM хорошим помічником розробнику електронних пристроїв.

3.2 Опис налагоджувача PROTEUS 8.6

3.2.1 Інсталяція Proteus 8.6

Proteus відповідної версії можна завантажити з файлообмінних мереж або придбати на сайті виробника. В даних методичних вказівках було використано програму Proteus версії 8.6. Після запуску файлу інсталятора з’явиться вікно привітання (рисунок 3.1).

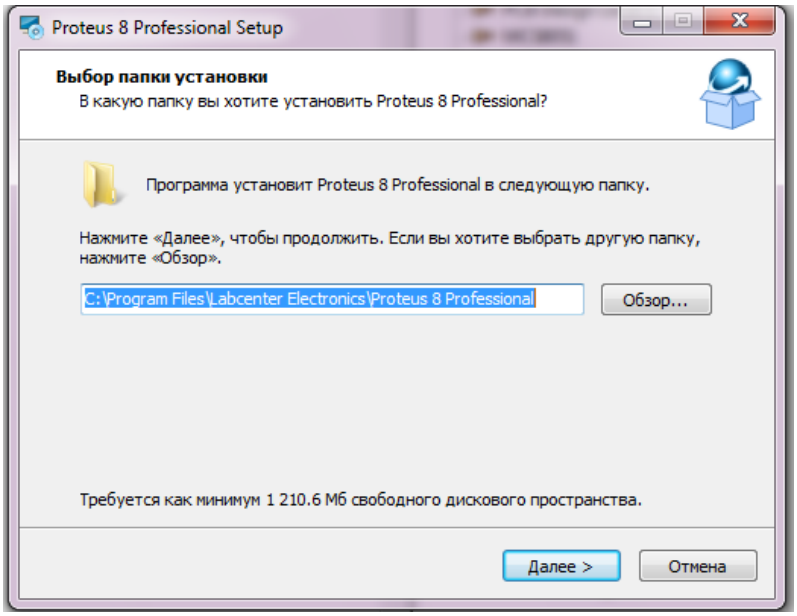


Рисунок 3.1 – Вікно привітання та вибір папки для програми

Далі необхідно виконати приведену нижче на рисунках послідовність дій. Перехід між екранами встановлення програми здійснюється за натисненням кнопки “Далее”.

Слідуйте рекомендаціям, які дає постачальник вашої версії системи Proteus (рисунок 3.2).

Екран пропонує почати встановлення програми. Якщо всі налаштування обрано правильно, то встановлення Proteus 8.6 почнеться без помилок. Зазвичай встановлення триває близько 5 хвилин (рисунок 3.3).

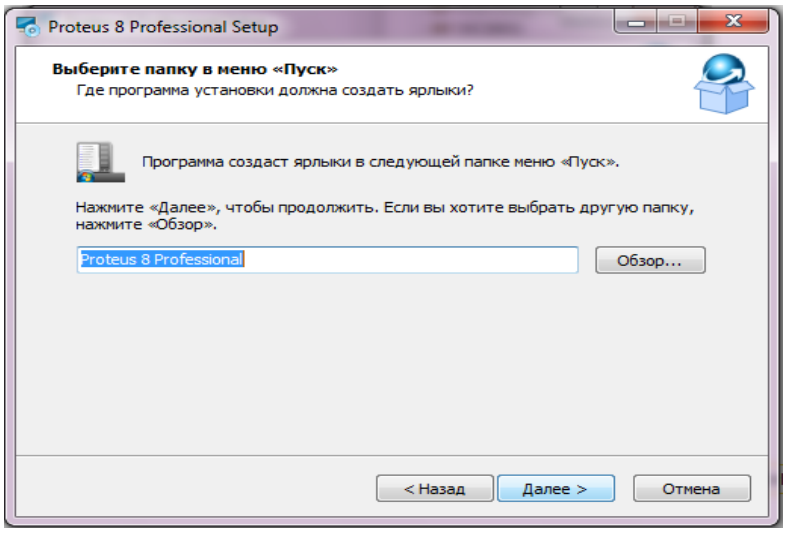


Рисунок 3.2 – Вибір локації у меню «Пуск»

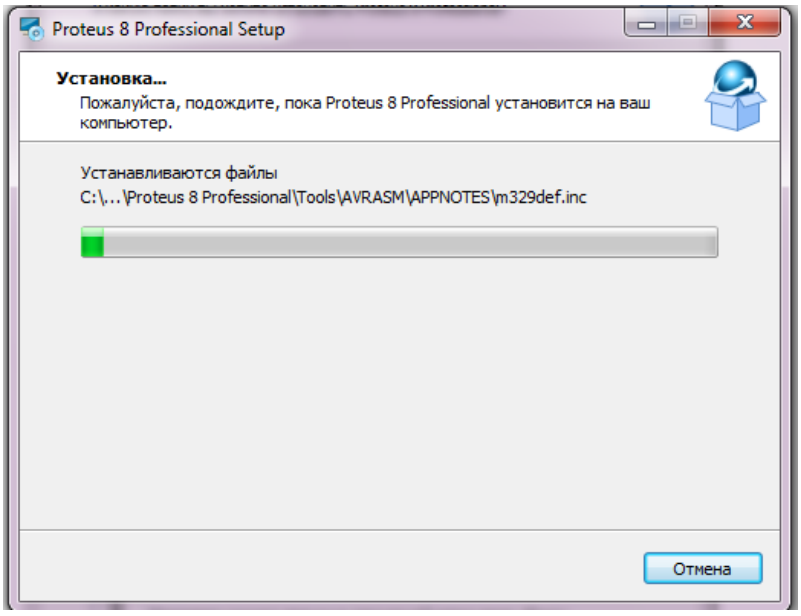


Рисунок 3.3 – Екран процесу встановлення

В кінці встановлення з’явиться вікно (рисунок 3.4) з кнопкою “Завершить”. Запустити програму можна з робочого стола або за кнопкою “ПУСК” (рисунок 3.5). Головний екран програми PROTEUS 8.6 зображено на рисунку 3.6.

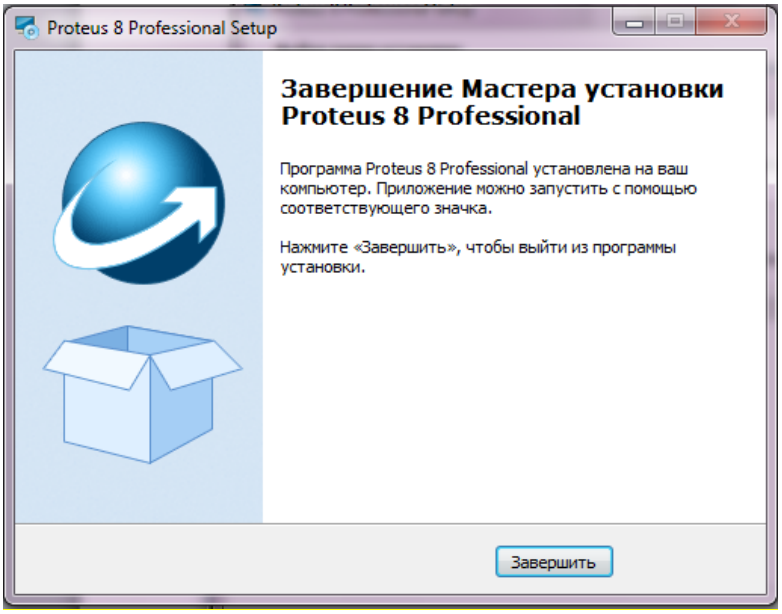


Рисунок 3.4 – Кінець інсталяції програми

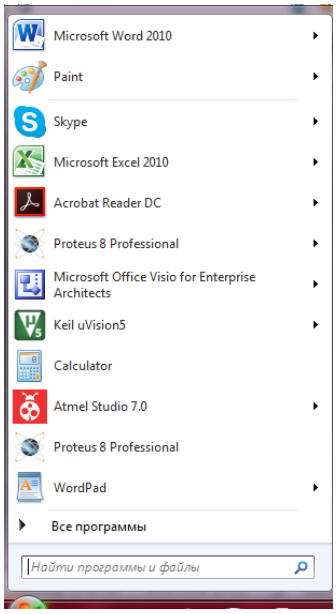


Рисунок 3.5 – Запуск програми за кнопкою “ПУСК”

3.2.2 Створення нового проекту

Щоб створити новий проект у Proteus потрібно у стартовому вікні програми (рисунок 3.7) у лівому верхньому кутку вибрати пункт меню File – > New Project або

застосувати скорочення клавіш (ctrl+N). Після цього з’явиться вікно налаштувань нового проекту, де можна обрати назву нового проекту та локацію для збереження на комп’ютері.

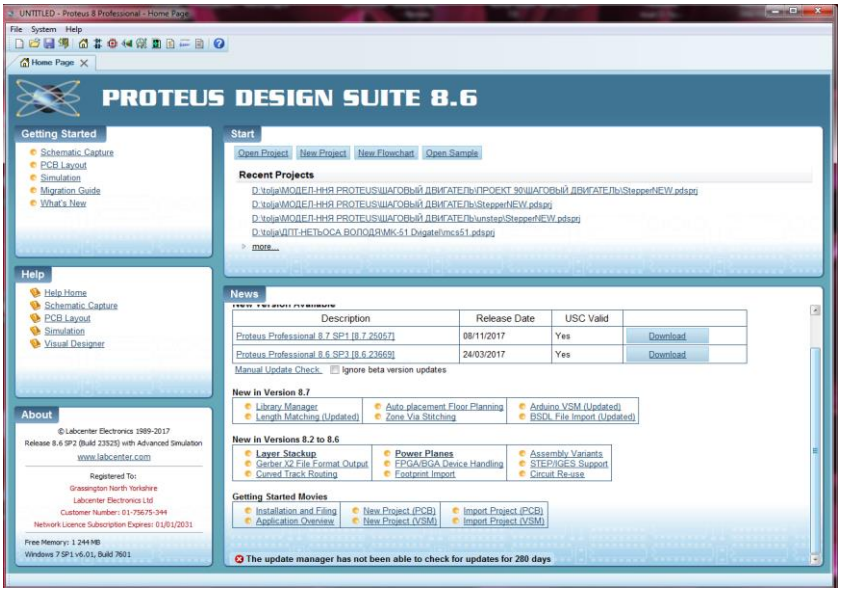


Рисунок 3.6 – Головний екран програми

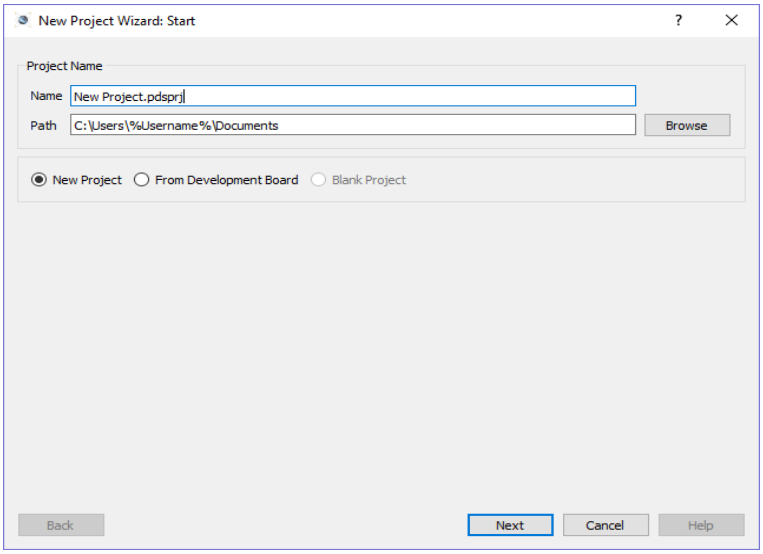


Рисунок 3.7 – Створення нового проекту у Proteus

Можна також обрати один з уже запропонованих проектів. Для цього треба обрати пункт From Development Board (рисунок 3.8).

Обравши налаштування вручну, ідемо на наступний крок – вибір розміру полотна для схеми (рисунок 3.9). Будемо обирати тип DEFAULT – за замовчуванням.

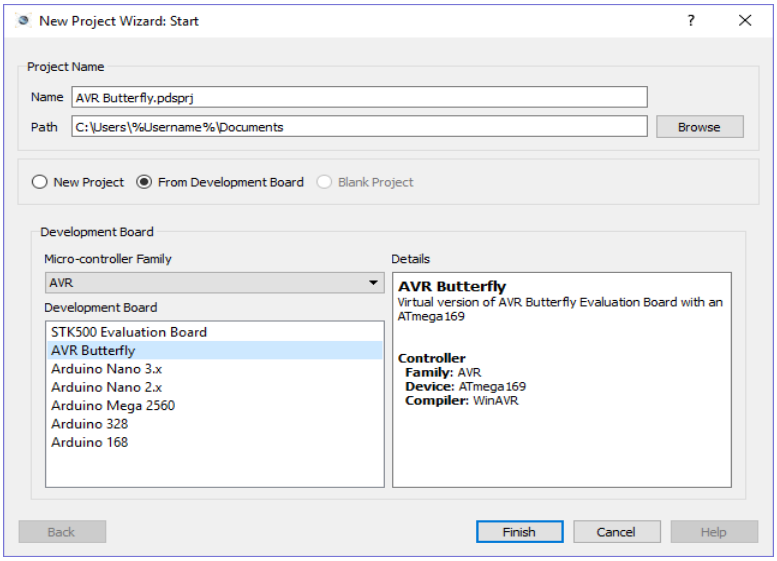


Рисунок 3.8 – Заздалегідь налаштовані проекти

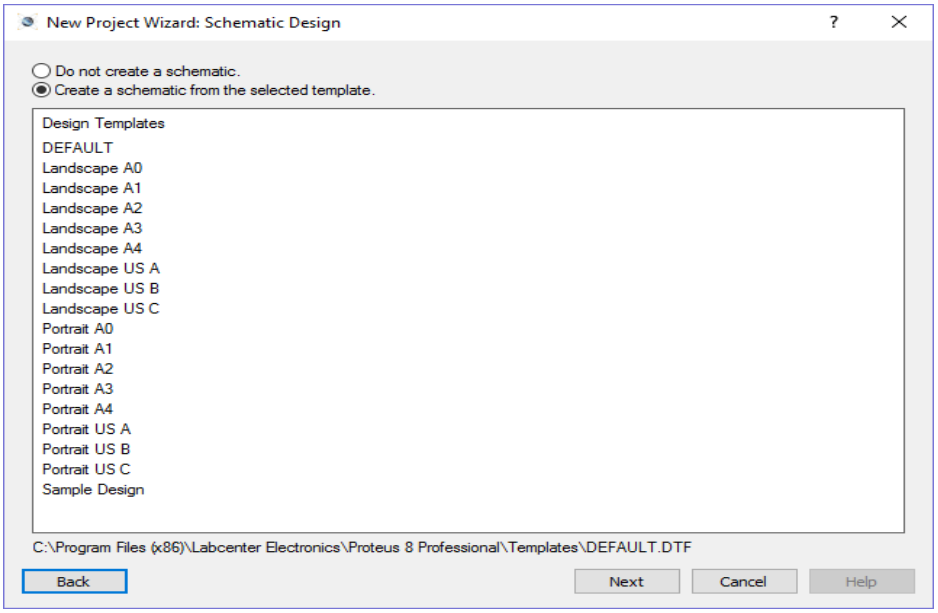


Рисунок 3.9 – Налаштування розмірів полотна схеми

В наступному вікні (рисунок 3.10) також оберемо тип DEFAULT.

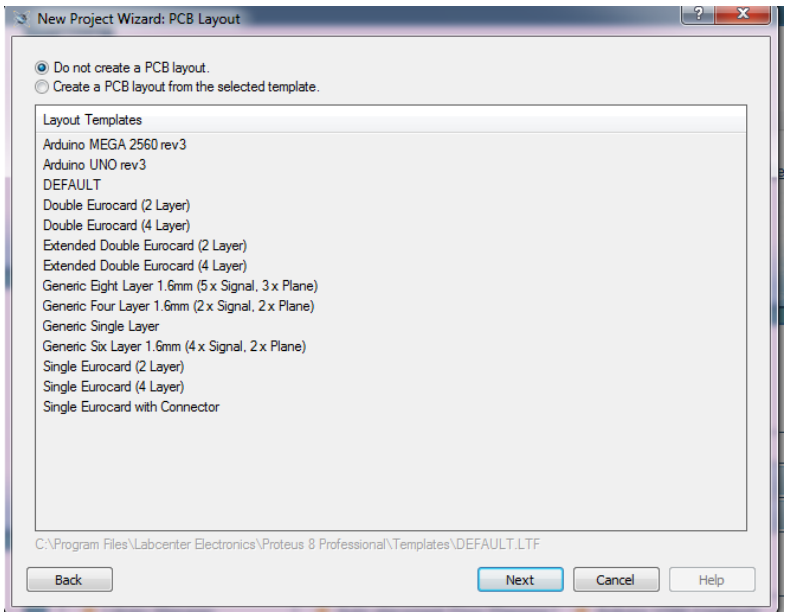


Рисунок 3.10 – Вибір схеми

У наступному вікні (рисунок 3.11) можна обрати налаштування для створення проекту прошивки. А далі закінчуємо процес попереднього налаштування проекту.

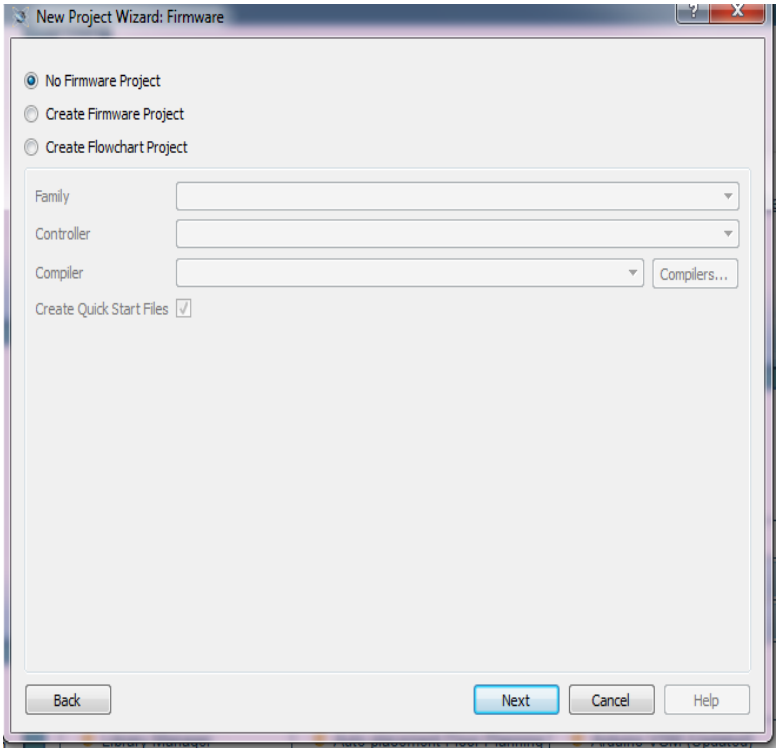


Рисунок 3.11 – Вибір налаштувань проекту прошивки

Оберемо перший пункт (без прошивки) та натиснемо кнопку NEXT (далі), після чого відкриється підсумкове вікно створення проекту (рисунок 3.12).

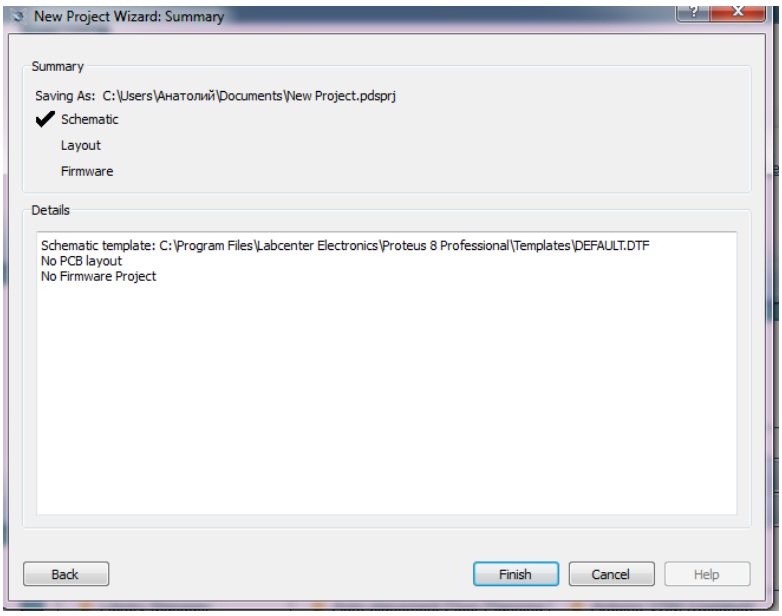


Рисунок 3.12 – Вікно з підсумками процесу налаштування

Врешті відкриється головний екран середовища розробки ISIS (рисунок 3.13)

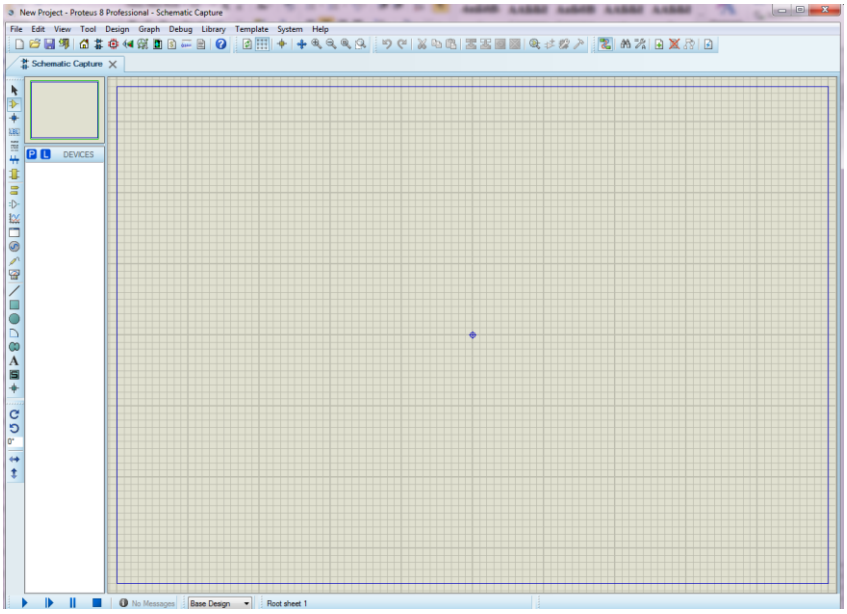


Рисунок 3.13 – Вигляд середовища ISIS після налаштувань

3.2.3 Відкриття існуючого проекту

Щоб відкрити вже існуючий ISIS-проект для Proteus треба обрати пункт у меню зліва зверху File – > Open project або сполучення клавіш (ctrl+o). Відобразиться вікно вибору файлу проекту з диска (рисунок 3.14). Файл має бути попередньо збережений у форматі .pdsprj.

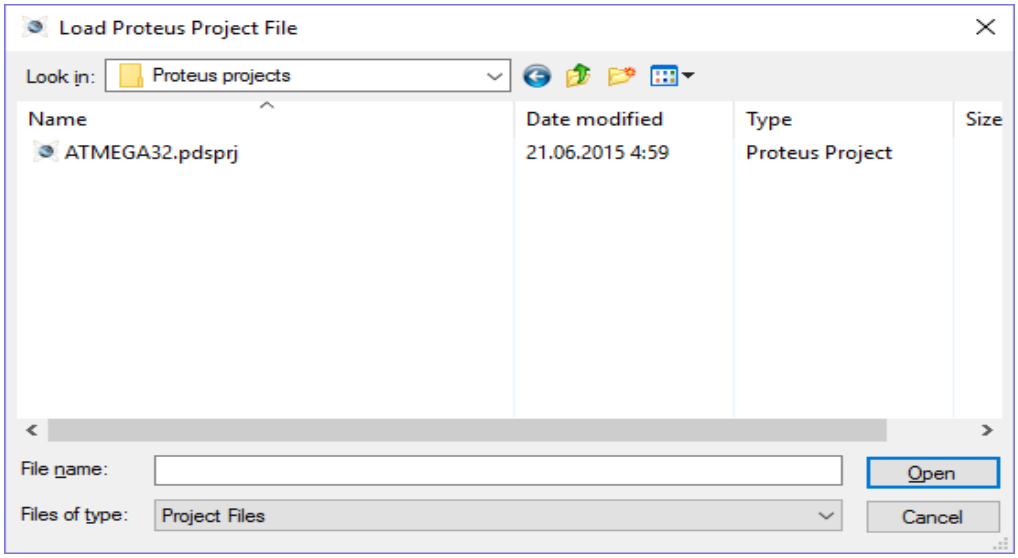


Рисунок 3.14 – Вікно вибору файлу

Приклад відкритого проекту, який розглядається у [1], наведено на рисунку 3.15.

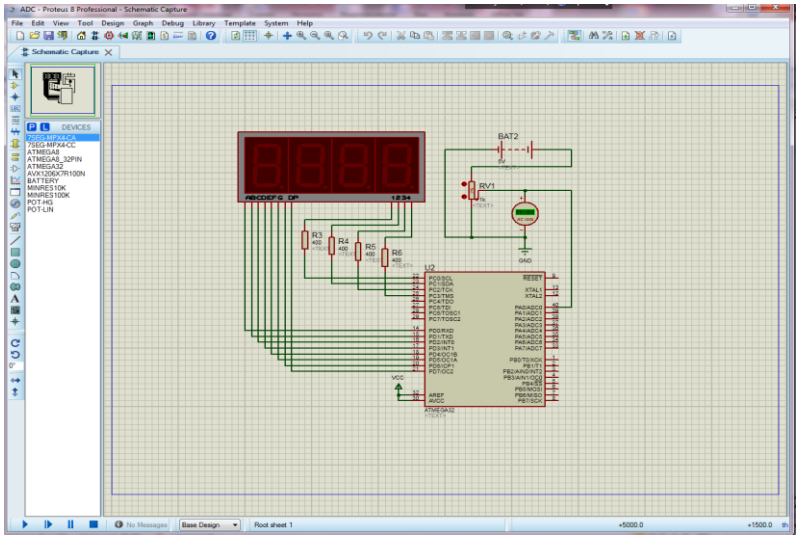


Рисунок 3.15 – Приклад відкритого проекту

3.2.4 Знайомство з інтерфейсом середовища ISIS

Це питання розглянемо на прикладі моделі принципової схеми цифрового вольтметру, яку ми далі побудуємо самостійно (рисунок 3.15).

На рисунку 3.16 наведено список компонентів, які було використані для побудови даної схеми.

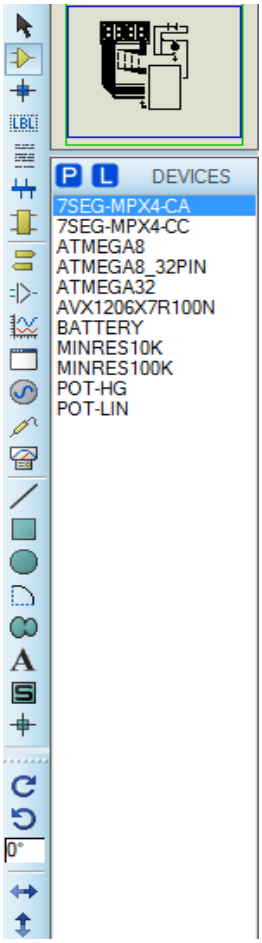


Рисунок 3.16 – Вид інтерфейсу бокового меню та списку компонентів

Стрілки повороту та стрілки напрямів у меню посередині дозволяють більш точно рухати елемент на схемі.

Для запуску схеми використовується меню, що знаходиться внизу вікна ISIS (рисунок 3.17). Щоб розпочати виконання програми треба натиснути на синій трикутник, а щоб завершити – на квадрат.

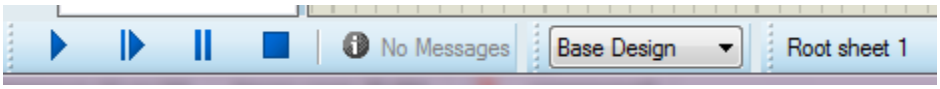


Рисунок 3.17 – Меню запуску моделювання

Меню на верхній панелі надає можливість обрати вигляд схеми. Наприклад, обравши перший зліва на рисунку 3.18 пункт меню Schematic capture (схематичне зображення) перейдемо до стандартного вигляду схеми, який ми частіше за все використовуємо.

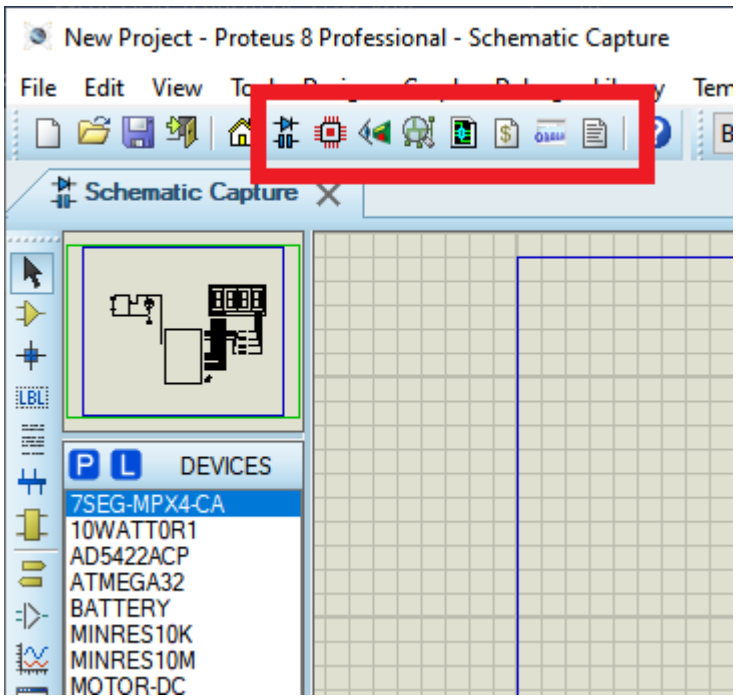


Рисунок 3.18 – Меню вибору вигляду схеми

Наступний пункт PCB Layout – показує вигляд схеми з дотриманням масштабу, розмірів деталей та їх фізичних параметрів, а також показує розташуваня кріплень до мікросхеми.

3.2.5 Додавання моделей

Щоб додати будь-яку модель на схему потрібно слідувати крокам, що описані далі (рисунок 3.19).

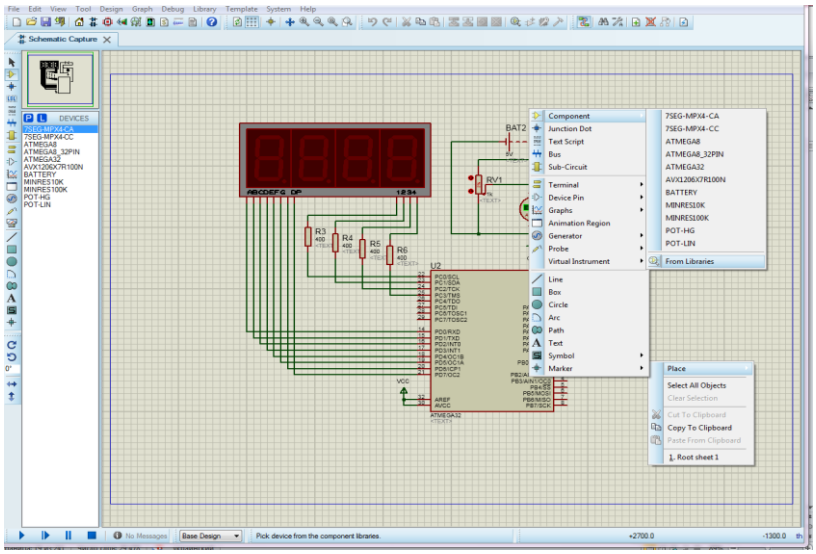


Рисунок 3.19 – Додавання моделі

1. У головному вікні схематичного представлення мікросхеми треба натиснути правою кнопкою миші по вільній ділянці полотна. Відкриється невелике меню, з якого треба буде обрати перший пункт – Place (помістити).
2. У наступному меню потрібно обрати, який саме елемент ми хочемо помістити на схему. Обираємо варіант Component (компонент).
3. Третє меню дозволяє обрати недавно використані елементи. Ми ж оберемо останній пункт меню – From libraries (з бібліотек), щоб самостійно обрати потрібний елемент з бібліотеки.

4. У вікні, що з'явилося, впишемо у рядок пошуку в лівому верхньому кутку вікна ключове слово, наприклад, motor. Після чого буде знайдено усі елементи, що використовують це слово у своїй назві (рисунок 3.20).

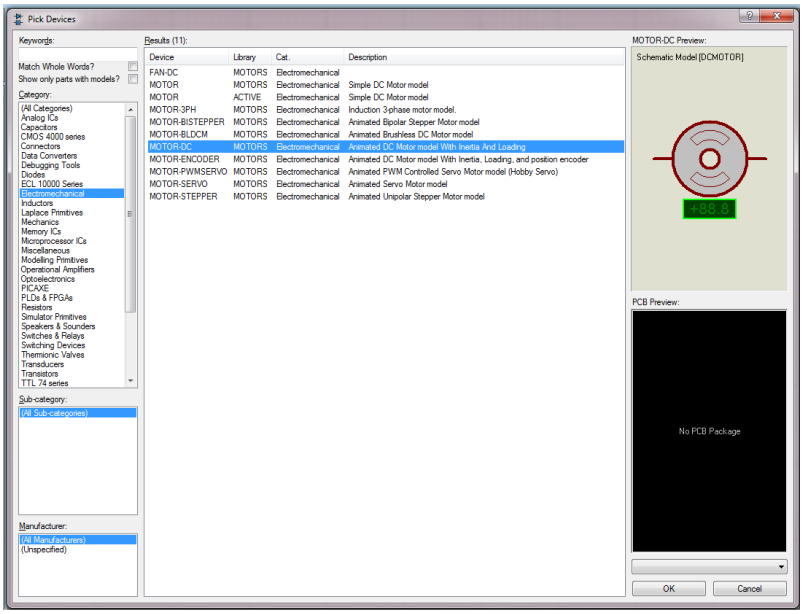


Рисунок 3.20 – Вікно вибору потрібного елемента

5. Обравши потрібний елемент (Motor-DC з бібліотеки Motors) натиснемо кнопку ОК та розмістимо елемент на схемі клацнувши по ній лівою кнопкою миші.

Нижче наведено приклади додавання світлодіода, двигуна постійного струму, осцилографа, вольтметра та крокового двигуна.

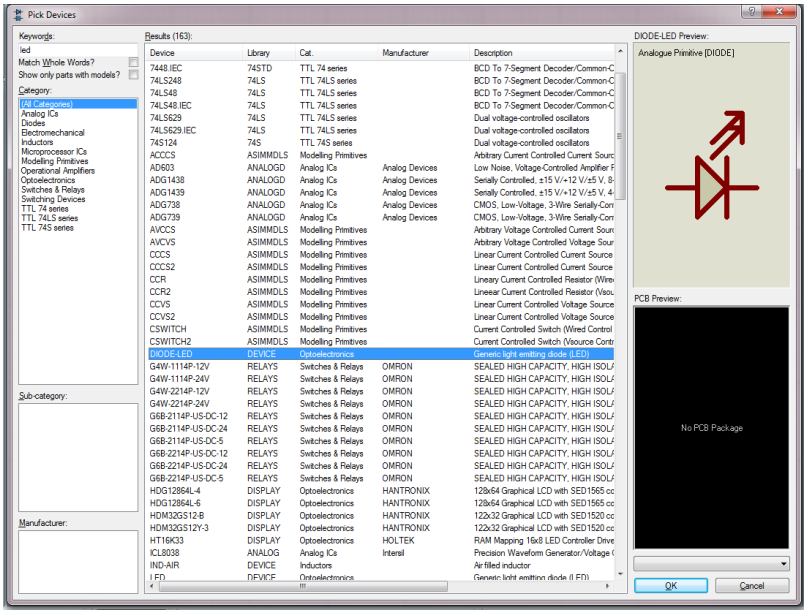


Рисунок 3.21 – Вибір потрібного елемента зі списку

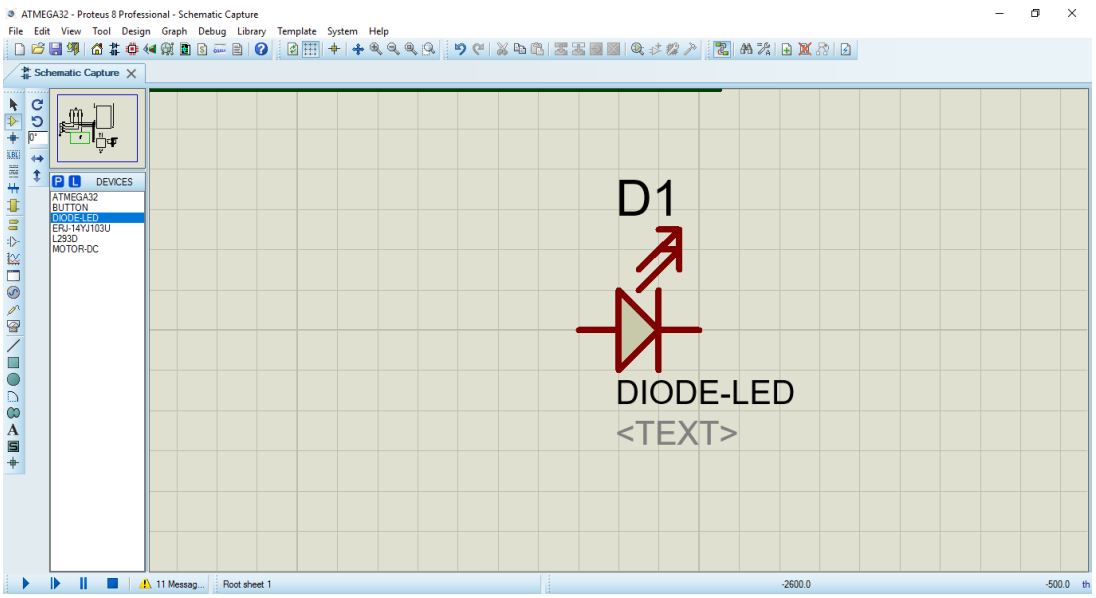


Рисунок 3.22 – Розміщення елемента на полотні

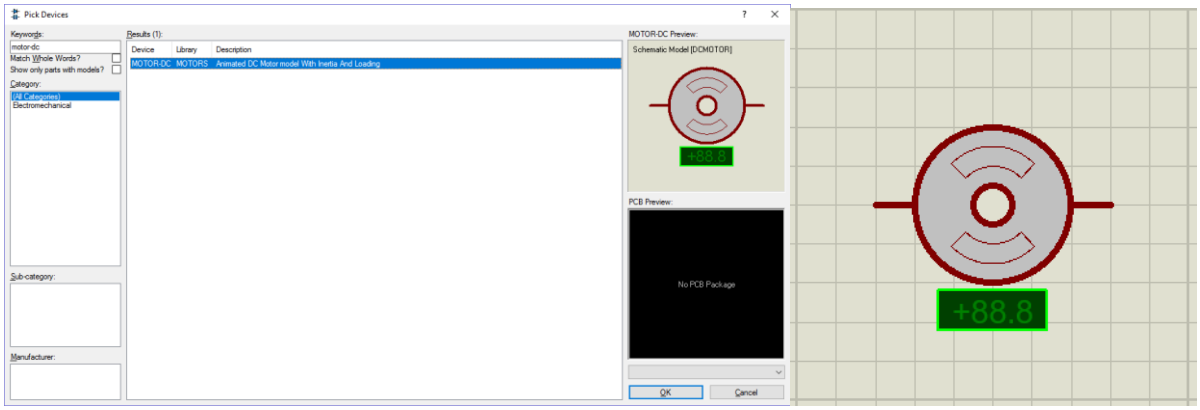


Рисунок 3.23 – Розміщення двигуна постійного струму

Щоб додати осцилограф та вольтметр потрібно переключитися у режим Instruments в меню на рисунку 3.24. Відкриється панель інструментів, з якої можна обрати потрібний нам.

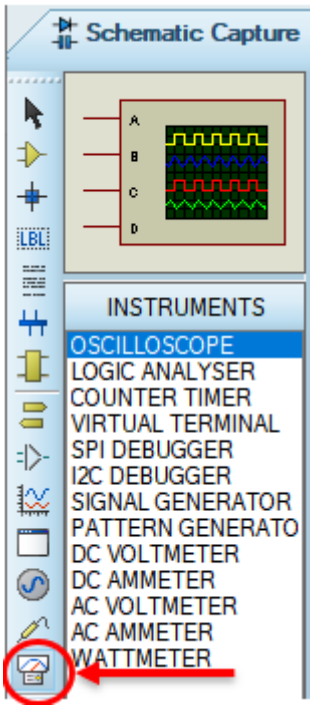


Рисунок 3.24 – Меню вибору інструментів

Додамо осцилограф з меню компонентів на схему цифрового вольтметра, яку виконано на мікроконтролері ATMEGA32 (рисунки 3.25, 3.29). Для цього після вибору осцилографа в меню натиснемо двічі по порожній ділянці на схемі, в цьому місці одразу з’явиться осцилограф (рисунок 3.25).

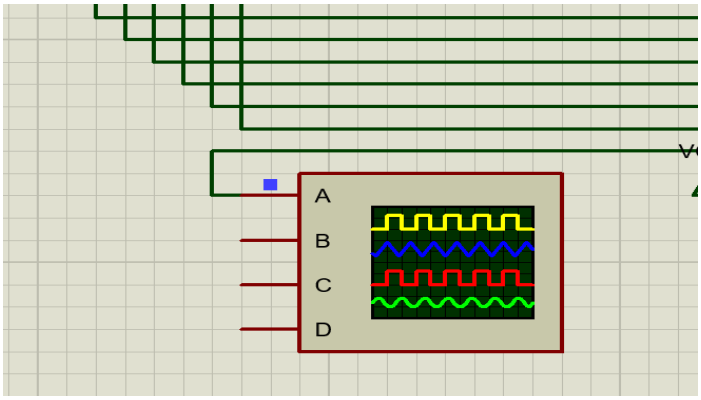


Рисунок 3.25 – Осцилограф на схемі

Під’єднаємо осцилограф до схеми. Для цього треба клікнути лівою кнопкою миші по одному з виходів осцилографа, має з’явитися лінія, що слідує за курсором миші (рисунок 3.26). Під’єднаємо інший кінець цього дроту до ділянки кола, яку

нам треба продивитися. Для цього клацнемо ще раз по ділянці, до якої потрібно під'єднатися. З'явиться лінія між осцилографом та ділянкою кола.

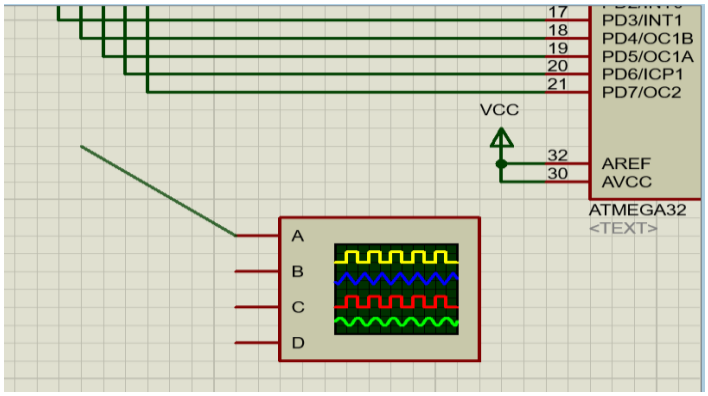


Рисунок 3.26 – Приклад приєднання осцилографа

Щоб відкрити вікно осцилографа, треба запустити схему. Для цього треба натиснути на синій трикутник в нижній частині екрану (рисунок 3.27). Схема запуститься, і трикутник змінить колір на зелений.

Рисунок 3.27 – Увімкнення системи

Далі треба у пункті меню Debug, на верхній панелі, у головному вікні програми обрати зі списку компонентів системи осцилограф і натиснути на його назву (рисунок 3.28). Одразу ж відкриється вікно осцилографа з даними (рисунок 3.29).

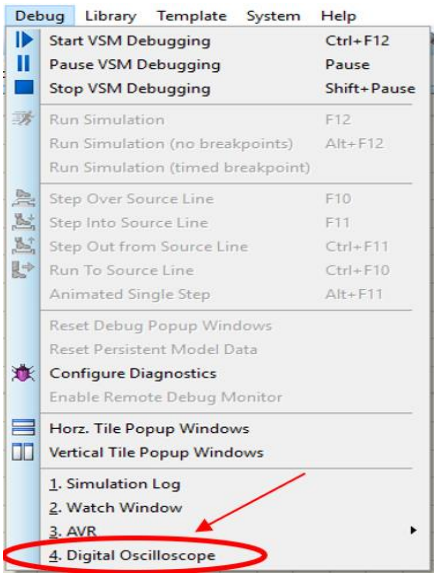


Рисунок 3.28 – Список компонентів у системі

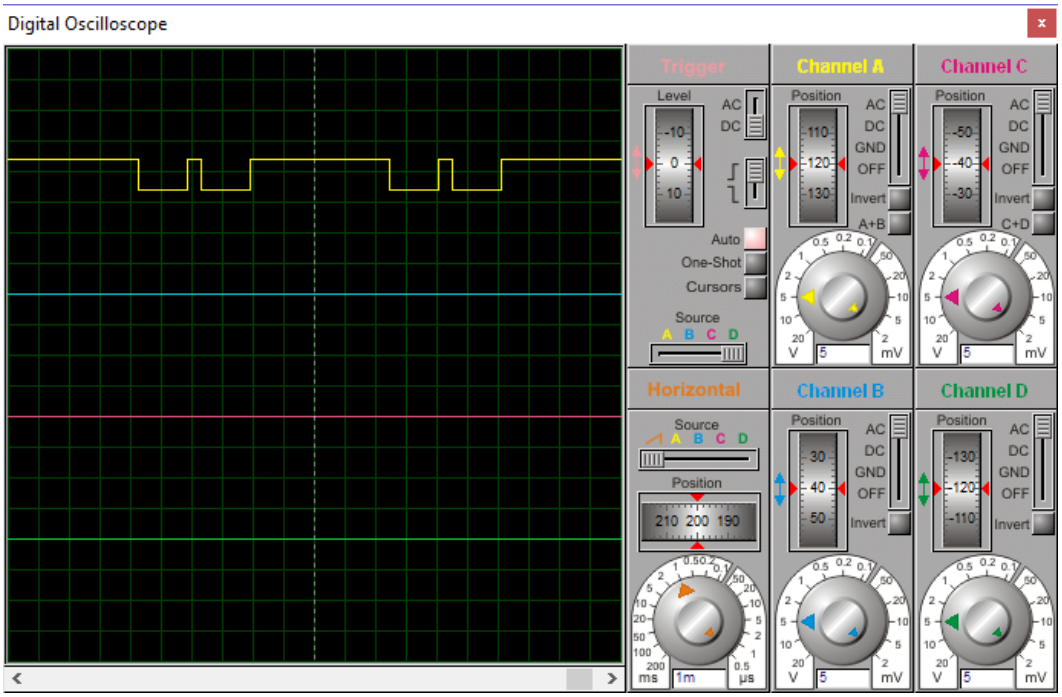


Рисунок 3.29 – Вікно осцилографа

Зупиняємо симуляцію за допомогою меню, що зображено на рисунку 3.27. Після цього аналогічно додамо на схему вольтметр. Оберемо у меню зліва компонент AC VOLTMETER і розмістимо його на схемі (рисунок 3.30).

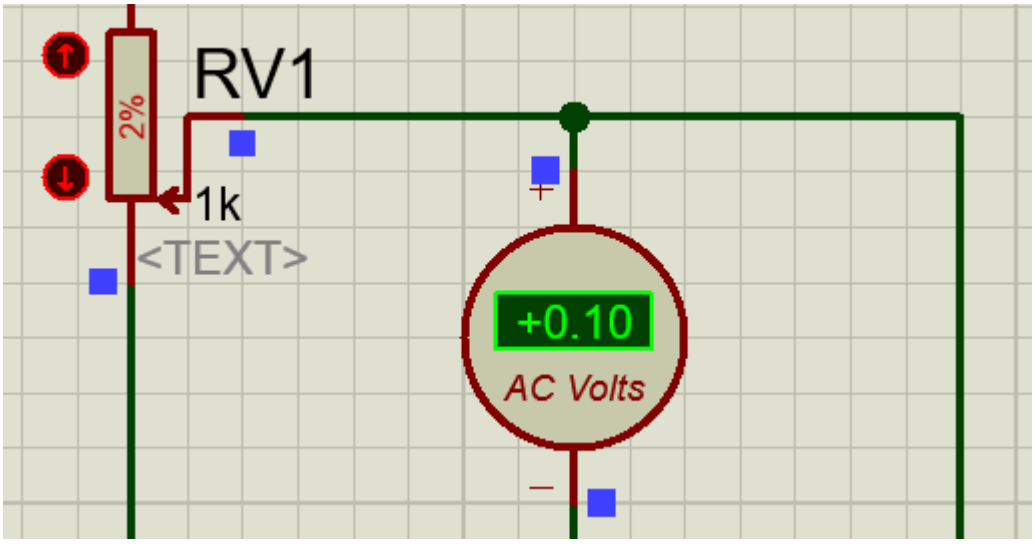


Рисунок 3.30 – Вольтметр, який розміщено на схемі

Додамо на схему семисегментний світлодіодний дисплей з маркуванням, наприклад, 7SEG-MPX1-CA. Для додаванная семисегментного індикатора потрібно виконати аналогічні кроки, як і при доданні попередніх компонентів.

Знайдемо у бібліотеці компонентів червоний чотирьохцифровий індикатор за ключовим словом 7SEG-MPX4-CA (рисунок 3.31).

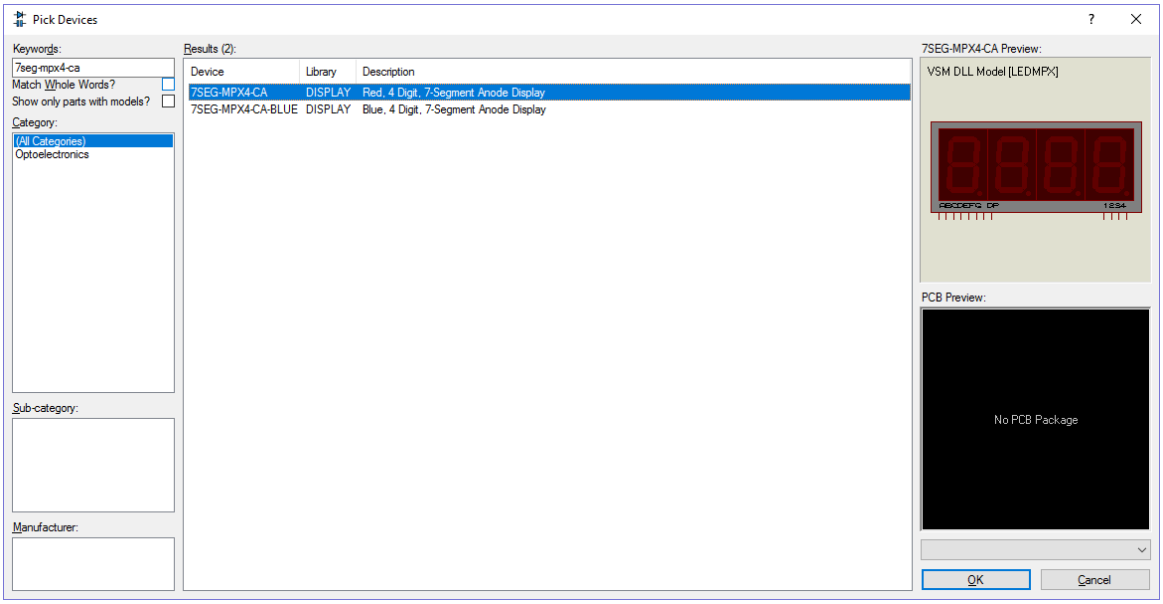


Рисунок 3.31 – Вікно додавання семисегментного індикатора

Виберемо місце на схемі та розмістимо індикатор, натиснувши ліву кнопку миші (рисунок 3.32).

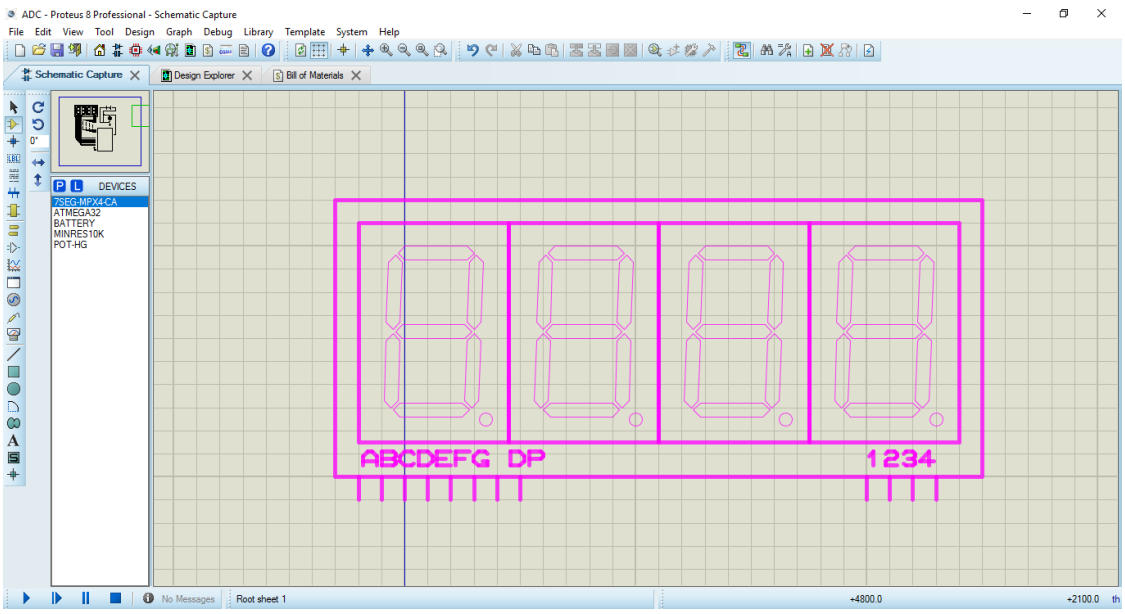


Рисунок 3.32 – Розміщення індикатора на схемі

Після такої операції на полотні схеми з’явиться новий елемент, який згодом можна буде під’єднати до схеми.

Додавання резисторів на схему відбувається за аналогічні кроки: спочатку ми шукаємо потрібний елемент у бібліотеці компонентів у бібліотеці RESISTORS. В даній бібліотеці за ключовим словом MINRES присутні різноманітні резистори з різними характеристиками (рисунок 3.33).

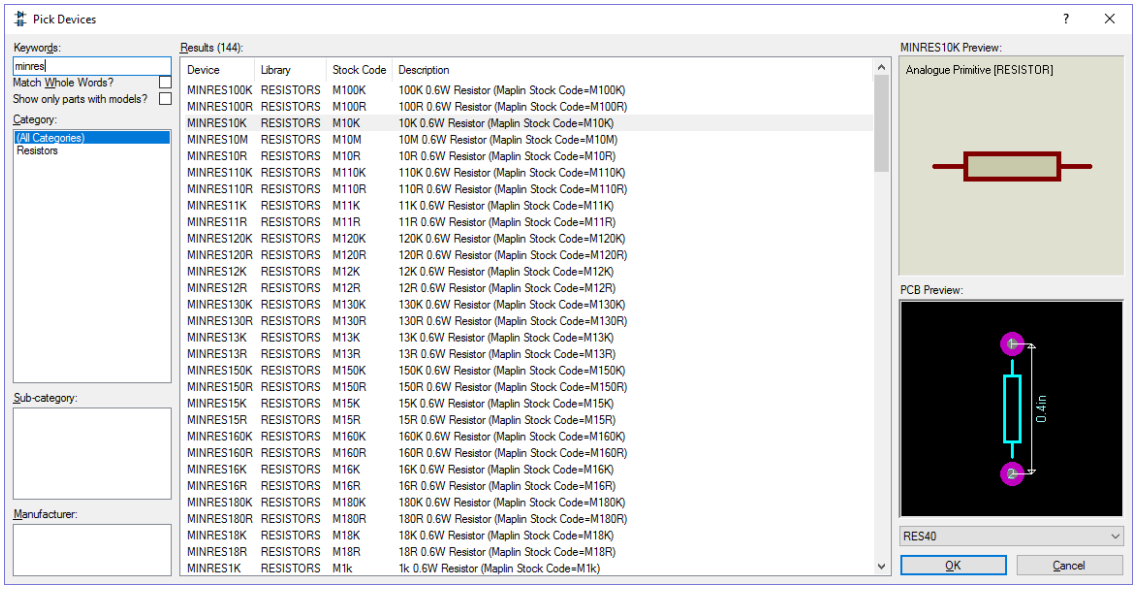


Рисунок 3.33 – Вікно вибору потрібного резистора

А далі потрібний резистор розміщається на схему. За бажанням можна змінити опір резистора. Для цього потрібно двічі натиснути лівою кнопкою миші по елементу, відкриється вікно як на рисунку 3.34. Після зміни опору у полі Resistance тиснимо клавішу Enter, після чого у резистора зміниться опір.

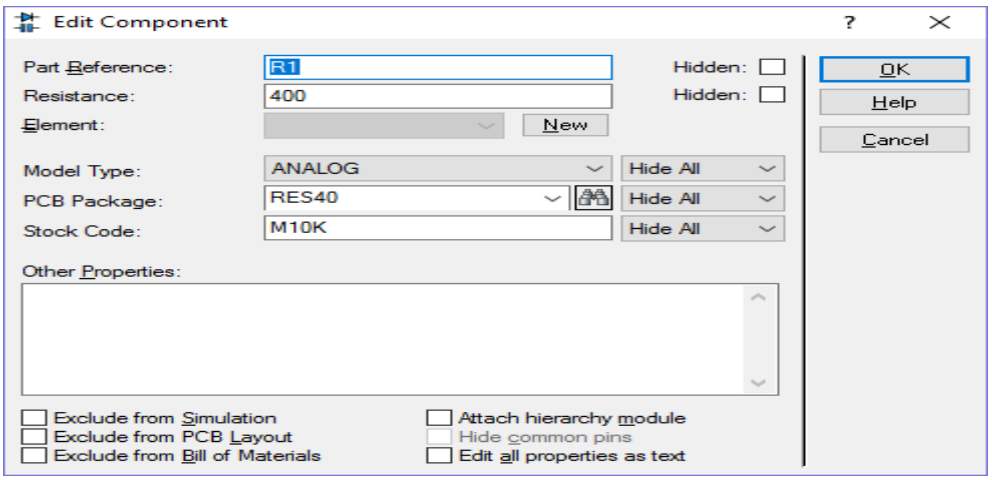


Рисунок 3.34 – Вікно налаштувань резистора

Додамо до схеми кроковий двигун. Вибираємо категорію “Electromechanical”. Для створення уніполярного крокового двигуна обираємо MOTOR-STEPPER (рисунок 3.35), для біполярного крокового двигуна – MOTOR-BISTEPPER (рисунок 3.36).

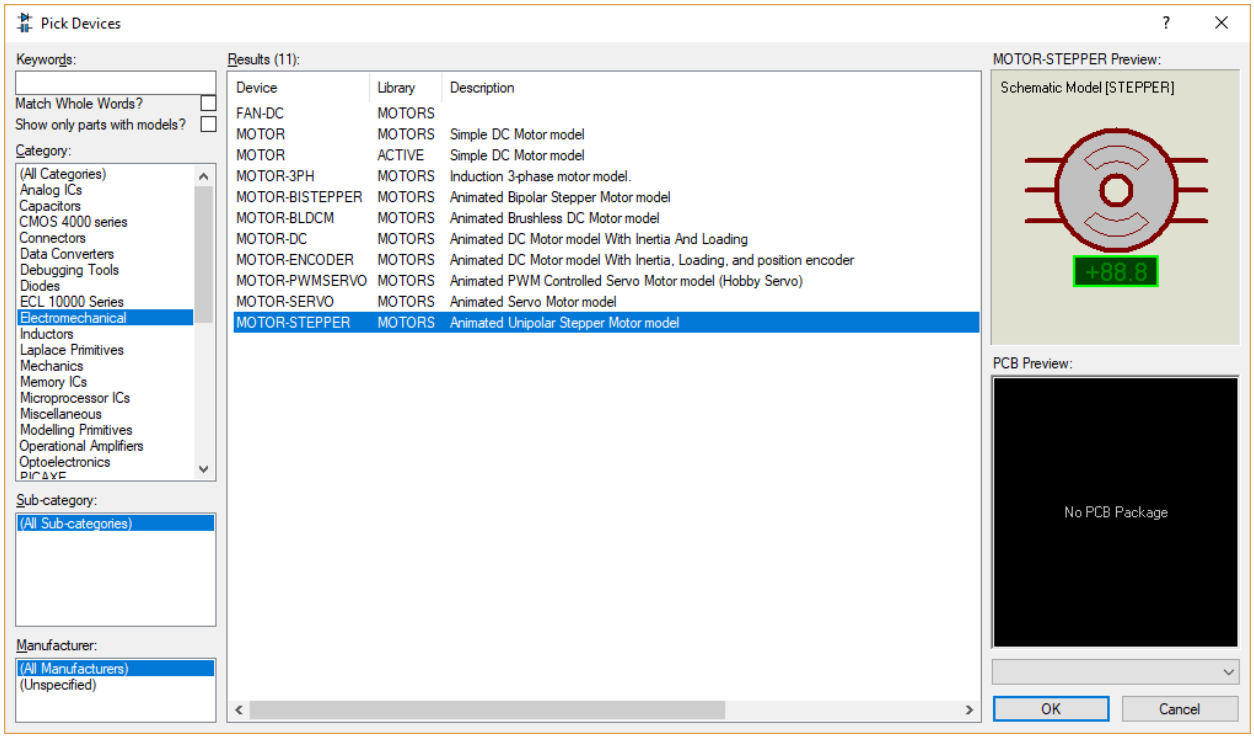


Рисунок 3.35 – Уніполярний кроковий двигун (STEPPER)

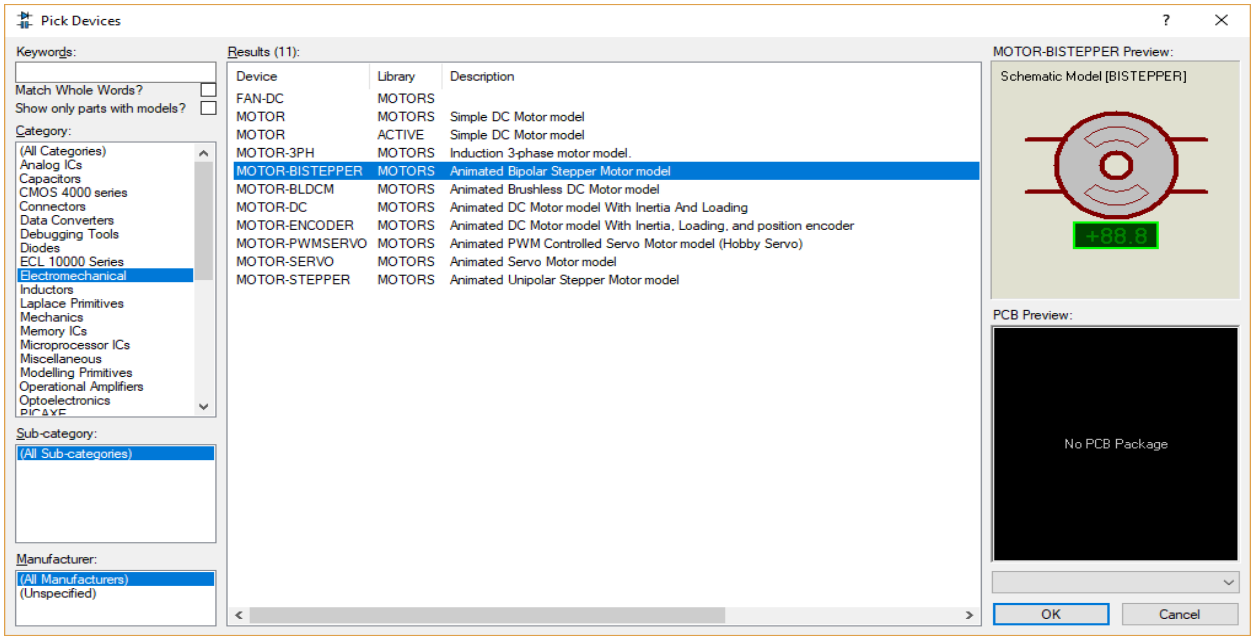


Рисунок 3.36 – Біполярний кроковий двигун (BISTEPPER)

3.2.6 Додавання мікроконтролера

Щоб додати у проект мікроконтролер, наприклад, ATMEGA32 з архітектурою AVR, оберіть режим роботи “Component mode” (рисунок 3.37) у меню, що знаходиться зліва на головному екрані середовища ISIS, та натисніть правою кнопкою миші на вільній частині робочого простору проекту. У меню, що з’явилося після натискання на кнопку миші, оберіть Place –> Component –> From Libraries (рисунок 3.38).

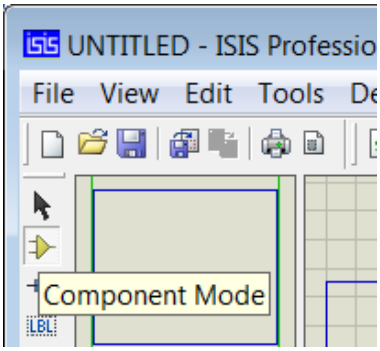


Рисунок 3.37 – Перехід у режим Component Mode

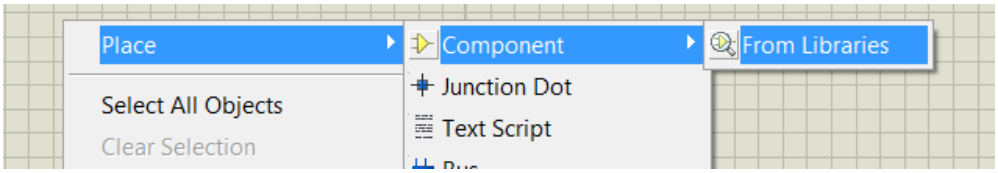


Рисунок 3.38 – Меню додавання нового компоненту

У меню, що відкрилося, введіть ключове слово пошуку “AVR” у верхньому лівому кутку вікна, оберіть бажаний AVR-сумісний МК та натисніть “OK” (рисунок 3.39). Далі треба лівою кнопкою миші розташувати мікроконтролер на полотні, двічі натиснувши на ліву кнопку.

3.2.7 Приклад побудови схеми цифрового вольтметра

З компонентів, які розглянуто вище, побудуємо схему моделі цифрового вольтметра, яку зображено на рисунок 3.40.

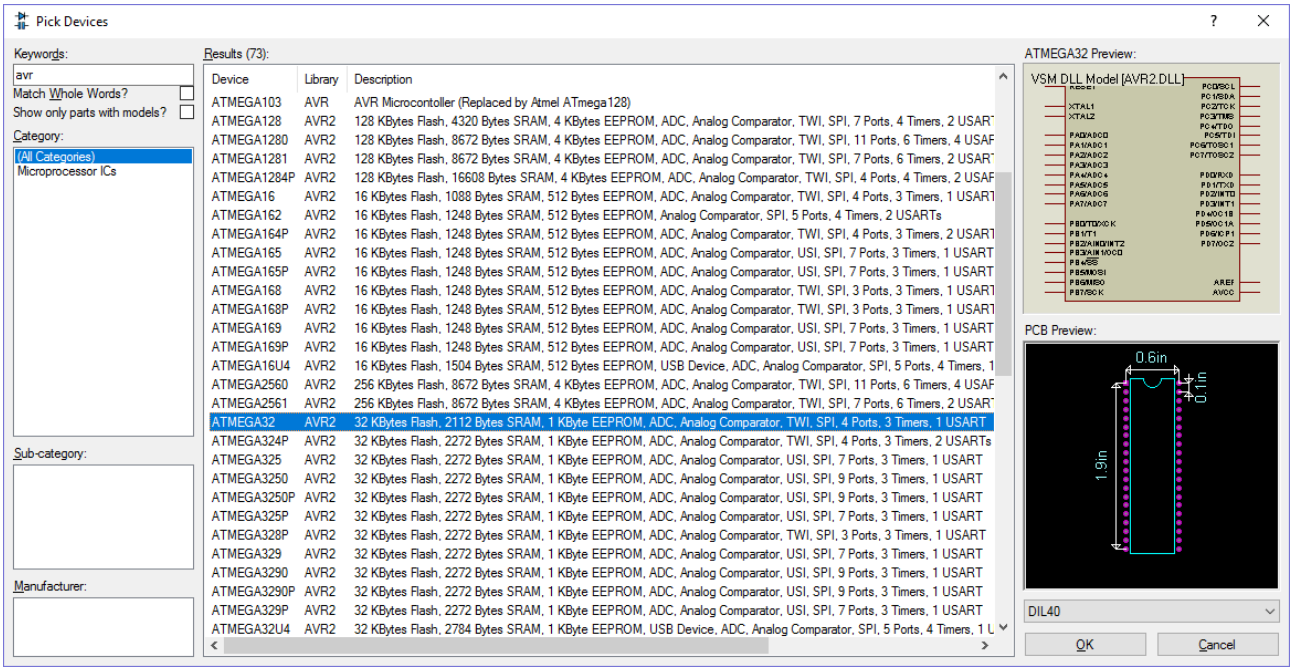


Рисунок 3.39 – Вибір мікроконтролера

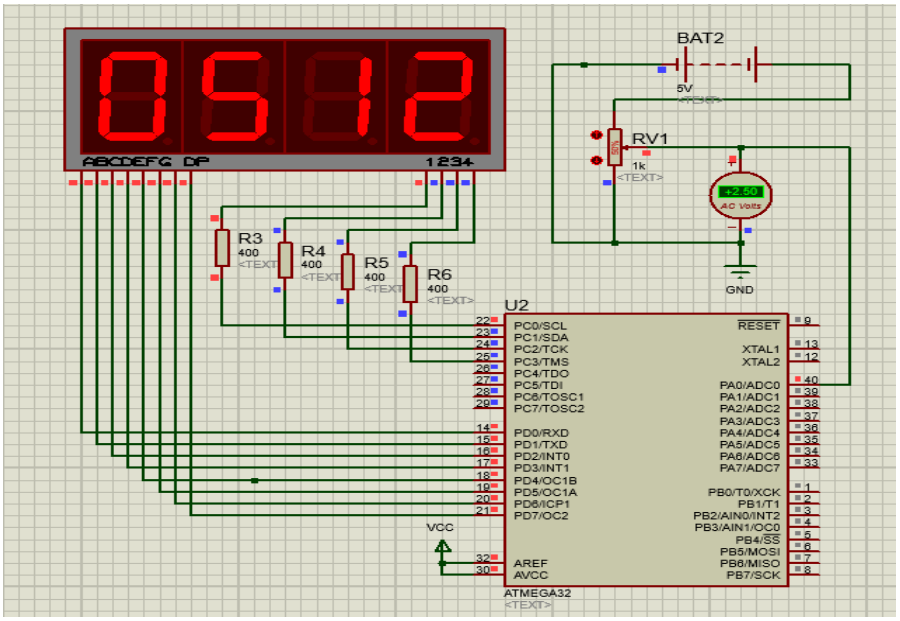


Рисунок 3.40 – Приклад схеми цифрового вольтметра

Роботу цієї схеми буде розглянуто у підрозділі 14.4/ Нижче розглядається побудова цієї схеми для її моделювання у PROTEUS.

Почнемо побудову схеми з додаваня до неї мікроконтролера. Для цього оберемо в бібліотеці компонентів МК ATMEGA32, який знаходиться першим у списку мікроконтролерів у бібліотеці AVR (рисунок 3.41).

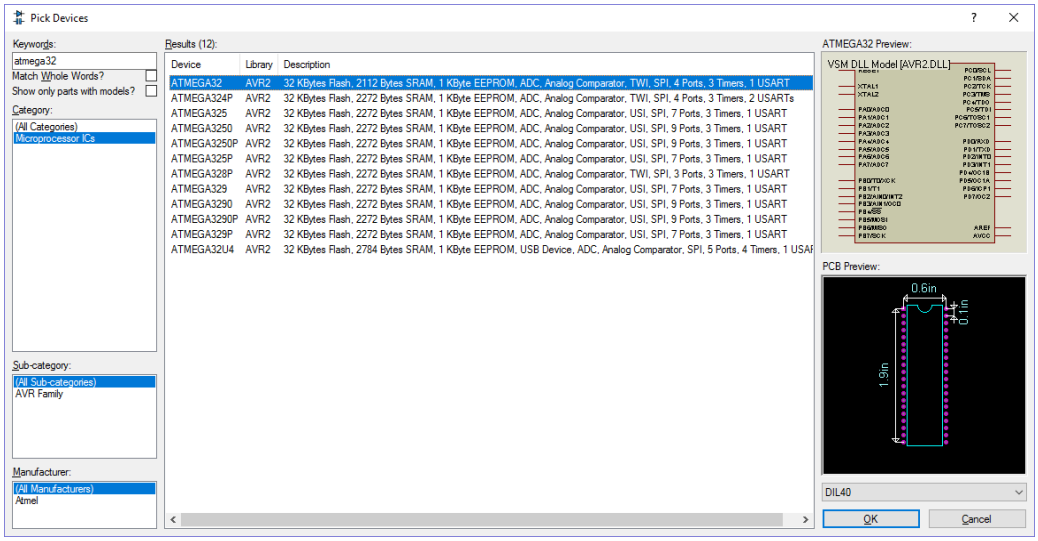


Рисунок 3.41 – Пошук МК у бібліотеці компонентів

Зображення обраного МК на схемі моделі наведене на рисунку 3.42.

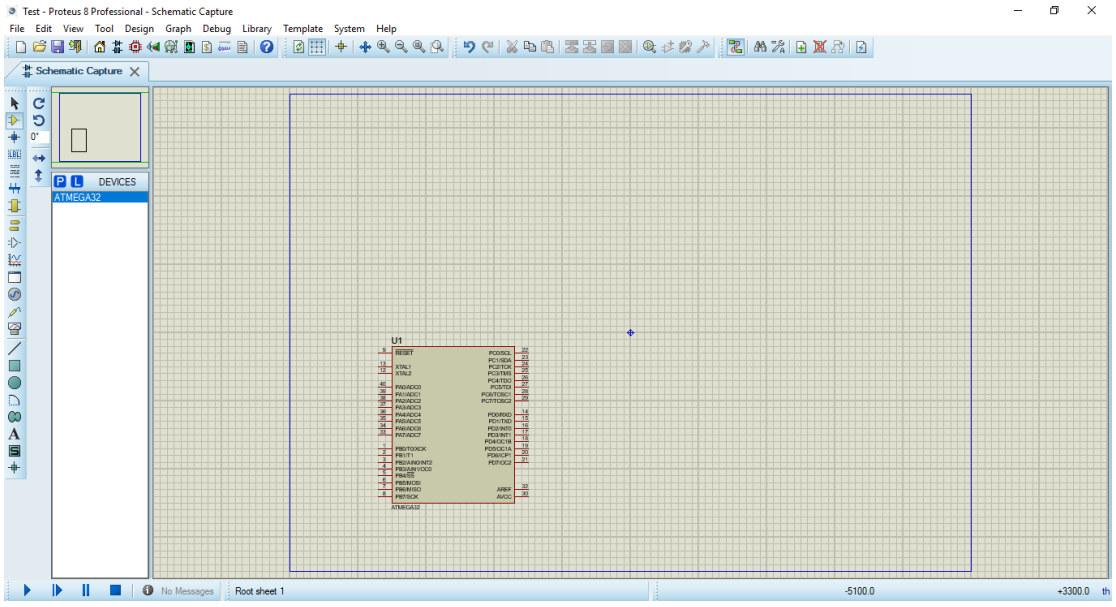


Рисунок 3.42 – Зображення обраного МК на схемі моделі

Оберемо зі списку семисегментний індикатор. Цей процес детальніше був розглянутий вище. Додавши індиатор на схему, з’єднаємо його з мікроконтролером, що вже міститься на схемі (рисунок 3.43).

Далі на схему потрібно додати чотири резистори, як це вказано вище. Оберемо з бібліотеки резистори MINRES10K та змінимо їх опір на 400 Ом кожен. Далі з’єднаємо їх із семисегментним індикатором так, як це показано на рисунку 3.44.

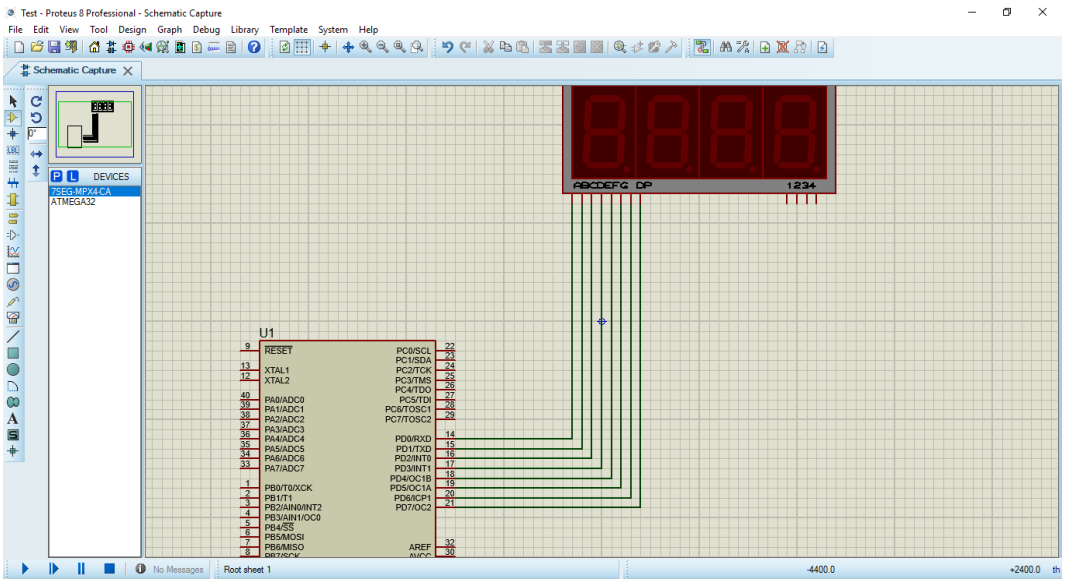


Рисунок 3.43 – З’єднання МК з індикатором

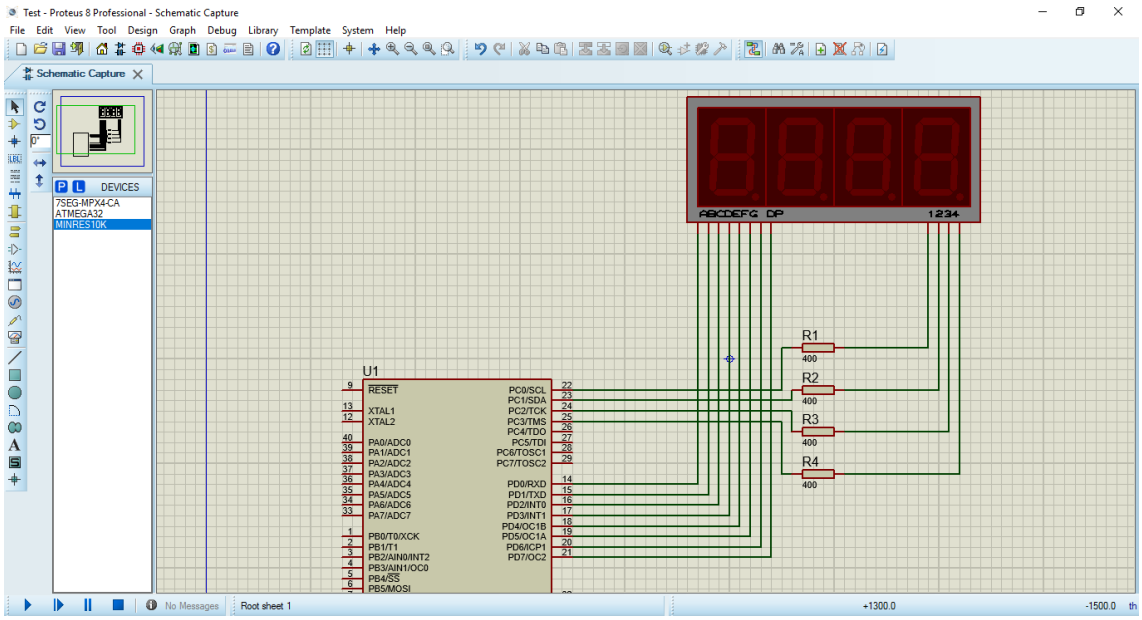


Рисунок 3.44 – Під’єднання резисторів

Додамо на схему потенціометр, знайшовши його у бібліотеці компонентів за ключовим словом POT-HG (рисунок 3.45).

За ключовим словом BATTERY знайдемо блок батарей і додамо їх на схему. Натиснемо двічі по даному блоку та у полі Voltage змінимо напругу елемента живлення на 5V. Розмістимо елементи на схемі, як показано на рисунку 3.46.

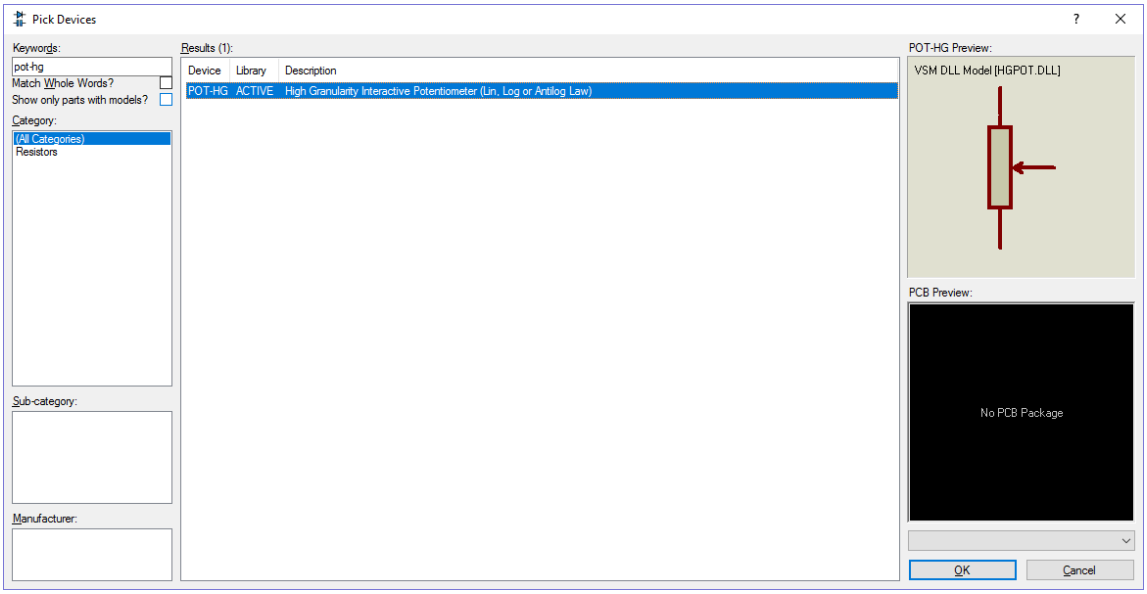


Рисунок 3.45 – Додавання на схему потенціометра

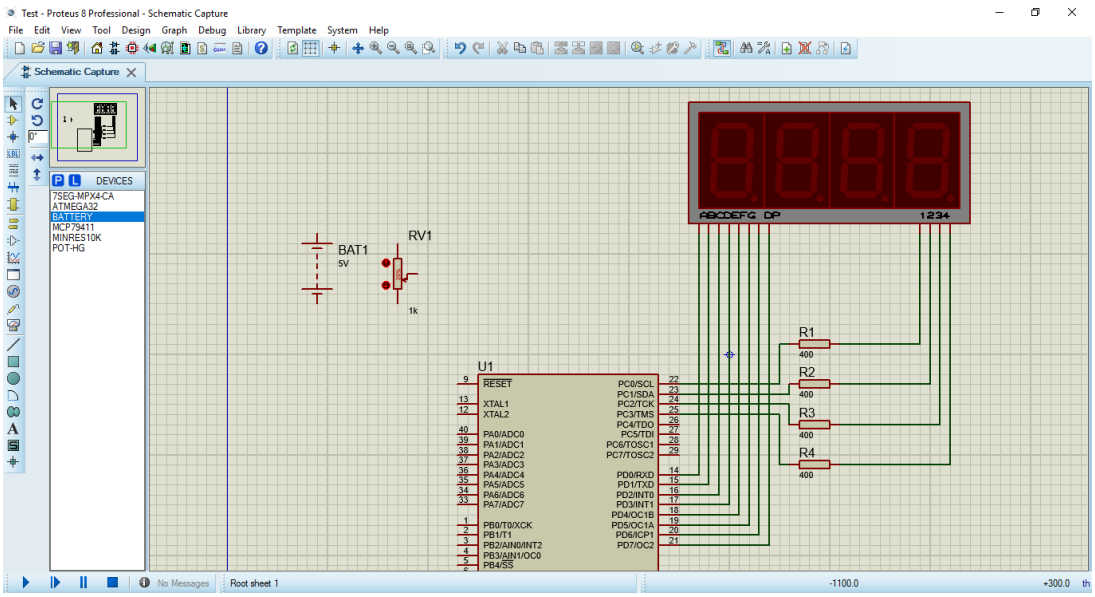


Рисунок 3.46 – Розташування на схемі батарей та потенціометра

Нарешті додамо на схему звичайний вольтметр, виконавши кроки, які описано вище. З’єднаємо останні елементи та мікроконтролер (рисунок 3.47).

Тепер важливо підключити до схеми заземлення. Для цього у боковому меню вікна розробки потрібно обрати підпункт Terminals mode. У цьому меню обираємо компонент GROUND, натискаємо лівою кнопкою миші двічі по місцю схеми, яке потрібно заземлити. В тому місці з’явиться значок заземлення (рисунок 3.48).

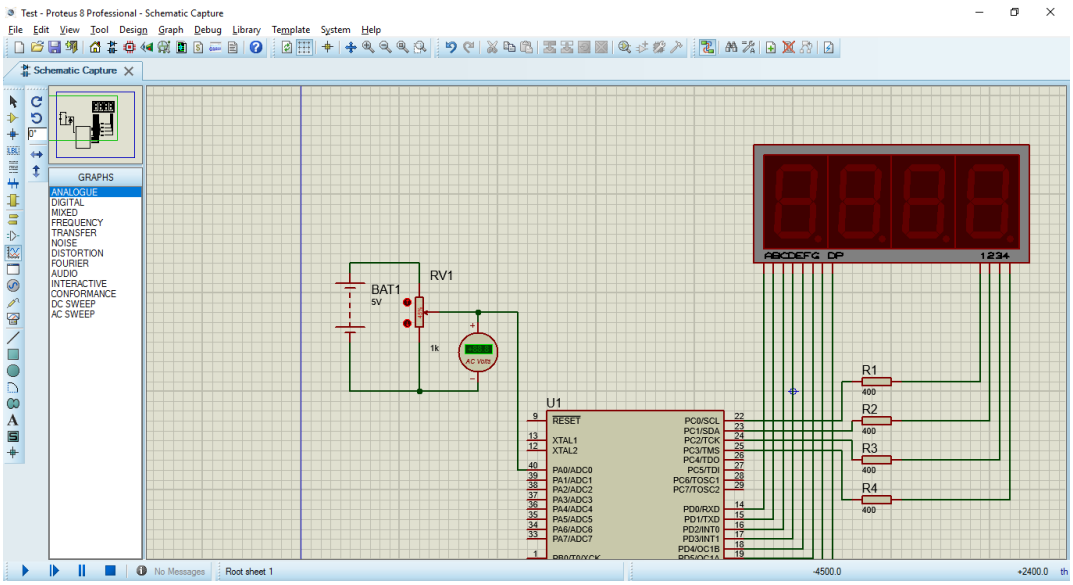


Рисунок 3.47 – Схема з’єднання компонентів

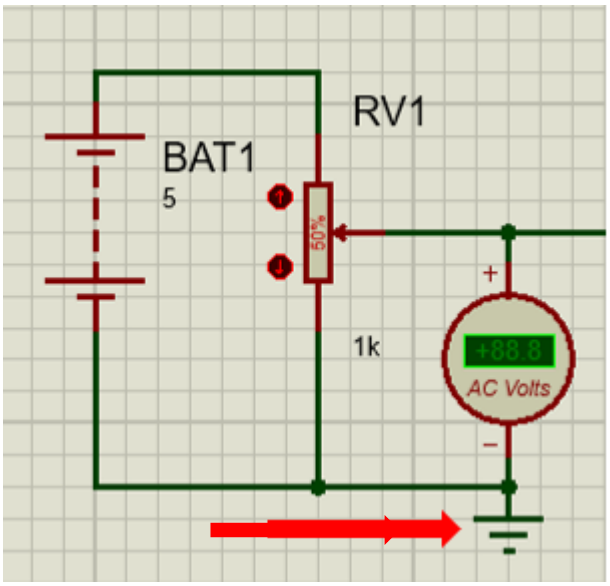


Рисунок 3.48 – Значок заземлення на схемі

З того ж меню потрібно зліва обрати пункт POWER та додати даний елемент у місце, яке показане на рисунку 3.49.

Заключним кроком буде процес завантаження .hex файла у пам’ять мікроконтролера для його подальшого виконання. Даний процес описано у наступному підрозділі.

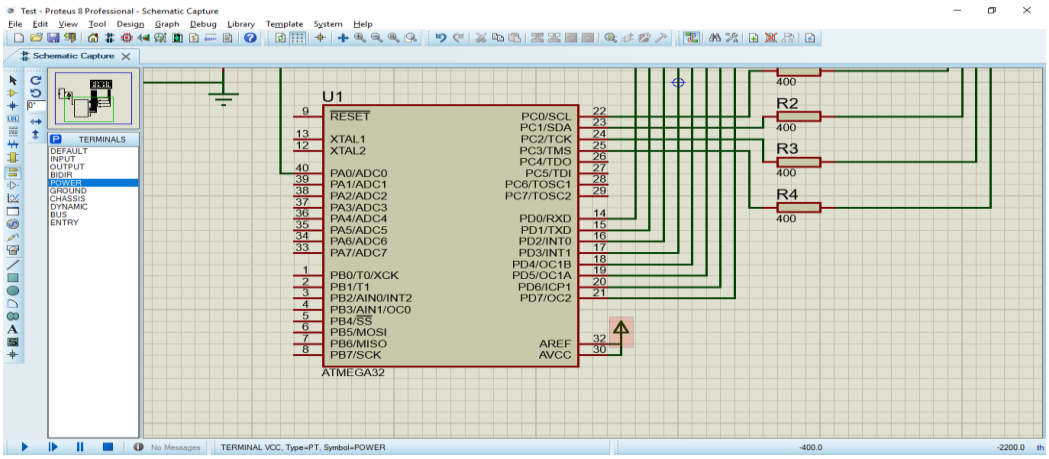


Рисунок 3.49 – Місце під’єднання живлення

3.2.8 Завантаження коду у пам’ять мікроконтролера

Процес завантаження вихідного коду у пам’ять мікроконтролера демонструє наведена нижче послідовність дій та ілюстрацій.

1. Створити новий проект у середовищі ISIS та зберегти його.
2. Додати у проект мікроконтролер з архітектурою AVR, який планується програмувати, наприклад, ATMEGA32.
3. Щоб завантажити вже скомпільований файл з програмою в пам’ять мікроконтролера, потрібно лівою кнопкою миші двічі натиснути по МК, після чого відкриється вікно налаштувань (рисунок 3.50).
4. Далі потрібно натиснути на зображення файлів (виділено червоним на рисунку 3.50), після чого відкриється вікно вибору файлу з комп’ютера (рисунок 3.51). В даному вікні можна знайти та обрати файл у форматі HEX, OBJ або ELF.
5. Після вибору файлу треба зберегти налаштування. Після цього мікроконтролер буде в змозі виконувати завантажену програму.

3.3 Створення проекту та отримання hex-файла в Atmel Studio

Для програмування мікроконтролерів AVR може використовуватися Atmel Studio 7 – інтегроване середовище розробки для програмування та відлагодження програм для мікроконтролерів AVR та AVR32 в операційних системах Windows.

Після шостої версії середовище може працювати як з AVR-мікроконтролерами, так і з системами з ARM-архітектурою.

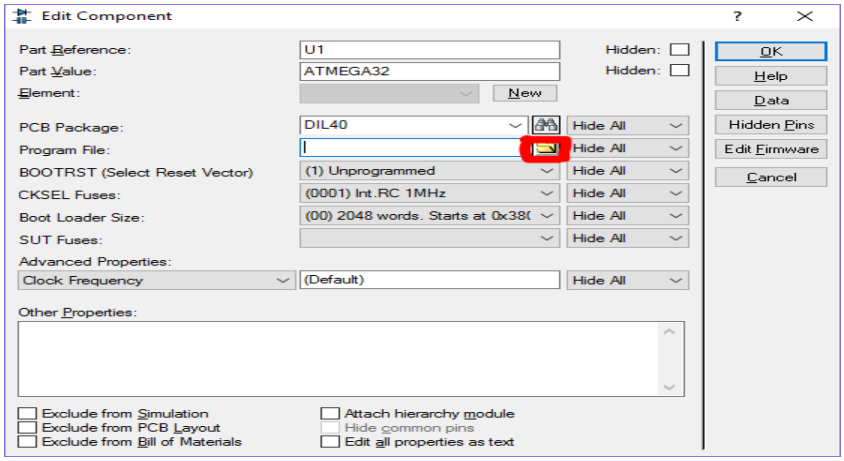


Рисунок 3.50 – Вікно налаштувань мікроконтролера

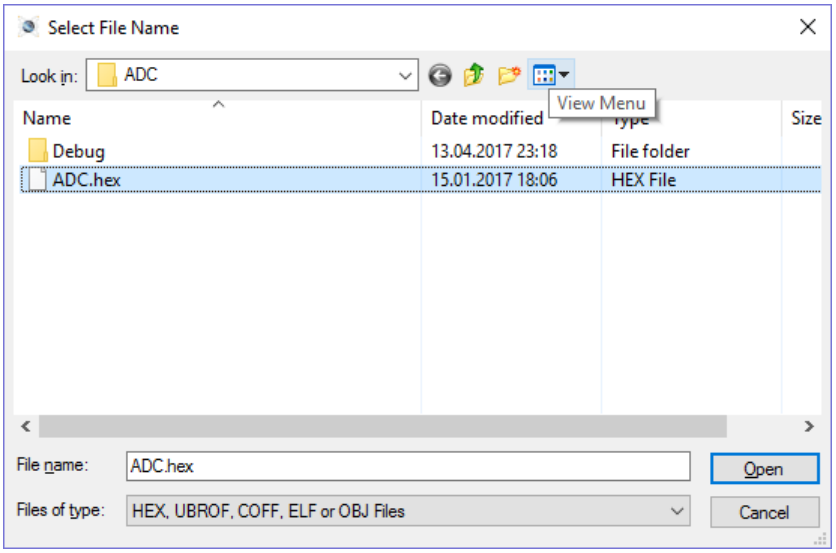


Рисунок 3.51 – Вибір файлу програми

Програмний пакет AVR Studio розробляється компанією Atmel з 2004 року. Починаючи з версії 6.0, компанія змінила назву програми на Atmel Studio та додала можливість програмувати системи на базі ARM-архітектури. Раніше існував і фірмовий асемблер під ОС Windows (wavrasm.exe) від Atmel, який поєднував асемблер і редактор, проте, невдовзі після появи AVR Studio, відмовились від його подальшого розвитку.

Atmel Studio містить в собі такі інструмени, як вбудований C/C++-компілятор, симулятор мікропроцесорної системи для відлагодження

програм, менеджер проектів, редактор коду, модуль внутрішньосхемного відлагодження, а також інтерфейс командного рядка. Крім стандартних елементів, середовище підтримує ряд інших інструментів, таких як компілятор GCC та плагін AVR RTOS операційної системи реального часу. Крім C/C++, середовище дозволяє програмувати також на асемблері. Завдяки зв'язці програмних пакетів Atmel Studio та Proteus від фирми Labcenter Electronics у нас є можливість програмування мікроконтролерів без наявності будь-якої матеріальної бази.

Остання версія Atmel Studio підтримує всі існуючі на сьогоднішній момент 8-бітові, 32-бітові AVR, SAM3 та SAM4-мікроконтролери і включає в себе велику кількість проектів з прикладами. Також доступні старі версії програми.

Програма останньої версії розроблялась за підтримки Microsoft Visual Studio, тому дизайн Atmel Studio схожий на їх продукт.

Дана програма є безкоштовною та доступна для завантаження з офіційного сайту виробника (рисунок 3.52).

Рисунок 3.52 – Зовнішній вигляд програми

Використання для моделювання в PROTEUS мікроконтролерів AVR було розглянуто вище у 3.2.6.

Нижче розглянуто використання програми Atmel Studio 7.

Після запуску файлу інсталятора з'явиться вікно привітання. Перед початком інсталяції з'явиться вікно ліцензійного погодження з правилами (рисунок 3.53). Погоджуємося з правилами та пройшовши ще декілька стандартних екранів

встановлюємо програму. Під час встановлення програма може просити дозволу для дозавантаження необхідих компонентів.

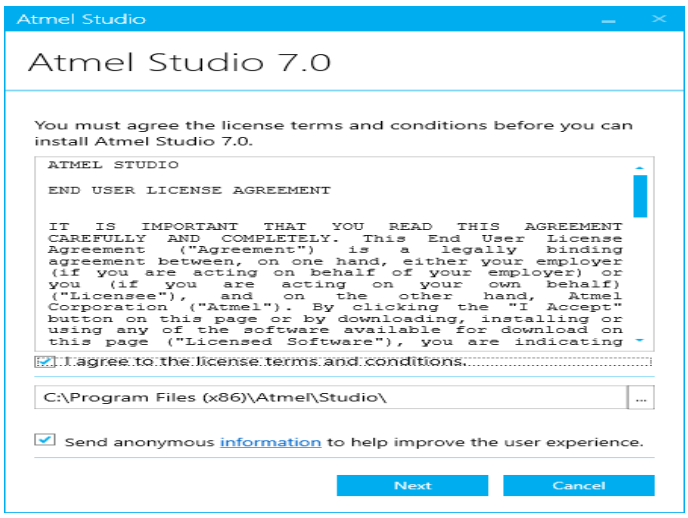


Рисунок 3.53 – Вікно ліцензійного погодження з правилами

Після запуску студії користувач потрапляє на стартову форму. На ній користувач може почати роботу із створення нового проекту або відкрити існуючий (рисунок 3.54).

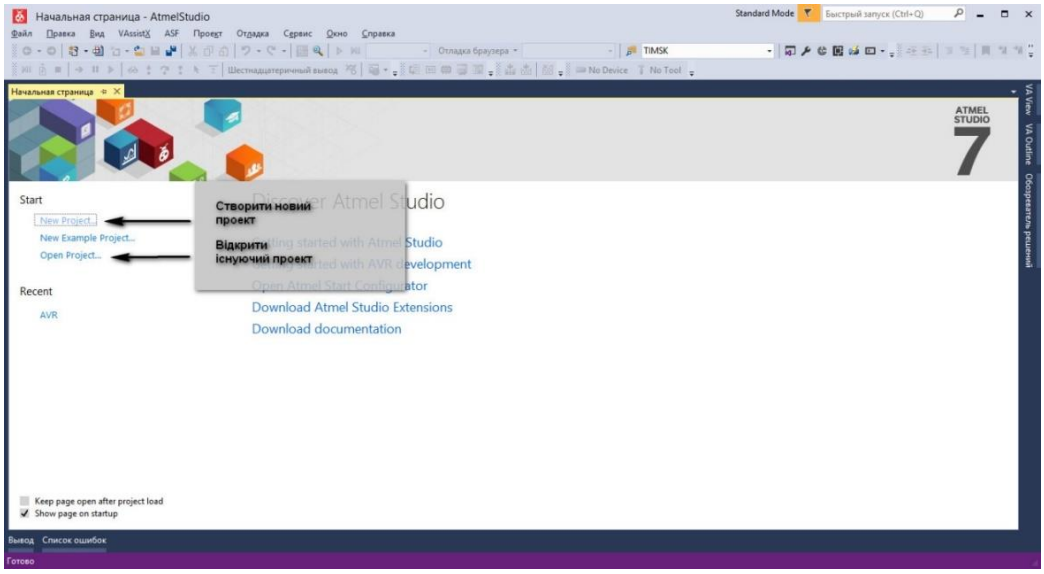


Рисунок 3.54 – Початкове вікно Atmel studio

Створимо новий проект. Для цього виберемо New Project, після чого потрапимо на форму параметрів (рисунок 3.55).

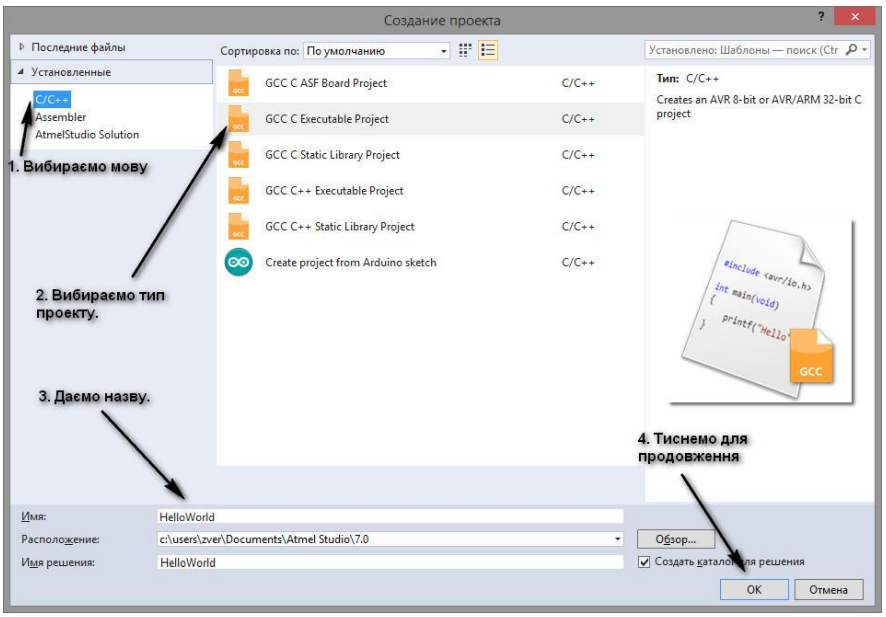


Рисунок 3.55 – Меню вибору мови, типу проекту та назви

З області шаблонів (вкладки встановлені) необхідно вибрати C/C ++ в якості нашого шаблону. В області з типами проектів вибираємо «Виконуваний проект». Після цього в нижній частині вікна даємо назву для нашого рішення та вибираємо його місце розташування. Тиснемо «ОК» та переходимо до вибору мікроконтролера (рисунок 3.56).

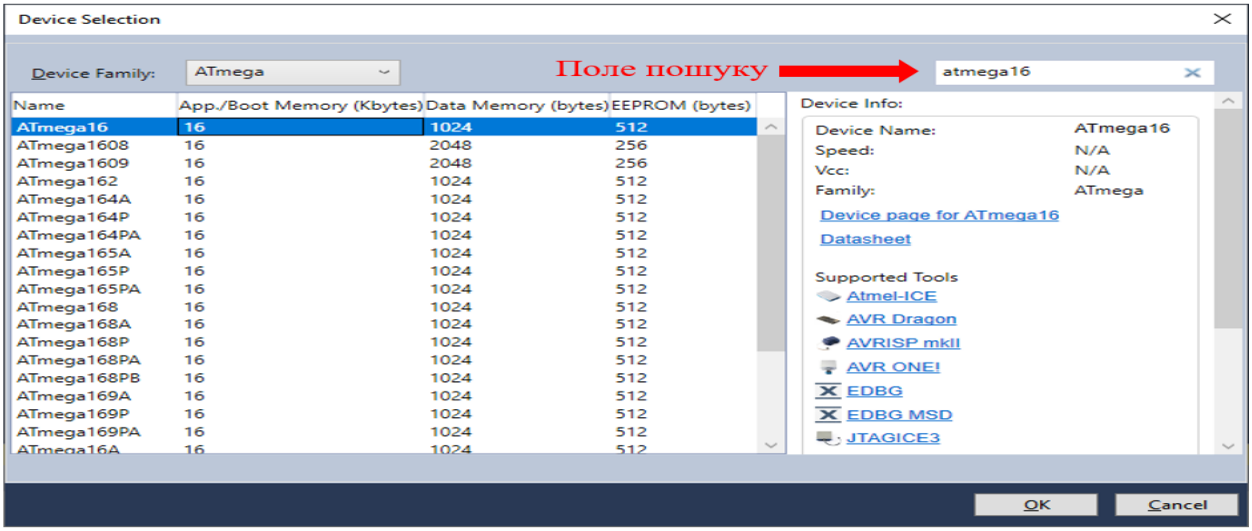


Рисунок 3.56 – Меню вибору мікроконтролера

Як приклад, вибираємо ATmega16 та будемо використовувати даний мікроконтролер як цільовий пристрій, для якого буде створюватись програма, яка

виконується. Після цього необхідно натиснути «ОК». Atmel Studio створить проект та покаже вікно середовища розробки (рисунок 3.57).

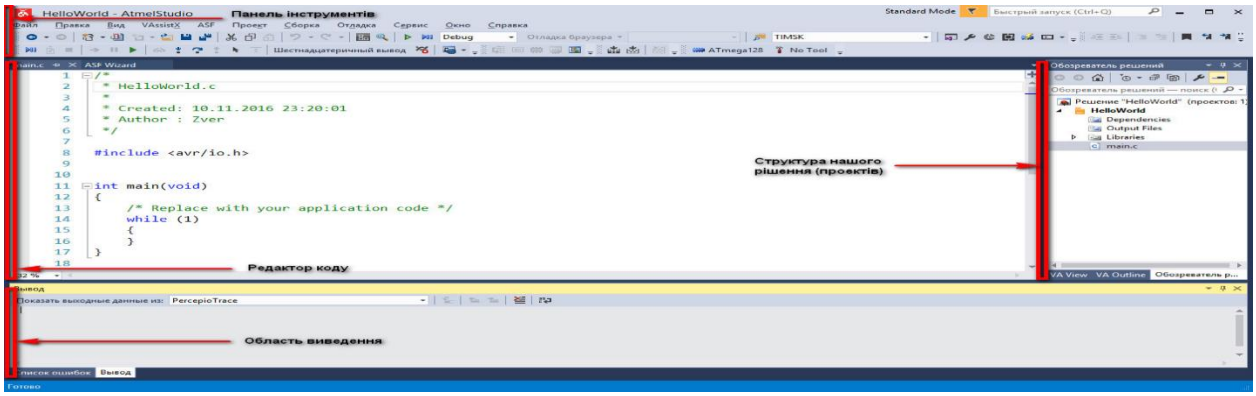


Рисунок 3.57 – Головне вікно середовища розробки

Як бачимо, середовище можна умовно поділити на зони. Зверху розташовується панель інструментів, посередині головна область коду, знизу – область виводу результату, справа – вікно перегляду структури проекту. Звісно, середовище можна налаштувати по-іншому, вибравши необхідні вікна для відображення.

Щоб створити hex-файл у Atmel studio 7 (рисунок 3.58) для проекту на мові С по-перше потрібно упевнитися, що в проекті існує файл з функцією main(), а також, що всі бібліотеки підключені.

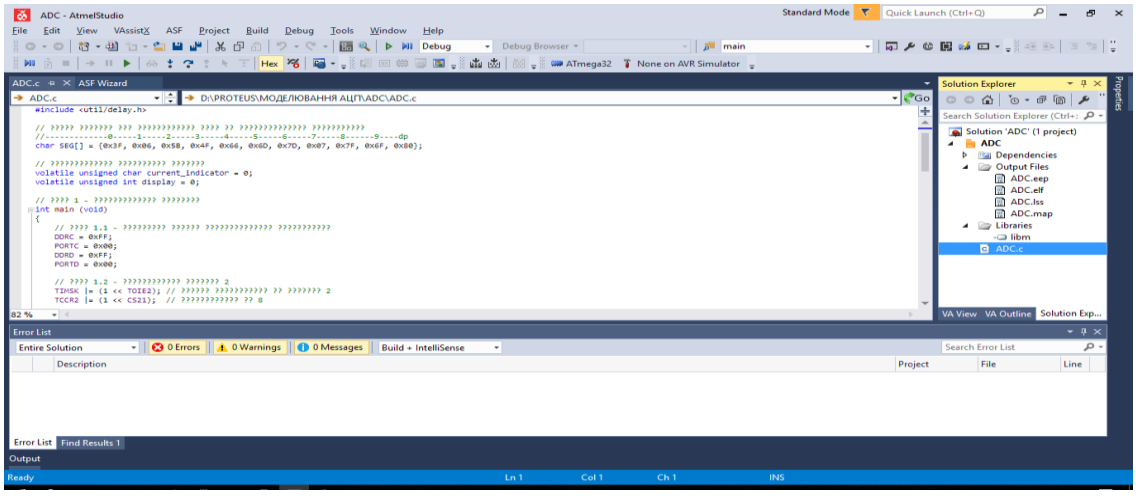


Рисунок 3.58 – Загальний вигляд середовища розробки

Після цього потрібно натиснути правою кнопкою миші на даний проект у меню Solution explorer, що знаходиться в основному вікні справа за замовчуванням (рисунок 3.59). В меню потрібно обрати пункт Build (збудувати) та зачекати декілька секунд.

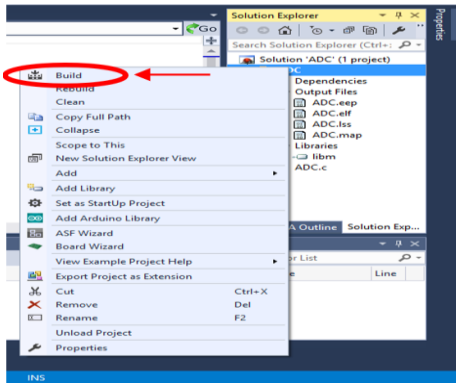


Рисунок 3.59 – Меню налаштувань проекту

Компілятор може вивести декілька зауважень (рисунок 3.60), які, проте, не заважають використовувати програму та її файли.

Рисунок 3.60 – Попередження від компілятора

Також буде автоматично згенеровано hex-файл для проекту, який збережеться у папку з проектом, як показано на рисунку 3.61.

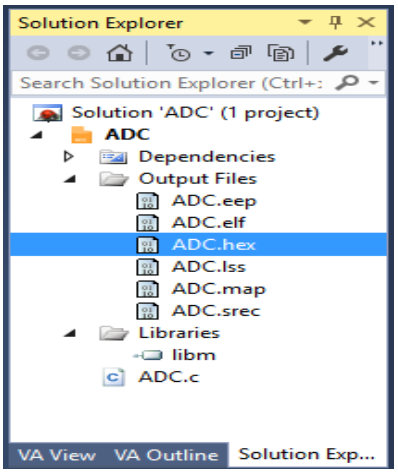


Рисунок 3.61 – Новий файл

Дані з файлу одразу можна продивитися, якщо двічі натиснути лівою кнопкою по файлу у списку (рисунок 3.62).

```
:10000000010E0A0E6B0E0E8E2F0E003C0C895319669
:100010000D92A036B107D1F720E0A0E6B0E001C014
:080020001D92A036B207E1F7C2
:00000001FF
```

Рисунок 3.62 – Дані з файлу

Контрольні запитання та завдання

5) Як виконується створення проекту та отримання hex-файла в Atmel Studio?
6) Як виконується налагодження програми в Atmel Studio?
7) Назвіть призначення та особливості програми PROTEUS VSM.
8) Опишіть послідовність створення нового проекту у Proteus.
9) Опишіть особливості інтерфейсу середовища ISIS.
10) Як виконується завантаження коду у пам’ять мікроконтролера?

4 ЛАБОРАТОРНА РОБОТА №1. ДОСЛІДЖЕННЯ КОМАНД ПЕРЕСИЛАННЯ, АРИФМЕТИЧНИХ, ЛОГІЧНИХ, РОБОТИ З ОКРЕМИМИ БІТАМИ ТА ЗСУВУ

Тема: команди пересилання даних, арифметичні, логічні, роботи з окремими бітами та зсуву.

Мета: користуючись налагоджувачем AVR Studio 4 дослідити виконання команд пересилання даних, арифметичних, логічних, роботи з окремими бітами та зсуву у покроковому режимі.

4.1 Порядок виконання лабораторної роботи

1) Вивчити теоретичні відомості з теми «Команди пересилання даних, арифметичні, логічні, роботи з окремими бітами та зсуву». Вміти коментувати команди, які наведено в таблиці 1.3.
2) Вивчити програмно-налагоджувальні засоби (ПНЗ) у AVR-Studio 4.

3) Ввести у симулятор AVR-Studio 4 програму з наведеними нижче прикладами команд: пересилання даних, арифметичних, логічних команд, команд зсуву та команд роботи з бітами. За допомогою ПНЗ проаналізувати виконання програми з цими командами. Переконатися в правильному виконанні програми. При негативному результаті здійснити зміну програми та повторити перевірку.
4) Розробити алгоритм для виконання індивідуального завдання.
5) Розробити програму для виконання індивідуального завдання.
6) Ввести програму індивідуального завдання у симулятор AVR-Studio 4.
7) За допомогою ПНЗ проаналізувати виконання індивідуальної програми. Переконатися в правильному виконанні індивідуального завдання, при негативному результаті здійснити зміну алгоритму або програми, повторити перевірку.
8) Роздрукувати лістинг правильно працюючої програми.
9) Відповісти на контрольні питання.

4.2 Команди пересилання даних

Стислі теоретичні відомості

Команди цієї групи призначено для пересилання вмісту комірок, що розташовані в просторі адрес статичної пам’яті даних (РЗП, РВВ та СОЗП). Поділ простору адрес на три частини (РЗП, РВВ, СОЗП) зумовлює різноманітність команд даної групи. Пересилання даних, яке виконується командами групи, може виконуватись в наступних напрямках:

- РЗП <=> РЗП;
- РЗП <=> РВВ;
- РЗП <=> статична пам’ять даних (РЗП, РВВ та СОЗП), 4 види адресації.

Також до даної групи можна віднести команди роботи зі стеком PUSH та POP (відсутні в AT90S1200), що дозволяють зберігати у стеку і відновлювати зі стека вміст РЗП.

На виконання команд даної групи потрібно від одного до трьох машинних циклів в залежності від команди (таблиця 1.3).

Нижче на рисунку 4.1 представлено типи команд пересилання, які визначають як вони будуть представлені у пам’яті програм процесора. Тип кожної команди зазначено в 6-му стовпці таблиці 1.3.

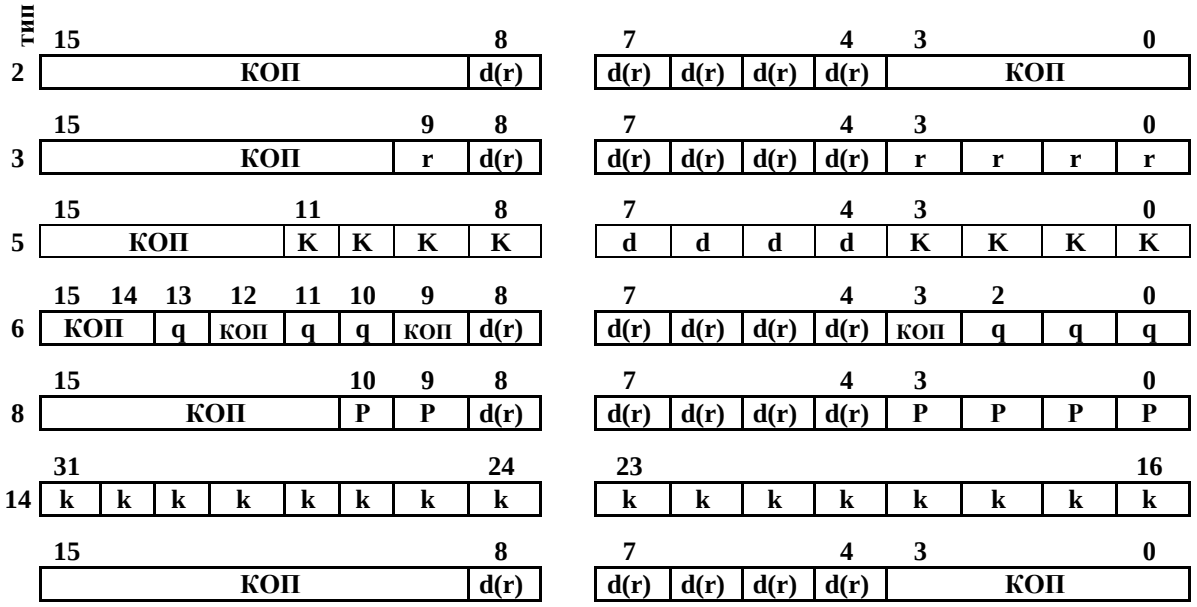


Рисунок 4.1 – Типи команд, що використовуються в командах пересилання даних

Нижче наведено приклад аналізу однієї з команд пересилання: MOV Rd, Rr.

Опис: команда копіює (пересилає) вміст одного з регістрів (Rr) в інший (Rd). Регістр-джерело Rr залишається незмінним. Регістр-приймач Rd завантажується копією регістра Rr. Номери регістрів Rr, Rd кодуються п’ятьма бітами і можуть приймати значення від 0 до 31 (2^5-1).

Приведена команда відповідає третьому типу (рисунок 4.1). Код операції (КОП) надає пристрою керування мікроконтролером інформацію про те, які дії потрібно виконати при виконанні команди. Його можна визначити за допомогою симулятора AVR STUDIO 4.

Якщо, наприклад, створити проект, який складається лише з однієї команди MOV R1, R31, та скомпілювати її, то область пам’яті програм матиме вигляд, який зображено на рисунку 4.2.

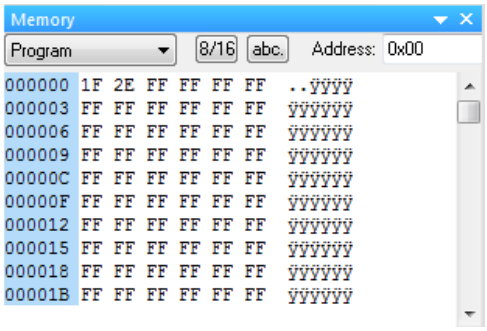


Рисунок 4.2 – Вигляд пам’яті програм після компіляції команди MOV R1, R31

Нижче наведено машинний код команди та зроблено його аналіз.

Номера бітів	15							8	7							0
Машинний код команди MOV R1, R31	КОП						r	d	d	d	d	d	r	r	r	r
Код команди у двійковій формі	0	0	1	0	1	1	1	0	0	0	0	1	1	1	1	1
Код команди у шістнадцятковій формі	2			E				1				F				

Машинний код команди, який представлено у шістнадцятковому коді: 2E 1F. Слід зазначити що після компіляції в пам’яті програм спочатку розміщається другий байт команди, потім – перший, тобто в пам’яті програм бачимо запис: 1F 2E.

З отриманого машинного коду команди можемо визначити, що КОП = 001011, регістр-джерело r = $11111_2 = 31_{10}$, регістр-приймач d = $00001_2 = 1_{10}$.

Знайдені значення r та d дозволили перевірити правильність наведеного вище аналізу команди. Розглядалась команда MOV R1, R31, тому отримані значення r = 31, d = 1, дозволяють зробити висновок, що аналіз проведено вірно.

Довжина команди: 1 слово (2 байти).

Час виконання команди при тактовій частоті 12МГц дорівнює 1 такту, або 1 мкс.

Нижче наведено приклад програми з використанням окремих команд пересилання даних.

Приклад програми з використанням команд пересилання даних

```
ldi R17, $32   ; Завантажити константу (R17 = $32)
ldi R23, $45   ; Завантажити константу (R23 = $45)
mov R17, R23; Копіювати регістр        (R17 = R23)
ldi R26, $60   ; Завантажити константу (R26 = $60)
ldi R27, $00   ; Завантажити константу (R27 = $00)
ld R3, X       ; Непряме завантаження  (R3 = X)
ld R4, X+      ; Непряме завантаження з постінкрементом (R4 = X, X = X + 1)
ld R2, -X      ; Непряме завантаження з преддекрементом  (X = X - 1, R2 = X)
ldi R28, $61   ; Завантажити константу (R28 = $61)
ldi R29, $00   ; Завантажити константу (R29 = $00)
ld R5, Y       ; Непряме завантаження (R5 = Y)
ld R6, Y+      ; Непряме завантаження з постінкрементом (R6 = Y, Y = Y + 1)
ld R7, -Y      ; Непряме завантаження з преддекрементом (Y = Y - 1, R7 = Y)
ldd R8, Y+2    ; Непряме завантаження зі зміщенням, в R8 завантажити вміст SRAM за
               ; адресою (Y + 2) = $63
ldi R30, $62   ; Завантажити константу (R30 = $62)
ldi R31, $00   ; Завантажити константу (R31 = $00)
ld R11, Z      ; Непряме завантаження (R11 = Z)
ld R10, Z+     ; Непряме завантаження з постінкрементом (R10 = Z, Z = Z + 1)
ld R12, -Z     ; Непряме завантаження з преддекрементом (Z = Z - 1, R12 = Z)
ldd R18, Z+5   ; Непряме завантаження зі зміщенням (R18 <= (Z + 5))
lds R9, $001   ; Безпосереднє завантаженн (R9 = $001)
lds R10, $022 ; Безпосереднє завантаження (R10 = $022)
lds R11, $080 ; Безпосереднє завантаження (R11 = $080)
st X, R2       ; Непряме збереження (X = R2)
st X+, R10     ; Непряме збереження з постінкрементом (X = R10, X = X + 1)
st -X, R11     ; Непряме збереження з преддекрементом (X = X - 1, X = R11)
st Y, R7       ; Непряме збереження (Y = R2)
st Y+, R6      ; Непряме збереження з постінкрементом (Y = R6, Y = Y + 1)
st -Y, R4      ; Непряме збереження з преддекрементом (Y = Y - 1, Y = R16)
std Y+2, R16   ; Непряме збереження зі зміщенням (Y + 2) <= R16)
st Z, R6       ; Непряме збереження (Z = R6)
st Z+, R7      ; Непряме збереження з постінкрементом (Z = R7, Z = Z + 1)
st -Z, R17     ; Непряме збереження с преддекрементом (Z = Z - 1, Z = R17)
std Z+4,R30    ; Непряме збереження зі зміщенням          (Z + 4) <= R30
sts $82, R7    ; Безпосереднє збереження  ($82 <= R7)
ldi R30, $01   ; Завантажити константу (R30 = $01)
lpm            ; Завантаження з пам’яті програм  (R0 = Z)
in R2, UDR     ; Зчитування порта (R2 <= UDR)
out $16, R7    ; Запис у порт ($16 (PINB) <= R7)
ldi R25, $DE   ; Завантажити константу (R25 = $DE)
out $3D, R25   ; Запис в порт ($3D (SP) <= R25)
push R4        ; Занесення регістра в стек  (STACK <= R4; SP = SP-1)
pop R9         ; Витягнення регістра зі стека (SP = SP+1, R9 <= STACK)
```

Нижче наведено варіанти індивідуальних завдань до використання команд пересилання.

Варіанти індивідуальних завдань до використання команд пересилання

Таблиця 4.1 – Завдання до використання команд пересилання

№	Текст індивідуального завдання
1	a) Вміст порту D (PIND) помістити в пам’ять даних за адресою 7F. b) Використовуючи непряму адресацію, занести вміст R0, R2, R5 в пам’ять даних, починаючи з адреси 6E. c) Вміст регістрів R0…R3 помістити в стек, попередньо встановивши вершину стека за адресою EF.
2	a) Вміст R5 записати в PORTD, використовуючи непряму адресацію та команду OUT. b) У регістри R3, R4 завантажити два байти з пам’яті програм, починаючи з адреси 01H. c) Зберегти вміст таймера/лічильника1 у пам’яті даних з адресою 6E, використовуючи непряму і безпосередню адресацію.
3	a) Заповнити регістри R0…R7 вмістом комірок пам’яті даних, починаючи з 60H, використовуючи непряму адресацію. b) Використовуючи команди роботи зі стеком, занести вміст пам’яті даних з адресою від 64H до 60H у довільно обрані регістри. c) Записати число 23H в PORTD.
4	a) Вміст молодшого байта таймера/лічільника1 занести в пам’ять даних за адресою 80H. b) Записати число 44H в PORTD використовуючи безпосередню адресацію, непряму адресацію і команди роботи зі стеком. c) Завантажити два байти в таймер/лічильник1 з пам’яті програм, починаючи з адреси 06H.
5	a) Вміст приймача/передавача (UDR) помістити в пам’ять даних. b) Організувати під стек пам’ять даних з адреси 70H і помістити в стек вміст таймера/лічильника1 та порту D (PIND). c) За адресою 60H в пам’яті даних знаходиться таблиця 3x5. Перші 3 елементи останнього рядка помістити в пам’ять даних, починаючи з адреси 80H.
6	a) Обміняти вміст комірки пам’яті даних 74H та таймеру/лічильника 0. b) Зберегти у стеку вміст приймача УАПП (UDR) та T/C1. c) Діагональні елементи матриці 3x3, що розташована в пам’яті даних за адресою 80H, помістити в регістри R1…R5.
7	a) Таймер/лічильник1 ініціалізувати значенням 4782D. b) Записати вміст регістра R2 в регістр R3 не використовуючи команду MOV. c) Вміст регістрів R5…R7 помістити в стек, попередньо встановивши вершину стека за адресою AF.
8	a) Вміст порту D (PIND) помістити в пам’ять даних за адресою 61H. b) В регістри R5, R6 завантажити два байти з пам’яті програм, починаючи з адреси 05H. c) Зберегти у стеку вміст порту D (PIND) та T/C1.
9	a) Записати вміст регістра R5 в регістр R4 не використовуючи команду MOV. b) Використовуючи команди роботи зі стеком, занести вміст пам’яті даних з адреси 64H до адреси 60H в довільно обрані регістри. c) Записати число 1FH в T/C0 використовуючи бузпосередню адресацію, непряму адресацію і команди работи зі стеком.

Закінчення таблиці 4.1

№	Текст індивідуального завдання
10	a) Організувати під стек пам'ять даних з адреси 80H і поместити в стек вміст таймера/лічильника1 та порту D (PIND). b) Зберегти вміст таймера/ лічильника1 в пам'яті даних за адресою 6E, використовуючи непряму і безпосередню адресацію. c) Записати число 3AH в PORTD.

4.3 Арифметичні та логічні команди, команди зсуву та роботи з окремими бітами

Стислі теоретичні відомості

Команди даної групи наведено у таблиці 1.3. Формати команд наведено на рисунку 1.22, а їх опис – у 1.4.7.

Нижче наведено приклад програми з використанням арифметичних, логічних команд та команд зсуву.

Приклад програми з використанням арифметичних, логічних команд та команд зсуву

```
ldi R17, $32    ; Завантажити константу (R17 = $32)
ldi R23, $45    ; Завантажити константу (R17 = $45)
add R17,R23  ; Додавання без перенесення (R17 = R17 + R23)
sec      ; Встановлення прапорця C  (C = 1)
adc R17,R23   ; Додавання с перенесенням (R17 = R17 + R23 + C)
ldi R24, $15    ; Завантажити константу (R24 = $15)
ldi R25, $13    ; Завантажити константу (R25 = $13)
adiw R24, 62 ; Додавання до двох байт константи (ZH:ZL = ZH:ZL+62)
sub R17, R23; Віднімання без перенесення (R17 = R17 - R23)
subi R17, 254 ; Віднімання константи (R17 = R17 – 254)
sbc R17, R23 ; Віднімання з перенесенням (R17 = R17 - R23 – C)
sbci R17, 254 ; Віднімання константи з перенесенням (R12 = R12 - 254 – C)
sbiw R24, 35 ; Віднімання з двох байт константи (ZH:ZL = ZH:ZL-35)
and R17, R23 ; Логічне І (R17 = R17 * R23)
andi R17, 230 ; Логічне І з константою (R17 = R17 * 230)
or R23, R17    ; Логічне АБО (R23 = R23 v R17)
ori R23, 45     ; Логічне АБО з константою (R23 = R23 v 45)
eor R17, R23 ; Логічне «виключне АБО»  (R17 = R17 ⊕ R23)
com R17         ; Побітова інверсія (R17 = $FF - R17)
neg R23         ; Зміна знака (додатковий код) (R23 = $00 - R23)
sbr R17, $55 ; Встановлення  парних бітів в регістрі (R17 = R17 v $55)
cbr R23, $AA ; Скидання непарних бітів  в регістрі (R23 = R23 * ($FF - $AA))
```

```
inc R17          ; Інкремент значення регістра (R17 = R17 + 1)
dec R23          ; Декремент значення регістра (R23 = R23 – 1)
tst R17          ; Перевірка на нуль або від'ємність (R17 = R17 * R17)
clr R23          ; Скидання регістра  (R23 = $00)
ser R17          ; Встановлення регістра (R17 = $FF)
swap R24         ; Обмін тетрадами регістра R24
ldi R25, $05     ; Завантаження константи в регістр (R25 = $05)
ldi R26, $C0     ; Завантаження константи в регістр (R26 = $C0)
lsl R25          ; Логічний зсув вліво регістра R25
lsr R26           ; Арифметичний зсув вліво R26
ldi R27, 12      ; Завантажити константу (R27 = 12)
ldi R28, $F4     ; Завантажити константу (R28 = $F4)
lsr R27          ; Логічний зсув вправо регістра  R27
asr R28        ; Арифметичний зсув вправо регістра R28
ldi R29, $A5 ; Завантаження константи в регістр  (R29 = $A5)
rol R28          ; Циклічний зсув вліво регістра R28
ror R28          ; Циклічний зсув вправо  регістра  R28
```

Нижче наведено варіанти індивідуальних завдань до використання арифметичних, логічних команд та команд зсуву.

Варіанти індивідуальних завдань до використання арифметичних, логічних команд та команд зсуву

Таблиця 4.2 – Завдання до використання арифметичних, логічних команд та команд зсуву

№	Текст індивідуального завдання
1	a) Вміст регістра R3 і значення за адресою пам'яті даних 6Ch додати і помістити за адресою 7Ch. b) Додати двухбайтне число, що міститься в R3 і R4 і двухбайтне число, що міститься в пам'яті даних 60h и 61h (попередньо помістивши туди значення). c) Відняти від двохбайтного числа, що міститься в PORTC і PORTD, число яке записано за адресою 0Dh і 1Dh попередньо помінявши в них тетради.
2	a) Відняти 51 з регістра R3, результат помістити за адресою 8Ch. b) Додати двобайтне число, що міститься за адресою 1Ch, 2Ch і двобайтне число 34 F4. c) Помножити двобайтне число на 4 і підрахувати кількість одиниці в отриманому результаті.

Закінчення таблиці 4.2

3	a) Знайти суму 3-х членів ряда натуральних чисел, починаючи з числа #03d b) Додати старшу і молодшу тетради регістра R26, результат записати в неупакованому форматі за адресою 70h. c) Додати два числа розташованих за адресою 0Dh і 0Ch, результат помістити в стек, попередньо встановивши вершину стека за адресою EF
4	a) Обчислити суму чисел -10, + 9, - …-2 +1. b) Додати два однобайтних числа в прямом коді (7-й біт знаковий). Результат – в молодші тетради регістрів R26-R29 . c) Дана матриця 1*3, яка розташована починаючи з адреси 60h. Знайти скільки елементів цієї матриці однакові з першим елементом. Результат записати в R29.
5	a) Обчислити суму від'ємних чисел від -1 до -10. b) Просумувати два числа, що розташовані за адресою 7Dh і 7Eh, результат помістити в регістр R29. c) Знайти суму діагональних (головної і додаткової діагоналей) елементів матриці 2*2, що розташована починаючи з адреси 60h.
6	a) Знайти суму цілих чисел, розташованих в діапазоні пам'яті даних, що обмежений вмістом PORTD і R0. (PORTD і R0 не мають нульових значень) b) Додати два однобайтних числа в прямому коді (7-й біт знаковий). Результат помістити в пам'ять даних. c) Виконати циклічний зсув вліво двобайтного числа, записаного за адресою 7Dh і 7Eh.
7	a) Вміст PORTB і PORTC додати і помістити за адресою 70h в неупакованому форматі. b) Додати двобайтне число, що розташовано в Т/Л1 і двобайтне число, розташоване в двох довільних регістрах. c) Відняти з двобайтного числа, розташованого в R23 і R24, число 34DEh, попередньо побітово проінвертувавши регістри.
8	a) Додати дві двобайтні константи, використавши команду NEG. b) Знайти суму 5623h і числа, розташованого в комірках 6Dh і 7Eh. c) Помістити масив 2x2, заповнений значеннями abs(i-j), де i – номер строки 0..1 і j – номер стовпця 0…1 починаючи з адреси 60h.
9	a) Помножити двобайтне число, розташоване в R22, на 5. b) В робочому регістрі R29 записано число в упакованому ДДК форматі. Додати тетради і результат помістить в пам'ять даних. c) Додати два двійкові числа з цілими частинами в R12 і R13 і десятковою в R14 і К15 відповідно.
10	a) Обчислити різницю чисел 77h і EEh. Результат помножити на 2. b) Обчислити кількість регістрів серед (R12…R16), в яких записано число 23h. c) Відняти від однобайтного числа двухбайтне.

Нижче наведено приклад програми з використанням команд роботи з бітами.

Приклад програми з використанням команд роботи з бітами

```
sbr R 30, $ 0F; Встановлення молодшої тетради регістра R 30. Старша тетрада без змін.
cbr R 30, $ 0F; Скидання молодшої тетради регістра R 30. Старша тетрада без змін.
sbi $ 1C, 0; Встановлення біта читання в регістрі EECR
cbi $ 12, 7; Скидання сьомого біта у порту D
bset 7       ; Встановлення прапорця регістра SREG (SREG (s = 7) <= 1)
bld R1, 7   ; Завантаження біта T в біт регістра (R1 (b = 7) <= T
```

```
bst R1, 7    ; Збереження біта регістра в біті T (T <= R1 (b = 7))
set          ; Встановлення прапорця перенесення (C <= 1)
clc          ; Очищення прапорця перенесення (C <= 0)
sen          ; Встановлення прапорця від'ємного числа (N <= 1)
cln          ; Очищення прапорця від'ємного числа (N <= 0)
sez          ; Встановлення прапорця нуля (Z <= 1)
clz          ; Скидання прапорця нуля (Z <= 0)
sei          ; Встановлення прапорця переривань (I <= 1)
cli          ; Скидання прапорця переривань (I <= 0)
ses          ; Встановлення прапорця числа зі знаком (S <= 1)
cls          ; Скидання прапорця числа зі знаком (S <= 0)
sev          ; Встановлення прапорця переповнення (V <= 1)
clv          ; Скидання прапорця переповнення (V <= 0)
set          ; Встановлення прапорця T (T <= 1)
clt          ; Скидання прапорця T (T <= 0)
seh          ; Встановлення прапорця допоміжного перенесення (H <= 1)
clh          ; Скидання прапорця допоміжного перенесення (H <= 0)
nop          ; Немає операції
sleep        ; Режим «Сну» (змешення енергоспоживання)
wdr          ; Скидання вартового таймера
```

Нижче наведено варіанти індивідуальних завдань до використання команд роботи з бітами.

Варіанти індивідуальних завдань до використання команд роботи з бітами

Таблиця 4.3 – Завдання до використання команд роботи з бітами

№	Текст індивідуального завдання
1	a) Скинути перший біт регістра керування таймера/лічильника 1 (TCCR1B). b) Реалізувати комбінаційну двійкову функцію, задану аналітично: $bit5 = \overline{(bit1 * bit4)} + \overline{(bit1 * bit3 + bit3)}$ c) Реалізувати функцію задану у таблиці 4.4.
2	a) Скинути перший біт регістра керування таймера/лічильника 0 (TCCR0B). b) Реалізувати комбінаційну двійкову функцію, задану аналітично: $bit1 = \overline{R21(3) + R20(4)}$ c) Реалізувати функцію задану у таблиці 4.5.
3	a) Встановити третій біт у порту B. b) Реалізувати комбінаційну двійкову функцію, задану аналітично: $bit1 = bit2 \vee \overline{(bit2 \wedge R23(2))} \wedge \overline{bit3}$ c) Реалізувати функцію задану у таблиці 4.6.

Закінчення таблиці 4.3

4	a) Скинути третій біт у порту С. b) Знайти алгебраїчну суму бітів bit1 + bit2 + bit3 + bit4. Результат помістити у у 3 молодших біта регістра R7. c) Реалізувати функцію задану у таблиці 4.7.
5	a) Переставити тетради регістра R3. b) Реалізувати комбінаційну двійкову функцію, задану аналітично: $R11(3) = \overline{(bit1 * bit2 + bit2)} * bit3 + \overline{bit1}$ c) Реалізувати функцію задану у таблиці 4.8.
6	a) Виконати циклічний зсув регістра R18 на 3 розряди вправо. b) Реалізувати комбінаційну двійкову функцію, задану аналітично: $bit1=bit2 \wedge \overline{(PORTD \wedge bit2)} \wedge bit3$ c) Реалізувати функцію задану таблиці 4.9.
7	a) Молодші 3 біта та старший біт регістра R12 помістити у старшу тетраду регістра R22. b) Реалізувати логічну функцію: $R13(0)=\overline{(bit1 * PORTC)} * \overline{R25(1)}$ c) Реалізувати функцію задану у таблиці 4.5.
8	a) Виконати логічний зсув регістра R17 на 5 розрядів вліво. b) Реалізувати логічну функцію: $bit1=\overline{(bit1 \vee R22(2)} \vee bit2) \wedge \overline{bit1}$ c) Записати в область прямоадресуємих бітів перші 3 біта регістрів R22 та R23.
9	a) Виконати арифметичний зсув регістра R17 на 5 розрядів вліво. b) Реалізувати логічну функцію: $PORTB(2) = \overline{R12(7) + bit1}$ c) Просумувати за модулем 2 біти регістра R5. Результат помістити у прапорець додаткового перенесення.
10	a) Проінвертувати перші 5 не зарезервованих біт з області прямоадресуємих біт і логічно підсумувати їх з 3-м бітом порту С. b) Реалізувати логічну функцію: $bit1=(bit1 \wedge bit2 \vee (bit1 \vee bit2)) \wedge bit1 \vee bit2 \vee bit3$ c) Помістити в регістр R12 перші вісім не зарезервованих прямоадресуємих біт.

Зауваження: bit, bit1, ... – будь-які біти з області прямоадресуємих біт.

Таблиця 4.4 – Булева функція 1

R17(0)	bit0	bit1	**F**
0	0	0	**1**
0	0	1	**0**
0	1	0	**0**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**0**
1	1	1	**1**

Таблиця 4.5 – Булева функція 2

R13(1)	bit1	PORTB(2	**F**
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**0**
1	0	1	**0**
1	1	0	**1**
1	1	1	**1**

Таблиця 4.6 – Булева функція 3

PIND6	bit	TXC	**F**
0	0	0	**1**
0	0	1	**0**
0	1	0	**0**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**1**
1	1	1	**1**

Таблиця 4.7 – Булева функція 4

SPE	bit	TXC	**F**
0	0	0	**1**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**0**
1	0	1	**1**
1	1	0	**0**
1	1	1	**1**

Таблиця 4.8 – Булева функція 5

PIND2	bit	TXEN	**F**
0	0	0	**0**
0	0	1	**0**
0	1	0	**1**
0	1	1	**1**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

Таблиця 4.9 – Булева функція 6

ACO	ACI	bit	**F**
0	0	0	**0**
0	0	1	**1**
0	1	0	**1**
0	1	1	**0**
1	0	0	**1**
1	0	1	**0**
1	1	0	**1**
1	1	1	**0**

Контрольні запитання та завдання

1) Які операції пересилання даних дозволяє виконувати система команд мікроконтролерів AVR?
2) Який спосіб адресації використовується у цих командах?
3) Опишіть операнди, що входять до складу команд.
4) Звідки береться цифра 12 у формулі розрахунку часу виконання команди у секундах?
5) Які арифметичні та логічні операції дозволяє виконувати система команд мікроконтролерів AVR?
6) Який спосіб адресації використовується у цих командах?
7) Як впливають арифметичні та логічні команди на прапорці?
8) Опишіть операнди, що входять до складу цих команд.
9) В якому поданні можуть бути використані числа при виконанні арифметичних та логічних операціях для мікроконтролерів AVR?

10) Які операції роботи з бітами дозволяє виконувати система команд мікроконтролерів AVR?
11) Які спосіби адресації використовуються у командах роботи з бітами?
12) Назвіть та прокоментуйте команди зсувів.
13) Опишіть операнди, що входять до складу цих команд.
14) Чим відрізняється арифметичний зсув від логічного?

5 ЛАБОРАТОРНА РОБОТА № 2. ДОСЛІДЖЕННЯ КОМАНД ПЕРЕДАЧІ КЕРУВАННЯ, ВИКЛИКУ ТА ПОВЕРНЕННЯ ІЗ ПІДПРОГРАМ

Тема: команди передачі керування**,** виклику та повернення із підпрограм

Мета: користуючись налагоджувачем дослідити у покроковому режимі виконання команд передачі керування, виклику та повернення із підпрограм

5.1 Порядок виконання лабораторної роботи

1) Вивчити теоретичні відомості з теми «Команди передачі керування, виклику та повернення із підпрограм». Дослідити формати (типи) команд, представлення операндів і роботу програми, що наведено у лабораторній роботі як приклад. Вміти коментувати команди, які наведено у таблиці 1.3.
2) Ввести у симулятор AVR-Studio 4 програму з наведеними нижче командами передачі керування, виклику та повернення із підпрограм.
3) За допомогою програмно-налагоджувальних засобів (ПНЗ) у AVR-Studio 4 проаналізувати виконання програми з наведеними нижче командами передачі керування, виклику та повернення із підпрограм. Переконатися в правильному виконанні програми. При негативному результаті здійснити зміну програми та повторити перевірку.
4) Розробити алгоритм для виконання індивідуального завдання.
5) Розробити програму для виконання індивідуального завдання.
6) Ввести програму індивідуального завдання у симулятор AVR-Studio 4.

7) За допомогою програмно-налагоджувальних засобів (ПНЗ) у AVR-Studio 4 проаналізувати виконання індивідуальної програми. Переконатися у правильному виконанні індивідуального завдання, при негативному результаті здійснити зміну алгоритму або програми, повторити перевірку.
8) Роздрукувати лістинг правильно працюючої програми.
9) Відповісти на контрольні питання.

5.2 Стислі теоретичні відомості

Команди даної групи наведено у таблиці 1.3. Формати команд наведено на рисунку 1.22, а їх опис – у підрозділі 1.4.7.

Нижче наведено приклад програми з використанням команд передачі керування, виклику та повернення із підпрограм.

Приклад програми з використанням команд передачі керування, виклику та повернення із підпрограм

```
      BEGIN:
      ldi r16, $32          ; Завантажити константу (r16 = $32)
      mov r0, r16           ; Скопіювати регістр (r0 = r16)
      mov r1, r0            ; Скопіювати регістр (r1 = r0)
      cpse r1, r0           ; Порівняти, пропустити, якщо рівні
      lds r11, $060         ; Завантажити константу (r11 = $32)
      ldi r30, $0d          ; Завантажити константу (r30 = $0d)
      ldi r31, $00          ; Завантажити константу (r16 = $00)
      ijmp                  ; Непрямий перехід по Z
      nop                   ; Порожня операція
      nop                   ; Порожня операція
      nop                   ; Порожня операція
      nop                   ; Порожня операція
 ok2: ldi r16, $80          ; Завантажити константу (r16 = $80)
      mov r0, r16           ; Скопіювати регістр (r0 = r16)
      cpse r1, r0           ; Порівняти, пропустити, якщо рівні
      rjmp ok               ; Відносний безумовний перехід
      nop                   ; Порожня операція
      nop                   ; Порожня операція
      nop                   ; Порожня операція
 ok:
      nop                   ; Порожня операція
      ldi r25, $de          ; Завантажити константу (r25 = $de)
      out $ 3D, r25         ; Записати r25 в порт $ 3D
```

```
        rcall ok1               ; Відносний безумовний виклик підпрограми
        ldi r21, $ab            ; Завантажити константу (r21 = $ab)
        ldi r22, $fe            ; Завантажити константу (r22 = $fe)
        cp r21, r22             ; Порівняти r21 з r22
        brne noteg              ; Перейти на noteg, якщо не рівні
        nop                     ; Порожня операція
        noteg:
        ldi r25, $de            ; Завантажити константу (r25 = $de)
        out $3D, r25            ; Запис r25 в порт $3D (SP)
        ldi r30, $8a            ; Завантажити константу (r30 = $8a)
        ldi r31, $00            ; Завантажити константу (r31 = $00)
        icall                   ; Непрямий виклик підпрограми
        ldi r31, $35            ; Завантажити константу (r31 = $35)
        ldi r30, $35            ; Завантажити константу (r30 = $35)
        ldi r29, $ab            ; Завантажити константу (r29 = $ab)
        ldi r28, $ab            ; Завантажити константу r28 = $ab
        cp r30, r28             ; Порівняти r30 з r28
        cpc r31, r29            ; Порівняти r31 з r29 і перенесенням
        brne noteg1             ; Перехід на noteg1, якщо не рівні
        nop                     ; Порожня операція
        nop                     ; Порожня операція
        noteg1:
        ldi r19, $35            ; Завантажити константу (r19 = $35)
        cpi r19, $35            ; Порівняти r19 з $35
        breq noteg2             ; Перехід на noteg2, якщо рівні
        nop                     ; Порожня операція
        nop                     ; Порожня операція
        noteg2:
        sbr r19, $80            ; Встановити 7-й біт в регістрі r19
        sbrs r19, 7             ; Пропустити, якщо  r19.7 = 1
        nop                     ; Порожня операція
        cbr r19, $80            ; Скинути 7-й біт в регістрі r19
        sbrc r19, 7             ; Пропустити, якщо r19.7 = 0
        nop                     ; Порожня операція
        sbi $11,0               ; Встановити 0-й біт порту $11 (DDRD)
        sbis $11,0              ; Пропустити, якщо $11.0 = 1
        nop                     ; Порожня операція
        cbi $11,0               ; Скинути 0-й біт порту $11
        sbic $11,0              ; Пропустити, якщо $11.0 = 0
        nop                     ; Порожня операція
        bset 5                  ; Встановлення прапорця SREG.5 (H)
        brbs 5, noteg3          ; Перехід на noteg3, якщо SREG.5 = 1
        nop                     ; Порожня операція
noteg3:
        nop                     ; Порожня операція
        bclr 5                  ; Скидання прапорця SREG.5
        brbc 5, noteg4          ; Перехід на noteg4, якщо SREG.5 = 0
        nop                     ; Порожня операція
noteg4:
        nop                     ; Порожня операція
        nop                     ; Порожня операція
        bset 0                  ; Встановлення прапорця SREG.0 (C)
```

```
        brcs greater        ; Перехід на greater, якщо C = 1
        nop                 ; Порожня операція
greater:
        nop                 ; Порожня операція
        bclr 0              ; Скидання прапорця SREG.0
        brcc greater1       ; Перехід на greater1, якщо C = 0
        nop                 ; Порожня операція
greater1:
        nop                 ; Порожня операція
        bset 2              ; Встановлення прапорця SREG.2 (N)
        brmi minus          ; Перехід на minus, якщо N = 1
        nop                 ; Порожня операція
minus:
        nop                 ; Порожня операція
        bclr 2              ; Скидання прапорця SREG.2
        brpl plus           ; Перехід на plus, якщо N = 0
        nop                 ; Порожня операція
plus:
        nop                 ; Порожня операція
        bset 0              ; Встановлення прапорця SREG.0 (C)
        brlo less           ; Перехід на less, якщо C = 1 (менше)
        nop                 ; Порожня операція
less:
        nop                 ; Порожня операція
        bclr 0              ; Скидання прапорця SREG.0
        brsh greater3       ; Перехід, якщо C = 0 (більше/дорівнює)
        nop                 ; Порожня операція
greater3:
        bset 4              ; Встановлення прапорця SREG.4 (S)
        brlt less1          ; Перехід, якщо S = 1 (менше зі знаком)
        nop                 ; Порожня операція
        less1:
        nop                 ; Порожня операція
        bclr 4              ; Скидання прапорця SREG.4
        brge greater2       ; Перехід, якщо S = 0 (більше/дорівнює)
        nop                 ; Порожня операція
greater2:
        nop                 ; Порожня операція
        bset 5              ; Встановлення прапорця SREG.5 (H)
        brhs H1             ; Перехід на H1, якщо H = 1
        nop                 ; Порожня операція
H1:
        nop                 ; Порожня операція
        bclr 5              ; Скидання прапорця SREG.5
        brhc H0             ; Перехід на H0, якщо H = 0
        nop                 ; Порожня операція
        H0:
        bset 6              ; Встановлення прапорця SREG.6 (T)
        brts T1             ; Перехід на T1, якщо T = 1
        nop                 ; Порожня операція
T1:
        nop                 ; Порожня операція
```

```
        bclr 6              ; Скидання прапорця SREG.6
        brtc T0             ; Перехід на T0, якщо T = 0
        nop                 ; Порожня операція
T0:
        bset 3              ; Встановлення прапорця SREG.3
        brvs V1             ; Перехід на V1, якщо V = 1
        nop                 ; Порожня операція
V1:
        nop                 ; Порожня операція
        bclr 3              ; Скидання прапорця SREG.3 (V)
        brvc V0             ; Перехід на V0, якщо V = 0
        nop                 ; Порожня операція
V0:
        bset 7              ; Встановлення прапорця SREG.7 (I)
        brie I1             ; Перехід на I1, якщо I = 1
        nop                 ; Порожня операція
I1:
        nop                 ; Порожня операція
        bclr 7              ; Скидання прапорця SREG.7
        brid I0             ; Перехід на I0, якщо I = 0
        nop                 ; Порожня операція
I0:
        rjmp BEGIN          ; Відносний безумовний перехід
        nop                 ; Порожня операція
        nop                 ; Порожня операція
ok1:
        ldi r19, 156        ; Завантажити константу (r19 = 156)
        ldi r20, 175        ; Завантажити константу (r20 = 175)
        ldi r21, 132        ; Завантажити константу (r21 = 132)
        ldi r22, 87         ; Завантажити константу (r22 = 87)
        add r21, r19        ; Додати r21 і r19 без перенесення
        adc r22, r20        ; Додати r22 і r20 з перенесенням
        ret                 ; Повернення з підпрограми
        ldi r23, $12        ; Завантажити константу (r23 = $12)
        ldi r22, $15        ; Завантажити константу (r22 = $15)
        ldi r21, $25        ; Завантажити константу (r21 = $25)
        ldi r20, $87        ; Завантажити константу (r20 = $87)
        sub r22, r20        ; Віднімання r20 з r22 без перенесення
        adc r23, r21        ; Віднімання r21 з r23 з перенесенням
        ret                 ; Повернення з підпрограми
        nop                 ; Порожня операція
```

Варіанти індивідуальних завдань

Таблиця 5.1 – Завдання до використання команд передачі керування, виклику та повернення із підпрограм

№	Текст індивідуального завдання
1	a) Вибрати з п’яти двобайтових чисел найбільше. b) Дана послідовність з 8-ми байт, починаючи з адреси 71h. Виділити з послідовності байти з певними властивостями і помістити їх за адресою, заданою регістром R13. Алгоритм виділення визначається вмістом регістра R14: якщо [R14] < > 27h, то виділяти байти, менші вмісту регістра R15 у два рази, якщо ж [R14] = 27h, то виділяти байти, більшого вмісту регістра R15.
2	a) У виконуваному коді є 4 підпрограми. Початкові адреси дані в 4-х парах регістрів R11 і R12, R13 і R14, ... В залежності від вмісту R30, який дорівнює 1, 2, 3 або 4, перейти до виконання однієї з підпрограм. b) b) У регістрах R11 і R12 містяться числа. Якщо при складанні цих чисел прапорець переповнення встановиться в 1, то необхідно в регістр R17 записати число 21H, в іншому випадку в регістр R18 записати число 31H h.
3	a) Вміст регістра R21 необхідно помістити в СПД з адресою: 70h, якщо [R21] > 11h; 71h, якщо [R21] < 11h. b) Написати підпрограму, яка виконувала б одну з дій: +/- над вмістом регістрів R21 і R22 (результат у R23). Вид операції задається молодшим бітом регістра R24.

Закінчення таблиці 5.1

4	a) Написати підпрограму, що визначає найбільше і найменше з п'яти однобайтних чисел, розташованих у пам'яті починаючи з адреси 71h. Номери знайдених чисел, зміщення відносно базової адреси, записати відповідно у регістри R21 і R22. a) Перевірити область пам'яті з адресами від 71H до 91H на наявність послідовності з 5 нульових або одиничних біт. Якщо така послідовність є, то у регістр R21 записати адресу початку послідовності.
5	b) Написати підпрограму, яка визначала б, до якого інтервалу належить вміст регістра R30: 0…99; 100…199 або 200…255. Для видачі результату використовуються три молодших біти регістра R21. c) Дана послідовність з 9 байт. 9-й байт містить біти контролю парності кожного з 8 попередніх байт. Написати підпрограму перевірки правильності контрольних біт. У разі невідповідності підпрограма повинна видавати номери неправильних біт у 9-му байті.
6	a) Написати підпрограму, яка сортує 10 беззнакових чисел за зростанням. Адреса послідовності починається з 71h. b) Залежно від складання двох знакових чисел записати у регістр R21 числа 31H і 32H наступним чином: – якщо результат від'ємний, то 31H; – якщо результат додатний, то 32H.
7	a) Вміст регістра SREG необхідно помістити в: – R21, якщо [R19] >1Fh; – R22, якщо [R19] < 1Fh. c) Написати підпрограму, яка переписує послідовність чисел з одних регістрів у інші. Якщо молодший біт регістра R21 встановлений, то запис відбувається так: R12→ R22, R13 → R23, R14 → R24, інакше R22→ R12, R23 → R13, R24 → R14
8	a) Написати підпрограму, яка визначала б, до якого інтервалу належить вміст регістра R21: 0…99; 100…199 або 200…255. Для видачі результату використовуються три молодших біта регістра R30. d) Дана послідовність з 10-ти байт, починаючи з адреси 71h. Виділити з послідовності байти з певними властивостями і помістити їх за адресою, яку задано у регістрі R21. Алгоритм виділення визначається вмістом регістрів R22 і R23: якщо [R22]<[R23], то виділяти байти, менші вмісту регістра R23, якщо ж [R22]≥[R23], то виділяти байти, більші вмісту регістра R23.
9	a) Вибрати з п'яти двобайтових чисел найменше. b) Дана послідовність з 10 байт. Знайти суму всіх парних чисел в цій послідовності.
10	a) Написати підпрограму, яка сортує 10 беззнакових чисел за спаданням. Адреса послідовності дорівнює 71h. b) Написати підпрограму, яка виконувала б одну з дій: +/- над вмістом регістрів R21 і R22 (результат у R23). Вид операції задається молодшим бітом регістра R30.

Контрольні запитання та завдання

1) Які операції передачі керування, виклику та повернення із підпрограм дозволяє виконувати система команд мікроконтролерів AVR?
2) Які спосіби адресації використовується у цих командах?
3) Опишіть операнди, що входять до складу цих команд.
4) Як впливають команди передачі керування на прапорці?
5) У чому відмінність між командами RETI та RET? Яка помилка виникне, якщо в кінці підпрограми обробки переривання замість RETI застосувати RET?

6 ЛАБОРАТОРНА РОБОТА №3. ДОСЛІДЖЕННЯ НОВИХ КОМАНД МІКРОКОНТРОЛЕРІВ СІМ`Ї AVR

Тема: Нові команди AVR-мікроконтролерів

Мета: Користуючись налагоджувачем дослідити виконання у покроковому режимі нових команд AVR-мікроконтролерів.

6.1 Порядок виконання лабораторної роботи

1) Вивчити теоретичні відомості з теми «Нові команди AVR-мікроконтролерів». Дослідити формати (типи) команд, представлення операндів і роботу програми, що наведено у лабораторній роботі як приклад. Вміти коментувати нові команди, які наведено у таблиці 1.5.
2) Ввести програму з використанням нових команд у симулятор AVR-Studio 4.
3) За допомогою програмно-налагоджувальних засобів в AVR-Studio4 проаналізувати виконання цієї програми та переконатися у правильному її виконанні. При негативному результаті здійснити зміну програми та повторити перевірку.
4) Роздрукувати лістинг правильно працюючої програми.
5) Відповісти на контрольні питання.

6.2 Стислі теоретичні відомості

Команди даної групи наведено у таблиці 1.5, а їх опис – у підрозділі 1.4.8.

Нижче наведено приклад програми з використанням нових команд.

Приклад програми з використанням нових команд

```
ok3:
nop                 ; Порожня операція
nop                 ; Порожня операція
                    ; Множення
ldi r19, $2f        ; Завантаження константи: r19 = $2f
ldi r20, $78        ; Завантаження константи: r20 = $78
MUL r19, r20        ; Множення беззнакових чисел
                    ; Інструкція переходу
jmp ok              ; Відносний безумовний перехід
nop                 ; Порожня операція
nop                 ; Порожня операція
nop                 ; Порожня операція
ok:
                    ; Множення
ldi r17, $2f        ; Завантаження константи: r17 = $2f
ldi r18, $78        ; Завантаження константи: r18 = $78
clr r0              ; Скидання r0
clr r1              ; Скидання r1
MULS r17, r18       ; Множення знакових чисел
ldi r17, $2f        ; Завантаження константи: r17 = $2f
ldi r18, $78        ; Завантаження константи: r18 = $78
clr r0              ; Скидання r0
clr r1              ; Скидання r1
MULSU r17, r18      ; Множення знакового числа на беззнакове
                    ; Інструкція переходу
ldi r30, $1a        ; Завантаження константи: r30 = $1a
ldi r31, $00        ; Завантаження константи: r31 = $00
EIJMP               ; Безумовний непрямий перехід в пам’яті програм ємністю 4 Мслів
nop                 ; Порожня операція
nop                 ; Порожня операція
nop                 ; Порожня операція
ok2:
                    ; Множення
ldi r17, $2f        ; Завантаження константи: r17 = $2f
ldi r18, $78        ; Завантаження константи: r16 = $78
MUL R1, R2;         ; Множення беззнакових чисел
clr r0              ; скидання r0
clr r1              ; скидання r1
FMUL r17, r18       ; Множення дробових беззнакових чисел
ldi r17, $2f        ; Завантаження константи (r17 = $2f)
ldi r18, $78        ; Завантаження константи (r18 = $78)
```

```
clr r0                ; Скидання r0
clr r1                ; Скидання r1
FMULSU R17, R18       ; Множення дробового знакового числа на беззнаковое
                      ; Виклик підпрограми
ldi r25, $de          ; Завантаження константи: r25 = $de
out $3D, r25          ; Запис регістра r25 в порт з адресою $3D
call ok1              ; Відносний безумовний виклик підпрограми
                      ; Множення
ldi r17, $2f          ; Завантаження константи: r17 = $2f
ldi r18, $78          ; Завантаження константи: r18 = $78
clr r0                ; Скидання r0
clr r1                ; Скидання r1
FMULS r17, r18        ; Множення дробових знакових чисел
                      ; Виклик підпрограми
ldi r30, $4b          ; Завантаження константи: r30 = $4b
ldi r31, $00          ; Завантаження константи: r31 = $00
eicall                ; Відносний безумовний виклик підпрограми
                      ; Передача даних
ldi r19, $00          ; Завантаження константи: r19 = $00
ldi r18, $1b          ; Завантаження константи: r18 = $1b
clr r31               ; Скидання регістра r31
clr r30               ; Скидання r30
MOVW r30, r18         ; Пересилання слова
LPM r1, Z             ; Завантаження регістра r1 з пам’яті програм з адресою в
                      ; індексному регістрі Z
LPM r2, Z+            ; Завантаження регістра r2 з пам’яті програм з адресою в
                      ; індексному регістрі Z з постінкрементом вмісту Z
SPM                   ; Команда самопрограмування мікроконтролера, у якого є відповідний
                      ; блок
clr r0                ; Скидання r0
ELPM                  ; Завантаження регістра r0 з розширено пам’яті програм з
                      ; адресою в регістрах RAMPZ та Z
ELPM r3, Z            ; Завантаження регістра r3 з розширеної пам’яті програм з
                      ; адресою в регістрах RAMPZ та Z
ELPM r4, Z+           ; Завантаження регістра r4 з розширеної пам’яті програм з адресою в
                      ; регістрах RAMPZ і Z з подальшим інкрементом вмісту RAMPZ: Z
nop                   ; Порожня операція
nop                   ; Порожня операція
nop                   ; Порожня операція
                      ; Інструкція переходу
jmp ok3               ; Відносний безумовний перехід
ok:
ldi r19, 156          ; Завантаження константи:  r19 = 156
ldi r20, 175          ; Завантаження константи:  r20 = 175
ldi r21, 132          ; Завантаження константи:  r21 = 132
ldi r22, 87           ; Завантаження константи:  r22 = 87
nop                   ; Порожня операція
nop                   ; Порожня операція
nop                   ; Порожня операція
add r21, r19          ; Додавання регістра r21 до  регістра r19 без перенесення
adc r22, r20          ; Додавання регістра r22 до  регістра r20 з перенесенням
ret                   ; Повернення з підпрограми
ok4:
```

```
ldi r23, $12        ; Завантаження константи:  r23 = $12
ldi r22, $15        ; Завантаження константи:  r22 = $15
ldi r21, $25        ; Завантаження константи:  r21 = $25
ldi r20, $87        ; Завантаження константи:  r20 = $87
sub r22, r20        ; Віднімання регістра r20 від регістра r22 без перенесення
adc r23, r21        ; Віднімання регістра r21 від регістра r23 з перенесенням
ret                 ; Повернення з підпрограми
```

Контрольні запитання та завдання

1) Які операції дозволяють виконувати нові команди, розглянуті у лабораторній роботі?
2) Які способи адресації використано у розглянутих командах?
3) Як представлені дробові числа у розглянутих командах?
4) Для чого використовують команду SPM та в чому її особливість?

7 ЛАБОРАТОРНА РОБОТА №4. ДОСЛІДЖЕННЯ МОДЕЛІ ПРИСТРОЮ КЕРУВАННЯ КРОКОВИМ ДВИГУНОМ

Тема: Моделювання пристрою керування кроковим двигуном з використанням мікроконтролера ATMega128.

Мета: Користуючись пакетом Proteus 8.6 дослідити роботу пристрою керування кроковим двигуном.

7.1 Моделювання уніполярного крокового двигуна

7.1.1 Опис моделі

Крокові двигуни, крім відмінностей у загальній конструкції, відрізняються ще й схемою включення обмоток. Є декілька варіантів їх конфігурацій, залежно від якої двигуни поділяються на уніполярні (англ. «Unipolar») і біполярні (англ. «Bipolar»).

Уніполярні крокові двигуни, так само як і біполярні, мають дві обмотки, і кожна з них має центральне відведення. Залежно від необхідного напрямку магнітного поля, в роботу включається відповідна половина обмотки, що досягається простим перемиканням ключів і істотно спрощує схему драйвера. Це

дозволяє змінювати напрямок магнітного поля, створюваного обмоткою, перемиканням її половинок. Подібний механізм дозволяє в якості керуючої системи використовувати найпростіший уніполярний драйвер з чотирма ключами.

Як правило, уніполярний двигун має 6 виводів, але середні виводи обмоток можуть бути об’єднані в середині самого двигуна, тому такий двигун може мати й 5 виводів. Таким чином, якщо вам в руки потрапив невідомий двигун з шість чи п’ятьма виводами – це гарантовано уніполярний кроковий двигун.

З біполярними кроковими двигунами справа йде трохи інакше. Дані двигуни мають тільки одну обмотку в одній фазі. Щоб змінювати напрямок магнітного поля з метою змінити напрямок струму в обмотці керуюча схема біполярного двигуна повинна бути набагато складніше. Цього можна досягти за допомогою мостової схеми (H-bridge).

До того ж, для спрощення завдання можна придбати кілька драйверних чіпів, які вам допоможуть. Біполярні крокові двигуни, на відміну від уніполярних мають два виводи на одну фазу, жоден з яких не є спільним.

Біполярні крокові двигуни трохи складніше в управлінні, але при схожих габаритах, біполярний двигун здатний забезпечити більший момент, в порівнянні з уніполярним. Однак уніполярний двигун, на противагу біполярному, простіше в експлуатації, і цілком згодиться для приводу пристроїв з невеликою потужністю – побутова техніка (пральна машина, холодильник), магнітофони тощо.

Робочу модель пристрою керування уніполярним кроковим двигуном показано на рисунку 7.1.

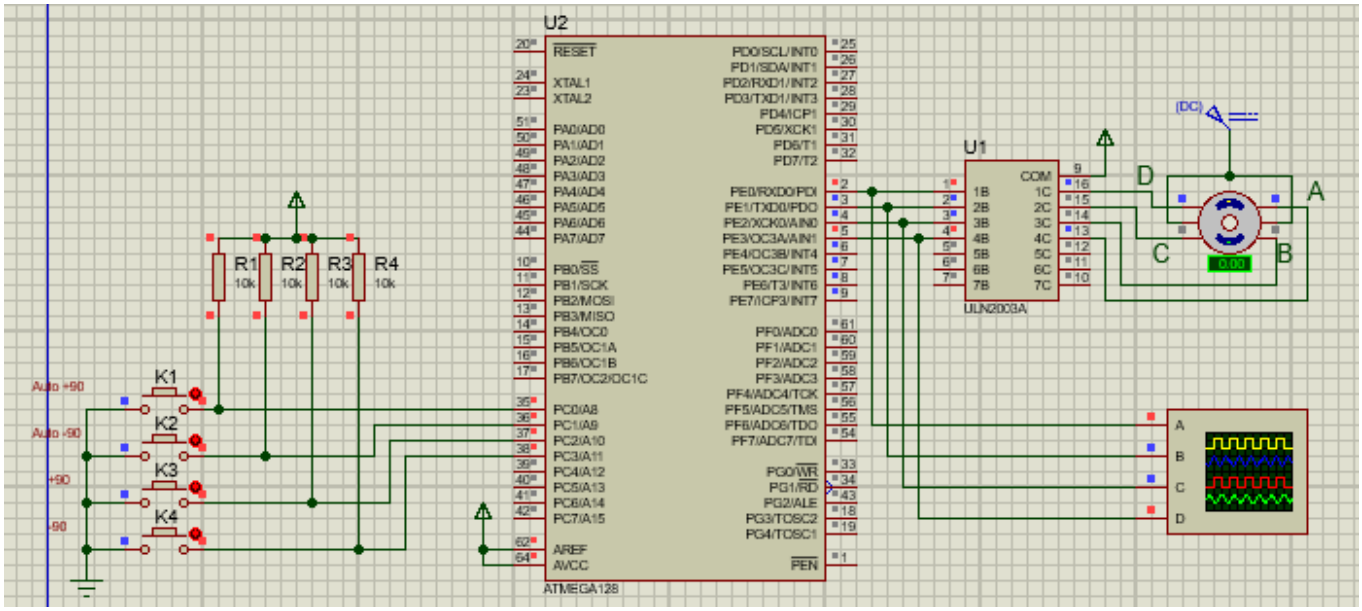


Рисунок 7.1 – Схема моделі пристрою керування уніполярним кроковим двигуном

Розберемо цю модель докладніше. З лівого боку на рисунку 7.1 зображено кнопки керування: K1, K2, K3, K4 на які, через резистори R1, R2, R3, R4 подається напруга: +5В. У правому верхньому куті знаходиться уніполярний кроковий двигун, а трохи лівіше мікросхема ULN2003A, через яку мікроконтролер і керує двигуном. По центру зображено мікроконтролер сім`ї AVR – ATMega128. Зовнішні резистори використовуються через те, що на вхідних лініях мікроконтролера відключено внутрішні резистори, що підтягують. На рисунку 7.2 наведено позначення виводів мікроконтролера. Нижче більш докладніше пояснюється, що і куди підключено в моделі.

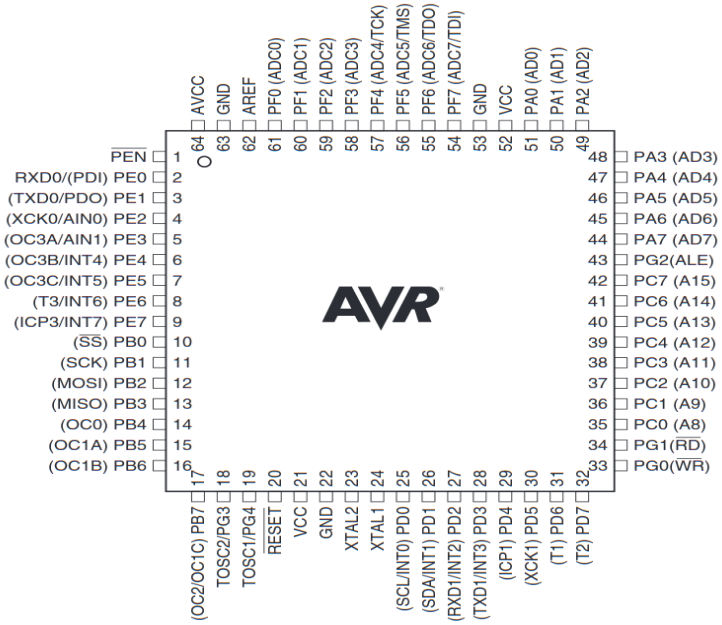


Рисунок 7.2 – Позначення виводів мікроконтролера ATMega128

На рисунку 7.3 наведено збільшений вигляд кнопок, за допомогою яких в моделі здійснюється керування крововим двигуном. Кнопки є нормально розімкненими. У вихідному стані на входи мікроконтролера від кнопок подається високий рівень напруги, тобто логічна одиниця. А коли кнопки натискаються, на входах мікроконтролера спостерігається низький рівень напруги, тобто логічний нуль. При відпусканні кнопок на входи мікроконтролера від кнопок знову подається високий рівень напруги, тобто логічна одиниця.

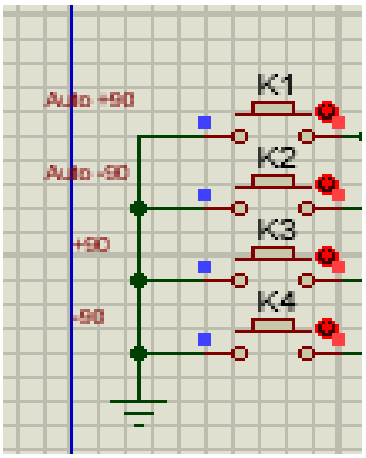


Рисунок 7.3 – Порядок розташування елементів керування

Кнопку K1 (Step +90 Auto Mode) підключено до виводу PC0 мікроконтролера, який програмується, як вхід.

Вона відповідає за вмикання та вимикання двигуна. Тобто коли двигун вимкнено, то вона його ввімкне. Якщо натиснути на кнопку, то двигун буде періодично виконувати цикл кроків від 0 до 360 градусів. Щоб зупинити двигун треба відпустити кнопку.

Кнопка K2 (Step -90 Auto Mode) працює так само як і K1, але при натисканні, відбуватиметься цикл кроків в зворотному порядку до циклу кроків кнопки K1. Кнопку підключено до виводу PC1 мікроконтролера, який програмується, як вхід.

Кнопка K3 (Step +90) відповідає за виконання одного кроку двигуна за годинниковою стрілкою. Крок двигуна у кроковому режимі дорівнює 90 градусів. Кнопку підключено до виводу PC2 мікроконтролера, який програмується, як вхід .

Кнопка K4 (Step -90) відповідає за виконання одного кроку двигуна проти годинникової стрілки. Крок двигуна дорівнює 90 градусів. Кнопку підключено до виводу PC3 мікроконтролера, який програмується, як вхід.

На рисунку 7.4 наведено підключення до мікроконтролера мікросхеми ULN2003A (U2).

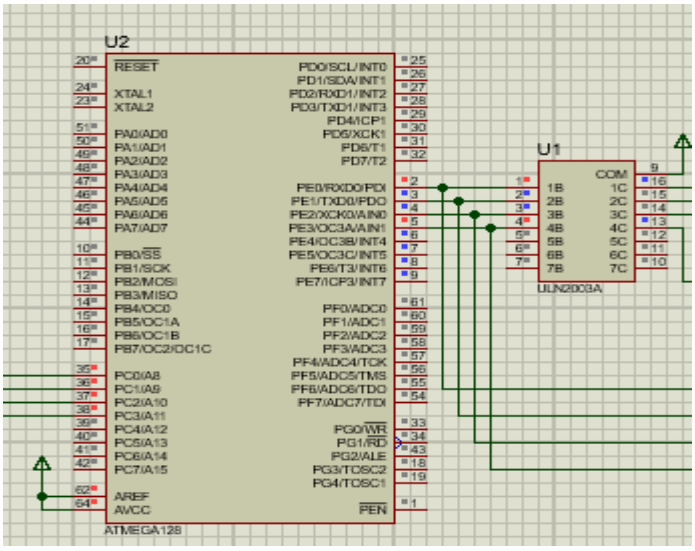


Рисунок 7.4 – Підключення до мікроконтролера мікросхеми ULN2003A

Мікросхема ULN2003A – це транзисторна схема Дарлінгтона з вихідними ключами підвищеної потужності з відкритим колектором.

Виводи 2, 3, 4 та 5 мікроконтролера, тобто PE0, PE1, PE2 та PE3 запрограмовані як виходи і підключені, відповідно, до входів 1B, 2B, 3B та 4B

мікросхеми ULN2003A. Мікросхема підсилює потужність цих сигналів: кожен вихідний канал ULN2003A розрахований на навантаження 500мА і витримує максимальний струм до 600мА.

Інші виводи мікросхеми, які зображено на рисунку 7.4, використовуються наступним чином: вивід GND підключено до землі, а на вхід COM подається напруга живлення самої мікросхеми ULN2003A: +5В.

Підключення крокового двигуна та мікросхеми ULN2003A зображено на рисунку 7.5.

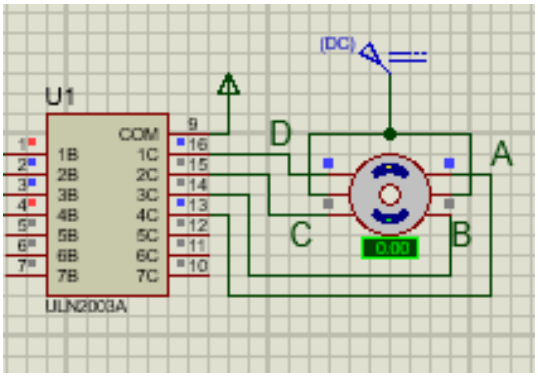


Рисунок 7.5 – Підключення крокового двигуна до мікросхеми ULN2003A

Чотири обмотки двигуна підключено до 13, 14, 15 та 16 виводів мікросхеми ULN2003A, а саме 4C, 3C, 2C та 1C. На ці виходи за допомогою мікросхеми ULN2003A подаються керуючі сигнали від мікроконтролера, що і повертають наш кроковий двигун в положення, яке визначається заданою кроковою послідовністю.

На вхід (AC) подається напруга живлення уніполярного крокового двигуна: +12В.

Деякі фрагменти, які демонструють роботу системи, наведено на рисунках 7.6…7.7.

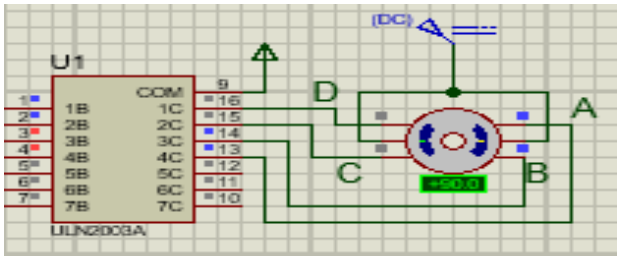


Рисунок 7.6 – Стан моделі після натискання кнопки «Step +90»

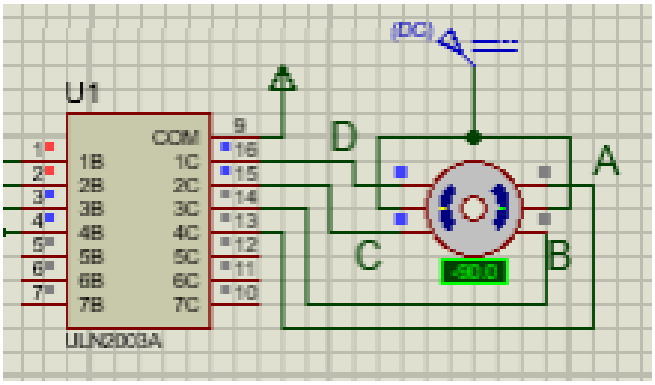


Рисунок 7.7 – Стан моделі після активації режиму реверсу, відповідна кнопка «Step-90»

Мікроконтролер

Для контролю та виконання всіх функцій, які покладено на модель, було обрано МК сім`ї AVR – ATmega 128. Він є малопотужним 8-розрядним КМОН-мікроконтролером, який має розширену RISC-архітектуру [1; 2; 10]. Швидкодія МК досягає 1 мільйона операцій в секунду, оскільки більшість команд виконується за один машинний такт.

ATmega 128 має 64 виводи типу вхід – вихід. У мікроконтролер вбудовано Flash-пам’ять програм, яку можна програмувати та перепрограмувати.

Основні характеристики мікроконтролера:

1) Висока продуктивність.
2) Мале споживання енергії від джерела живлення.
3) Розвинена RISC-архітектура:
    а) 131 основна команда, більшість з яких виконується за один машинний такт;
    б) 32 робочих регістри загального призначення;
    в) повністю статичний режим роботи.
4) Енергонезалежна пам’ять програм та даних.
5) 128 Кбайт внутрішньої Flash-пам’яті програм з кількістю циклів перепрограмування до 10 000.
6) 4096 байт EEPROM-пам’яті даних із кількістю циклів перепрограмування до 100 000.

7) 64 виводи типу вхід – вихід.
8) JTAG (IEEE1149.1 сумісний) інтерфейс.
9) Можливе сканування пам’яті відповідно до JTAG-стандарту.
10) Периферійні функції:
    а) два 8-бітових таймери/лічильники із програмованим режимом порівняння;
    б) два 16-бітових таймери/лічильники із програмованим режимом порівняння та захоплення;
    в) лічильник реального часу із програмованим генератором;
    г) чотири ШІМ-генератори;
    д) 8 вхідних каналів 10-бітового АЦП.
11) Синхронні послідовні SPI- та I^2C-інтерфейси, та аналоговий компаратор.
32) Спеціальні функції:
    а) функція Reset за включенням живлення і функція вимикання за зниженням напруги живлення;
    б) внутрішній калібрований RC-генератор;
    в) зовнішні та внутрішні джерела переривання;
13) 64 вивідний корпус TQFP та 64-контактний MLF.
14) Напруга живлення: від 4.5 В до 5.5 В.
15) Тактова частота: від 0 до 16 МГЦ.

Опис виводів мікроконтролера:

- VCC – напруга живлення;
- GND – спільна земля;
- порт A (PA7...PA0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду. При введенні лінії порту A будуть діяти як джерело струму, якщо зовні діє низький рівень і включено резистори, що підтягують;

– порт B (PB7...PB0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
– порт C (PC7...PC0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
– порт D (PD7...PD0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
– порт E (PE7...PE0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду;
– порт F (PF7...PF0) – 8-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду. Якщо активовано інтерфейс JTAG, то резистори на лініях PF7 (TDI), PF5 (TMS) і PF4 (TCK) будуть підключені, навіть якщо виконується скидання. Порт F виконує також функції інтерфейсу JTAG;
– порт G (PG4...PG0) – 5-розрядний порт двонаправленого введення/виведення із внутрішніми підтягуючими резисторами, які вибираються окремо для кожного розряду. Порт G також виконує деякі спеціальні функції;
– RESET – вхід скидання. Якщо на цей вхід подати низький рівень напруги тривалістю більше мінімально необхідної, то буде згенеровано скидання незалежно від роботи схеми синхронізації. Дія імпульсу меншої тривалості не гарантує генерацію скидання;
– XTAL1 – вхід інвертуючого підсилювача внутрішнього генератора та вхід зовнішньої синхронізації;
– XTAL2 – вихід інвертуючого підсилювача внутрішнього генератора;

– AVCC – вивід для подачі живлення порту F і аналого-цифровому перетворювачу. Він повинен бути зовні пов’язаний з VCC-входом, навіть якщо АЦП не використовується. При використанні АЦП цей вивід пов’язано з VCC через фільтр низьких частот [1; 2; 10];
– AREF – вхід підключення джерела опорної напруги для АЦП;
– PEN – вхід дозволу програмування для режиму послідовного програмування через інтерфейс SPI. Якщо під час дії скидання при подачі живлення на цей вхід подати низький рівень, то мікроконтролер переходить у режим послідовного програмування через SPI.

Уніполярний кроковий двигун

Уніполярні КД (Unipolar Stepper Motors) аналогічні біполярним, але мають відвід від середини кожної з обмоток Це дозволяє змінювати напрямок магнітного поля простими ключовими каскадами, а не мостовими схемами У уніполярному КД середні виводи обмоток можуть електрично зєднуватися всередині корпусу, тому в такому двигуні назовні виходять не 6, а 5 відводів Іноді буває й 8 відводів, якщо половина кожної обмотки зроблено окремою секцією. В останньому випадку уніполярний КД легко перетворити на біполярний.

Драйвер уніполярного крокового двигуна

В якості драйвера двигуна обрано мікросхему ULN2003A.

Мікросхема ULN2003A (рисунок 7.8) – це транзисторна схема Дарлінгтона з вихідними ключами підвищеної потужності з відкритим колектором, що має на виходах діоди, які захищають. Вони потрібні для захисту керуючих електричних ланцюгів від оберненого викиду напруги від індуктивного навантаження.

Рисунок 7.8 – Зовнішній вигляд мікросхеми ULN2003A

Кожен канал в ULN2003A розрахований на навантаження 500мА та витримує максимальний струм до 600мА. Входи і виходи розміщені в корпусі мікросхеми один напроти другого, що облегшує розводку друкованої плати. ULN2003A відносяться до сім`ї мікросхем ULN200X.

Різні версії цієї мікросхеми призначені для певної логіки, а саме мікросхема ULN2003A пристосована до роботи з ТТЛШ та КМОН-логікою (живлення: +5В). Мікросхема може бути застосована для керування навантаженням значної потужності, включаючи електромагнітні реле, двигуни постійного струму, електромагнітні клапани, крокові двигуни і т. ін. На рисунку 7.9 наведено схему підключення драйвера до крокового двигуна.

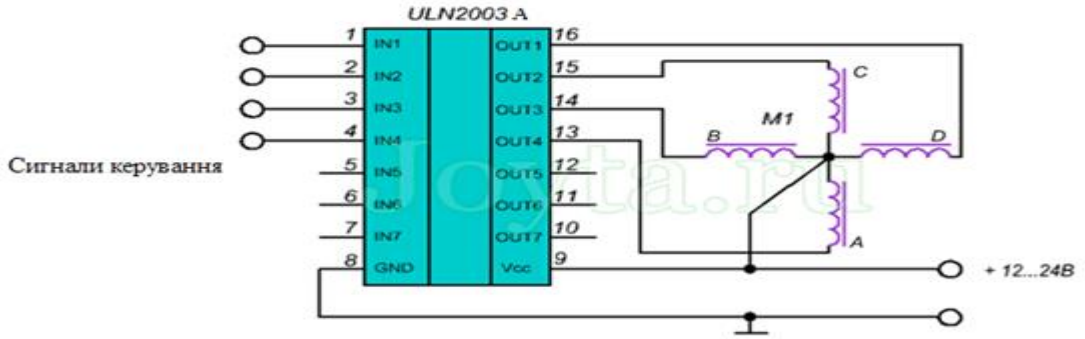


Рисунок 7.9 – Схема підключення драйвера до крокового двигуна

Керування кроковими двигунами можна забезпечити різними способами, але найчастіше використовуються такі крокові послідовності:

– хвильова;
– крокова;
– напівкрокова.

Порядок подачі напруги на обмотки КД для цих крокових послідовностей наведено на рисунку 7.10.

Крок	A	B	C	D
1	ON	OFF	OFF	OFF
2	OFF	ON	OFF	OFF
3	OFF	OFF	ON	OFF
4	OFF	OFF	OFF	ON

а)

Крок	A	B	C	D
1	ON	OFF	OFF	ON
2	ON	ON	OFF	OFF
3	OFF	ON	ON	OFF
4	OFF	OFF	ON	ON

б)

Крок	A	B	C	D
1	ON	OFF	OFF	OFF
2	ON	ON	OFF	OFF
3	OFF	ON	OFF	OFF
4	OFF	ON	ON	OFF
5	OFF	OFF	ON	OFF
6	OFF	OFF	ON	ON
7	OFF	OFF	OFF	ON
8	ON	OFF	OFF	ON

в)

Рисунок 7.10 – Послідовності збудження обмоток КД : а) хвильова, б) крокова, в) напівкрокова

Найпростіший спосіб керування кроковими двигунами – хвильова послідовність збудження. Обмотки збуджуються послідовно одна за одною. При цьому двигун починає обертатися в сторону, протилежну порядку збудження обмоток. Оскільки в свій час збуджується тільки одна обмотка, то момент двигуна, що обертає, є невеликим. Для його збільшення використовується крокова послідовність.

Крокова послідовність – аналогічна попередній, але тут одночасно збуджуються дві обмотки, завдяки чому збільшується обертаючий момент двигуна.

Напівкрокова послідовність збудження – це комбінація перших двох. Під час одного обороту ротора кількість циклів збудження подвоюється. При цьому режимі в 2 рази зменшується величина кроку і двигун працює рівніше.

Вище на рисунку 7.1 було наведено схему моделі пристрою керування уніполярним кроковим двигуном. На цих рисунках обмотки двигуна позначено: A, B, C та D.

Відповідно до схеми моделі розроблено схему алгоритму роботи (рисунок 7.11) та керуючу програму мовою С, які наведено нижче.

В цих алгоритмі та програмі реалізовано крокову послідовність збудження обмоток.

Уніполярний крокoвий двигун, який використовується в моделі, за один крок у крокoвому режимі обертається на кут у 90 градусів. Згідно схемі моделювання, яку наведено на рисунку 7.1, керуюча послідовність імпульсів передається через лінії мікроконтролера: P2.0 (обмотка D), P2.1 (обмотка C), P2.2 (обмотка B) та P2.3 (обмотка A). Ця послідовність у чотирьохрозрядному двійковому коді для крокового режиму виглядає наступним чином: 1001 для кута 0 градусів; 0011 для кута 90 градусів; 0110 для кута 180 градусів, 1100 для кута 270 градусів та 1001 для повернення у початковий стан (кут дорівнює 0 градусів). З виходів 1C, 2C, 3C та 4C драйвера ULN2003A імпульси керування подаються безпосередньо на обмотки двигуна, які на рисунку 7.1 позначено як D, C, B та A.

Якщо перепрограмувати роботу моделі у напівкрокoвий режим згідно з рисунком 7.14, в, то за один крок двигун буде обертатися на кут у 45 градусів.

7.1.2 Схема алгоритму роботи моделі уніполярного крокового двигуна для крокового режиму роботи

Схему алгоритму роботи моделі уніполярного крокового двигуна для крокового режиму роботи наведено на рисунку 7.11.

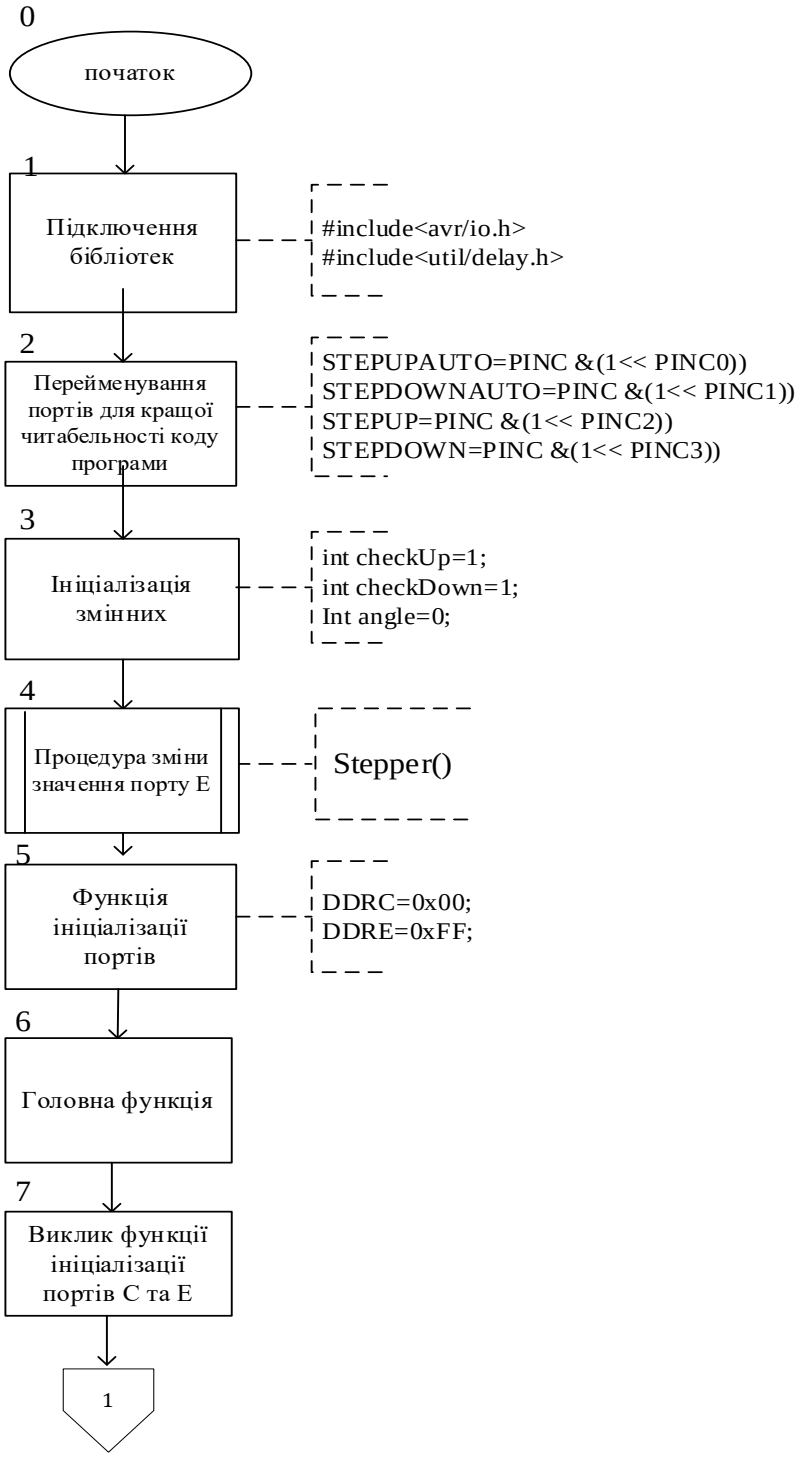
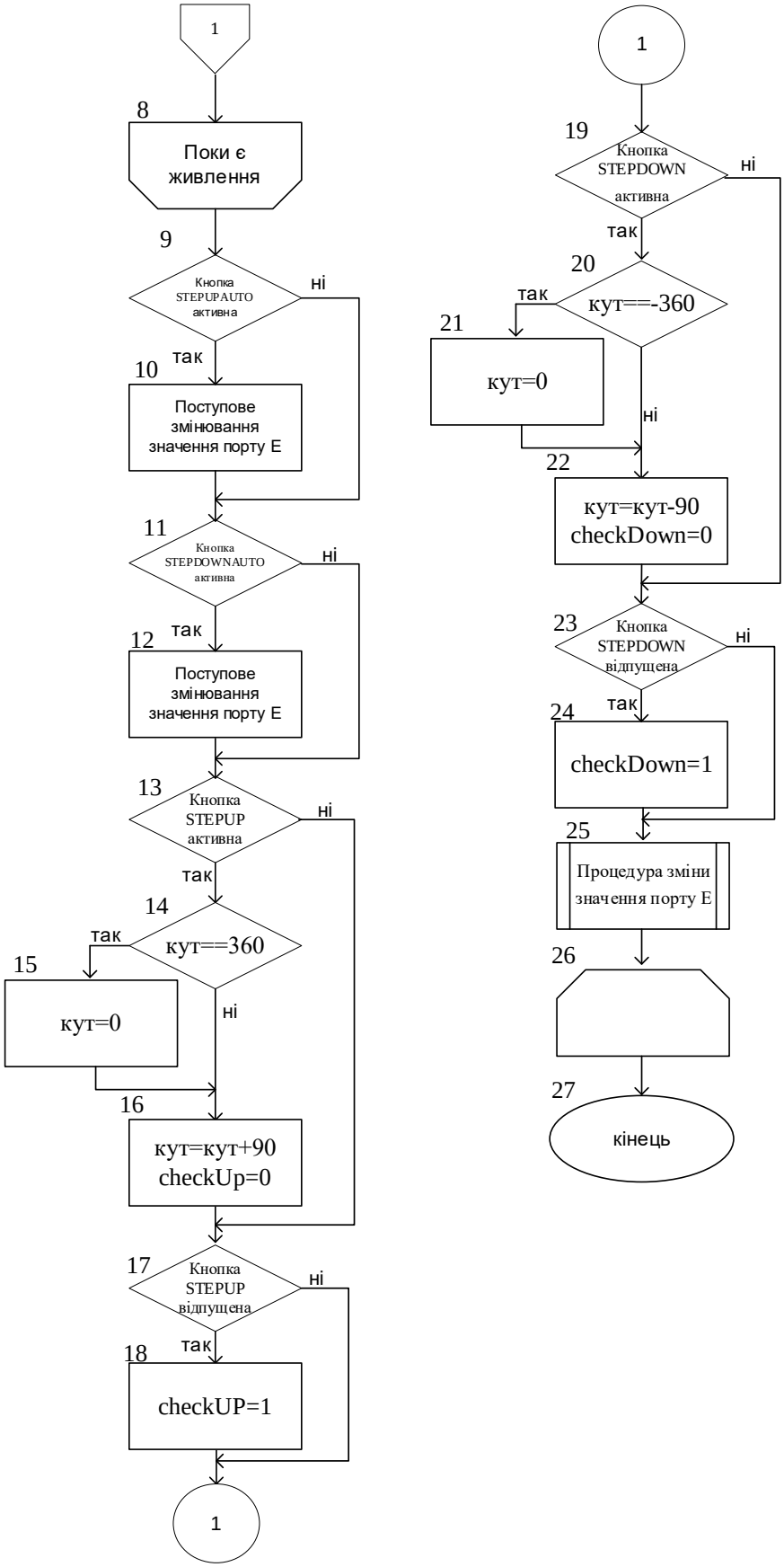
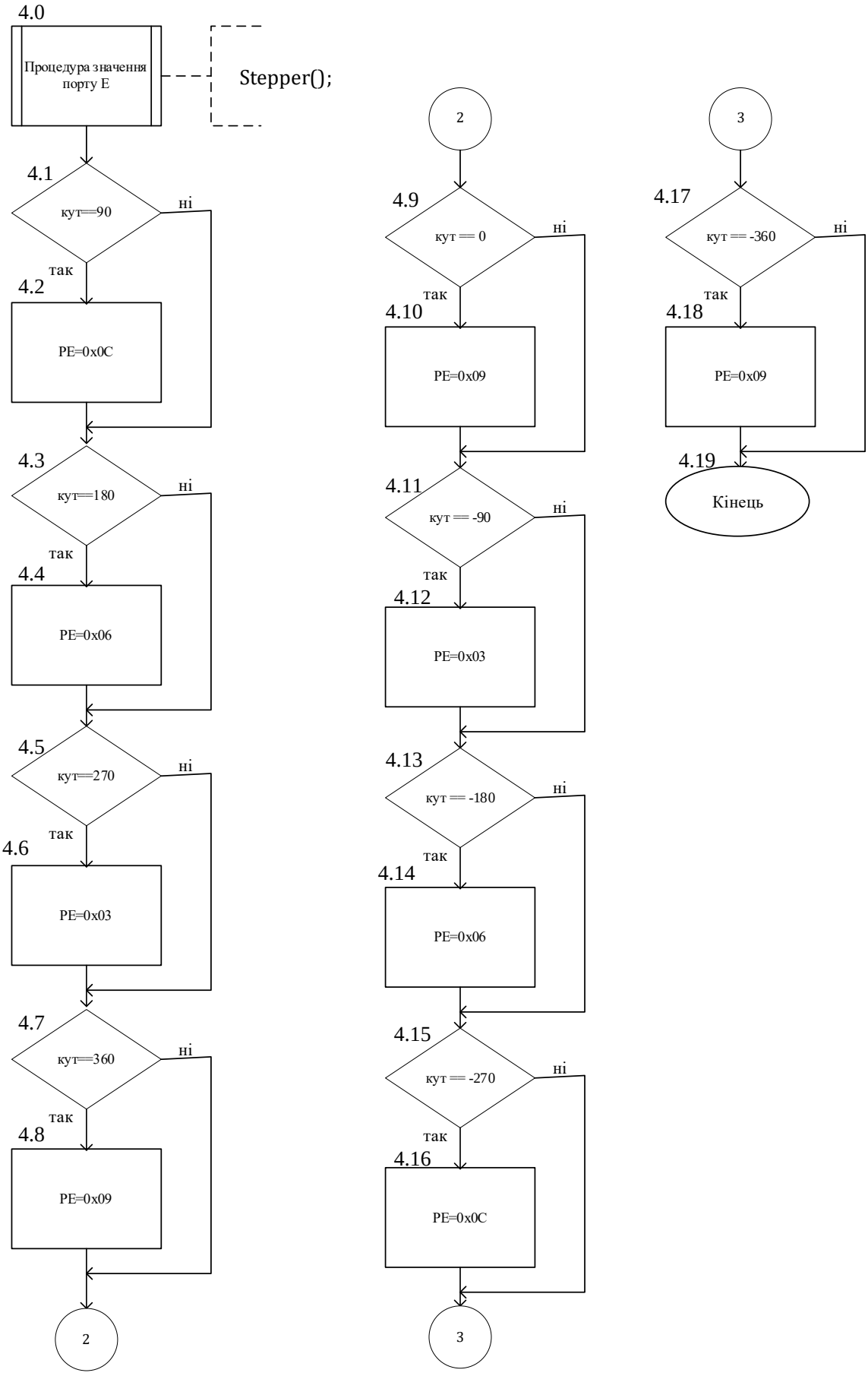


Рисунок 7.11 – Схема алгоритму роботи моделі уніполярного крокового двигуна для крокового режиму роботи



Продовження рисунку 7.11 (стор.160)



Закінчення рисунку 7.11 (стор.160, 161)

7.1.3 Робоча програма моделі уніполярного двигуна для крокового режиму роботи

Лістинг керуючої програми:

```
#include <avr/io.h>  // 1. Підключення бібліотек
#include <util/delay.h>

#define STEPUPAUTO (PINC & (1 << PINC0)) // 2. Перейменування портів для кращої читабельності коду
#define STEPDOWNAUTO (PINC & (1 << PINC1))
#define STEPUP (PINC & (1 << PINC2))
 #define STEPDOWN (PINC & (1 << PINC3))

int checkUp = 1; // 3. Ініціалізація змінних
int checkDown = 1;
int angle = 0;

void stepper()  // 4. Функція зміни значення порту E, порт виводу керуючого впливу на двигун в залежності від кута
{
if(angle==90)
PORTE=0x0C;
if(angle==180)
PORTE=0x06;
if(angle==270)
PORTE=0x03;
if(angle==360)
PORTE=0x09;
if (angle==0)
PORTE=0x09;
if(angle == -90)
PORTE=0x03;
if(angle== -180)
PORTE=0x06;
if(angle== -270)
PORTE=0x0C;
if(angle== -360)
PORTE=0x09;
}

void init_ports()  // 5. Функція ініціалізації портів
{
```

```
DDRC = 0x00; DDRE = 0xFF;//порт C на введення; порт E на виведення
}

int main(void) // 6. Головна функція
{
init_ports(); // 7. Виклик функції ініціалізації портів C та E
while (1) // 8. Поки є живлення, працює цикл
{

if(!STEPUPAUTO) // 9. Якщо натиснута кнопка STEPUPAUTO двигун обертається за годинниковою стрілкою
{
PORTE = 0x0C;    //10. Поступове змінювання значення порту E
_delay_ms(1000);
PORTE = 0x06;
_delay_ms(1000);
PORTE = 0x03;
_delay_ms(1000);
PORTE = 0x09;
_delay_ms(1000);
}

if(!STEPDOWNAUTO) // 11. Якщо натиснута кнопка STEPDOWNAUTO двигун обертається проти годинникової стрілки
{
PORTE = 0x03;              //12. Поступове змінювання значення порту E
_delay_ms(1000);
PORTE = 0x06;
_delay_ms(1000);
PORTE = 0x0C;
_delay_ms(1000);
PORTE = 0x09;
_delay_ms(1000);
}

if(!STEPUP && checkUp == 1) // 13. Виконати крок за годинниковою стрілкою, змінити значення змінної angle на +90 та викликати функцію stepper()
{
if(angle == 360)    //14.  Якщо кут становить 360,
{
angle = 0;              //15. то встановлюємо кут в 0
}

angle += 90;         //16. Значення кута змінюється на 90
```

```
checkUp = 0;        //  скидання прапорця checkUp  в 0
}

if(STEPUP)   //17. Якщо кнопка відпущена
{
checkUp = 1;  //18. Прапорець checkUp  встановлюється в 1
}

if(!STEPDOWN && checkDown == 1) // 19. Виконати крок проти годинникової
стрілки, змінити значення змінної angle на -90 та викликати функцію stepper()
{
if(angle == -360)   //20. Якщо кут становить -360,
{
angle = 0;              //21. то встановлюємо кут в 0
}

angle -= 90;           //22. Значення кута змінюється на -90
checkDown = 0;    //  скидання прапорця check Down  в 0
}

if(STEPDOWN)  //23. Якщо кнопка відпущена
{
checkDown = 1;   //24. Прапорець check Down встановлюється в 1
}

stepper(); // 25.  Виклик функції зміни кута повороту двигуна в покроковому режимі
}                //26. Закінчення циклу
}                //27. Закінчення програми
```

На рисунку 7.12 наведено керуючі послідовності для уніполярного двигуна для крокового режиму, які відображає осцилограф після натискання кнопки «Step +90 Auto Mode».

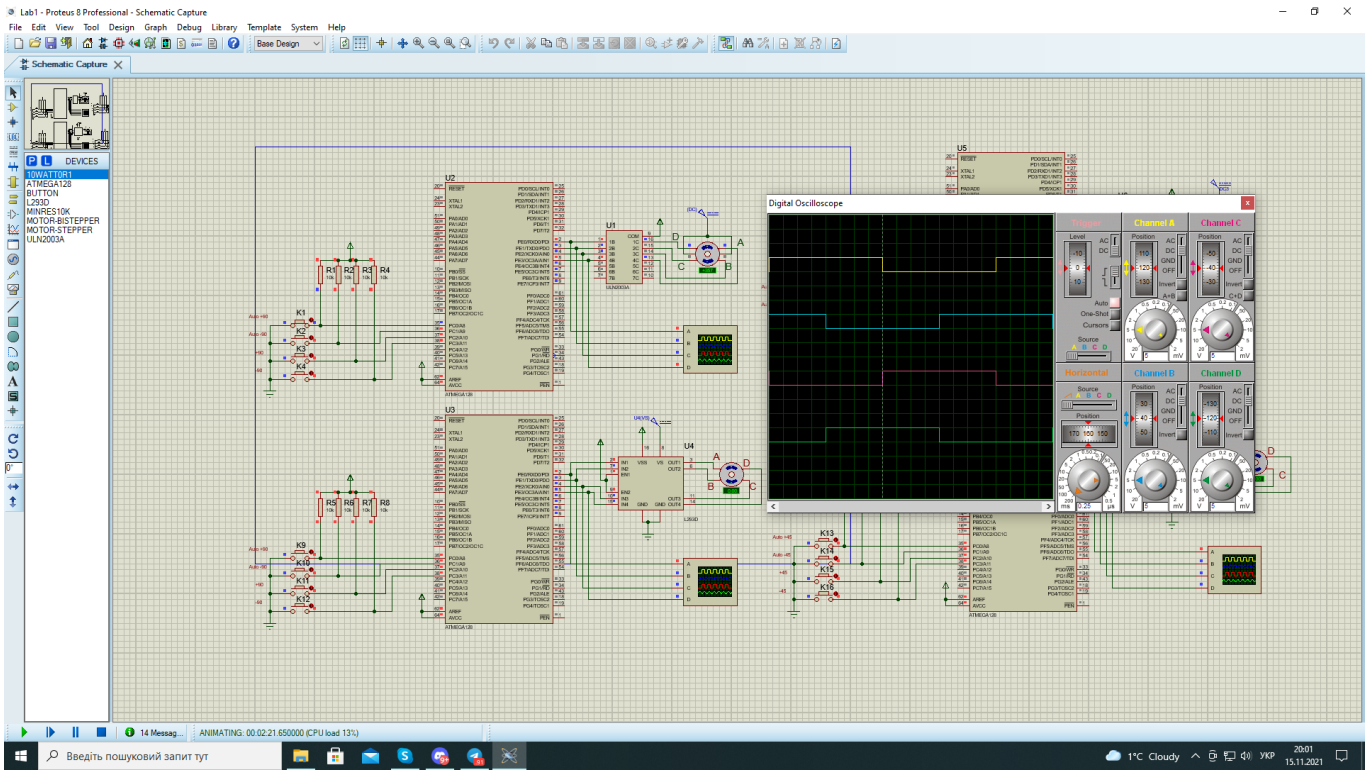


Рисунок 7.12 – Керуючі послідовності для уніполярного двигуна для крокового режиму роботи, які відображає осцилограф після натискання кнопки «Step +90 Auto Mode»

7.1.4 Схема алгоритму роботи моделі уніполярного крокового двигуна для напівкрокового режиму роботи

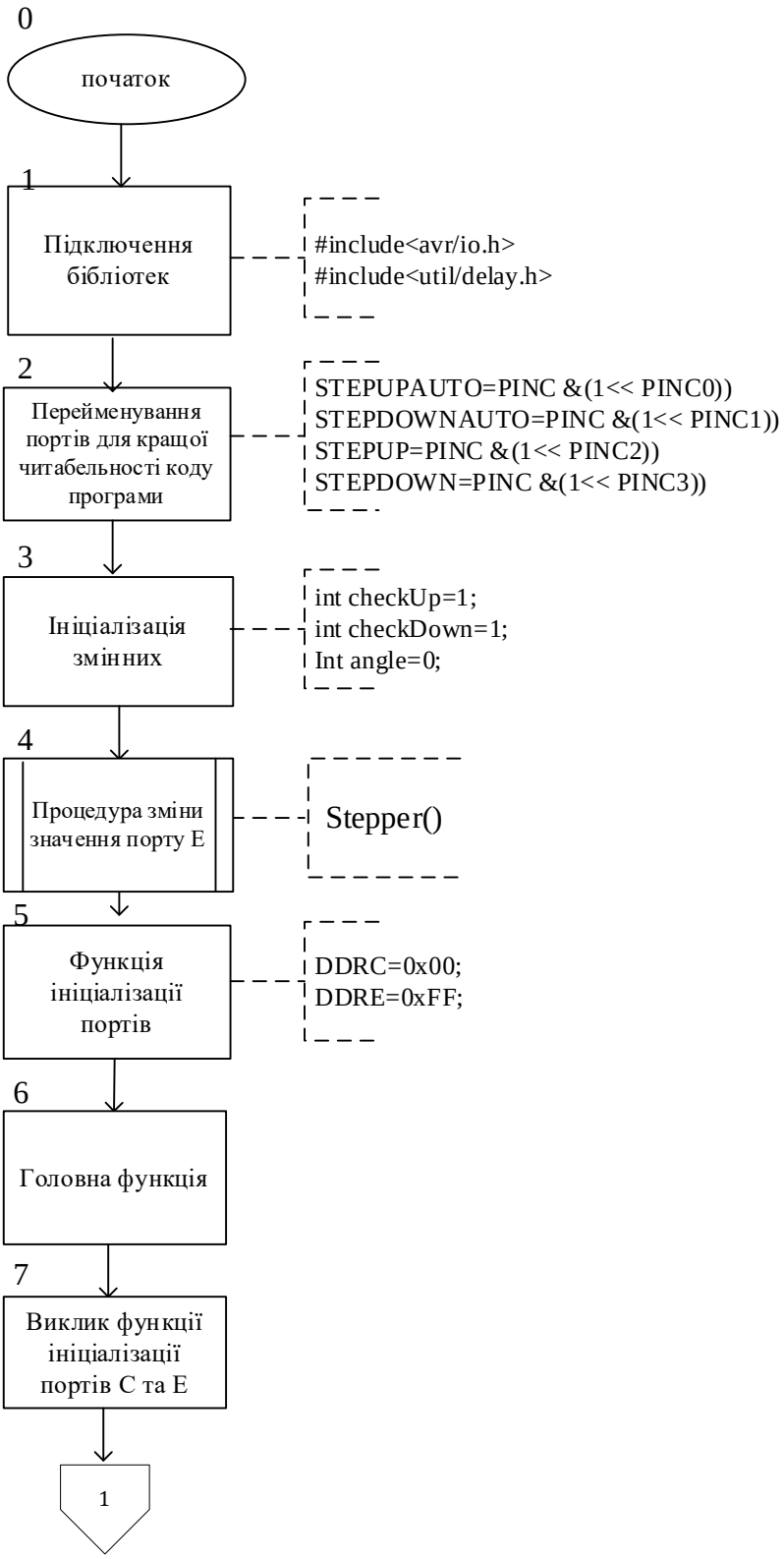
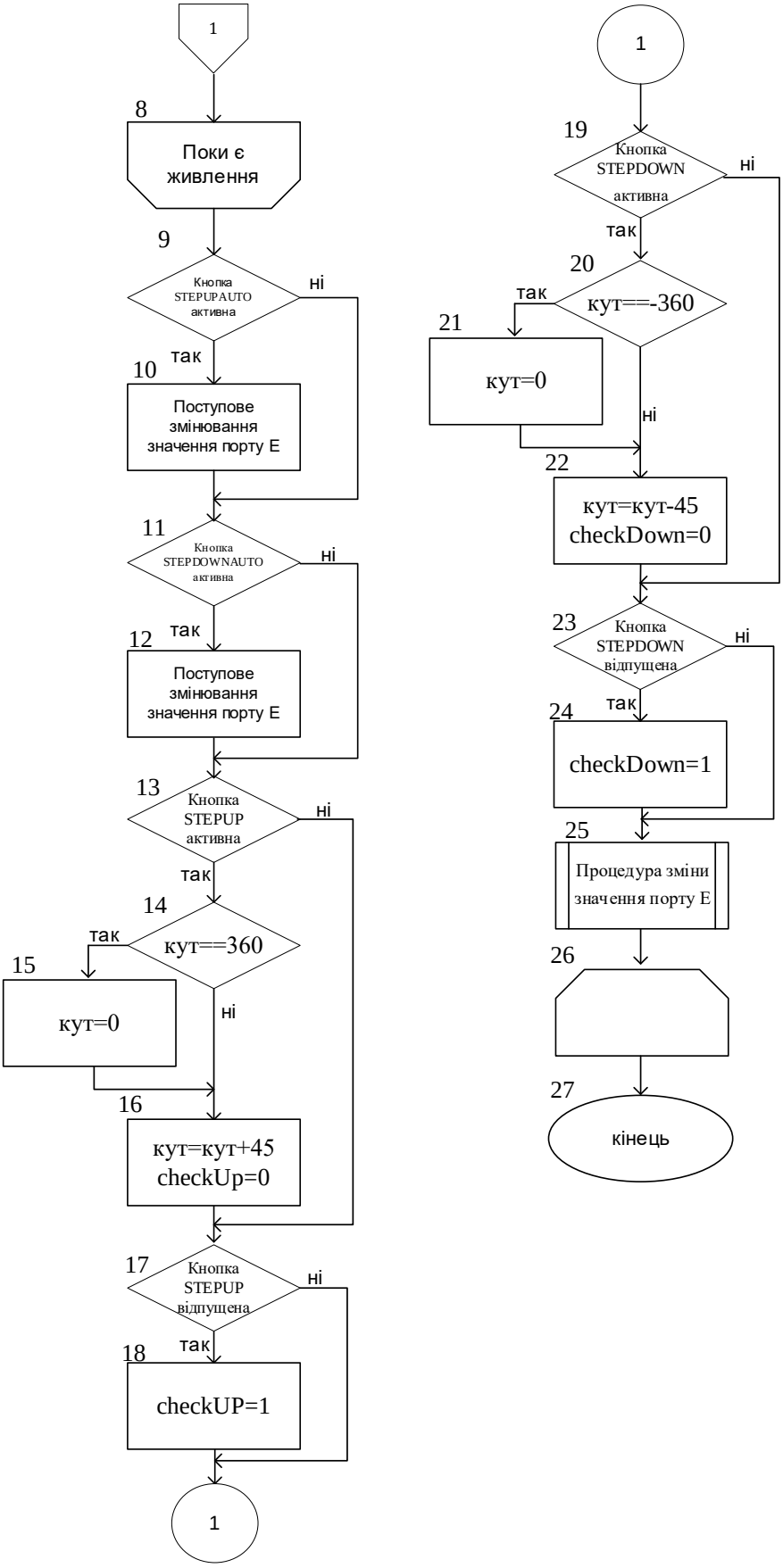
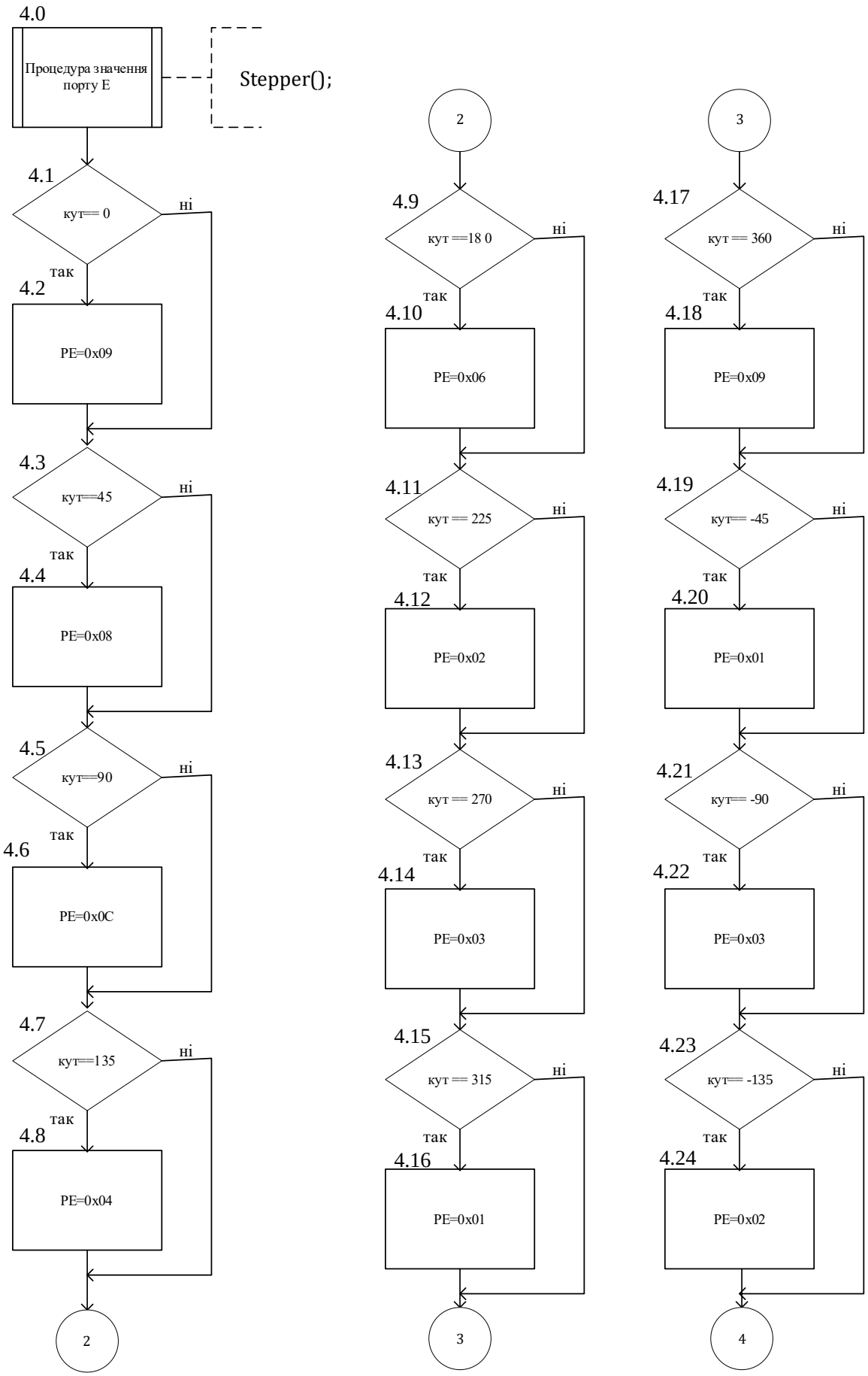


Рисунок 7.13 – Схема алгоритму роботи моделі уніполярного крокового двигуна для напівкрокового режиму роботи



Продовження рисунку 7.13 (стор. 167)



Закінчення рисунку 7.13 (стор. 167, 168)

7.1.5 Робоча програма моделі уніполярного крокового двигуна для напівкрокового режиму

```
#include <avr/io.h>  // 1. Підключення бібліотек
#include <util/delay.h>

#define STEPUPAUTO (PINC & (1 << PINC0)) // 2. Перейменування портів для
кращої       читабельності коду
#define STEPDOWNAUTO (PINC & (1 << PINC1))
#define STEPUP (PINC & (1 << PINC2))
#define STEPDOWN (PINC & (1 << PINC3))

int checkUp = 1; // 3. Ініціалізація змінних
int checkDown = 1;
int angle = 0;

    void   stepper()              // 4. Функція зміни значення порту E, порт виводу
    {                             керуючого впливу на двигун в залежності від кута
  if(angle  == 0)
  PORTE=0x09;
  if(angle == 45)
  PORTE=0x08;
  if(angle == 90)
  PORTE=0x0C;
  if(angle == 135)
  PORTE=0x04;
  if (angle == 180)
  PORTE=0x06;
 if(angle == 225)
 PORTE=0x02;
 if(angle == 270)
 PORTE=0x03;
 if(angle == 315)
 PORTE=0x01;
 if(angle == 360)
 PORTE=0x09;
 if(angle == -45)
 PORTE=0x01;
 if(angle == -90)
 PORTE=0x03;
 if(angle == -135)
 PORTE=0x02;
```

```
   if (angle == -180)
   PORTE=0x06;
   if(angle == -225)
   PORTE=0x04;
   if(angle == -270)
   PORTE=0x0C;
   if(angle == -315)
   PORTE=0x08;
   if(angle == -360)
   PORTE=0x09;
  }

void init_ports()     // 5. Функція ініціалізації портів
{
DDRC = 0x00; DDRE = 0xFF;//порт C на введення; порт E на виведення
}

int main(void) // 6 .Головна функція
{
init_ports();    // 7. Виклик функції ініціалізації портів C та E


while (1)  // 8. Поки є живлення, працює цикл
{
if(!STEPUPAUTO) // 9. Якщо натиснута кнопка STEPUPAUTO двигун обертається
за годинниковою стрілкою
{
PORTE = 0x08;              //10. Поступове змінювання значення порту E
_delay_ms(1000);
PORTE = 0x0C;
_delay_ms(1000);
PORTE = 0x04;
_delay_ms(1000);
PORTE = 0x06;
_delay_ms(1000);
PORTE = 0x02;
_delay_ms(1000);
PORTE = 0x03;
_delay_ms(1000);
PORTE = 0x01;
_delay_ms(1000);
PORTE = 0x09;
//_delay_ms(1000);
}
```

```
if(!STEPDOWNAUTO) // 11. Якщо натиснута кнопка STEPDOWNAUTO двигун обертається проти годинникової стрілки
{
PORTE = 0x01;              //12. Поступове змінювання значення порту E
_delay_ms(1000);
PORTE = 0x03;
_delay_ms(1000);
PORTE = 0x02;
_delay_ms(1000);
PORTE = 0x06;
_delay_ms(1000);
PORTE = 0x04;
_delay_ms(1000);
PORTE = 0x0C;
_delay_ms(1000);
PORTE = 0x08;
_delay_ms(1000);
PORTE = 0x09;
//_delay_ms(1000);
}

if(!STEPUP && checkUp == 1) // 13. Виконати крок за годинниковою стрілкою, змінити значення змінної angle на +45 та викликати функцію stepper()
{
if(angle == 360)    //14.  Якщо кут становить 360,
{
angle = 0;             //15. то встановлюємо кут в 0
}
angle += 45;         //16. Значення кута змінюється на 45
checkUp = 0;         //  скидання прапорця checkUp  в 0
}

if(STEPUP)        //17. Якщо кнопка відпущена
{
checkUp = 1;     //18. Прапорець checkUp  встановлюється в 1
}

if(!STEPDOWN && checkDown == 1) // 19. Виконати крок проти годинникової стрілки, змінити значення змінної angle на -45 та викликати функцію stepper()
{
if(angle == -360)      //20. Якщо кут становить -360,
{
angle = 0;          //21. то встановлюємо кут в 0
```

```
}
angle -= 45;        //22. Значення кута змінюється на -45
checkDown = 0;    //  скидання прапорця check Down  в 0
}

if(STEPDOWN)     //23. Якщо кнопка відпущена
{
checkDown = 1;   //24. Прапорець check Down встановлюється в 1
}

stepper(); // 25. Виклик функції зміни кута повороту двигуна в покроковому режимі
}              //26. Закінчення циклу
}              //27. Закінчення програми
```

На рисунку 7.14 наведено керуючі послідовності для уніполярного крокового двигуна для напівкрокового режиму, які відображає осцилограф після натискання кнопки «Step - 45 Auto Mode».

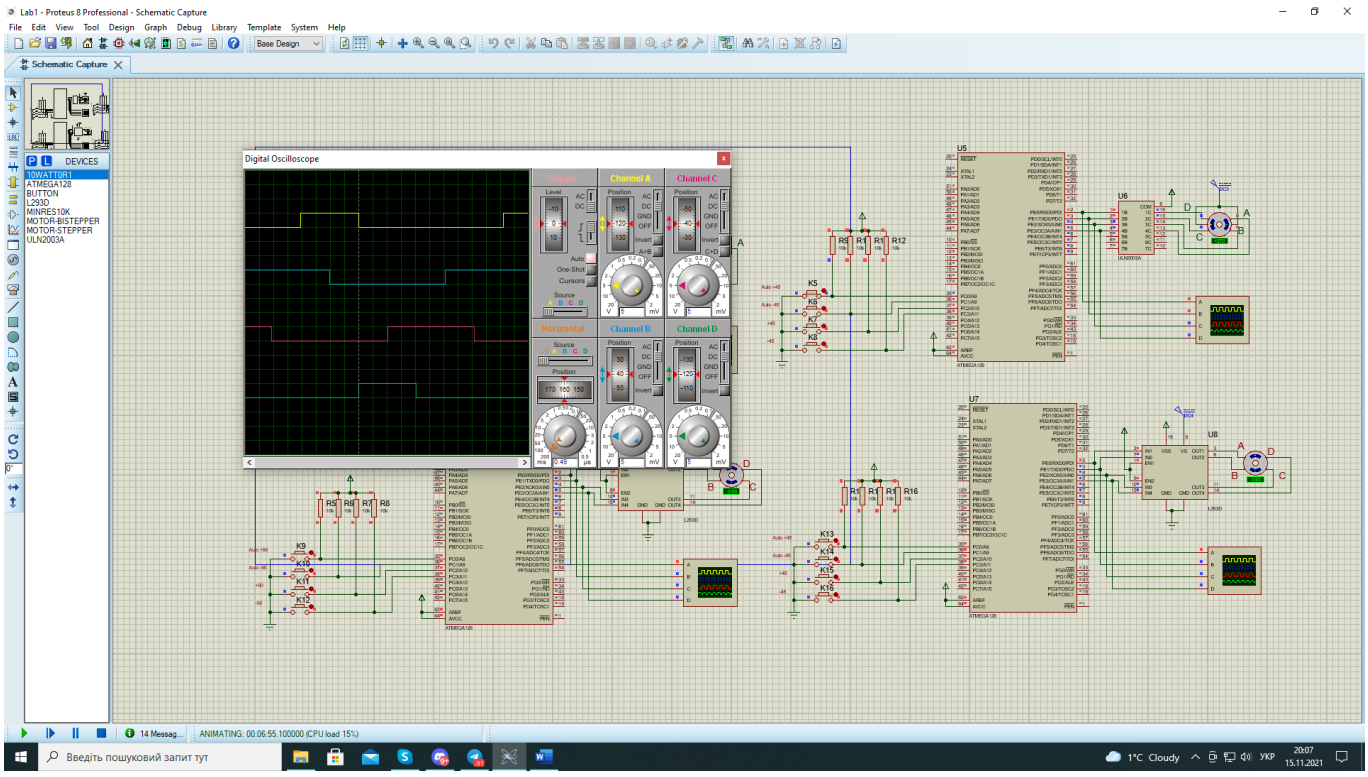


Рисунок 7.14 – Керуючі послідовності для уніполярного крокового двигуна для напівкрокового режиму, які відображає осцилограф після натискання кнопки «Step - 45 Auto Mode»

7.2 Моделювання біполярного крокового двигуна

7.2.1 Опис моделі

Робочу модель пристрою керування біполярним кроковим двигуном показано на рисунку 7.15.

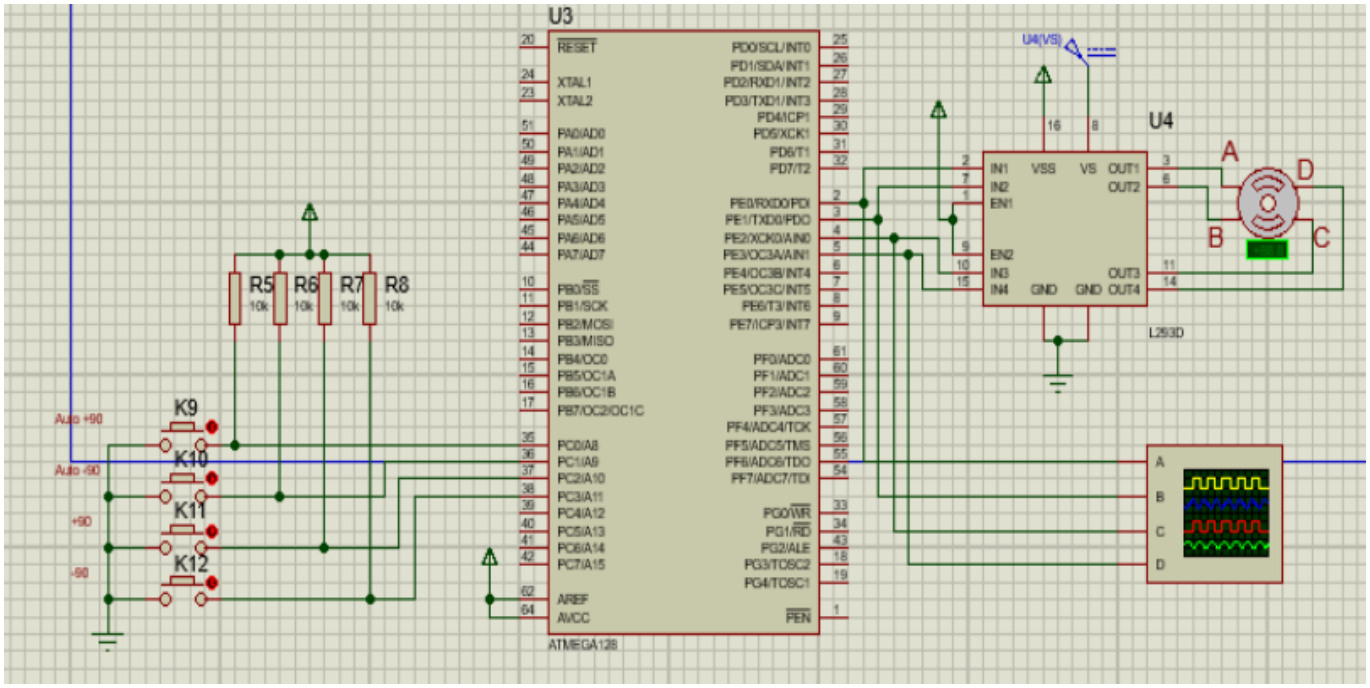


Рисунок 7.15 – Схема моделі пристрою керування біполярним кроковим двигуном

З лівого боку рисунка 7.15 наведено кнопки керування: K9, K10, K11, K12, на які через резистори R5, R6, R7, R8, відповідно, подається напруга: +5В. У правому куті зображено біполярний кроковий двигун, а трохи лівіше мікросхему L293D (U4), через яку мікроконтролер керує двигуном. Сам мікроконтролер ATMega128 знаходиться по центру. Зовнішні резистори використовуються через те, що на вхідних лініях мікроконтролера не використовуються внутрішні резистори, що підтягують. Вище на рисунку 7.2 наведено позначення виводів мікроконтролера.

Нижче більш докладніше пояснюється, що і куди підключається в моделі. Почнемо з кнопок, за допомогою яких здійснюється керування двигуном. Їх збільшений вигляд наведено на рисунку 7.16.

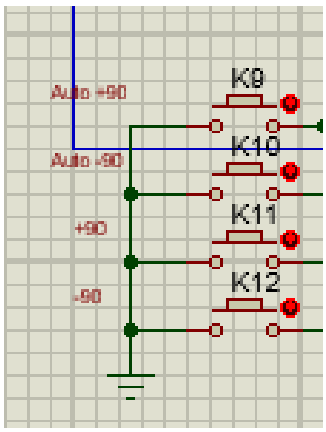


Рисунок 7.16 – Порядок розташування елементів керування

Кнопки є нормально розімкненими та не фіксованими, тобто після зняття прикладеного зусилля вони повертаються у вихідний стан. Це у нашому випадку означає наступне: у вихідному стані на входи мікроконтролера від кнопок подається високий рівень напруги, тобто логічна одиниця. А коли кнопки натискаються, на входах мікроконтролера спостерігається низький рівень напруги, тобто логічний нуль. Після наступного натискання на кнопку вона размикається і на відповідному вході МК знову з`являється логічна одиниця.

Кнопка K9 (Step +90 Auto Mode) відповідає за вмикання та вимикання двигуна. Тобто коли двигун вимкнено, то вона його ввімкне. Якшо натиснути на кнопку, то наш двигун виконає цикл кроків від 0 до 360 градусів та повернеться в початковий стан – 0 градусів. У однофазному режимі з цілим кроком крок двигуна дорівнює 90 градусів. Кнопку підключено до виводу P1.0 мікроконтролера, який програмується, як вхід.

Кнопка K10 (Step –90 Auto Mode) працює так само як і K9, але при натисканні, відбуватиметься цикл кроків в зворотному порядку до циклу кроків кнопки K9. У однофазному режимі з цілим кроком крок двигуна дорівнює 90 градусів. Кнопку підключено до виводу P1.1 мікроконтролера, який програмується, як вхід .

Кнопка K11 (Step +90) відповідає за виконання одного кроку двигуна за годинниковою стрілкою.

У однофазному режимі з цілим кроком крок двигуна дорівнює 90 градусів. Кнопку підключено до виводу P1.2 мікроконтролера, який програмується, як вхід .

Кнопка K12 (Step -90) відповідає за виконання одного кроку двигуна проти годинникової стрілки. У однофазному режимі з цілим кроком крок двигуна дорівнює 90 градусів. Кнопку підключено до виводу P1.3 мікроконтролера, який програмується, як вхід .

На рисунку 7.17 наведено підключення мікросхеми L293D до мікроконтролера.

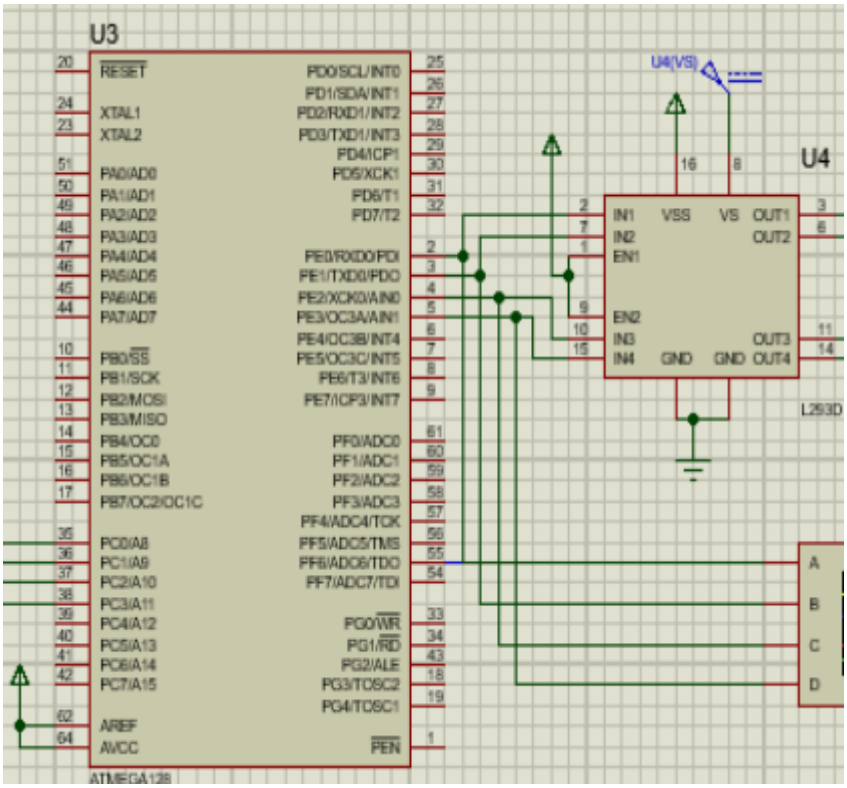


Рисунок 7.17 – Підключення мікросхеми L293D до мікроконтролера

Виводи 2, 3, 4 та 5 мікроконтролера, тобто PE.0, PE.1, PE.2 та PE.3 запрограмовані як виходи і підключені, відповідно, до входів IN1, IN2, IN3 та IN4 мікросхеми L293D (U4).

В залежності від сигналів керування, на виходах PE.0, PE.1, PE.2 та PE.3 буде або високий рівень напруги, або низький, і в залежності від цього мікросхема буде керувати кроковим двигуном.

Нижче описано використання інших виводів мікросхеми L293D, яку зображено на рисунку 7.17. Виводи GND підключено до землі. Виводи EN1, EN2 підключено до напруги живлення: +5В.

На вхід VSS подається напруга живлення самої мікросхеми L293D: +5В. На вхід VS подається напруга живлення біполярного крокового двигуна: +12В.

Підключення до мікросхеми L293D крокового двигуна зображено на рисунку 7.18. Біполярний кроковий двигун підключено до виводів 3, 6, 11 та 14 мікросхеми L293D, а саме OUT1, OUT2, OUT3 та OUT4.

На ці виходи подаються керуючі імпульсні сигнали, що і повертають наш кроковий двигун в положення, яке відповідає заданому кроку.

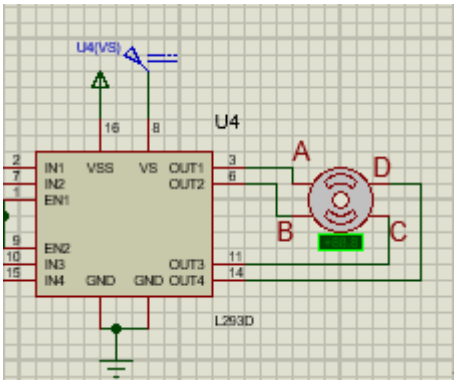


Рисунок7.18 – Схема підключення драйвера до крокового двигуна

Драйвер біполярного крокового двигуна

В якості драйвера біполярного крокового двигуна обрано мікросхему L293D (рисунок 7.19).

Рисунок 7.19 – Зовнішній вигляд мікросхеми L293D

Мікросхема L293D включає два драйвери для керування двома електродвигунами відносно невеликої потужності.

Мікросхема має дві пари входів для керування напрямом обертання двигунів і дві пари виходів для підключення електродвигунів. Крім того, у L293D є входи для вмикання кожного з драйверів.

Також, L293D забезпечує розділене живлення для мікросхеми і для керованих нею двигунів, що дозволяє підключати електродвигуни з більшою напругою живлення, ніж у мікросхеми. Розділення живлення мікросхеми і електродвигунів також необхідне для зменшення завад, які викликано стрибками напруги при керуванні двигунами.

Обидва драйвери, що входять у склад мікросхеми, мають ідентичний принцип роботи, тому розглянемо принцип роботи лише одного з них. Схематичний вигляд одного з драйверів мікросхеми L293D наведено на рисунку 7.20.

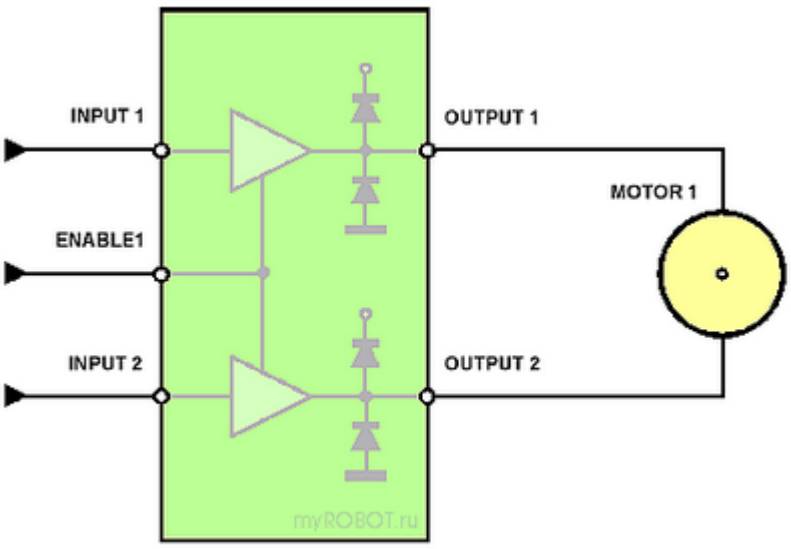


Рисунок 7.20 – Схематичний вигляд одного з драйверів мікросхеми L293D

До виходів OUTPUT1 та OUTPUT2 підключається двигун (MOTOR1). На вхід ENABLE1, який відповідає за ввімкнення драйвера, подають керуючий сигнал, наприклад, підключають його до додатного полюсу джерела живлення: +5V. Якщо

при цьому на входи INPUT1 та INPUT2 не подаються відповідні керуючі сигнали, то двигун обертатися не буде.

Якщо вхід INPUT1 з'єднати з додатним полюсом джерела живлення, а вхід INPUT2 – з від'ємним, то двигун почне рухатися за годинниковою стрілкою. Якщо вхід INPUT1 з'єднаємо з від`ємним полюсом джерела живлення, а вхід INPUT2 – з додатним, то двигун почне рухатися в іншу сторону.

Якщо подати сигнали одного рівня одразу на два керуючих входи INPUT1 та INPUT2, тобто підключити обидва входи до додатного полюсу джерела живлення або до від`ємного, то двигун рухатися не буде.

Якщо прибрати керуючий сигнал з входу ENABLE1, то при будь яких варіантах сигналів на входах INPUT1 та INPUT2 двигун рухатися не буде. В нашій моделі входи ENABLE1 та ENABLE2 підключаються до напруги: +5В. Змінюючи сигнали керування на входах INPUT1 та INPUT2 ми будемо подавати відповідні сигнали керування на обмотки біполярного двигуна.

На рисунку 7.21 наведено нумерацію та позначення виводів мікросхеми.

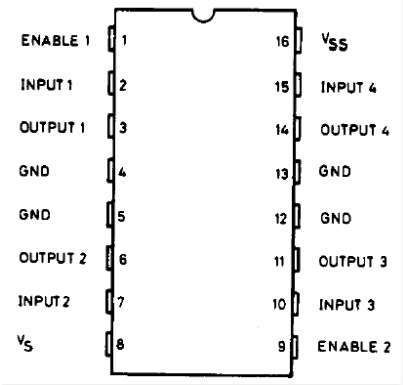


Рисунок 7.21 – Нумерація та позначення виводів мікросхеми L293D

Призначення виводів:

- входи ENABLE1 та ENABLE2 відповідають за вмикання відповідного драйвера, що входить у склад мікросхеми. В нашій моделі ці входи підключаються до напруги: +5В;
- входи INPUT1 та INPUT2 керують формуванням відповідних сигналів керування на обмотці OUTPUT1, OUTPUT2 біполярного двигуна;

– входи INPUT3 та INPUT4 керують формуванням відповідних сигналів керування на обмотці OUTPUT3, OUTPUT4 біполярного двигуна;
– контакт Vs з'єднують з додатним полюсом джерела живлення двигунів або просто з додатним полюсом джерела живлення, якщо схема і двигуни живляться від одного джерела. Простіше кажучи, цей контакт відповідає за живлення електродвигунів;
– контакт Vss з'єднують з додатним полюсом джерела живлення. Цей контакт забезпечує живлення самої мікросхеми;
– чотири контакти GND з'єднують з "землею" (спільним дротом – від`ємним полюсом джерела живлення). Крім того, за допомогою цих контактів зазвичай забезпечують тепловідвід від мікросхеми, тому їх краще всього паяти на достатньо широку контактну поверхню на друкованій платі.

Дана мікросхема має наступні основні технічні характеристики:

– напруга живлення двигунів (Vs): 4,5...36 В;
– напруга живлення мікросхеми (Vss): +5 В;
– допустимий струм навантаження: 600 мА (на кожен канал);
– піковий (максимальний) струм на виході: 1,2 А (на кожен канал);
– логічний "0" вхідної напруги: до 1,5 В;
– логічна "1" вхідної напруги: 2,3...7 В;
– в мікросхему вбудовано захист від перегріву.

Біполярний кроковий двигун

Біполярний кроковий двигун (Bipolar Stepper Motors) – це електричний двигун з двома обмотками, які вище на рисунку 7.15 позначено як A, B та C, D. Подача електричного струму на обмотки двигуна у відповідному напрямку та послідовності приводить до того, що його ротор фіксується в строго певному положенні та обертається на заданий кут. Струм в обмотках повинен переполюсовуватися драйвером мостового типу, наприклад, L293D. Завдяки цьому кут повороту ротора залежить від кількості послідовних перемикань обмоток, а

швидкість обертання ротора дорівнює частоті перемикання обмоток, яку помножено на кут повороту ротора за одне перемикання.

Існують три можливі послідовності включення обмоток. Перша послідовність – це включення обмоток у порядку: AB, CD, BA та DC (BA та DC означає зворотне включення обмоток, коли струм через обмотку протікає у протилежному напрямку). Ця послідовність зветься як однофазний режим з цілим кроком. У кожен фіксований момент часу включено тільки одну обмотку.

Наступний режим зветься двофазним з цілим кроком, коли обидві фази включаються одночасно та ротор завжди центрує себе між двома полюсами. Друга послідовність має такий вигляд: AB та CD; BA та CD; BA та DC та AB та DC.

Ще один варіант – це включення фаз у послідовності: AB; AB та CD; CD; BA та CD; BA; BA та DC; DC; AB та DC. Ця послідовність зветься як режим з напівкроком. В цьому режимі біполярний двигун, який використовується в нашій моделі, за один крок буде обертатися на кут у 45 градусів.

Згідно схемі моделювання, яку наведено на рисунку 7.15, керуюча послідовність імпульсів передається через лінії PE.0, PE.1, PE.2 та PE.3 мікроконтролера. Ця послідовність у чотирьохрозрядному двійковому коді виглядає наступним чином: 0001 для кута 0 градусів (IN1=1, IN2=0, IN3=0, IN4=0); 1000 для кута 90 градусів (IN1=0, IN2=0, IN3=0, IN4=1); 0010 для кута 180 градусів (IN1=0, IN2=1, IN3=0, IN4=0); 0100 для кута 270 градусів (IN1=0, IN2=0, IN3=1, IN4=0) та 0001 для повернення у початковий стан (кут дорівнює 0 градусів, IN1=1, IN2=0, IN3=0, IN4=0).

Нижче наведено схему алгоритму роботи біполярного двигуна у однофазному режимі з цілим кроком та керуючі програми для однофазного режиму з цілим кроком та напівкрокового режиму.

7.2.2 Схема алгоритму роботи біполярного крокового двигуна у однофазному режимі з цілим кроком

Схему алгоритму роботи біполярного крокового двигуна у однофазному режимі з цілим кроком наведено на рисунку 7.22.

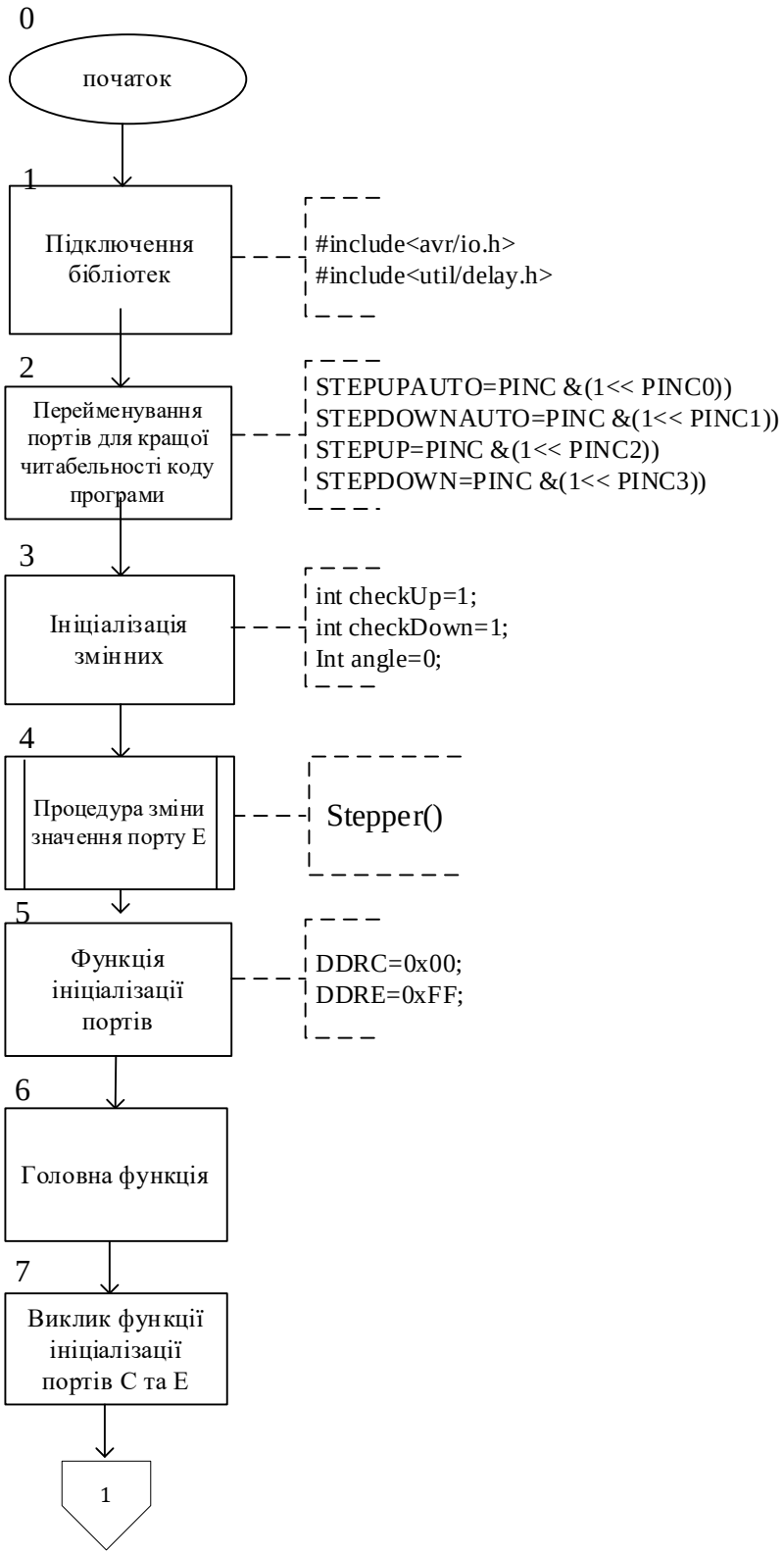
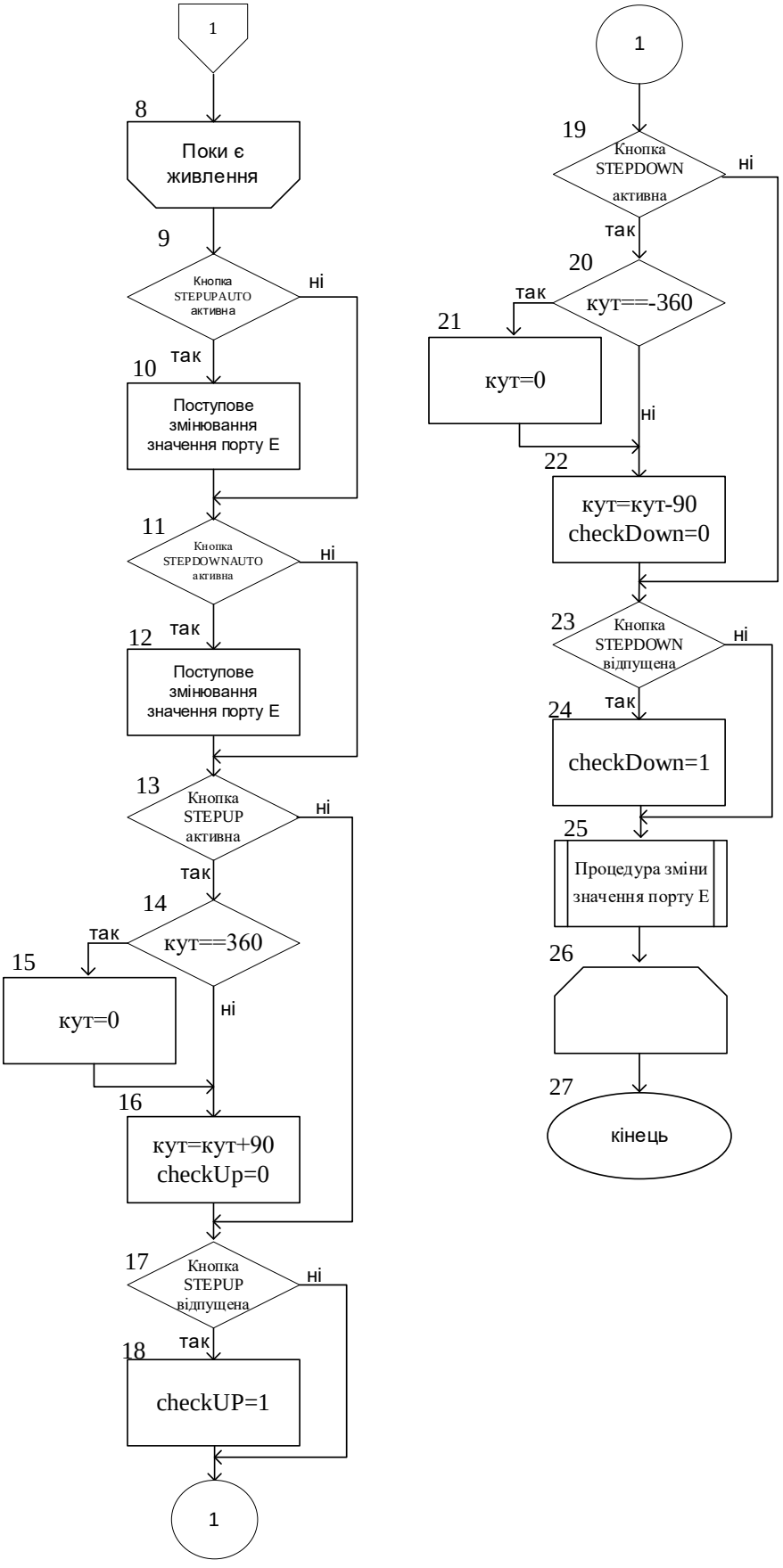
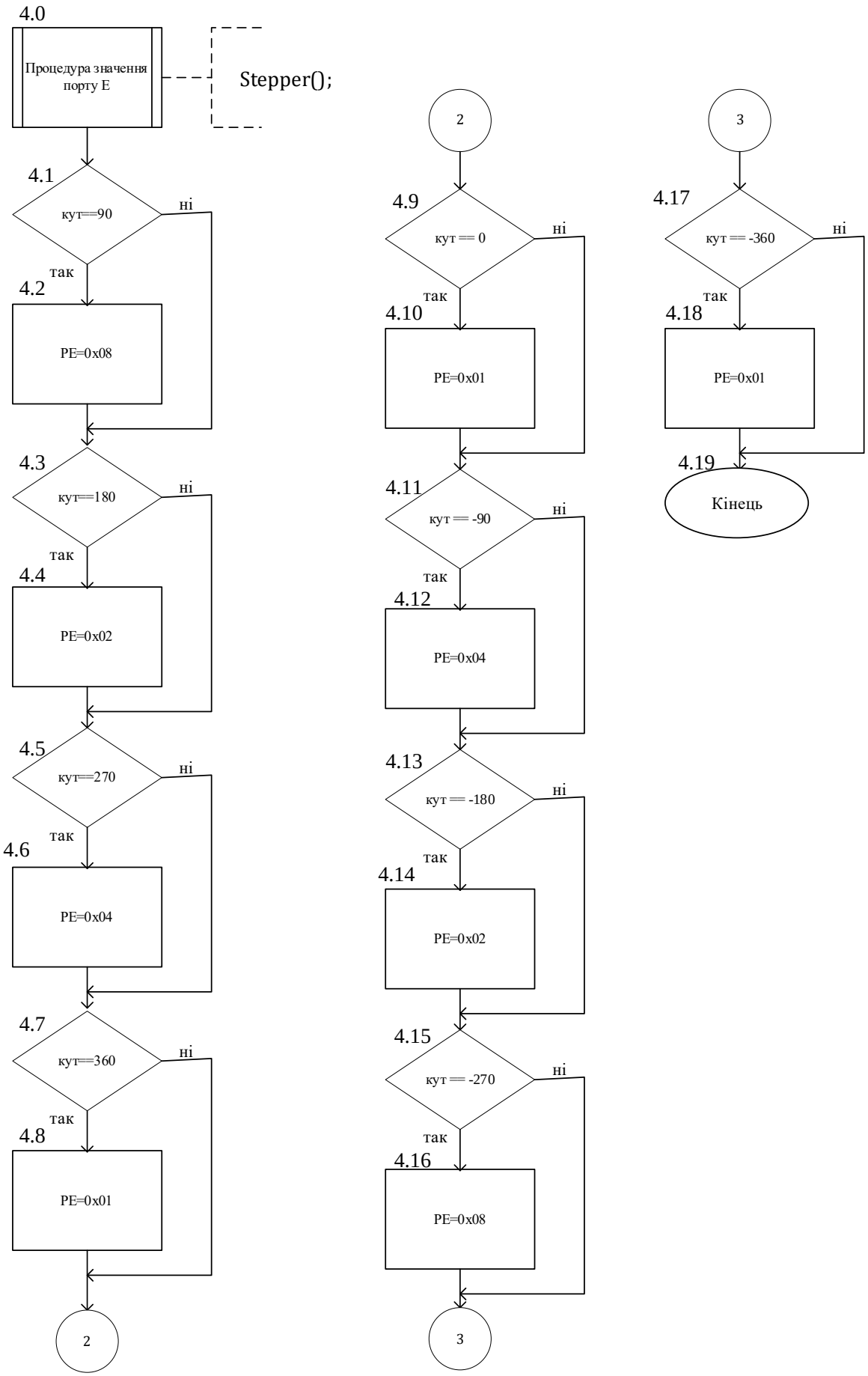


Рисунок 7.22 – Схема алгоритму роботи моделі біполярного крокового двигуна для режиму роботи з цілим кроком



Продовження рисунку 7.22 (стор. 182)



Закінчення рисунку 7.22 (стор. 182, 183)

7.2.3 Робоча програма роботи біполярного крокового двигуна у однофазному режимі з цілим кроком

```
#include <avr/io.h> // 1. Підключення бібліотек
#include <util/delay.h>
#define STEPUPAUTO (PINC & (1 << PINC0)) // 2. Перейменовуємо порти для
                                                кращої читабельності коду
#define STEPDOWNAUTO (PINC & (1 << PINC1))
#define STEPUP (PINC & (1 << PINC2))
#define STEPDOWN (PINC & (1 << PINC3))
int checkUp = 1; // 3. Ініціалізація змінних
int checkDown = 1;
int angle = 0;

void stepper()    // 4. Функція зміни значення порту E, порт виводу керуючого впливу
на двигун в залежності від кута
{
if(angle == 90)
PORTE = 0x08;
if(angle == 180)
PORTE = 0x02;
if(angle == 270)
PORTE = 0x04;
if(angle == 360)
PORTE = 0x01;
if(angle == 0)
PORTE = 0x01;
if(angle == -90)
PORTE = 0x04;
if(angle == -180)
PORTE = 0x02;
if(angle == -270)
PORTE = 0x08;
if(angle == -360)
PORTE = 0x01;
}

void init_ports() // 5. Функція ініціалізації портів
{
DDRC = 0x00; // порт C на введення
DDRE = 0xFF;// порт E на виведення
}
```

```
int main(void) // 6. Головна функція
{
init_ports(); // 7.  Виклик функції ініціалізації портів C та E


while (1) // 8. Поки є живлення, працює цикл
{
if(!STEPUPAUTO) // 9. Якщо натиснута кнопка STEPUPAUTO двигун обертається за годинниковою стрілкою
{
PORTE = 0x08;              //10. Зміна значення порту E
_delay_ms(1000);
PORTE = 0x02;
_delay_ms(1000);
PORTE = 0x04;
_delay_ms(1000);
PORTE = 0x01;
_delay_ms(1000);
}

if(!STEPDOWNAUTO) // 11. Якщо натиснута кнопка STEPDOWNAUTO двигун обертається проти годинникової стрілки
{
PORTE = 0x04;              //12. Зміна значення порту E
_delay_ms(1000);
 PORTE = 0x02;
_delay_ms(1000);
PORTE = 0x08;
_delay_ms(1000);
PORTE = 0x01;
_delay_ms(1000);
}

if(!STEPUP && checkUp == 1) // 13.  Виконати крок за годинниковою стрілкою, змінити значення змінної angle на +90 та викликати функцію stepper()
{
if(angle == 360)    //14.  Якщо кут становить 360,
{
angle = 0;             //15. то встановлюємо кут в 0
}

angle += 90;         //16. Значення кута змінюється на 90
checkUp = 0;         //  скидання прапорця checkUp  в 0
```

```
}

if(STEPUP)   //17. Якщо кнопка відпущена
{
checkUp = 1;  //18. Прапорець checkUp  встановлюється в 1
}

if(!STEPDOWN && checkDown == 1) // 19. Виконати крок проти годинникової стрілки, змінити значення змінної angle на -90 та викликати функцію stepper()
{
if(angle == -360)   //20. Якщо кут становить -360,
{
angle = 0;              //21. то встановлюємо кут в 0
}

angle -= 90;           //22. Значення кута змінюється на -90
checkDown = 0;    //  скидання прапорця check Down  в 0
}

if(STEPDOWN)  //23. Якщо кнопка відпущена
{
checkDown = 1;   //24. Прапорець check Down встановлюється в 1
}

stepper(); // 25.  Виклик функції зміни кута повороту двигуна в покроковому режимі
}                //26. Закінчення циклу
}                //27. Закінчення програми
```

На рисунку 7.23 наведено керуючі послідовності, які відображає осцилограф після натискання кнопки «Step +90 Auto Mode», для біполярного крокового двигуна, який працює у однофазному режимі з цілим кроком.

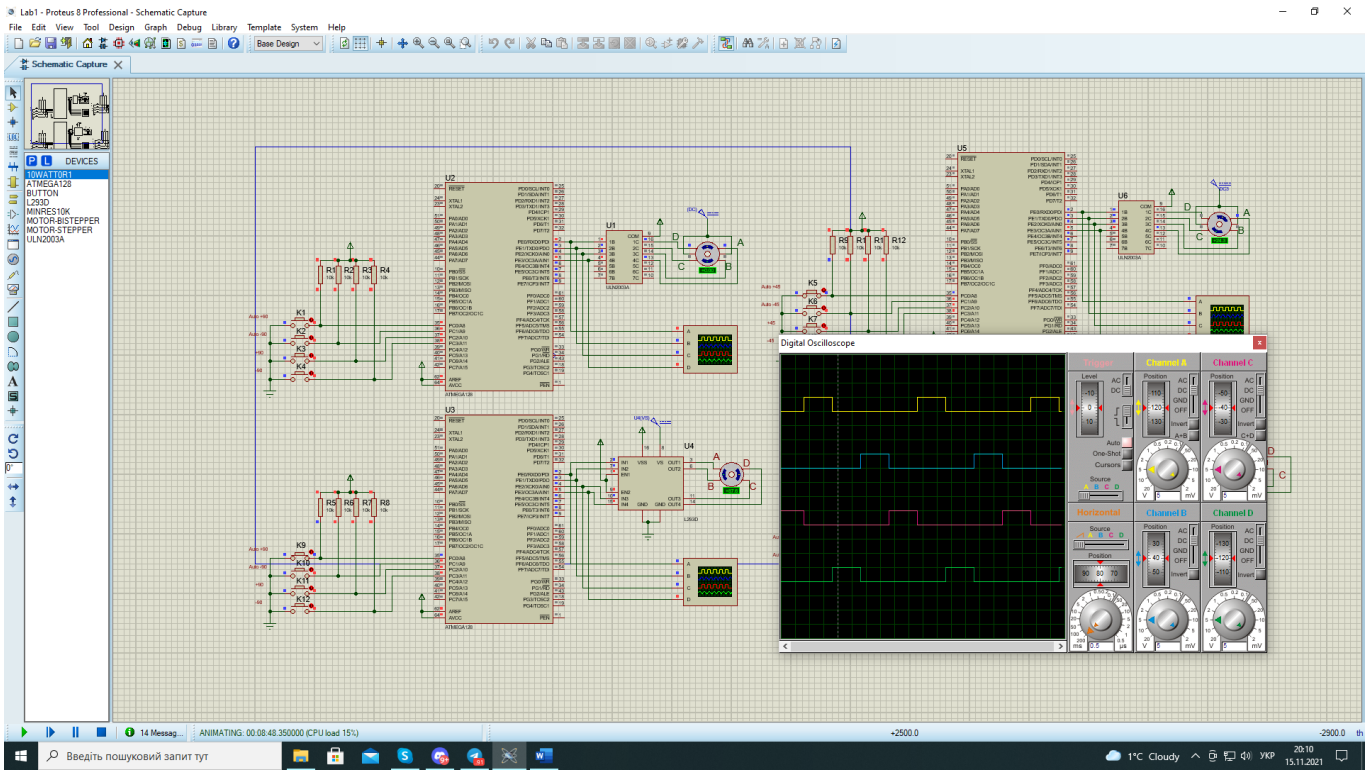


Рисунок 7.23 – Керуючі послідовності для біполярного крокового двигуна для однофазного режиму з цілим кроком, які відображає осцилограф після натискання кнопки «Step +90 Auto Mode»

7.2.4 Схема алгоритму роботи біполярного крокового двигуна для напівкрокового режиму роботи

Схему алгоритму роботи біполярного крокового двигуна у напівкроковому режимі наведено на рисунку 7.24.

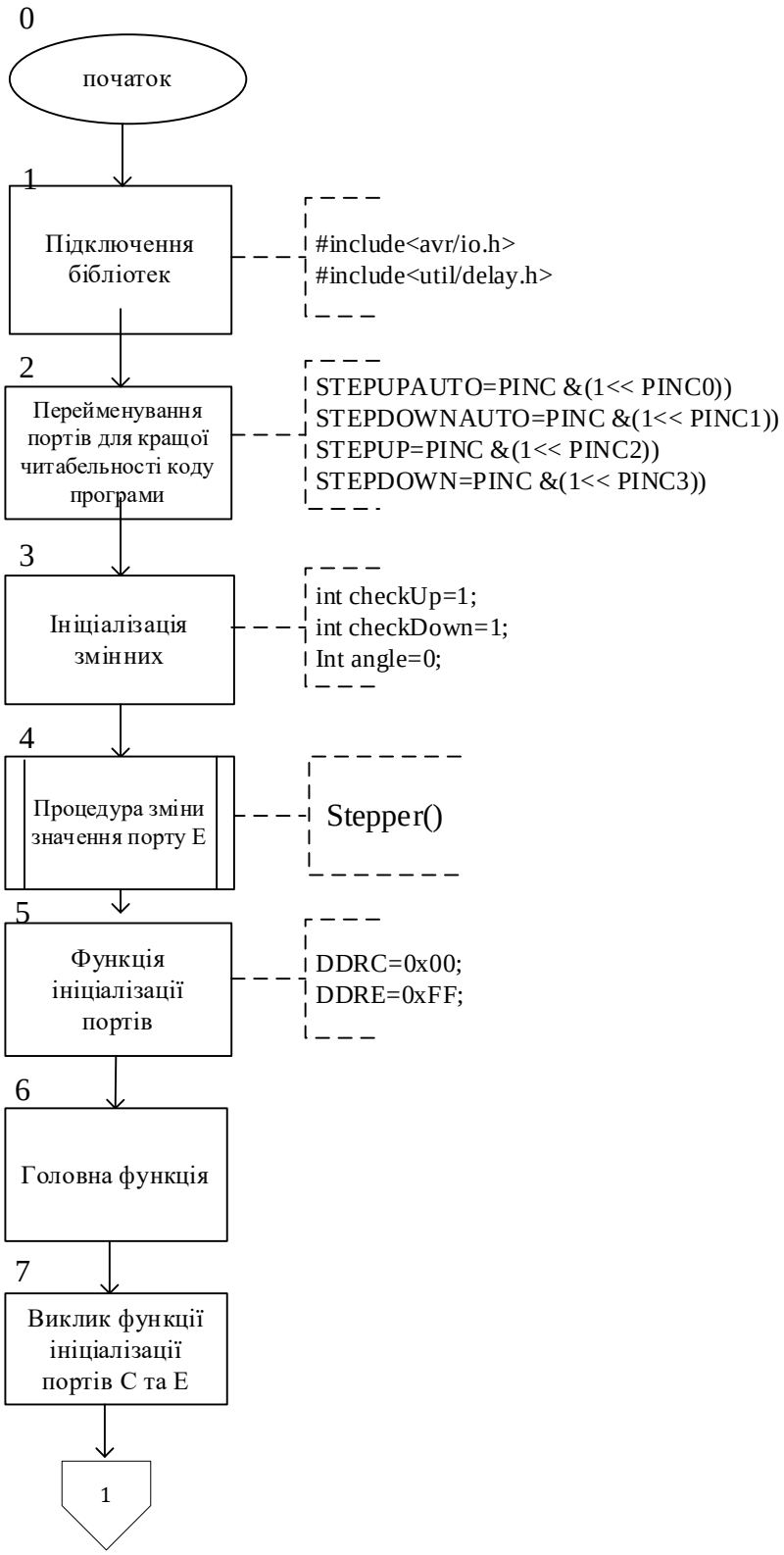
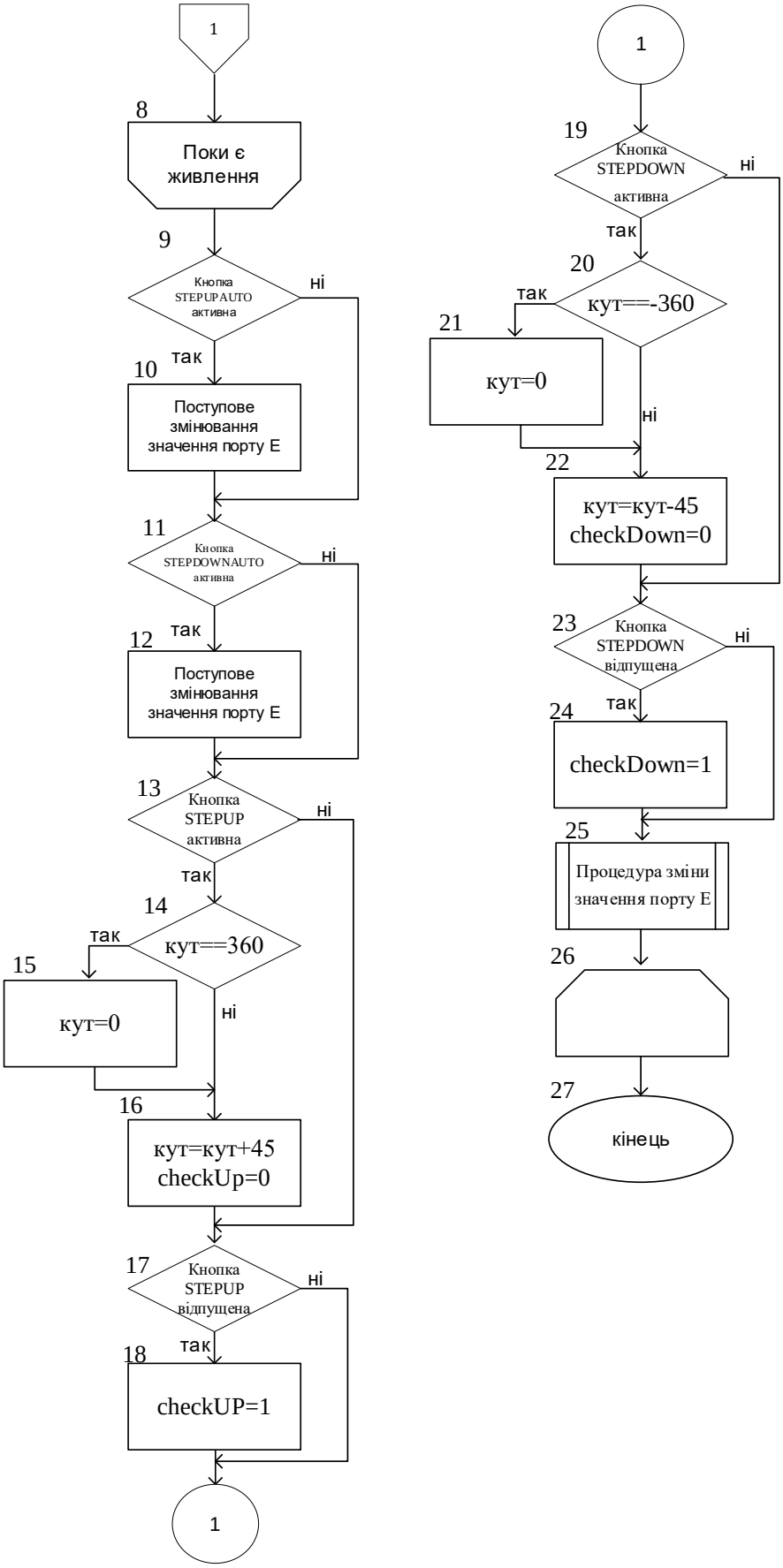
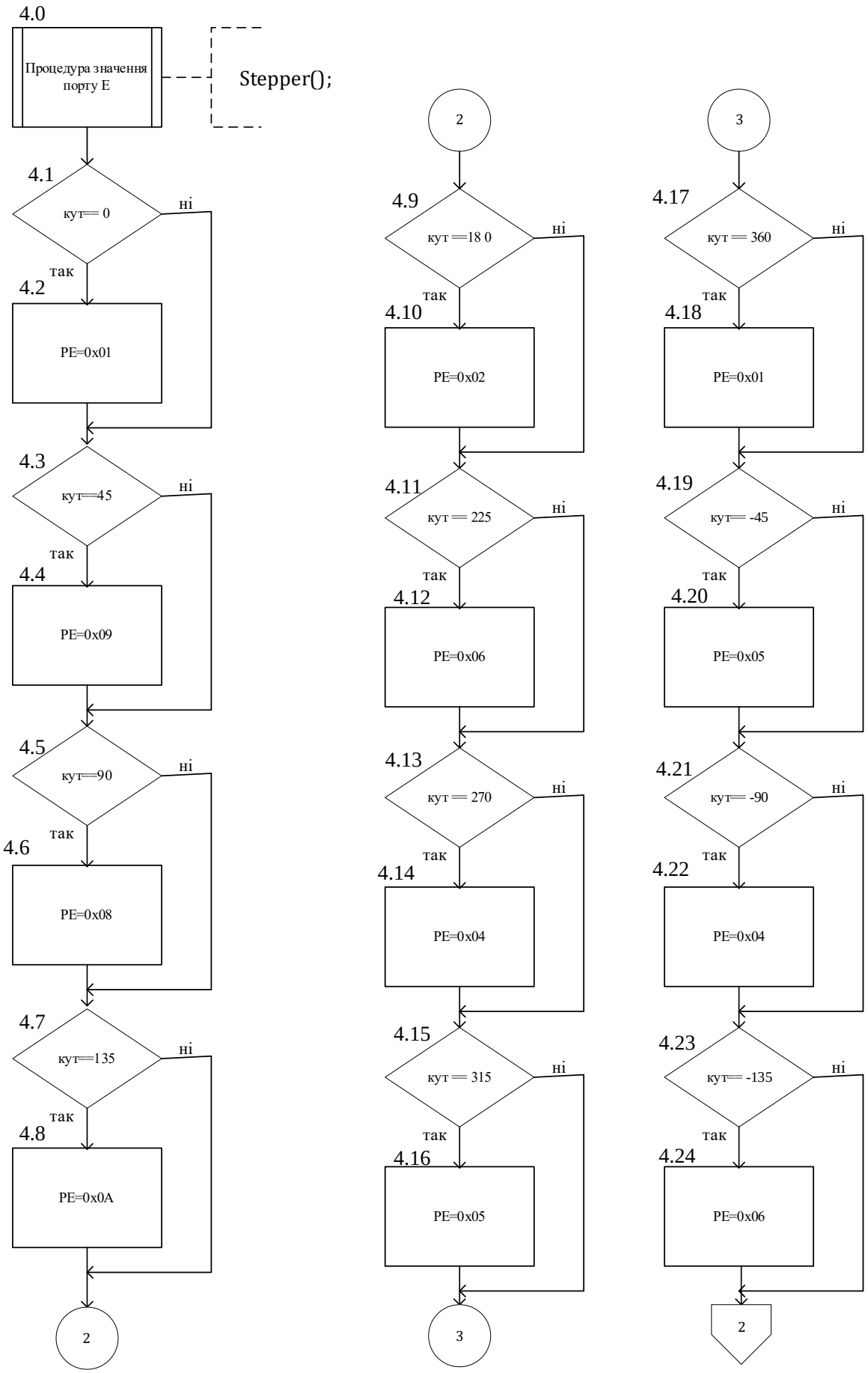


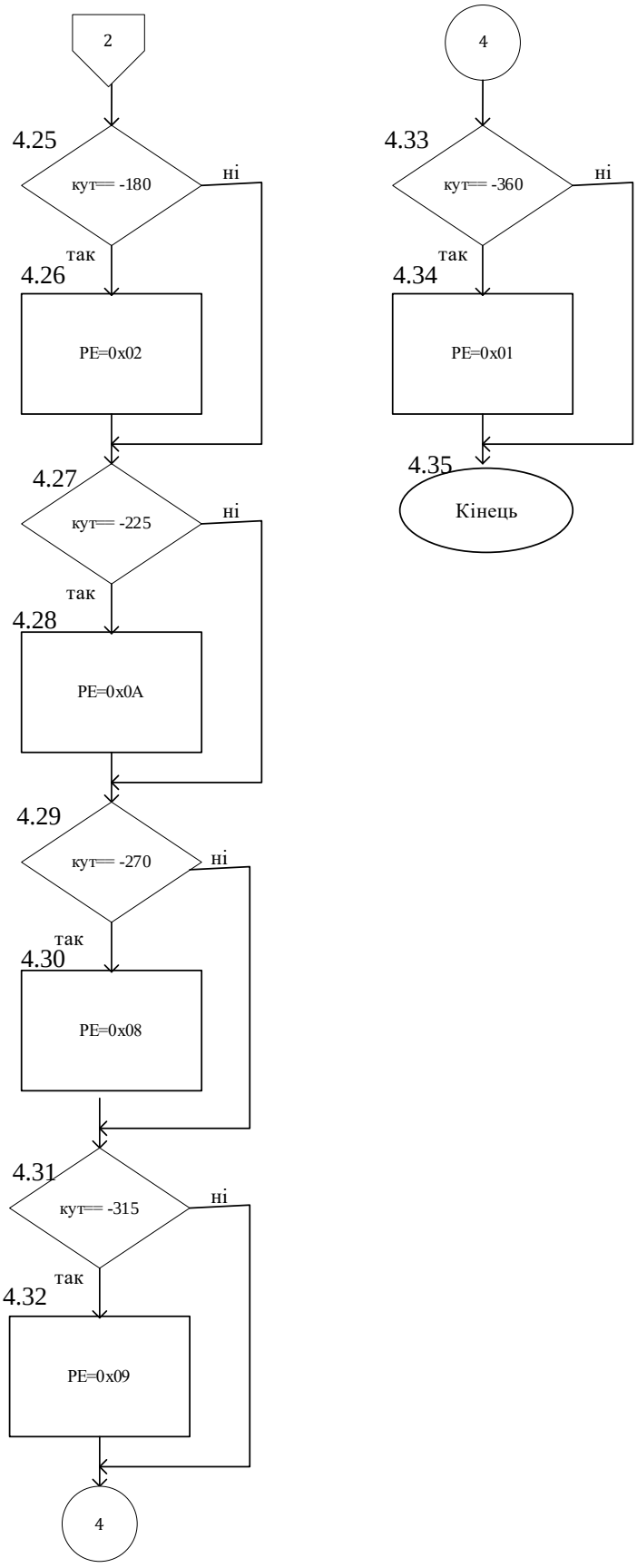
Рисунок 7.24 – Схема алгоритму роботи моделі біполярного крокового двигуна для напівкрокового режиму роботи



Продовження рисунку 7.24 (стор.189)



Продовження рисунку 7.24 (стор.189, 190)



Закінчення рисунку 7.24 (стор.189…191)

7.2.5 Робоча програма роботи біполярного крокового двигуна у напівкроковому режимі

```
#include <avr/io.h> // 1. Підключення бібліотек #include <util/delay.h>

#define STEPUPAUTO (PINC & (1 << PINC0)) // 2. Перейменування портів для кращої читабельності коду
#define STEPDOWNAUTO (PINC & (1 << PINC1))
 #define STEPUP (PINC & (1 << PINC2))
#define STEPDOWN (PINC & (1 << PINC3))

int checkUp = 1; // 3. Ініціалізація змінних
int checkDown = 1;
int angle = 0;

void stepper()        // 4. Функція зміни значення порту E, порт виводу керуючого впливу на двигун в залежності від кута
{
if(angle == 0)
PORTE=0x01;
if(angle == 45)
PORTE=0x09;
if(angle == 90)
PORTE=0x08;
if(angle == 135)
PORTE=0x0A;
if (angle == 180)
PORTE=0x02;
if(angle == 225)
PORTE=0x06;
if(angle == 270)
PORTE=0x04;
if(angle == 315)
PORTE=0x05;
if(angle == 360)
PORTE=0x01;
if(angle == -45)
PORTE=0x05;
if(angle == -90)
PORTE=0x04;
if(angle == -135)
PORTE=0x06;
```

```
if (angle == -180)
PORTE=0x02;
if(angle == -225)
PORTE=0x0A;
if(angle == -270)
PORTE=0x08;
if(angle == -315)
PORTE=0x09;
if(angle == -360)
PORTE=0x01;
}

void init_ports()   //5.  Функція ініціалізації портів
{
DDRC = 0x00; //порт C на введення;
DDRE = 0xFF;// порт E на виведення
}

int main(void) // 6. Головна функція
{
init_ports(); // 7. Виклик функції ініціалізації портів C та E

while (1) // 8. Поки є живлення, працює цикл

{
if(!STEPUPAUTO) // 9. Якщо натиснута кнопка STEPUPAUTO двигун обертається за годинниковою стрілкою
{
PORTE = 0x09;              //10. Поступове змінювання значення порту E
_delay_ms(1000);
PORTE = 0x08;
_delay_ms(1000);
PORTE = 0x0A;
_delay_ms(1000);
PORTE = 0x02;
_delay_ms(1000);
PORTE = 0x06;
_delay_ms(1000);
PORTE = 0x04;
_delay_ms(1000);
PORTE = 0x05;
_delay_ms(1000);
PORTE = 0x01;
//_delay_ms(1000);
```

```
}

if(!STEPDOWNAUTO) // 11. Якщо натиснута кнопка STEPDOWNAUTO двигун обертається проти годинникової стрілки
{
PORTE = 0x05;              //12. Поступове змінювання значення порту E
_delay_ms(1000);
 PORTE = 0x04;
_delay_ms(1000);
 PORTE = 0x06;
_delay_ms(1000);
PORTE = 0x02;
_delay_ms(1000);
PORTE = 0x0A;
_delay_ms(1000);
 PORTE = 0x08;
_delay_ms(1000);
PORTE = 0x09;
_delay_ms(1000);
PORTE = 0x01;
//_delay_ms(1000);
}

if(!STEPUP && checkUp == 1) // 13. Виконати крок за годинниковою стрілкою, змінити значення змінної angle на +45 та викликати функцію stepper()
{
if(angle == 360)      //14.  Якщо кут становить 360,
{
angle = 0;              //15. то встановлюємо кут в 0
}
angle += 45;          //16. Значення кута змінюється на 45
checkUp = 0;        //  скидання прапорця checkUp  в 0
}
if(STEPUP)           //17. Якщо кнопка відпущена
{
checkUp = 1;          //18. Прапорець checkUp  встановлюється в 1
}

if(!STEPDOWN && checkDown == 1) // 19. Виконати крок проти годинникової стрілки, змінити значення змінної angle на -45 та викликати функцію stepper()
{
if(angle == -360)           //20. Якщо кут становить -360,
{
angle = 0;                    //21. то встановлюємо кут в 0
```

```
}

angle -= 45;            //22. Значення кута змінюється на -45
checkDown = 0;    //  скидання прапорця check Down  в 0
}

if(STEPDOWN)  //23. Якщо кнопка відпущена
{
checkDown = 1; //24. Прапорець check Down встановлюється в 1
}

stepper(); // 25. Виклик функції зміни кута повороту двигуна в покроковому режимі
}                //26. Закінчення циклу
}                //27. Закінчення програми
```

На рисунку 7.25 наведено керуючі послідовності, які відображає осцилограф після натискання кнопки «Step +45 Auto Mode», для біполярного крокового двигуна, який працює у напівкроковому режимі.

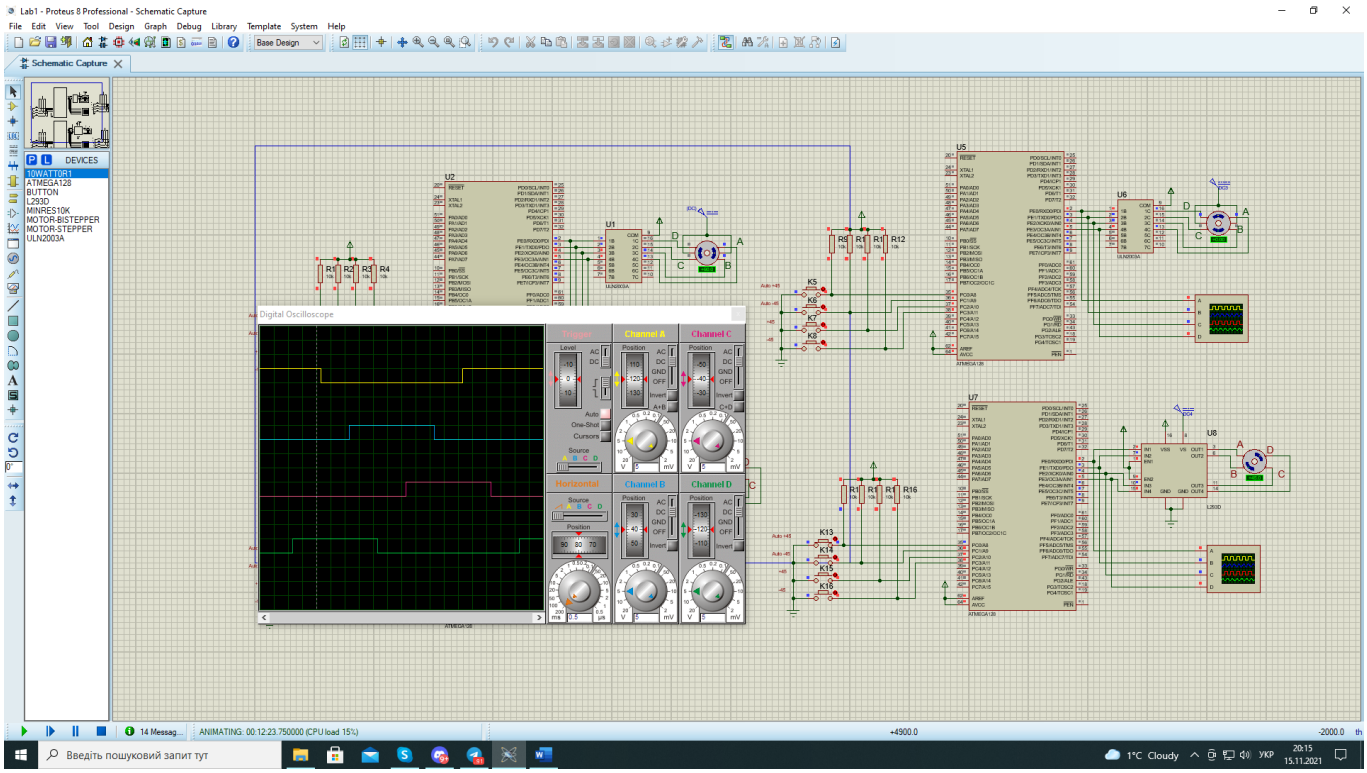


Рисунок 7.25 – Керуючі послідовності для біполярного крокового двигуна для напівкрокового режиму, які відображає осцилограф після натискання кнопки «Step +45 Auto Mode»

7.3 Зміст звіту

Звіт по роботі повинен містити:

- схему моделі;
- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

Контрольні запитання та завдання

1) Дайте визначення уніполярним кроковим двигунам.
2) Дайте визначення біполярним кроковим двигунам.
3) Як уніполярний кроковий двигун підключається до мікроконтролера?
4) Опишіть роботу драйвера уніполярного крокового двигуна.
5) Назвіть та опишіть способи керування кроковими двигунами.
6) Наведіть та пояснить схему алгоритму роботи та робочу програму моделі уніполярного крокового двигуна для крокового режиму роботи.
7) Наведіть та пояснить схему алгоритму роботи та робочу програму моделі уніполярного крокового двигуна для напівкрокового режиму роботи.
8) Поясніть результати моделювання уніполярного та біполярного крокових двигунів.
9) Як біполярний кроковий двигун підключається до мікроконтролера?
10) Опишіть роботу драйвера біполярного крокового двигуна.
11) Опишіть алгоритм та робочу програму моделі біполярного крокового двигуна.
12) Дайте порівняльну характеристику крокових двигунів.

8 ЛАБОРАТОРНА РОБОТА №5. ДОСЛІДЖЕННЯ МОДЕЛІ ПРИСТРОЮ КЕРУВАННЯ ДВИГУНОМ ПОСТІЙНОГО СТРУМУ

Тема: Моделювання пристрою керування двигуном постійного струму з використанням мікроконтролера сім'ї AVR.

Мета: Користуючись пакетом Proteus 8.6 дослідити роботу пристрою керування двигуном постійного струму.

8.1 Порядок виконання роботи

1) Створити модель пристрою в пакеті Proteus 8.6.
2) Розробити схему алгоритму роботи моделі та робочу програму.
3) Створити hex-файл та підключити його до мікроконтролера.
4) Запустити модель та виконати її дослідження згідно методичних вказівок.
5) Зробити відповідні висновки.

8.2 Загальна характеристика

8.2.1 Опис моделі

Робочу модель пристрою керування двигуном постійного струму показано на рисунку 8.1.

З лівого боку моделі наведено кнопки керування: K1, K2, K3, K4, на які через резистори R1, R2, R3, R4 відповідно, подається напруга живлення. У правому нижньому куті зображено двигун постійного струму (DC Motor), а трохи лівіше – мікросхему L293D, через яку мікроконтролер керує двигуном за допомогою ШІМ-сигналу. В якості мікроконтролера обрано мікросхему ATmega128, яка знаходиться у правому куті моделі.

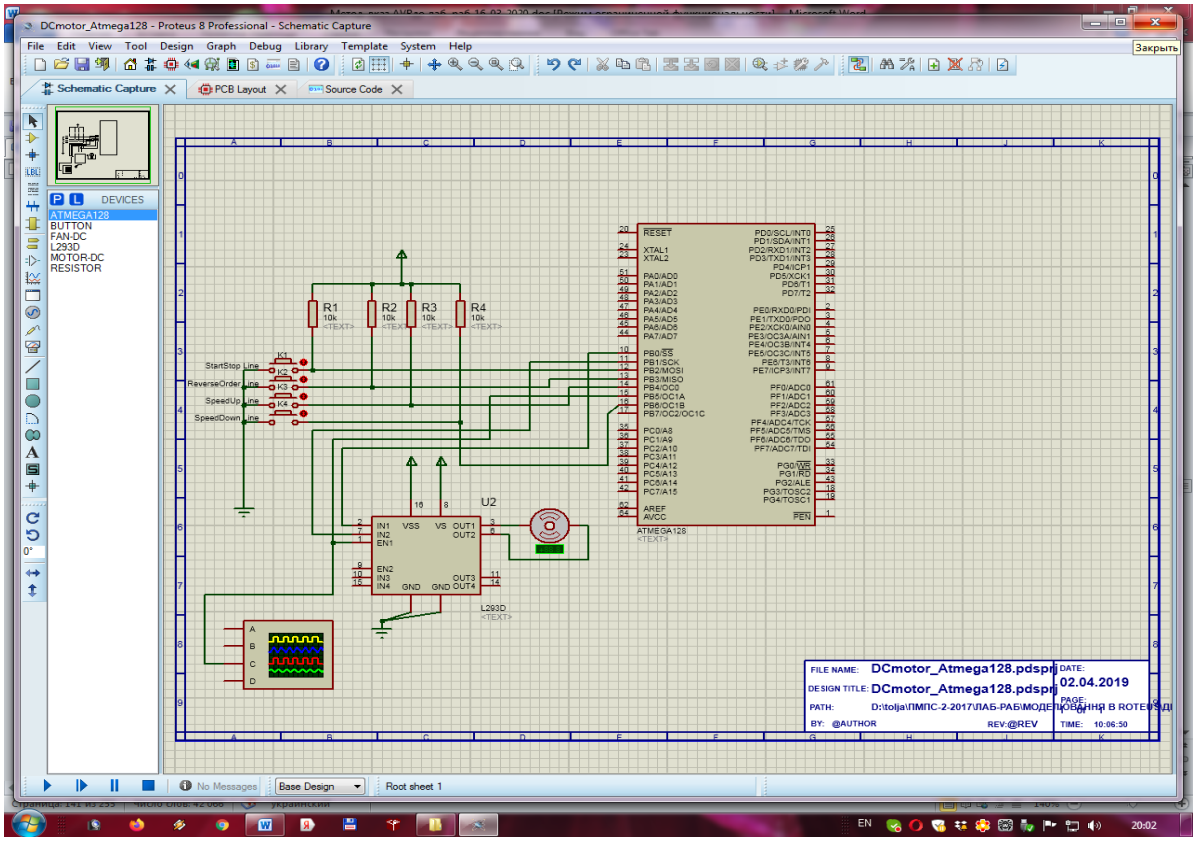


Рисунок 8.1 – Схема моделі пристрою керування двигуном постійного струму

Зовнішні резистори використовуються через те, що на вхідних лініях мікроконтролера не підключаються внутрішні підтягуючі резистори.

На рисунку 8.2 наведено зовнішній вигляд та позначення виводів мікроконтролера.

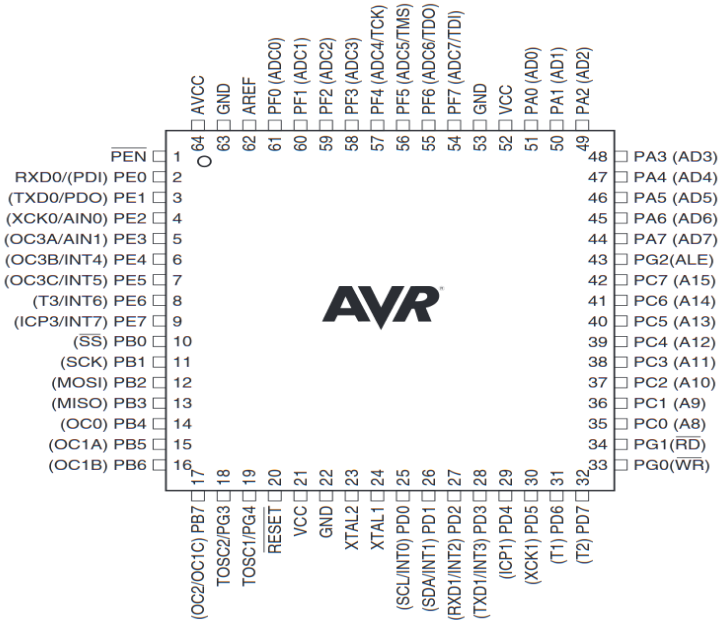


Рисунок 8.2 – Зовнішній вигляд та позначення виводів мікроконтролера

На рисунку 8.3 наведено збільшений вигляд кнопок керування та порядок їх розташування.

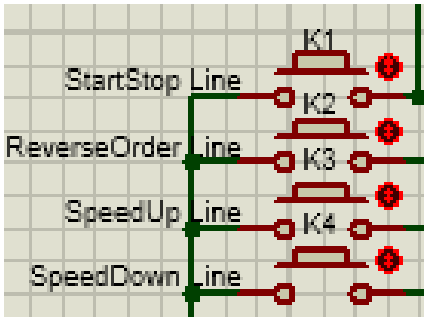


Рисунок 8.3 – Порядок розташування кнопок керування

У вихідному стані кнопки є нормально розімкненими. В цьому випадку на відповідні лінії порту В мікроконтролера (PB) подаються високі рівні напруги, тобто логічні одиниці. Коли кнопки замикаються, на входи мікроконтролера подаються низькі рівні напруги, тобто логічні нулі. Перемикання сигналу з логічного нуля у логічну одиницю від обраної кнопки сприймається мікроконтролером як відповідний сигнал керування.

Кнопка K1 (StartStop Line) відповідає за вмикання та вимикання двигуна. Тобто коли двигун вимкнено, то вона його ввімкне. Коли двигун ввімкнений та обертається, то вона його вимкне. Кнопку K1 підключено до 12-го виводу мікроконтролера – PB2, який повинен бути запрограмований як вхід.

Кнопка K2 (ReverseOrder Line) відповідає за зміну напрямку обертання двигуна. Одне натискання – одна зміна напрямку. Потрібно мати на увазі, що двигун пристрій інерційний. Він не може зупинитися миттєво та одразу почати обертатися в інший бік. Після зміни напрямку програмно, у нього витрачається певний період часу, щоб фізично зупинитись, та почати обертатись у іншу сторону. Кнопку K2 підключено до 13-го виводу ATmega128, а саме до PB3, який попередньо запрограмовано як вхід.

Кнопка K3 (SpeedUp Line) відповідає за збільшення швидкості обертання двигуна шляхом поступового зменшення шпаруватості та збільшення постійної еквівалентної напруги імпульсного ШІМ-сигнала до максимуму, який дорівнює амплітуді імпульса – значенню напруги живлення на виводі VS мікросхеми L293D

(рисунок 8.4). Оскільки двигун інерційний, то швидкість обертання збільшується не стрибкоподібно, а поступово, і потребує певного проміжку часу, щоб вийти на новий програмно встановлений рівень.

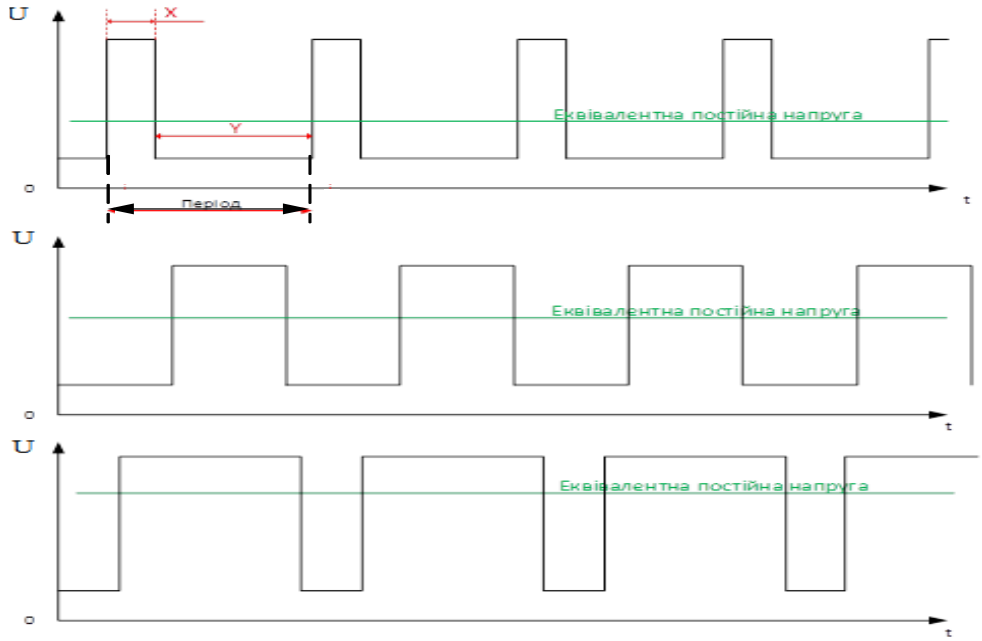


Рисунок 8.4 – Формування імпульсного ШІМ-сигналу

Кнопку K3 підключено до 14-го виводу мікроконтролера – PB4, який повинен бути запрограмований як вхід.

Кнопка K4 (SpeedDown Line) відповідає за зменшення швидкості обертання двигуна, для чого вона поступово збільшує шпаруватість та зменшує значення постійної еквівалентної напруги імпульсного ШІМ-сигнала (рисунок 8.4).

Через інерційність двигуна швидкість обертання зменшується не стрибками, а поступово і вимагає для свого зменшення певного проміжку часу. Кнопку K4 підключено до 16-го виводу мікроконтролера – PB6, який повинен бути запрограмований як вхід.

На рисунку 8.5 наведено підключення до мікроконтролера драйвера ШІМ – мікросхеми L293D (U2).

До входів IN1 та IN2 цієї мікросхеми підключені виводи 10 та 11 мікроконтролера: PB0 та PB1, які запрограмовані як виходи.

В залежності від комбінації сигналів керування на цих входах – нуль або одиниця, двигун буде або стояти, або обертатись у відповідному напрямку.

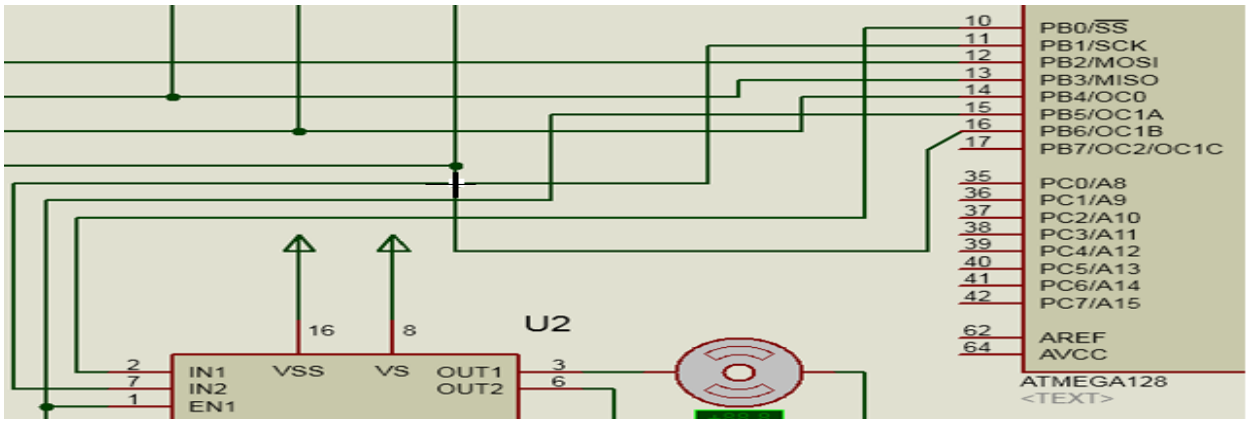


Рисунок 8.5 – Підключення мікросхеми L293D та мікроконтролера ATmega128

15-й вивід мікроконтролера – PB5 запрограмовано як вихід і підключено до входу EN1 мікросхеми L293D. З цього виходу мікроконтролера знімається ШІМ-сигнал. Вхід EN1 відповідає за роботу першої мостової схеми у складі мікросхеми (рисунок 8.14). Коли на ньому високий рівень, на перший міст подається живлення, яке підведено до виводу VS мікросхеми (рисунок 8.15). Коли на вході EN1 низький рівень, на перший міст напруга живлення не подається. Завдяки ШІМ-сигналу на цьому вході можна змінювати швидкість обертання двигуна.

Два входи GND мікросхеми L293D заземлено. На вхід VSS (рисунок 8.15) подається напруга живлення самої мікросхеми, а на вхід VS – напруга живлення двигуна.

На рисунку 8.6 зображено підключення до мікросхеми L293D двигуна.

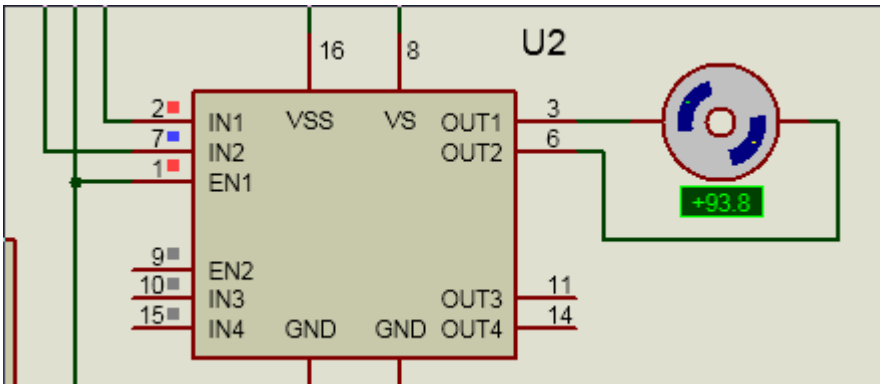


Рисунок 8.6 – Підключення до мікросхеми L293D двигуна

Двигун підключено до 3 та 6 виводів мікросхеми L293D, а саме OUT1 та OUT2. На ці виходи, в залежності від того, чи ввімкнене живлення першої мостової схеми, що керується входом EN1, та в залежності від стану ключів у першій мостовій схемі, які керуються входами IN1 та IN2, або подається напруга живлення, яку ми подали на вхід VS, або напруга не подається.

У випадку, коли на один з виводів – OUT1 чи OUT2 подається напруга живлення, а на інший не подається, виникне різниця потенціалів. Це викличе протікання струму обмоткою збудження двигуна, внаслідок чого він почне обертатися.

Як видно з рисунку 8.6 в моделі сигнал на вході IN1 позначено червоним кольором, тобто від мікроконтролера подано високий рівень (логічну одиницю). Сигнал на вході IN2 позначено синім кольором, тобто подається низький рівень – логічний нуль. При такій комбінації на входах IN1 та IN2 двигун почав обертатися за годинниковою стрілкою. При протилежній комбінаії двигун буде обертатися проти годинникової стрілки. При однакових сигналах на входах IN1 та IN2 двигун обертатися не буде.

На рисунках 8.7…8.10 наведено деякі фрагменти, які демонструють роботу моделі.

На рисунку 8.11 наведено стан усієї системи після запуску сценарію.

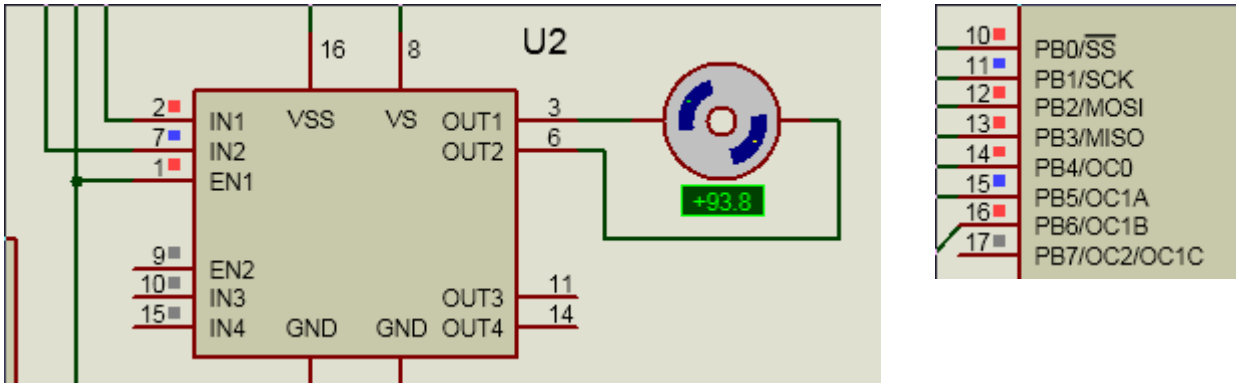


Рисунок 8.7 – Стан моделі після натискання кнопки «старт/стоп»

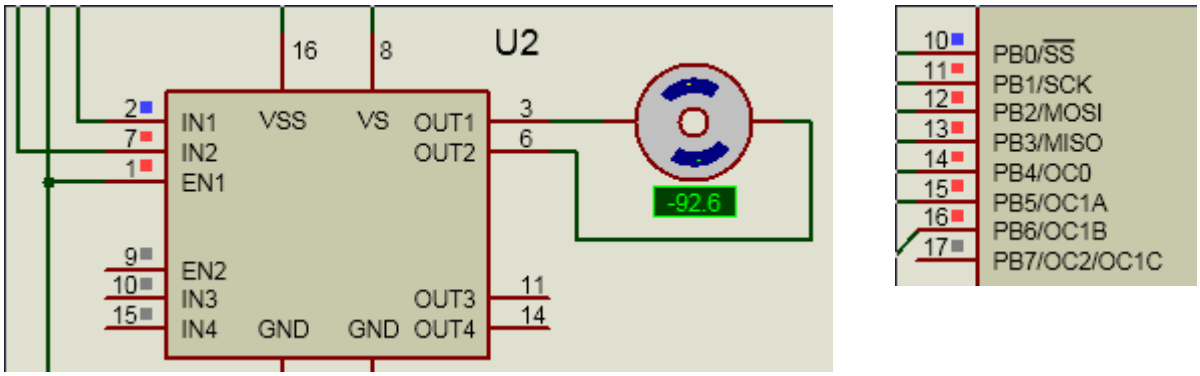


Рисунок 8.8 – Стан моделі після активації режиму реверсу

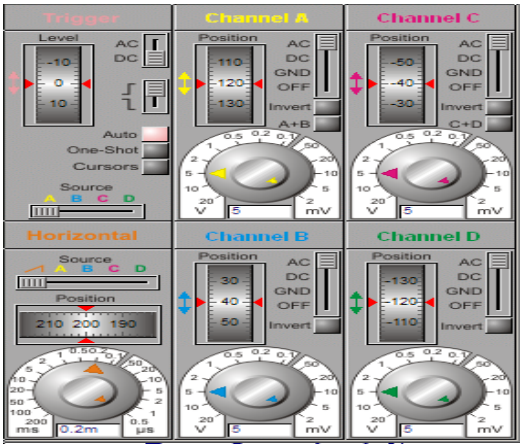


Рисунок 8.9 – Налаштування осцилографа

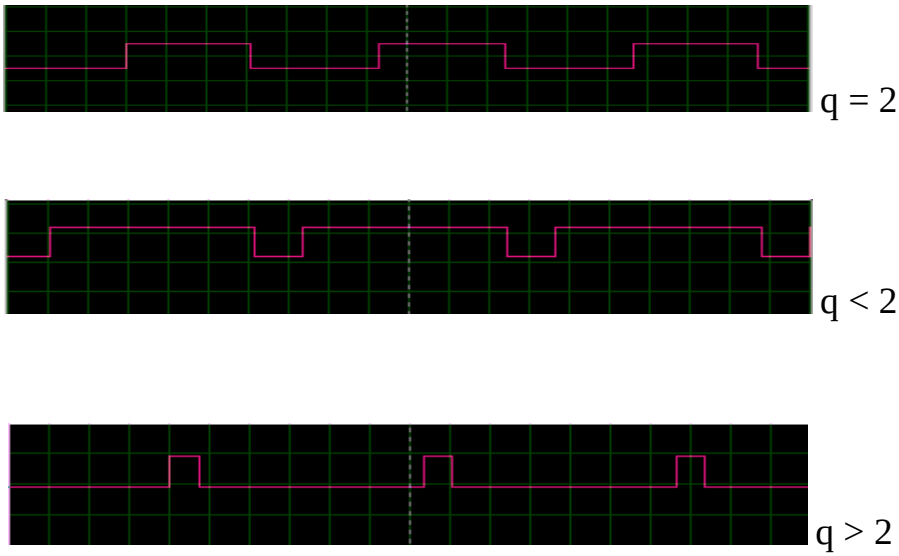


Рисунок 8.10 – ШІМ-сигнал, що подається на двигун (шпаруватість q = 2, q < 2, q > 2)

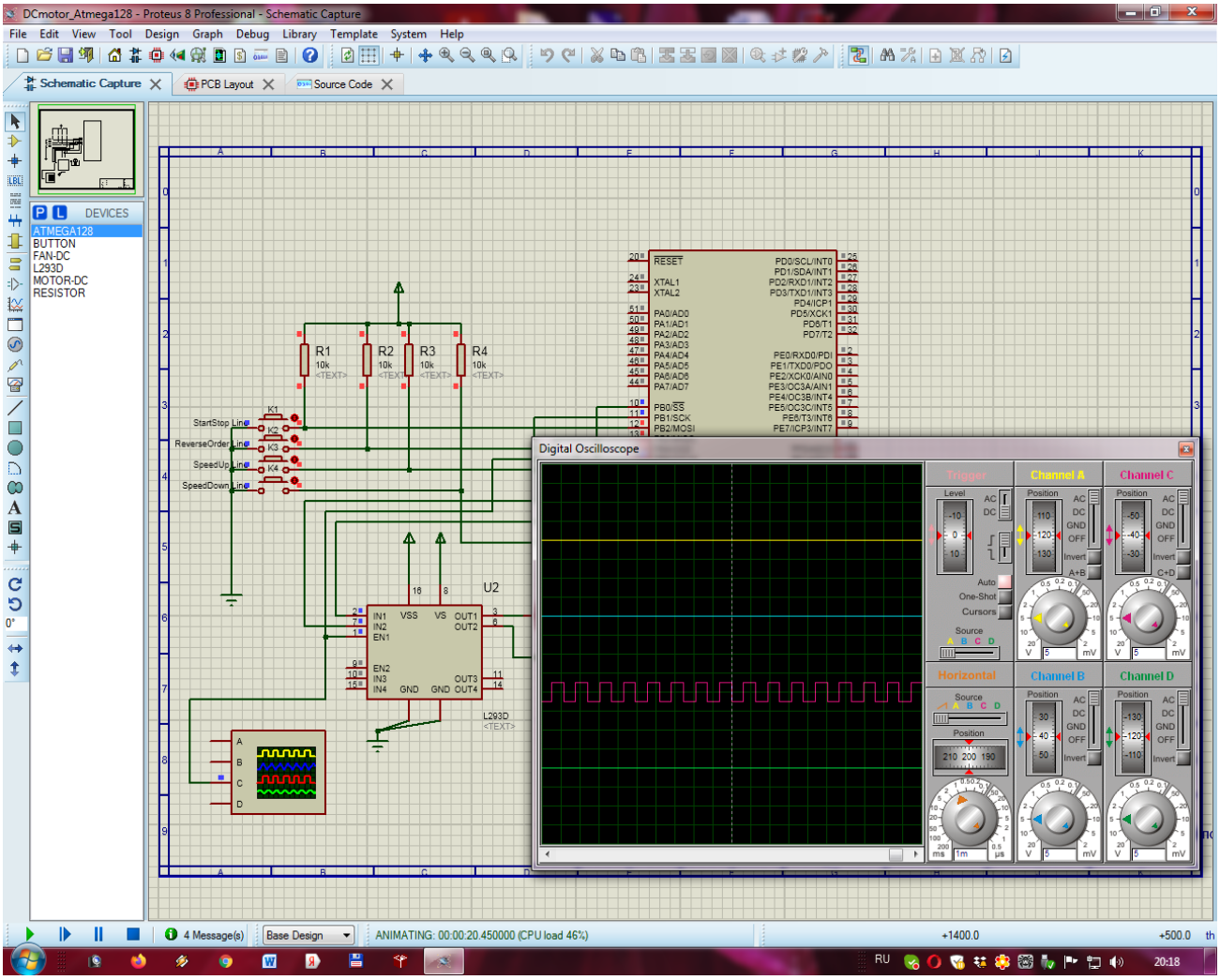


Рисунок 8.11 – Стан усієї системи після запуску сценарію (шпаруватість ШІМ-сигналу дорівнює 2)

Мікроконтролер

Для контролю та виконання всіх функцій, які покладено на модель, було обрано МК сім`ї AVR – ATmega 128. Він є малопотужним 8-розрядним КМОН-мікроконтролером, який має розширену RISC-архітектуру [1; 2; 10]. Його основні характеристики та опис виводів розглянуто вище у 7.1.1.

Модуль таймера

Для керування двигуном постійного струму в моделі використовується модуль таймера мікроконтролера, який працює в режимі широтно-імпульсної модуляції. Регулювання швидкості обертання двигуна забезпечується зміною шпаруватості керуючих імпульсів, що у свою чергу викликає зміну постійної

складової імпульсного сигналу (рисунок 8.10). Чим менше шпаруватість, а відповідно більше величина відношення «тривалість імпульсу/період», тим більша напруга надходить на двигун, який буде обертатись із більшою швидкістю.

Двигун

В моделі використовується електродвигун постійного струму. Регулювання швидкості обертання двигуна зазвичай забезпечується за допомогою формування керуючих імпульсів у вигляді сигналів з широтно-імпульсною модуляцією. Подібний підхід дозволяє регулювати середню величину потужності, що надходить на двигун. В даному випадку, чим більше величина відношення «тривалість імпульсу/період» (менше шпаруватість), тим більша потужність надходить на двигун.

Частота сигналу з широтно-імпульсною модуляцією повинна перевищувати 20 кГц, що дозволяє виключити звукові ефекти, пов’язані з формуванням звукових сигналів самим двигуном при зміні магнітного поля, яке в ньому утворюється [1].

Для формування ШІМ-сигналів в моделі використовується модуль таймера мікроконтролера.

Драйвер ШІМ

Двигун постійного струму підключається до мікроконтролера за допомогою відповідної мостової схеми – драйвера, яку зображено на рисунку 8.12.

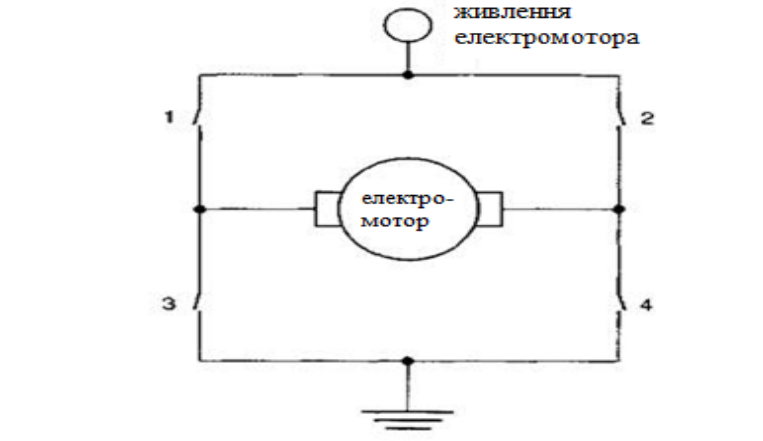


Рисунок 8.12 – Мостова схема підключення двигуна постійного струму до мікроконтролера

Подібну схему мають драйвери двигунів, які використовуються для перетворення керуючих сигналів малої потужності від мікроконтролера у сигнали, потужність яких достатня для керування двигунами.

Окрім підсилення потужності керуючих сигналів драйвери керують зміною напрямку обертання двигнів. Існує декілька схем драйверів, які відрізняються як потужністю, так і елементною базою, на якій вони виконані. В моделі застосовано драйвер, який виконано у вигляді готової до роботи мікросхеми – L293D. Її зовнішній вигляд показано на рисунку 8.13.

Рисунок 8.13 – Зовнішній вигляд мікросхеми L293D

Мікросхема L293D включає два драйвери та може керувати двома електродвигунами відповідної потужності. Також мікросхема забезпечує розділене живлення для самої мікросхеми та для керованих нею двигунів, що дозволяє підключати електродвигуни з більшою напругою живлення, чим у мікросхеми. Розділення живлення мікросхеми і електродвигунів необхідне також для зменшення завад, які викликано стрибками напруги, що пов’язані з роботою двигунів.

Обидва драйвери, що входять у склад мікросхеми мають ідентичні принципи роботи, тому нижче розглянуто принцип роботи лише одного з них, спрощений вид якого наведено на рисунку 8.14.

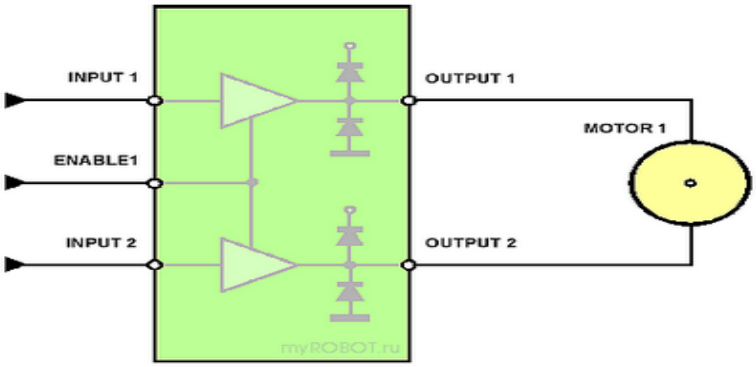


Рисунок 8.14 – Спрощений вид драйвера мікросхеми L293D

До виходів OUTPUT1 та OUTPUT2 підключається двигун постійного струму MOTOR1. Вхід ENABLE1, який відповідає за ввімкнення драйвера, під’єднують до додатного полюсу джерела живлення: +5V. Якщо при цьому відповідні керуючі сигнали на входи INPUT1 та INPUT2 не подаються, то двигун обертатися не буде.

Якщо вхід INPUT1 з’єднати з додатним полюсом джерела живлення, а вхід INPUT2 – з від’ємним, то двигун почне обертатися за годинниковою стрілкою.

Якщо з’єднати вхід INPUT1 з від`ємним полюсом джерела струму, а вхід INPUT2 – з додатним, то двигун почне обертатися в іншу сторону.

Якщо на керуючі входи INPUT1 та INPUT2 подати сигнали одного рівня, тобто під’єднати обидва входи до додатного полюсу джерела живлення або до від`ємного, то двигун обертатися не буде.

Якщо прибрати керуючий сигнал зі входу ENABLE1, то при будь-яких варіантах сигналів на входах INPUT1 та INPUT2 двигун також обертатися не буде. На вхід ENABLE1 подається імпульсний ШІМ-сигнал, який формується модулем таймера мікроконтролера. Змінюючи шпаруватість цього сигналу ми змінюємо постійну складову імпульсного сигналу, що в свою чергу змінює швидкість обертання двигуна.

На рисунку 8.15 показано нумерацію та позначення виводів мікросхеми L293D.

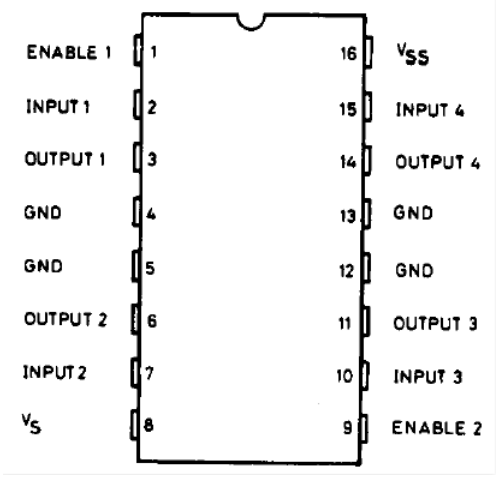


Рисунок 8.15 – Нумерація та позначення виводів мікросхеми L293D

Нижче описано призначення виводів мікросхеми:

- входи ENABLE1 та ENABLE2 відповідають за вмикання відповідного драйвера, що входить у склад мікросхеми. На ці входи подається ШІМ-сигнал;
- входи INPUT1 та INPUT2 керують напрямом обертання двигуна, який підключено до виходів OUTPUT1 та OUTPUT2;
- входи INPUT3 та INPUT4 керують напрямом обертання другого двигуна, який підключено до виходів OUTPUT3 та OUTPUT4;
- контакт Vs відповідає за живлення електродвигунів. Його з'єднують з додатним полюсом джерела живлення двигунів або просто з додатним полюсом джерела живлення мікросхеми, якщо схема і двигуни живляться від одного джерела;
- контакт Vss з'єднують з додатним полюсом джерела живлення. Цей контакт забезпечує живлення самої мікросхеми;
- чотири контакти GND з'єднують з "землею" (спільним дротом – від`ємним полюсом джерела живлення). Крім того, за допомогою цих контактів зазвичай забезпечують тепловідвід від мікросхеми, тому їх краще всього паяти на достатньо широку контактну поверхню.

Дана мікросхема має наступні технічні характеристики:

- напруга живлення двигунів Vs: 4,5...36 В;
- напруга живлення мікросхеми Vss: +5 В;
- допустимий струм навантаження на кожен канал: 600 мА;
- максимальний струм на виході на кожен канал: 1,2 А;
- логічний нуль вхідної напруги: до 1,5 В;
- логічна одиниця вхідної напруги: 2,3...7 В;
- швидкість перемикань: до 5 кГц;
- мікросхема має захист від перегріву.

Інший спосіб керування двигунами постійного струму засновано на використанні мостових схем типу L298N [1]. Це двоканальний пристрій, який працює від ТТЛШ-рівнів з напругою до 46 В та струмом до 2 А на кожен канал.

Розташування видів та спрощену внутрішню структуру мікросхеми зображено на рисунку 8.16.

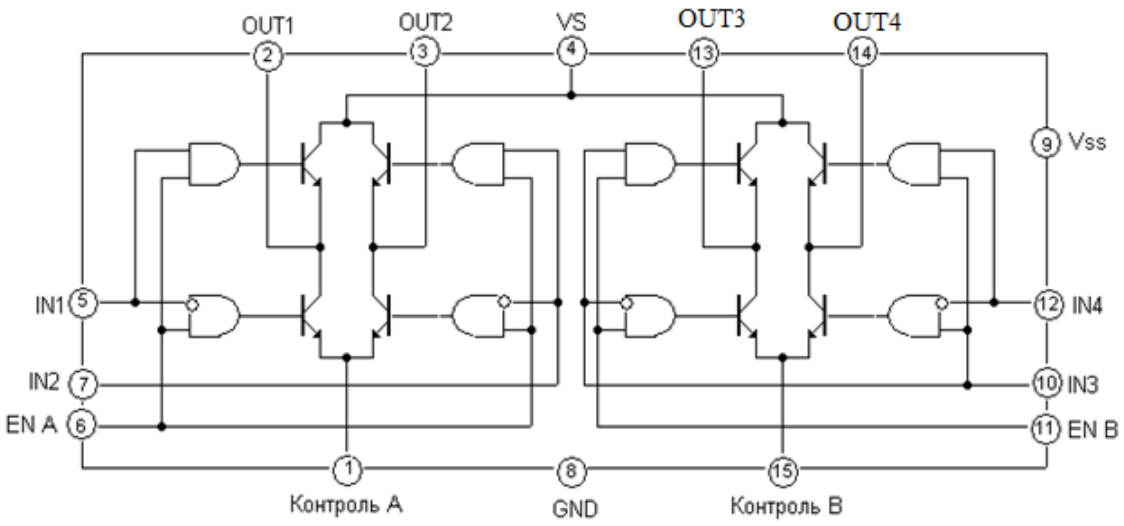


Рисунок 8.16 – Розташування видів і спрощена внутрішня структура мікросхеми L298N

Через вивід Vs (контакт 4) надходить напруга живлення для двигуна. На вивід Vss (контакт 9) подається напруга живлення мікросхеми схеми: +5 В.

На входи ENA і ENB (контакти 6 і 11) подаються керуючі ШІМ-сигнали. Коли на вході ENA/ENB низький рівень, ці входи заблоковано і двигун не обертається. Якщо на цей вхід подати високий рівень, входи відкриваються.

Входи IN1 і IN2 (контакти 5 і 7) керують напрямком обертання двигуна для першого каналу, а IN3 і IN4 – другого:

- IN1 – l, IN2 – 0 – двигун обертається за годинниковою стрілкою;
- IN1 – 0, IN2 – 1 – двигун обертається проти годинникової стрілки;
- INI = IN2 = 0/1 двигун не обертається.

Емітери транзисторів з’єднано для підключення зовнішніх контролюючих датчиків.

Для підвищення до 5 А значення вихідного струму, що надходить безпосередньо на обмотку двигуна постійного струму, на вихід схеми підключають додаткові пари транзисторів, шунтуючи їх захисними діодами (рисунок 8.17).

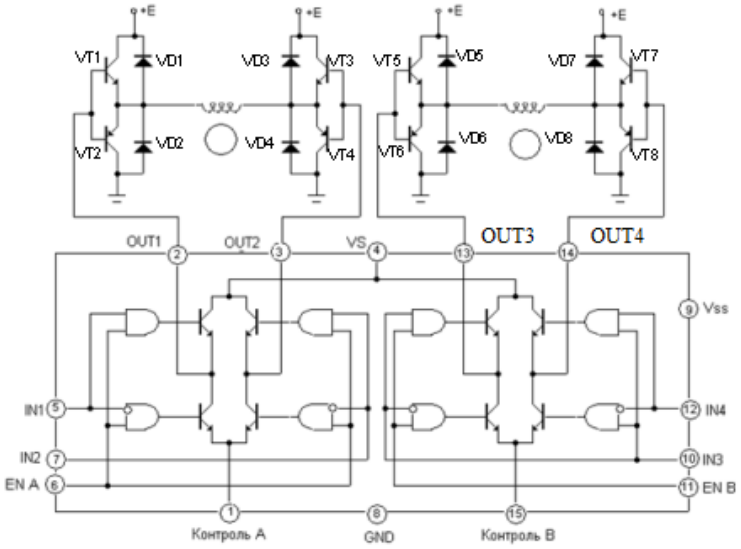


Рисунок 8.17 – Схема потужного ШІМ-драйвера

Захисні діоди

Як видно з рисунків 8.14, 8.17 схема драйверів містить 8 захисних діодів VD1…VD8. Ці діоди на рисунку 8.17 призначено для захисту транзисторних ключів (VT1...VT8) від додатних (VT1, VT3, VT5, VT7) і від'ємних (VT2, VT4, VT6, VT8) паразитних імпульсів досить високої амплітуди, які з'являються на обмотках двигуна при комутації обмоток. Додатні імпульси виникають при запиранні (вимиканні) ключів, а від'ємні – при включенні. Механізм виникнення цих імпульсів описано нижче.

Коли сила струму в котушці змінюється, змінюється і магнітний потік в середині котушки, який збуджує у ній електрорушійну силу індукції (ЕІ). Ця ЕІ протидіє зміні магнітного потоку (правило Ленца). Якщо, наприклад, струм у котушці різко збільшується (включення ключа), то наростаючий магнітний потік індукує в котушці ЕІ, під дією якої виникає струм, протилежний первісному струму і прагне загальмувати його. При цьому на ключі виникає від'ємний імпульс (рисунок 8.18, а) достатньо великої амплітуди, який може вивести ключ з ладу.

Якщо ж струм в котушці різко зменшується (при виключенні ключа), то убиваючий магнітний потік збуджує ЕІ, яка створює струм, спрямований аналогічно вихідного струму, що підтримує в котушці початковий струм.

При цьому на ключі виникає додатний імпульс (рисунок 8.18, б) достатньо великої амплітуди, який може вивести ключ з ладу.

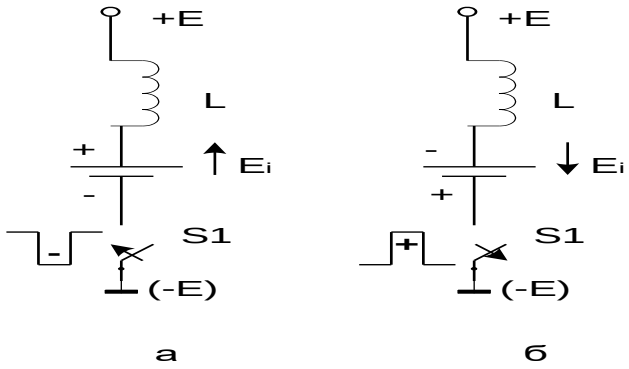


Рисунок 8.18 – Виникнення паразитних імпульсів при комутації ключів:
а – від'ємний імпульс при включенні ключа;
б – додатний імпульс при відключенні ключа

Розрахунок ШІМ-модуля та його програмування мовою Асемблер

Вхідні дані

Треба виконати розрахунок та програмування ШІМ-модуля та основі таких вхідних даних:

- тип МК-ра: АТmega 128;
- номер таймера: Т/С1;
- на виході РВ5 треба сформувати інвертований ШІМ-сигнал;
- частота ШІМ-сигналу: f_{PB5} = 25 кГц;
- частота тактового сигналу підсистеми введення/виведення: $f_{CLKI/0}$= 16 МГц;
- режим роботи таймера: Phase and Frequency Correct PWM (PFCPWM);
- шпаруватість ШІМ-сигналу: Q_{PB5} = 2.

Завдання

Розрахувати:

- коефіцієнт ділення переддільника: Кділ = N;
- модуль лічби: TOP;
- період ШІМ-сигналу: T_{PB5};
- тривалість імпульсу ШІМ-сигналу: $t_{імпPB5}$.

Написати мовою Асемблер фрагмент програми, який забезпечує формування ШІМ-сигналу згідно вхідних даних.

Рішення завдання

На рисунку 8.19 наведено формування таймером ШІМ-сигналу в режимі Phase and Frequency Correct PWM [1, 2].

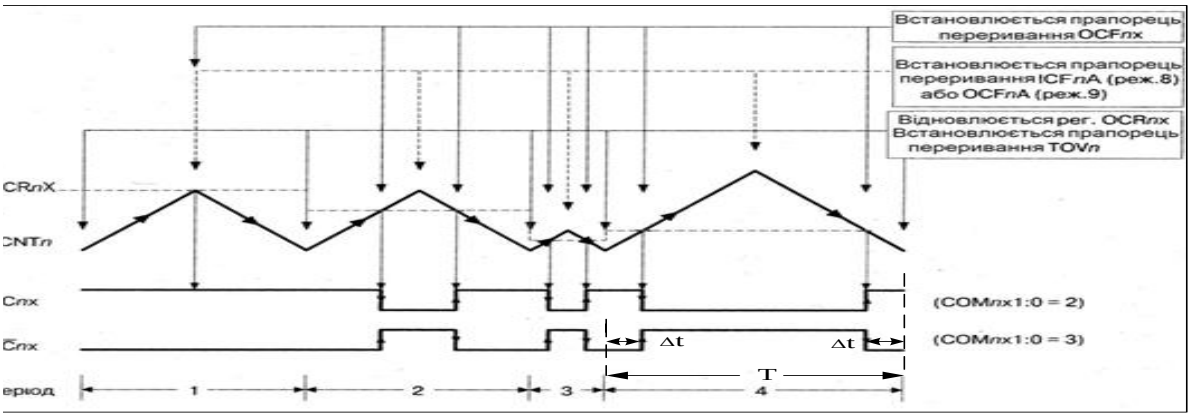


Рисунок 8.19 – Формування таймером ШІМ-сигналу в режимі Phase and Frequency Correct PWM

Величина частоти ШІМ-сигналу f_{PL3} визначається виразом [1; 2]:

$$f_{PB5} = \frac{f_{CLK\ I/O}}{2N(TOP+1)}, \qquad (8.1)$$

де N – коефіцієнт ділення попереднього дільника таймера; TOP – модуль лічби; $f_{clkI/0}$ – частота генератора тактових імпульсів підсистеми введення/виведення мікроконтролера.

Згідно таблиці 8.1 обираємо номер режиму роботи таймера: 8.

Значення модуля лічби TOP визначається вмістом регістра ICR1. Згідно формули 8.1 при N = 8 та значенні f_{PB5}, яке задано в завданні та дорівнює 25кГц, розраховуємо значення ICR1 = TOP:

$$ICR1 = \frac{f_{CLKI/O}}{2*N*f_{PB5}} = \frac{16*10^6}{2*8*25*10^3} = 40\ .$$

Обираємо значення ICR1 = 40 = 00101000B = $0028.

Період ШІМ-сигналу:

$$T_{PB5} = \frac{1}{f_{PB5}} = \frac{1}{25000} = 40 \text{ мкс.}$$

Таблиця 8.1 – Режими роботи 16-розрядних таймерів/лічильників

Номер режиму	WGMn3	WGMn2	WGMn1	WGMn0	Режим роботи таймера/лічильника Tn	Модуль лічби (TOP)	Оновлення регістрів TOVn	Момент встановлення прапорця TOVn
0	0	0	0	0	Normal	$FFFF	Негайно	$FFFF
1	0	0	0	1	Phase correct PWM, 8-розрядний	$00FF	При TOP	$0000
2	0	0	1	0	Phase correct PWM, 9-розрядний	$01FF	При TOP	$0000
3	0	0	1	1	Phase correct PWM, 10-розрядний	$03FF	При TOP	$0000
4	0	1	0	0	CTC (скидання за збігом)	OCRnA	Негайно	$FFFF
5	0	1	0	1	Fast PWM, 8-розрядний	$00FF	При TOP	При TOP
6	0	1	1	0	Fast PWM, 9-розрядний	$01FF	При TOP	При TOP
7	0	1	1	1	Fast PWM, 10-розрядний	$03FF	При TOP	При TOP
8	1	0	0	0	Phase and Frequency Correct PWM	ICRn	$0000	$0000
9	1	0	0	1	Phase and Frequency Correct PWM	OCRnA	$0000	$0000
10	1	0	1	0	Phase correct PWM	ICRn	При TOP	$0000
11	1	0	1	1	Phase correct PWM	OCRnA	При TOP	$0000
12	1	1	0	0	CTC (скидання за збігом)	ICRn	Негайно	$FFFF
13	1	1	0	1	Зарезервовано	–	–	–
14	1	1	1	0	Fast PWM	ICRn	При TOP	При TOP
15	1	1	1	1	Fast PWM	OCRnA	При TOP	При TOP

Примітка. n = 1, 3, 4 або 5.

При шпаруватості ШІМ-сигналу $Q_{PB5} = 2$ потрібна тривалість імпульсу:

$$t_{імпPB5} = \frac{T_{PB5}}{Q} = \frac{40\ мкс}{2} = 20\ мкс.$$

Згідно з рисунком 8.19

$$t_{імпPB5} = T_{PB5} - 2\,\Delta\,t. \qquad (8.2)$$

$$\Delta t = \frac{T_{PB5} - t_{імпPB5}}{2} = \frac{40 - 20}{2} = 10 \text{ мкс.}$$

Згідно роботи таймера у режимі Phase and Frequency Correct PWM [1]

$$\Delta t = T_{CLKI/O} * N * OCR1A, \tag{8.3}$$

де OCR1A – регістр порівняння каналу А таймера 1.

Із виразу (8.3)

$$OCR1A = \frac{\Delta t}{T_{CLKI/O} * N} = \frac{10 * 10^6}{\frac{1}{16} * 10^{-6} * 8} = 20 = 00010100B = \$0014.$$

Згідно рисунку 8.19

$$t_{імпPB5} = T_{PB5} - 2\,\Delta t.$$

Тоді шпаруватість

$$Q = \frac{T_{PB5}}{T_{PB5} - 2\,\Delta t} = \frac{T_{PB5}}{T_{PB5} - 2\,T_{CLKI/O} * N * OCR1A} = \frac{1}{1 - \frac{2\,T_{CLKI/O} * N * OCR1A}{T_{PB5}}} =$$

$$= \frac{1}{1 - 2 * \frac{1}{f_{CLKI/O}} * N * OCR1A * f_{PB5}}. \tag{8.4}$$

За останньою формулою при OCR1A = 20 розрахуємо значення шпаруватості Q, яке повинно дорівнювати 2.

$$Q = \frac{1}{1 - 2 * \frac{1}{f_{CLKI/O}} * N * OCR1A * f_{PB5}} = \frac{1}{1 - 2 * \frac{1}{16 * 10^6} * 8 * 20 * 25 * 10^3} = 2.$$

Обчислимо значення Q при збільшенні OCR1A на 5 та при зменшенні на 5:

$$Q_{(OCR1A=25)} = \frac{1}{1 - 2 * \frac{1}{16 * 10^6} * 8 * 25 * 25 * 10^3} = 2{,}667.\ Q_{(OCR1A=15)} = \frac{1}{1 - 2 * \frac{1}{16 * 10^6} * 8 * 15 * 25 * 10^3} = 1{,}6.$$

Розробка окремих фрагментів програми мовою Асемблер

Зупинка таймера

Для зупинки таймера треба записати в регістр керування TCCR1B (рисунок 8.20), адреса якого згідно із таблицею 8.2 дорівнює $004E, наведене нижче керуюче слово КС1 = 00000000B = $00:

7 р.	6р.	5р.		4р.	3р.	2р.	1р.	0р.
ICNC1	ICES1	–		WGM13	WGM12	CS12	CS11	CS10
0	0	0		0	0	0	0	0 ($00 = КС1)

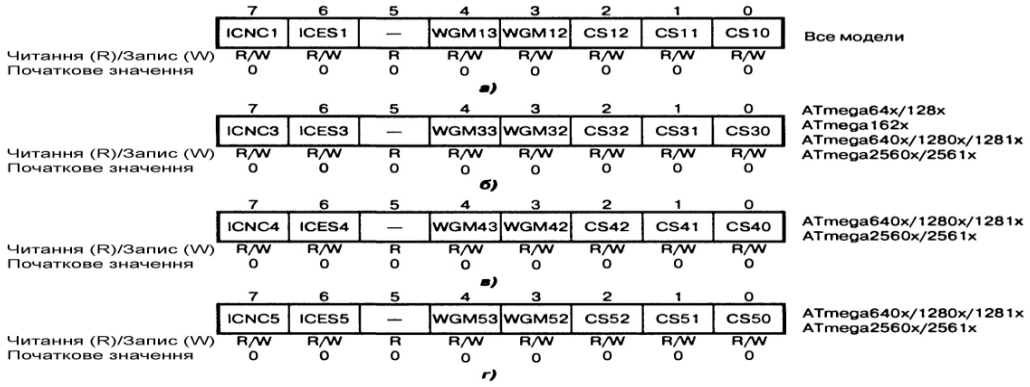


Рисунок 8.20 – Формат регістрів TCCR1B (а), TCCR3B (б), TCCR4B (в), TCCR5B (г)

Таблиця 8.2 – Адреси регістрів керування 16 – розрядних таймерів/лічильників

Регістр	Адреса	ATmega8515x	ATmega8535x	ATmega8x	ATmega16x/32x	ATmega64x/128x	ATmega48x/88x/168x	ATmega162x	ATmega164x/324x/644x	ATmega165x	ATmega325x/3250x, ATmega645x/6450x	ATmega640x, ATmega1280x/1281x, ATmega2560x/2561x
TCCR1A	$2F($4F)	•	•	•	•	•		•				
	($80)						•		•	•	•	•
TCCR1B	$2E ($4E)	•	•	•	•	•		•				
	($81)						•		•	•	•	•
TCCR1C	($7A)					•						
	($82)						•		•	•	•	•
TCNT1	$2D:$2C ($4D:$4C)	•	•	•	•	•		•				
	($85:$84)						•		•	•	•	•
OCR1A	$2B:$2A ($4B:$4A)	•	•	•	•	•		•				
	($89:$88)						•		•	•	•	•
OCR1B	$29:$28 ($49:$48)	•	•	•	•	•		•				
	($8B:$8A)						•		•	•	•	•

Закінчення таблиці 8.2

OCR1C	($79:$78)					•						
	($8D:$8C)											•
ICR1	$27:$26 ($47:$46)		•	•	•	•						
	$25:$24 ($45:$44)	•						•				
	($87:$86)						•		•	•	•	•
TCCR3A	($8B)					•		•				
	($90)											•
TCCR3B	($8A)					•		•				
	($91)											•
TCCR3C	($8C)					•						
	($92)											•
TCNT3	($89:$88)					•		•				
	($95:$94)											•
OCR3A	($87:$86)					•		•				
	($99:$98)											•
OCR3B	($85:$84)					•		•				
	($9B:$9A)											•
OCR3C	($83:$82)					•						
	($9D:$9C)											•
ICR3	($81:$80)					•		•				
	($97:$96)											•
TCCR4A	($A0)											•
TCCR4B	($A1)											•
TCCR4C	($A2)											•
TCNT4	($A5:$A4)											•
OCR4A	($A9:$A8)											•
OCR4B	($AB:$AA)											•
OCR4C	($AD:$AC)											•
ICR4	($A7:$A6)											•
TCCR5A	($120)											•
TCCR5B	($121)											•
TCCR5C	($122)											•
TCNT5	($125:$124)											•
OCR5A	($129:$128)											•
OCR5B	($12B:$12A)											•
OCR5C	($12D:$12C)											•
ICR5	($127:$126)											•

Тоді програма зупинки таймера має вигляд:

```
LDI R18, $00; R18← КС1 = $00;
LDI R27,  $00;  R27 ← $00
LDI R26,  $4E; R26 ← $4E } X(R27,R26) ← $004E;
ST X, R18; TCCR1B ← R18 = $00, зупинка таймера.
```

Завантаження регістра TCCR1A

Згідно рисунку 8.21 формат регістра керуючого слова (КС2), яке завантажується у регістр TCCR1A МК-ра AT Mega 128 має вид:

7 р.	6р.	5р.	4р.	3р.	2р.	1р.	0р.	КС2
COM1A1	COM1A0	COB1B1	COM1BO	COM1C1	COM1C0	WGM11	WGM10	
1	1	0	0	0	0	0	0 B=$C0	

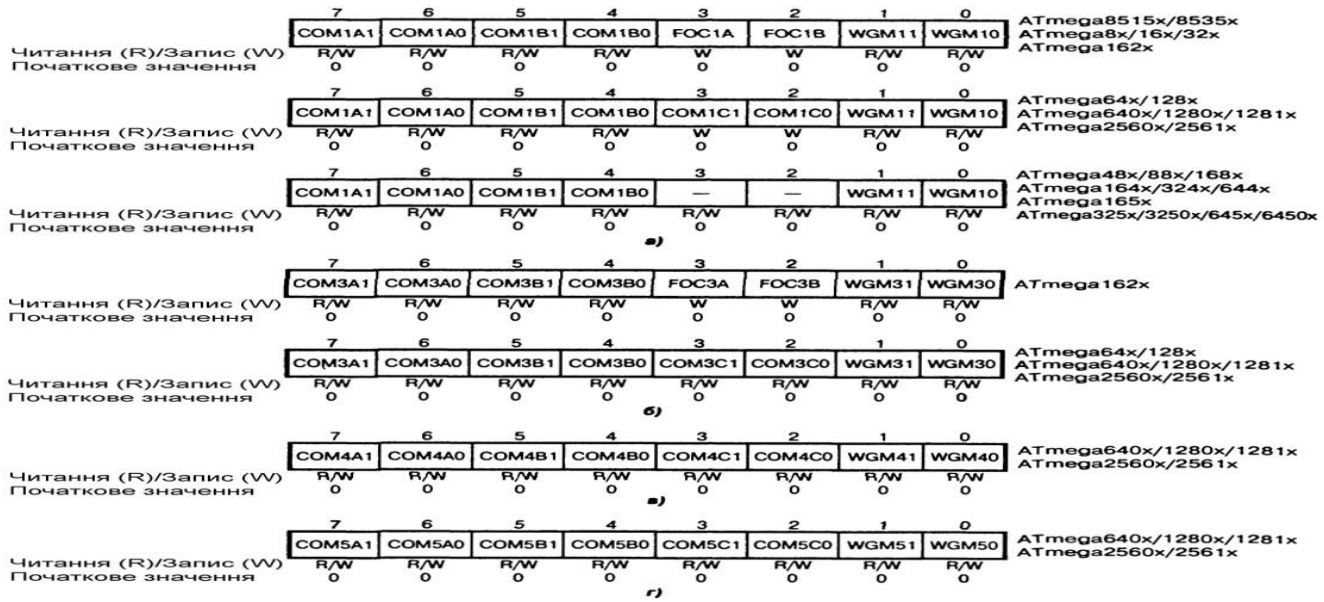


Рисунок 8.21 – Формат регістрів TCCR1A (а), TCCR3A (б), TCCR4A (в), TCCR5A (г)

Для нашої задачі для формування інвертованого ШІМ-сигналу згідно таблиці 8.3 необхідно встановити біти COM1A1 = COM1A0 = 1.

Для програмування режиму роботи 8 згідно таблиці 8.1 необхідно запрограмувати біти: WGM11 = 0; WGM10 = 0.

Інші біти регістра TCCR1A у нашому прикладі не використовуються. Тому запишемо в них нулі.

Тоді керуюче слово КС2 = 11000000B = $C0.

Згідно таблиці 8.2 адреса регістра TCCR1A = $004F.

Таблиця 8.3 – Поведінка виводу OCnA/OCnB/OCnC в режимі Phase and Frequency Correct PWM

COMnx1	COMnx0	Опис
0	0	Таймер/лічильник Tn відключено від виводу OCnA/OCnB/OCnC
0	1	WGMn3 = «0»: таймер/лічильник Tn відключено від виводу OCnA/OCnB/OCnC; WGMn3 = «1»: стан виводу OCnA змінюється на протилежний
1	0	Скидається в "0" при прямій лічбі і встановлюється в "1" при зворотній лічбі (неінвертований ШІМ-сигнал)
1	1	Встановлюється в "1" при прямій лічбі і скидається в "0" при зворотній лічбі (інвертований ШІМ-сигнал)

Примітка. n = 1, 3, 4 або 5; x= A, B або C.

Тоді програма завантаження регістра TCCR1A має вигляд:

```
LDI R17, $C0; R17← KC2 = $C0;
LDI R29, $00; R27 ← $00
LDI R28, $4F; $26 ← $4F } Y(R29,R28) ← $004F;

ST Y, R17; TCCR5A←R17 = $C0.
```

Програмування лінії PB5 на виведення

Для програмування лінії PB5 на виведення необхідно встановити 5-й розряд регістра DDRB в одиницю. Згідно таблиці 8.4 адреса регістра: $0037.

Тоді програма програмування лінії PB5 на виведення має вигляд:

```
LDI R31, $00;                } Z(R31,R30) ← $0037
LDI R30, $37; R30 ← $37 ;
LD R19, Z; R19← DDRB

ORI R19, $20; R19←00100000B + R19; R19.5←1, інші біти без змін;
ST Z, R19; DDRB←R19; DDRB.5 ←1.
```

Завантаження регістра ICR1

Згідно таблиці 8.2 16-розрядний регістр ICR1 має адресу: $0047 (СБ) : $0046 (МБ). В ICR1 треба завантажити: TOP = 40 = 00101000B = = $0028 (див. вище).

Тоді програма завантаження регістра ICR1 має вигляд:

```
LDI R23, $00; R23← $00;
```

```
LDI R27, $00; R27 ← $00;
LDI R26,$47; R26 ← $47 } X(R27,R26) ← $0047;
```

ST X, R23; СБ ICR1←R23 = $00 .

LDI R24, $28; R24← $28

```
LDI R29, $00; R29 ← $00
LDI R28, $46; R28 ← $46 } Y(R29,R28) ← $0046;
```

ST Y, R24; МБ ICR1←R24 = $28.

Таблиця 8.4 – Адреси регістрів портів введення/виведення

Порт	Регістр	ATmega8515x	ATmega8535x	ATmega8x	ATmega16x	ATmega162x	ATmega64x, ATmega128x	ATmega48x/88x/168x	ATmega164x/324x/644x	ATmega165x, ATmega325x/645x	ATmega3250x/6450x	ATmega1281x/2561x	ATmega640x, ATmega1280x/2560x
A	PORTA	$1B ($3B)		–	$1B ($3B)			–	$02 ($22)				
	DDRA	$1A ($3A)		–	$1A ($3A)			–	$01 ($21)				
	PINA	$19 ($39)		–	$19 ($39)			–	$00 ($20)				
B	PORTB	$18 ($38)						$05 ($25)					
	DDRB	$17 ($37)						$04 ($24)					
	PINB	$16 ($36)						$03 ($23)					
C	PORTC	$15 ($35)						$08 ($28)					
	DDRC	$14 ($34)						$07 ($27)					
	PINC	$13 ($33)						$06 ($26)					
D	PORTD	$12 ($32)						$0B ($2B)					
	DDRD	$11 ($31)						$0A ($2A)					
	PIND	$10 ($30)						$09 ($29)					
E	PORTE	$07 ($27)		–	–	$07 ($27)	$03 ($23)	–	–	$0E ($2E)			
	DDRE	$06 ($26)		–	–	$06 ($26)	$02 ($22)	–	–	$0D ($2D)			
	PINE	$05 ($25)		–	–	$05 ($25)	$01 ($21)	–	–	$0C ($2C)			

Закінчення таблиці 8.4

F	PORTF	$03 ($23)		–	–	–	–	($62)	–	–			
	DDRF	$02 ($22)		–	–	–	–	($61)	–	–			
	PINF	$01 ($21)		–	–	–	–	$00 ($20)	–	–			
G	PORTG	–	–	–	–	–	($65)	–	–	$14 ($34)			
	DDRG	–	–	–	–	–	($64)	–	–	$13 ($33)			
	PING	–	–	–	–	–	($63)	–	–	$12 ($32)			
H	PORTH	–	–	–	–	–	–	–	–	–	($DA)	–	($102)
	DDRH	–	–	–	–	–	–	–	–	–	($D9)	–	($101)
	PINH	–	–	–	–	–	–	–	–	–	($D8)	–	($100)
J	PORTJ	–	–	–	–	–	–	–	–	–	($DD)	–	($105)
	DDRJ	–	–	–	–	–	–	–	–	–	($DC)	–	($104)
	PINJ	–	–	–	–	–	–	–	–	–	($DB)	–	($103)
K	PORTK	–	–	–	–	–	–	–	–	–	–	–	($108)
	DDRK	–	–	–	–	–	–	–	–	–	–	–	($107)
	PINK	–	–	–	–	–	–	–	–	–	–	–	($106)
L	PORTL	–	–	–	–	–	–	–	–	–	–	–	($10B)
	DDRL	–	–	–	–	–	–	–	–	–	–	–	($10A)
	PINL	–	–	–	–	–	–	–	–	–	–	–	($109)

Завантаження регістра OCR1A

Згідно таблиці 8.2, 16-розрядний регістр OCR1A має адресу: $004B (СБ): $004A (МБ). В OCR1A треба завантажити: 20 = 00010100B = $0014 (див. вище).

Тоді програма завантаження регістра OCR1A має вигляд:

```
LDI R24, $00; R24← $00;
LDI R31,$00; R31 ← $00;
LDI R30, $4B; R30 ← $4B } Z(R31,R30) ← $004B;
ST Z, R24; СБ OCR1A←R24 = $00.
LDI R25, $14; R25 ← $14;
LDI R31, $00; R31 ← $00;
LDI R30, $4A; R30 ← $4A } Z(R31,R30) ← $004A;
```

ST Z, R25; МБ OCR1A ← R25 = $14.

Програмування $К_{діл}$ = N = 8, режиму роботи номер 8 та запуск таймера Т/С1

Вище при завантаженні регістра TCCR1A було записано WGM11 = 0 та WGN10 = 0, що разом з двома бітами WGM12 = 0 та WGM13 = 1 регістра TCCRB програмують режим роботи № 8 (таблиця 8.1).

Для програмування Кділ = 8 (таблиця 8.5) та запуску таймера також використовують регістр TCCR1B (рисунок 8.20, див вище), в якій треба завантажити керуюче слово: КС3 = $12.

7 р.	6р.	5р.	4р.	3р.	2р.	1р.	0р.
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
0	0	0	1	0	0	1	0 (КС3 =12)

(режим 8) (Кділ = 8)

Таблиця 8.5 – Вибір джерела тактового сигналу таймерів/лічильників Tn

CSn2	CSn1	CSn0	Джерело тактового сигналу	
			Т3 в моделях ATmega162x	Інші
0	0	0	Таймер/лічильник зупинений	Таймер/лічильник зупинений
0	0	1	$f_{clkI/0}$	$f_{clkI/0}$
0	1	0	$f_{clkI/0}$/8	$f_{clkI/0}$/8
0	1	1	$f_{clkI/0}$/64	$f_{clkI/0}$/64
1	0	0	$f_{clkI/0}$/256	$f_{clkI/0}$/256
1	0	1	$f_{clkI/0}$/1024	$f_{clkI/0}$/1024
1	1	0	$f_{clkI/0}$/16	Вивід Tn, лічба виконується за спадаючим фронтом
1	1	1	$f_{clkI/0}$/32	Вивід Tn, лічба виконується за наростаючим фронтом

Примітка. N = 1, 3, 4 або 5.

Адреса регістра TCCR1B згідно таблиці 8.2 (див. вище) дорівнює $004E.

Тоді програма програмування Кділ = N = 8, режиму роботи номер 8 та запуск таймера Т/С1має вигляд:

LDI R17, $12; R17← $12;

LDI R29, $00; R29← $00;

LDI R28, $4E; R28← $4E;

ST Y, R17; TCCR1B ← R17 = $12.

8.2.2 Схема алгоритму роботи моделі

Для моделювання пристрою керування двигуном постійного струму у пакеті PROTEUS 8.6 (рисунок 8.11) розроблено схему алгоритму роботи моделі (рисунок 8.22) та робочу програму мовою С, які наведено нижче.

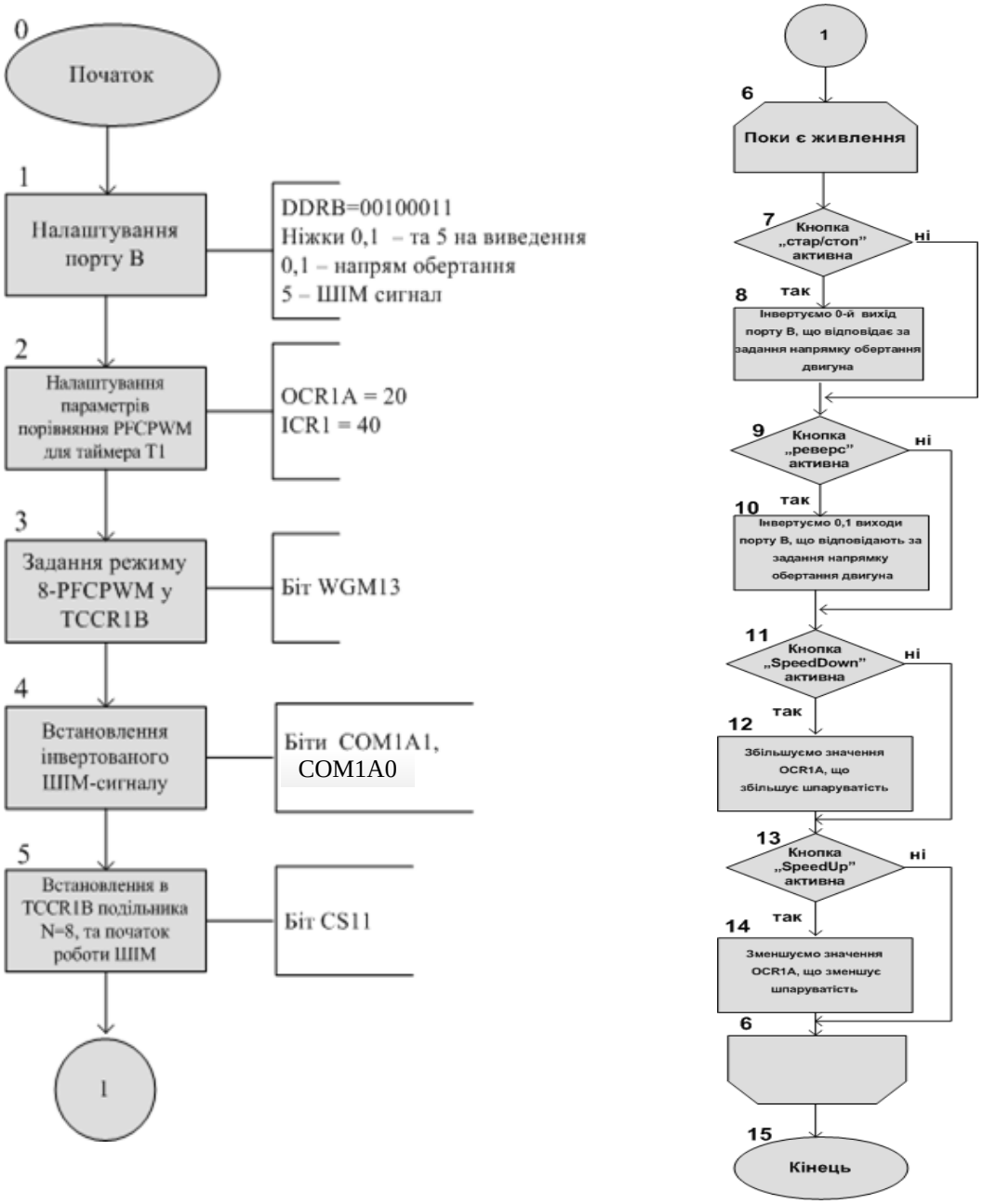


Рисунок 8.22 – Схема алгоритму роботи

8.2.3 Робоча програма

Нижче наведено робочу програму мовою С.

```
#include <inttypes.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <avr/sleep.h>
#include <util/delay.h>
#include F_CPU 16000000L
int main(void) // 0
{
// 1: Init port B
DDRB = 0b00100011;
PORTB = 0b00000000;
// 2: Init 16-bit timer 1 values
OCR1A = 20;
ICR1 = 40;
int delay = 1;//змінна delay, яка використовується при зміні
              //шпаруватості
// Init Phase and frequency correct PWM
TCCR1B = _BV(WGM13);  // 3
TCCR1A = 0; // 4
TCCR1A |= _BV(COM1A0); // 4
TCCR1A |= _BV(COM1A1);
TCCR1B |= _BV(CS11);  // 5
//об'явлення змінних-прапорців
char flag_start_stop=0,flag_reverse=0,flag_speedup=0,flag_speeddown=0;
while(1) // 6
{
// 7: Start-Stop button action
if(!(PINB&4))
{ flag_start_stop = 1; _delay_ms(10); }
if(( flag_start_stop==1 )&&(PINB&4))
{PORTB^=1; flag_start_stop = 0; } // 8
// 9: Reverse button action
if(!(PINB&8))
{ flag_reverse = 1; _delay_ms(10); }
if(( flag_reverse==1 )&&(PINB&8))
{PORTB^=3; flag_reverse = 0; }     // 10
// 11: Speed -  button action
if(!(PINB&16))
{ flag_speeddown=1; _delay_ms(10); }
if(( flag_speeddown==1 )&&(PINB&16))
{ if (OCR1A!=40) OCR1A+=delay; flag_speeddown = 0; } // 12
// 13: Speed + button action
if(!(PINB&64))
{ flag_speedup = 1; _delay_ms(10); }
if(( flag_speedup==1 )&&(PINB&64))
{
if (OCR1A!=0)OCR1A- =delay; // 14
flag_speedup = 0;
```

```
48 }
49 } // 6: End while
50 } // 15: End of proram
```

Нижче наведено деякі пояснення окремих фрагментів програми, яку наведено вище.

Ініціалізація порту B

Напрям передачі даних через контакт введення/виведення порту B визначає відповідний біт DDxn регістра DDRB. Якщо цей біт встановлено в одиницю, то n-й вивід порту являє собою вихід, якщо ж цей біт скинуто у нуль, то цей вивід функціонує як вхід. Загальний вигляд регістра DDRx показано на рисунку 6.23.

Рисунок 8.23 – Вигляд регістра DDRx

В моделі виводи 0, 1 програмуються на виведення для передачі сигналів на входи драйвера двигуна, за допомогою яких відбувається керування напрямом обертання двигуном постійного струму. Вивід 5 також програмується на вихід, оскільки з нього знімається ШІМ-сигнал, який керує швидкістю обертання двигуна постійного струму. Тоді в регістр DDRB записується значення: 0b00100011.

Через можливе близьке знаходження двигуна до мікроконтролера і наведення завад вбудовані у мікроконтролер підтягуючі резистори не використовуються, але в моделі застосовано зовнішні резистори по 10кОм.

Відповідний біт PORTBn регістра PORTB виконує подвійну функцію. Якщо вивід порту запрограмовано на виведення, то цей біт визначає стан виходу порту. Коли він встановлений в одиницю, на виході встановлюється напруга високого рівня, коли ж він скинутий в нуль, на виході встановлюється напруга низького рівня.

Якщо ж вивід порту запрограмовано на введення, то біт PORTBn визначає стан внутрішнього підтягуючого резистора для даного виводу. При встановленні біта PORTBn в одиницю підтягуючий резистор підключається між виводом мікроконтролера та лінією живлення. Загальний вигляд регістра PORTx показано на рисунку 8.24. В моделі регістр PORTB програмується нулями: PORTB = 0b00000000 (внутрішні підтягуючі резистори відключено).

PORTx Register

Bit No.	7	6	5	4	3	2	1	0
Name	Px7	Px6	Px5	Px4	Px3	Px2	Px1	Px0
Initial Value	0	0	0	0	0	0	0	0

Upper Nibble Lower Nibble

Рисунок 8.24 – Вигляд регістра PORTx

Запуск та зупинка двигуна постійного струму

Для керування запуском/зупинкою двигуна постійного струму за допомогою драйвера та мікроконтролера, нам потрібно читати вхід з защіпки порту, до якого підключена відповідна кнопка. Для цього будемо перевіряти біт Px2 регістра PINB. Загальний вигляд регістра PINx показано на рисунку 8.25.

PINx Register

Bit No.	7	6	5	4	3	2	1	0
Name	Px7	Px6	Px5	Px4	Px3	Px2	Px1	Px0
Initial Value	0	0	0	0	0	0	0	0

Upper Nibble Lower Nibble

Рисунок 8.25 – Вигляд регістра PINx

Якщо на вхід PB2 порту буде подаватись високий рівень напруги, то біт Px2 прийматиме значення одиниці, якщо низький рівень напруги то біт Px2 прийматиме значення нуля.

Щоб запобігти зайвим спрацюванням при натисканні кнопки керування, введено прапорець натискання та затримку 10 мс.

Керуюча програма буде змінюватися вже після відпускання кнопки. ШІМ-сигнал поступає на вхід EN1 драйвера двигуна одразу після ініціалізації таймера. Проте двигун не обертається тому що на входах IN1 та IN2 драйвера присутні логічні нулі. На виводі PB2 порту B, через який подається сигнал керування запуском/зупинкою двигуна постійного струму, спочатку присутній високий рівень напруги. Після натискання кнопки на другий вхід порту B подається низький рівень напруги, встановлюється в одиницю прапорець натискання та робиться пауза на 10 мс. Наступна умова спрацює тоді, коли при встановленому прапорці, кнопку знову розімкнули. Після спрацювання цієї умови скидається прапорець, та інвертується сигнал на нульовому виході порту B. На керуючих входах драйвера встановлюються сигнали: IN1=1, IN2=0. Двигун починає обертатися за годинниковою стрілкою. Почерговою зміною рівня напруги на вході PB2 можна керувати запуском/зупинкою двигуна постійного струму.

Нижче приведено фрагмент коду на мові C, який все це реалізовує.

```
//Start-Stop button action
 if(!(PINB&4))
 { flag_start_stop = 1; _delay_ms(10); }
 if(( flag_start_stop==1 )&&(PINB&4))
 {PORTB^=1; flag_start_stop = 0; }
```

Зміна напрямку обертання двигуна постійного струму

Для зміни напрямку обертання двигуна постійного струму, або ввімкнення реверсу, потрібно слідкувати за станом відповідної кнопки керування, та за її натисканням, а потім відпусканням змінювати напрям руху. Слідкування за натисканням кнопки робиться за допомогою зчитування рівня напруги з защіпки порту, до якого підключена ця кнопка. В моделі дану кнопку підключено до виводу PB3 мікроконтролера ATmega128, який налаштовано як вхід. При натисканні кнопки рівень напруги буде низький, тобто подається нуль. Коли кнопку розімкнено, то рівень напруги буде високий і подається одиниця.

Логіка керування зміною напрямку обертання двигуна постійного струму буде наступною.

Сторона, у яку обертається двигун залежить від значення сигналів на входах IN1 та IN2 драйвера двигуна L293D. Зміна сигналів буде робитися одразу на лініях PB0 та PB1 мікроконтролера, які налаштовані як виходи. Лінія з низьким потенціалом змінюється на високий рівень, а лінія з високим рівнем – на низький рівень. Вище було відмічено, що після запуску двигуна з лінії PB0 знімається високий рівень напруги, а з лінії PB1 – низький рівень напруги. Після натискання та відпускання кнопки K2 (ReverseOrder Line) за допомогою операції побітового XOR будуть інвертуватися два перші розряди регістра PORTB та змінюватися напрямок обертання двигуна.

Нижче наведено фрагмент коду на мові C, який виконує вказані вище операції.

```
  //Reverse button action
if(!(PINB&8))
{ flag_reverse = 1; _delay_ms(10); }
if(( flag_reverse==1 )&&(PINB&8))
{PORTB^=3; flag_reverse = 0; }
```

Зміна швидкості обертання двигуна постійного струму

Для зміни швидкості обертання двигуна постійного струму в моделі використовуються дві кнопки: перша буде збільшувати швидкість, а друга – зменшувати. Принцип, за яким буде змінюватися швидкість, ґрунтується на властивості ШІМ-сигналу, яку описано вище. Двигун до джерела живлення підключено через ключі за мостовою схемою, яку показано на рисунку 8.12.

В даному випадку для нас важливі комбінації, коли є різниця потенціалів на обмотці двигуна. Це наступні комбінації ключів: 1 – відкритий, 2 – закритий, 3 – закритий, 4 – відкритий; 1 – закритий, 2 – відкритий, 3 – відкритий, 4 – закритий.

Є комбінації, коли на обмотці двигуна немає різниці потенціалів, і ротор електродвигуна не обертається: 1 – відкритий, 2 – відкритий, 3 – закритий, 4 – закритий; 1 – закритий, 2 – закритий, 3 – відкритий, 4 – відкритий.

Тобто, коли ключі відкривати за діагоналлю, двигун починає обертатись. Для того, щоб регулювати швидкість обертання треба вмикати ці ключі на час X та вимикати на час Y з великою швидкістю (рисунок 8.4).

Швидкість обертання двигуна залежить від напруги, яка подається в обмотку збудження. На рисунку 8.4 ця напруга позначається, як еквівалентна постійна напруга. З цього рисунку видно, що при зміні тривалості імпульсу в межах періоду, який не змінюється, змінюється шпаруватість імпульсного сигналу, що викликає зміну еквівалентної постійної напруги. Коли буде збільшуватися тривалість імпульсу, буде зменшуватися шпаруватість та збільшуватися еквівалентна постійна напруга. У цьому випадку двигун буде обертатися швидше. Коли буде зменшуватися тривалість імпульсу, буде збільшуватися шпаруватість та зменшуватися еквівалентна постійна напруга. У цьому випадку двигун буде обертатися повільніше.

Таким чином, при зміні шпаруватості – відношенні періоду слідування імпульсів до їх довжини, змінюються еквівалентна постійна напруга та швидкість обертання двигуна постійного струму.

Для збільшення швидкості використовується кнопка К3 (SpeedUp Line). Кнопку підключено до виводу PB4 мікроконтролера, який запрограмовано як вхід. Кнопку у початковому стані не натиснуто, тому сигнал на вході PB4 має значення логічної одиниці. Коли кнопку натиснуто, цей сигнал прийме значення логічного нуля. При виявленні в програмі логічного нуля встановлюється прапорець натискання та очікується 10 мс. Наступна умова у нас виконається, якщо індикатор натискання знову встановлено в одиницю (на защіпці PB4 порту логічна одиниця, тобто кнопку вже відпущено). У цьому випадку збільшується значення 8-бітового регістра OCR1A на величену, яку записано у змінну delay, але не більше, ніж 20.

У цьому випадку постійна еквівалентна напруга буде мати значення, при якому двигун постійного струму буде обертається з максимальною швидкістю, яку ми можемо досягти при моделюванні.

Нижче наведено код на С, який відповідає за збільшення швидкості обертання двигуна постійного струму при натисканні, а потім відпусканні відповідної кнопки.

```
  //Speed + button action
   if(!(PINB&16))
   { flag_speedup = 1; _delay_ms(10); }
   if(( flag_speedup==1 )&&(PINB&16))
   { if (OCR1A!=40) OCR1A+=delay; flag_speedup = 0; }
```

Зменшення швидкості обертання двигуна постійного струму відбувається за схожим принципом. В програмі перевіряється рівень напруги на виводі PB6 регістра PINB, який запрограмовано на вхід. Коли кнопка не натиснена, на вході високий рівень, тобто одиниця. Коли кнопка натискається, значення біта змінюється з одиниці на нуль, і залишається таким до того часу, доки кнопка натиснена. Щоб запобігти зайвим спрацюванням при натисканні кнопки керування, введено прапорець натискання та затримку 10 мс. Керуюча програма буде змінюватися вже після відпускання кнопки. У цьому випадку зменшується значення 8-бітового регістра OCR1A на величену, яку записано у змінну delay, але не менше, ніж нуль.

```
//Speed - button action
   if(!(PINB&64))
   { flag_speeddown = 1; _delay_ms(10); }
   if(( flag_speeddown==1 )&&(PINB&64))
   {
      if (OCR1A!=0)OCR1A- =delay;
      flag_speeddown = 0;
   }
```

8.3 Зміст звіту

Звіт по роботі повинен містити:

- схему моделі;
- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

Контрольні запитання та завдання

1) Для чого може використовуватись таймер/лічильник?
2) Які регістри відповідають за дозвіл/заборону переривань від таймерів/лічильників?
3) Чим відрізняється режим роботи CTC від режиму Normal?
4) Для рішення яких завдань краще використання режиму Phase Correct PWM ніж режиму Fast PWM?
5) Для чого можуть використовуватись 16-розрядні таймери/лічильники?
6) При виникненні яких подій таймери/лічильники T1, T3, T4, T5 можуть генерувати переривання?
7) За допомогою яких регістрів виконується керування таймером/лічильником?
8) Які сигнали можуть використовуватись в якості тактового сигналу fclkn для таймерів/лічильників T1, T3, T4 і T5?
9) Яким чином здійснюється вибір джерела тактового сигналу, а також запуск і зупинка таймерів/лічильників?
10) Як увімкнути режим роботи Normal таймера/лічильника T3? Для чого може використовуватись даний режим роботи?
11) Що таке роздільна здатність лічильника таймера?
12) У чому полягає різниця між роботою 8-розрядних та 16-розрядних таймерів/лічильників у режимі Fast PWM?
13) Завдяки чому в режимі Phase and Frequency Correct PWM кожен період сигналу є повністю симетричним?
14) Поясніть, що таке ШІМ-сигнал?
15) Як обчислюється шпаруватість ШІМ-сигналу?
16) Як впливає значення шпаруватості на швидкість обертання двигуна?
17) Наведіть та поясніть модель пристрою керування дсигуном.
18) Наведіть та поясніть схему алгоритму роботи моделі.
19) Наведіть та поясніть робочу програму на мові Асемблер, що керує моделлю.
20) Наведіть та поясніть робочу програму на мові CI, що керує моделлю.

9 ЛАБОРАТОРНА РОБОТА №6. МОДЕЛЮВАННЯ МОДУЛЯ УНІВЕРСАЛЬНОГО АСИНХРОННОГО ПРИЙМАЧА-ПЕРЕДАВАЧА СІМ'Ї AVR

Тема: Моделювання модуля універсального асинхронного приймача-передавача сім'ї AVR

Мета: Користуючись пакетом PROTEUS дослідити моделювання модуля універсального асинхронного приймача-передавача сім'ї AVR

9.1 Порядок виконання роботи

1) Створити модель пристрою в пакеті Proteus 8.6.
2) Розробити схему алгоритму роботи моделі та робочу програму.
3) Створити hex-файл та підключити його до мікроконтролера.
4) Запустити модель та виконати її дослідження згідно методичних вказівок.
5) Зробити відповідні висновки.

9.2 Стислі теоретичні відомості

9.2.1 Архітектура модуля УСАПП в складі AVR-мікроконтролерів

Більшість мікроконтролерів сімейства AVR мають у своєму складі модуль універсального асинхронного приймача-передавача (УАПП), або універсального синхронно-асинхронного приймача-передавача (УСАПП). У деяких моделях міститься по декілька таких модулів [1; 10].

В іноземній літературі модуль УАПП позначається, як UART (Universal Asynchronous Receiver and Transmitter) – універсальний асинхронний приймач-передавач, а модуль УСАПП – як USART (Universal Synchronous and Asynchronous Receiver and Transmitter) – універсальний синхронно-асинхронний приймач-передавач.

Модулі УСАПП, які мають всі мікроконтролери сімейства Mega, при роботі в асинхронному режимі сумісні з модулями УАПП як за розміщенням розрядів

керуючих регістрів, так і за функціонуванням. В деяких моделях модулі УСАПП можуть використовуватись в якості ведучого шини SPI [1, 10].

Всі модулі УАПП/УСАПП підтримують дуплексний обмін за послідовним каналом. При цьому швидкість передачі даних може змінюватись у доволі широких межах. У модулях УАПП пакет може бути 8-ми чи 9-ти розрядним, а в модулях УСАПП його довжина може складати від 5-ти до 9-ти розрядів. Ще однією особливістю модулів УСАПП у порівнянні з УАПП є наявність схем формування та контролю парності.

Модулі УАПП/УСАПП можуть виявляти наступні не типові ситуації:

– переповнення;
– помилку кадрування;
– некоректний старт-біт.

В модулях є також корисна функція – фільтрація завад.

Для програмування модулів передбачено 3 переривання, запит на генерацію яких формується при виникненні наступних подій:

– передачу завершено;
– регістр даних передавача пустий;
– прийом завершено.

Виводи мікроконтролера, що використовуються модулями УАПП/УСАПП, є лініями портів введення/виведення загального призначення. Як приклад, виводи деяких мікроконтролерів, що використовуються модулями УСАПП сімейства Mega, та їх функції наведено у [1, 10].

Спрощену структурну схему модуля УАПП/УСАПП наведено на рисунку 9.1 у [1, 10]. Елементи, які виділені на рисунку сірим кольором, є тільки у складі модулів УСАПП.

Модуль має три частини: блок тактування, блок передавача та блок приймача.

Блок тактування включає схему синхронізації, яка використовується при роботі у синхронному режимі, і контролер швидкості передачі. У модулях УАПП блок тактування складається тільки з контролера швидкості передачі.

Блок передавача має однорівневий буфер UDR, регістр зсуву, схему формування біта парності (тільки УСАПП) і схему керування.

Блок приймача, включає схеми відновлення тактового сигналу і даних; схему контролю парності (тількиУСАПП); буфер UDR: дворівневий в УСАПП та однорівневий в УАПП; регістр зсуву, а також схему керування.

Буферні регістри приймача і передавача розміщуються за однаковою адресою простору регістрів введення/виведення і позначаються як регістр даних UDR (Universal Data Register – UDRn, де n = 0/1). У цих регістрах зберігаються молодші 8 розрядів даних, які приймаються чи передаються. Під час читання UDR виконується звернення до буферного регістра приймача, а у разі запису – до буферного регістра передавача. Розміщення регістрів даних для деяких моделей мікроконтролерів наведено у [1; 10].

Дворівневий буфер приймача у модулях УСАПП є FIFO-буфером (First In First Out, першим зайшов, першим вийшов). Зміна стану буфера відбувається при будь-якому зверненні до регістра UDR. Тому, не варто використовувати регістр UDR в якості операндів команд типу «читання/модифікація/запис» (SBI та CBI) та команди перевірки SBIC та SBIS, які також змінюють стан буфера приймача.

Для керування модулями УСАПП використовуються три регістри: UCSRA (UCSRnA), UCSRB (UCSRnB) та UCSRC (UCSRnC), де n = 0, 1, 2 або 3. Адреси цих регістрів, їх формати та опис окремих розрядів наведено у [1; 10].

9.2.2 Швидкість прийому/передачі даних

В асинхронному режимі, а також у синхронному режимі при роботі у якості ведучого, швидкість прийому та передачі даних задається контролером швидкості передачі, що функціонує як дільник системного тактового сигналу з програмованим коефіцієнтом ділення. Коефіцієнт визначається вмістом регістра контролера UBRR (UBRRn). Після дільника у блок приймача сигнал надходить одразу, а у блок передавача – через додатковий дільник, коефіцієнт ділення якого: 2, 8 чи 16 залежить від режиму роботи модуля УАПП/УСАПП.

У [1, 10] в якості прикладу наведено структурну схему блока синхронізації модуля УСАПП для мікроконтролера Mega 128.

Швидкість обміну в модулях УАПП/УСАПП визначається наступними формулами:

– звичайний асинхронний режим (U2Xn = 0):

$$BAUD = fclк / (16(UBRR+1)); \quad (9.1)$$

– пришвидшений асинхронний режим (U2Xn = 1):

$$BAUD = fclк / (8(UBRR+1)); \quad (9.2)$$

– синхронний ведучий режим:

$$BAUD = fclк / (2(UBRR+1)), \quad (9.3)$$

де BAUD – швидкість передачі у бодах, fclк – тактова частота мікроконтролера, UBRR – вміст регістра контролера швидкості передачі: 0...4095.

В якості прикладу в [1, 10] наведено значення регістра UBRR, що дозволяють отримати стандартні для асинхронного режиму швидкості передачі при використанні деяких резонаторів, а також значення похибок, що отримуються, відносно стандартних швидкостей.

При максимальному значенні швидкості UBRR = 0, похибка = 0,0%. Рекомендується використовувати значення регістра UBRR, при яких отримувана швидкість передачі відрізняється від необхідного значення менше ніж на 0,5% (у [1, 10] виділено жирним шрифтом).

9.2.3 Формат кадру

Сукупність одного слова (байта) даних і додаткової інформації під час передачі називається кадром (рисунок 9.1).

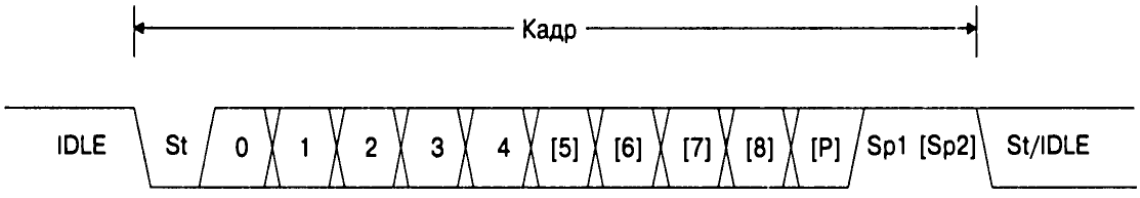


Рисунок 9.1 – Формат кадру

На рисунку 9.1 використано наступні скорочення:

– St – старт-біт, завжди має низький рівень;
– 0…8 – номер біта даних;
– P – біт паритету: парність або непарність;
– Sp1, Sp2 – стоп-біти, завжди мають високий рівень;
– IDLE – стан очікування, в якому призупинено обмін лінією RxD чи TxD.

У стані очікування на лінії має бути високий рівень.

До початку передачі в лінію передається високий рівень цифрового сигналу, що говорить приймачу про справність лінії зв`язку. Кадр починається із нульового старт-біта, який використовується схемою синхронізації приймача для підлаштування фази синхрочастоти приймача під отримувані посилки (біти). Далі передаються біти слова даних, починаючи з молодшого розряда. Після останнього старшого розряда слова даних передаються один або два стоп-біти. Якщо запрограмовано формування біта парності, то він розміщується між старшим розрядом слова даних і першим стоп-бітом.

Формат кадру визначається кількома розрядами регістрів UCSRB (UCSRnB) та UCSRC (UCSRnC) [1; 2]. Розмір слова даних в USART визначається відповідно до [1; 2].

У модулях УАПП слово даних може бути тільки 8- або 9-розрядним, що визначається станом прапорця CHR9 (CHR9n) регістра UCSRB (UCSRnB). Якщо цей прапорець скинуто в нуль, розмір слова дорівнює 8 розрядам, якщо встановлено в одиницю – 9 розрядам.

Вибір кількості стоп-бітів в модулях УСАПП здійснюється за допомогою розряду USBS (USBSn) регістра UCSRC (UCSRnC). Якщо цей розряд скинуто в нуль, блок передавача у кінці посилки формує один стоп-біт. Якщо розряд встановлено в одиницю, блок передавача формує два стоп-біти. Слід зазначити, що приймачем другий стоп-біт ігнорується, і, відповідно, помилки кадрування виявляються тільки для першого стоп-біта.

Функціонування схеми контролю парності модулів УСАПП визначають розряди UPM1:UPM0 (UPMn1:UPMn0) регістра UCSRC (UCSRnC) відповідно до [1, 2].

Значення біта парності отримується шляхом виконання операції «виключне АБО» над усіма розрядами слова даних, яке передається. Якщо використовується перевірка на парність (even parity), то отриманий результат інвертується:

$$P_{EVEN} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus 1,$$
$$P_{ODD} = d_{n-1} \oplus \ldots \oplus d_3 \oplus d_2 \oplus d_1 \oplus d_0 \oplus \underline{0}. \quad (9.4)$$

Якщо контроль парності включено, біт парності вставляється передавачем між старшим розрядом даних, які передаються, і першим стоп-бітом.

9.2.4 Передача даних

Для дозволу роботи передавача необхідно встановити в одиницю розряд TXEN (TXENn) регістра UCSRB (UCSRnB). Після цього вивід TXD (TXDn) підключається до передавача модуля УАПП/УСАПП і починає функціонувати, як вихід, незалежно від значень у регістрах керування напрямом обміну портом. Якщо використовується синхронний режим роботи, треба програмно перевизначити функціонування виводу ХСК (ХСКn) на введення або виведення.

В асинхронному режимі передача ініціюється записом даних, які передаються, у буферний регістр передавача – UDRn (рисунок 9.2, а).

З регістра UDRn дані пересилаються у регістр зсуву передавача. Якщо використовуються 9-розрядний формат даних, значення розряду TXB8 (TXB8n) регістра UCSRB (UCSRnB) копіюється у 9-й розряд регістра зсуву. 9-й розряд даних повинен бути завантажений у розряд TXB8 (TXB8n) до запису байта даних у регістр даних.

Можливі два варіанти реакції на запис даних у регістр UDR:

- якщо запис у регістр здійснюється в той момент, коли передавач знаходиться у стані очікування (попередні дані вже передано), то у цьому випадку дані пересилаються у регістр зсуву одразу ж після запису в регістр UDR;
- якщо запис у регістр UDR здійснюється під час передачі, то в цьому випадку дані пересилаються у регістр зсуву після передачі останнього стоп-біта поточного кадру.

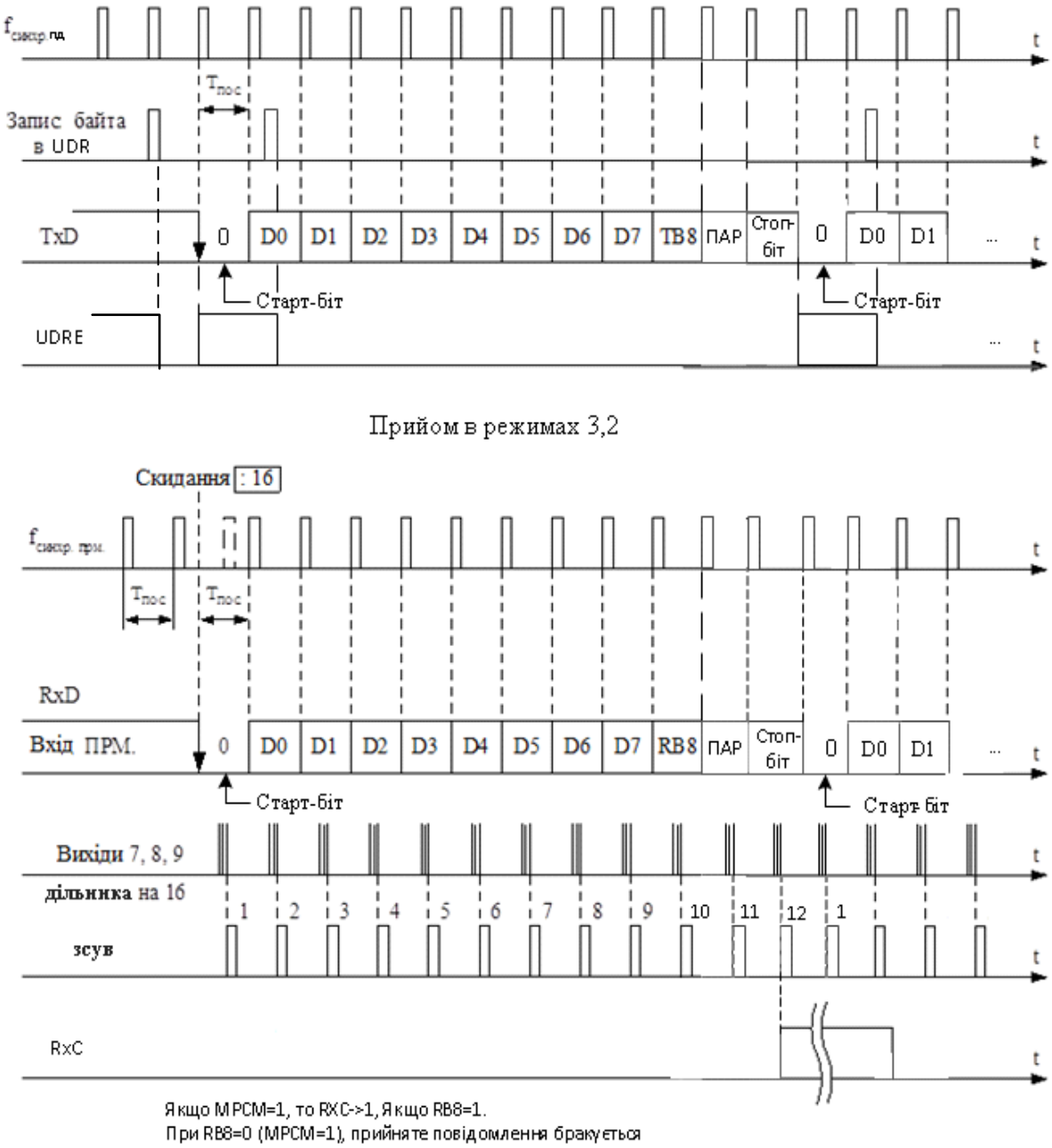


Рисунок 9.2 – Часові діаграми роботи інтерфейсу в асинхронному режимі: а – передача; б – прийом даних

Після отримання сигналу «RESET» та після пересилання слова даних у регістр зсуву, прапорець UDRE (UDREn) регістра UCSRA (UCSRnA) встановлюється в одиницю, що означає готовність передавача до отримання нового байта даних. У цьому стані прапорець залишається до наступного запису в буфер. Одночасно з пересиланням у регістрі зсуву формується службова інформація – старт-біт, можливий біт парності (тільки в USART), а також один або два стоп-біти.

Після завантаження регістра зсуву його вміст починає зсуватися вправо і поступати на вивід TXD (TXDn) у порядку, розглянутому на рисунку 9.1. Швидкість зсуву визначається налаштуванням контролера тактових сигналів.

Якщо під час передачі попереднього слова у регістр UDR було записано нове слово даних, то після передачі останнього стоп-біта попереднього слова в регістр зсуву пересилається нове слово. Якщо ж до моменту закінчення передачі кадру нового запису виконано не було, то встановлюється прапорець переривання TXC (TXCn) регістра UCSRA (UCSRnA) – «Передачу завершено». Скидання прапорця здійснюється апаратно при вході у підпрограму обробки переривання або програмно, записом в цей розряд одиниці.

Відключення передавача здійснюється скиданням розряду TXEN (TXENn) регістра UCSRB (UCSRnB). Якщо у момент виконання цієї команди здійснювалася передача, скидання розряду відбудеться тільки після завершення поточної та відкладеної передач, тобто після очищення буферного регістра та регістра зсуву передавача. При вимкненому передавачеві вивід TXD (TXDn) може використовуватися, як лінія введення/виведення загального призначення.

9.2.5 Прийом даних

Для дозволу роботи приймача треба встановити розряд RXEN (RXENn) регістра UCSRB (UCSRnB) [1; 10]. Після цього вивід RXD (RXDn) підключається до приймача УАПП/УСАПП і починає функціонувати як вхід, незалежно від значень регістра керування портом. Якщо використовується синхронний режим роботи, треба програмно перевизначити функціонування виводу XCK (XCKn).

Прийом даних починається після виявлення приймачем коректного старт-біта (див. вище рисунок 9.2, б). Потім кожен розряд кадру зчитується з частотою, яка визначається контролером швидкості передачі або тактовим сигналом XCK (XCKn). Отримані розряди даних послідовно розміщуються у регістр зсуву приймача до виявлення першого стоп-біта кадру. Після цього вміст регістра зсуву пересилається у буфер приймача, з якого прийняте значення може бути зчитано.

При використанні 9-розрядних слів даних значення старшого восьмого розряду, оскільки нумерація розрядів ведеться від нуля, може бути визначене за станом прапорця RXB8 (RXB8n) регістра UCSRB (UCSRnB). У модулях USART вміст цього розряду має бути зчитаний до звернення до регістра даних. Це пов'язано з тим, що прапорець RXB8 (RXB8n) відображає значення старшого розряду слова даних кадру, який знаходиться на верхньому рівні буфера приймача, стан якого при читанні регістра даних зміниться.

Якщо було запрограмовано контроль парності (тільки УСАПП), то схема контролю парності обчислює біт парності для всіх розрядів прийнятого слова даних і порівнює його з прийнятим бітом парності. Результат перевірки запам'ятовується у буфері приймача разом з прийнятим словом даних і стоп-бітами. Наявність або відсутність помилки контролю парності може бути потім визначено за станом прапорця UPE (UPEn) регістра UCSRA (UCSRnA). Цей прапорець встановлюється в одиницю якщо наступне слово, яке може бути прочитане з буфера, має помилку контролю парності. При вимкненому контролі парності прапорець UPE (UPEn) завжди читається як нуль.

У регістрі UCSRA(UCSRnA) блоку приймача модулів УАПП/УСАПП є ще два прапорці, які показують стан обміну: прапорець помилки кадрування FE (FEn) і прапорець переповнення DOR (DORn)/OR (Orn) [1; 10]. Прапорець FE (Fen) встановлюється в одиницю, якщо значення першого стоп-біта прийнятого кадру не відповідає потрібному, тобто дорівнює нулю.

Прапорці DOR (DORn) в УСАПП та OR (Orn) в УАПП відображають втрату даних через переповнення буфера приймача. В УАПП прапорець встановлюється в одиницю, якщо до моменту закінчення прийому кадру (заповнення регістра зсуву приймача) дані попереднього кадру не були зчитані з регістра даних. В УСАПП прапорець встановлюється в одиницю у разі прийому старт-біта нового кадру при заповнених буфері та регістрі зсуву приймача. Встановлений прапорець DOR (DORn)/OR (Orn) означає, що між минулим байтом, який зчитано з регістра UDR, та байтом, зчитаним у даний момент, відбулася втрата одного або декількох кадрів.

Обробка прапорців в модулях УАПП/УСАПП дещо відрізняється.

У модулях УАПП прапорець помилки кадрування FE (Fen) має бути прочитаний перед зверненням до регістра даних, а прапорець переповнення OR (Orn) – після звернення до цього регістра.

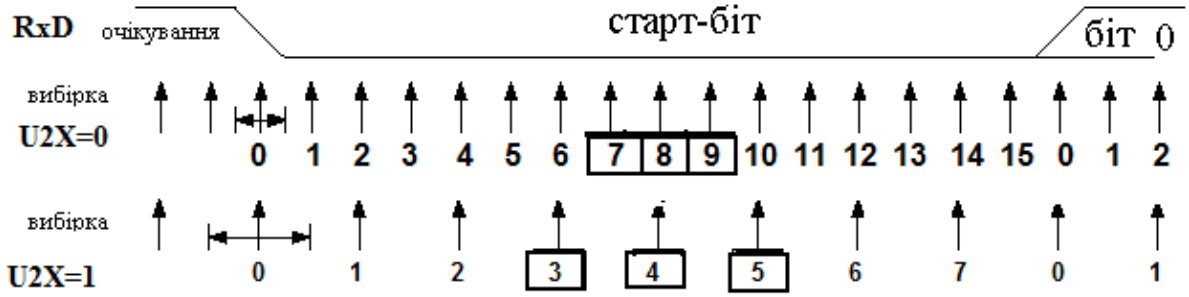
У модулях УСАПП всі прапорці помилок буферизуються разом із словом даних, тобто відповідні розряди регістра UCSRA (UCSRnA) відносяться до кадру, слово даних якого буде прочитано при наступному зверненні до регістра даних UDR (UDRn). Тому стан цих прапорців треба зчитати перед зверненням до регістра даних.

Для індикації стану приймача в модулях УАПП/УСАПП використовується прапорець переривання «Прийом завершено» RXC (RXCn) регістра UCSRA (UCSRnA). Цей прапорець встановлюється в одиницю за наявності в буфері приймача не прочитаних даних. У модулях УАПП цей прапорець скидається після читання регістра даних, а у модулях УСАПП – при звільненні буфера (після зчитування всіх даних, які знаходяться у ньому).

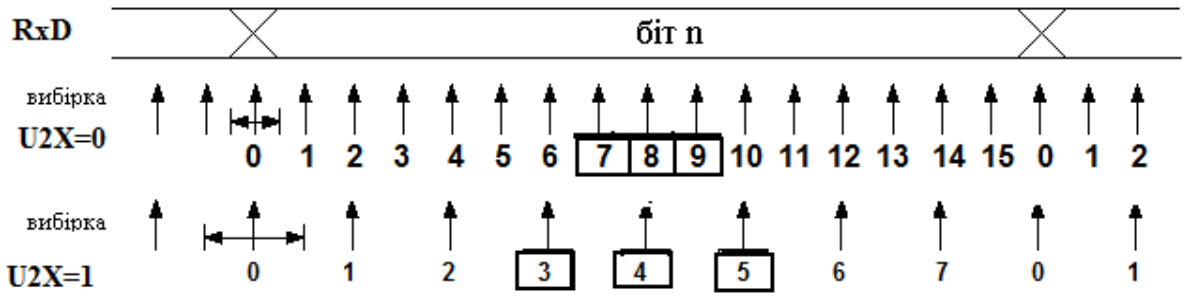
Вимкнення приймача здійснюється скиданням розряду RXEN (RXENn) регістра UCSRB (UCSRnB). На відміну від передавача, приймач вимикається відразу ж після скидання розряду, тобто кадр, який приймається у цей момент, втрачається. При вимкненні приймача очищується його буфер, тобто втрачаються також всі непрочитані дані. При вимкненому приймачі вивід RXD (RXDn) може використовуватися як лінія введення/виведення загального призначення.

При прийомі в асинхронному режимі роботи використовуються схеми відновлення тактового сигналу і даних. Схема відновлення тактового сигналу призначена для синхронізації внутрішнього тактового сигналу, який формується контролером швидкості передачі, і кадрів, які поступають на вивід RXD (RXDn) мікроконтролера. Схема відновлення даних здійснює читання та фільтрацію кожного розряду кадру, що приймається.

Для визначення старт-біта кадру схема відновлення тактового сигналу опитує вхід приймача (рисунок 9.3).



а



б

Рисунок 9.3 – Розпізнавання розрядів кадру: а – старт-біт; б – інші розряди

Частота опитування залежить від стану розряду U2X (U2Xn) регістра UCSRA (UCSRnA). У звичайному режимі (при U2Xn = 0) частота опитування у 16 разів перевищує швидкість передачі даних, а у прискореному режимі (при U2X = 1) – у 8 разів. Горизонтальні стрілки на рисунку 9.3 ілюструють можливий відхід синхронізації у процесі опитування. Більш високу розсинхронізацію у часі має прискорений обмін (біт U2X = 1).

Виявлення в процесі очікування зміни сигналу на виводі RXD (RXDn) з одиниці на нуль інтерпретується як можлива поява переднього фронту старт-біта. Коли виявлено цю зміну, скидається лічильник-дільник на 16 у ланцюзі формування сигналу «Синхр. RXD» (рисунок 9.2, б). В результаті цього відбувається суміщення моментів переповнення цього лічильника-дільника з межами зміни на вході RXD бітів кадру, що приймається. Шістнадцять станів лічильника-дільника ділять час, протягом якого кожен біт посилки, що приймається, присутній на вході RXD, на 16 фаз (вибірок), з 0-ї по 15-у для кожного біта.

У нормальному режимі перевіряється значення 7-ої, 8-ої і 9-ої вибірок вхідного сигналу, а в прискореному режимі – 3-ої, 4-ої і 5-ої вибірок (рисунок 9.3, а). Якщо значення хоча б двох вибірок із вказаних дорівнює одиниці, старт-біт вважається помилковим (завада), а приймач переходить до очікування наступної зміни вхідного сигналу з одиниці на нуль. Інакше вважається, що виявлено старт-біт нової послідовності даних, з яким синхронізується внутрішній тактовий сигнал приймача. Після цього починає працювати схема відновлення даних.

Рішення про значення кожного прийнятого розряду також ухвалюється за результатами 7-ої, 8-ої і 9-ої (3-ої, 4-ої і 5-ої) вибірок вхідного сигналу (рисунок 9.3, б). Станом розряду вважається логічне значення, яке було отримане щонайменше у двох з трьох вибірок. Процес розпізнавання повторюється для всіх розрядів кадру, що приймається, включаючи перший стоп-біт.

Таким чином, старт-біт нового кадру може передаватись відразу ж після останньої вибірки, яка використовується для визначення значення першого стоп-біта. У звичайному режимі роботи формування старт-біта може початися в момент А, а в прискореному режимі – в момент В (рисунок 9.4).

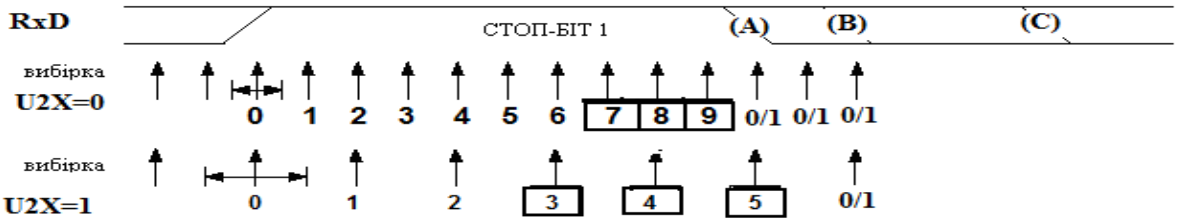


Рисунок 9.4 – Розпізнавання стоп-біта та наступного старт-біта

Момент С на рисунку визначає максимальну тривалість стоп-біта. Використання більш раннього виявлення старт-біта впливає на робочій діапазон синхрочастот приймача (допустиме розходження синхрочастоти передавача та приймача при фіксованій довжині посилки) [1; 10].

9.2.6 Обмін даними через інтерфейс УСАПП у мікроконтролерній мережі

У мікроконтролерних мережах може відбуватися обмін інформацією між одним ведучим та одним з декількох ведених мікроконтролерів, коли кожен ведений мікроконтролер мережі має свою унікальну адресу. Якщо ведучий мікроконтролер хоче що-небудь передати, то він посилає адресний байт, який визначає, до якого з ведених мікроконтролерів він збирається звернутися. Коли один з ведених мікроконтролерів розпізнав свою адресу, він переходить у режим прийому даних і відповідно приймає подальші байти як дані. Решта ведених мікроконтролерів ігнорують байти даних, що приймаються, до посилки ведучим нового адресного байта. Програмування режиму фільтрації кадрів даних, що приймаються і не містять адреси, здійснюється встановленням в одиницю розряду MPCM (MPCMn) регістра UCSRA (UCSRnA) [1; 2].

У мікроконтролері, який виконує роль ведучого, встановлюється режим передачі 9-розрядних даних. При передачі адресного байта старший – 8-й розряд, за умови, що перший (молодший) розряд при двійковому кодуванні має номер – нуль, встановлюється в одиницю, а при передачі байтів даних він скидається в нуль.

У ведених мікроконтролерів механізм прийому залежить від режиму роботи приймача. При прийомі 9-розрядних даних ідентифікація вмісту кадру здійснюється за старшим дев`ятим розрядом слова даних. Якщо вказаний розряд встановлено в одиницю, то кадр містить адресу, якщо скинуто в нуль – кадр містить дані.

Для програмування обміну даними у мікроконтролерній мережі виконуються наступні дії:

1) У всіх ведених мікроконтролерів встановлюється в одиницю розряд MPCM (MPCMn) регістра UCSRA (UCSRnA).

2) Всі мікроконтролери програмуються в режим передачі дев`ятирозрядних кадрів.

3) Ведучий мікроконтролер посилає адресний кадр з TB8 = 1, а всі ведені мікроконтролери його приймають. У кожному з ведених мікроконтролерів встановлюється прапорець RXC (RXCn) регістра UCSRA (UCSRnA).

4) Кожен ведений мікроконтролер читає вміст регістра даних (адресний кадр) та порівнює його із своєю мережевою адресою. Мікроконтролер, адреса якого співпала з адресою, посланою ведучим, скидає в нуль розряд MPCM (MPCMn).

5) Адресований мікроконтролер починає приймати кадри, що містять дані, оскільки їх дев`ятий розряд дорівнює нулю. У решті не адресованих ведених мікроконтролерів розряд MPCM (MPCMn) залишається встановленим в одиницю, через це кадри даних з бітом TB8 = 0 будуть ними ігноруватися.

6) Після прийому останнього байта даних адресований мікроконтролер встановлює в одиницю розряд MPCM (MPCMn) і знову чекає прихід кадру з адресою. Процес повторюється з пункту 3.

9.3 Опис моделі

Робочу модель дослідження універсального асинхронного приймача-передавача (УАПП) наведено на рисунку 9.5.

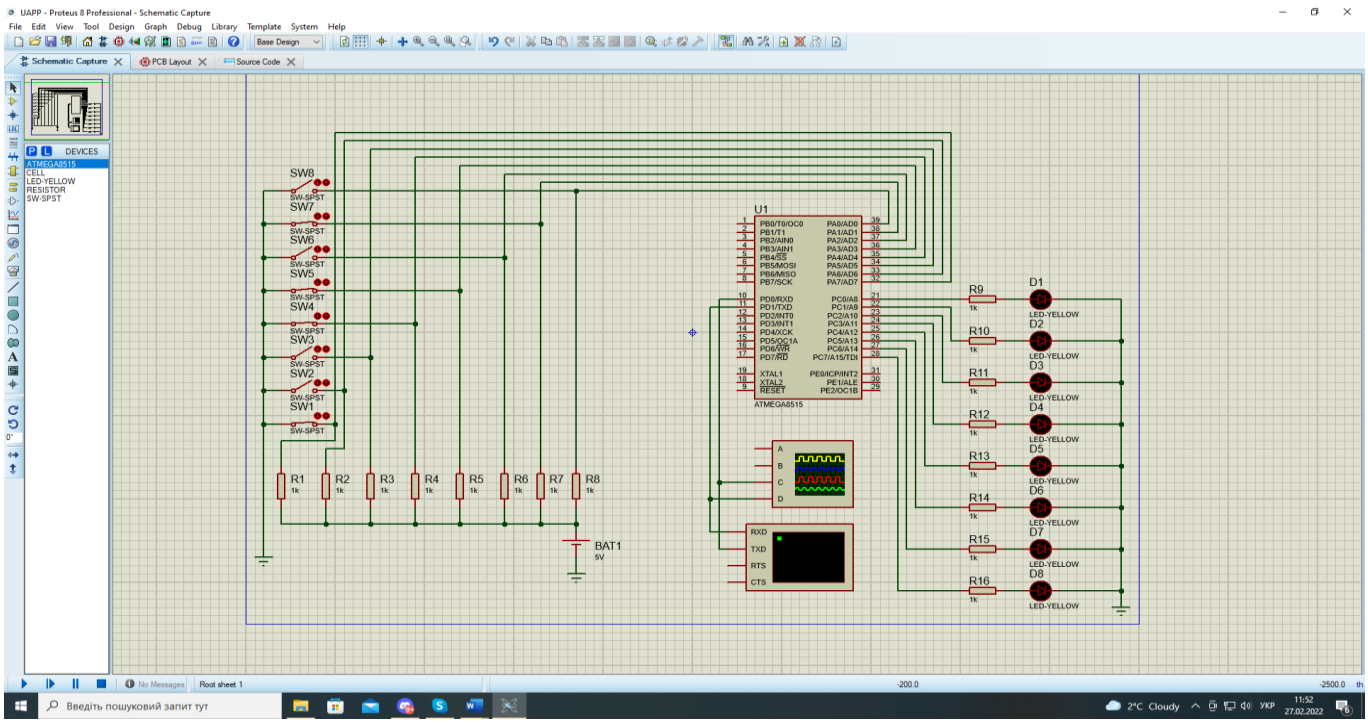


Рисунок 9.5 – Схема робочої моделі

Розберемо цю модель докладніше. Зліва знаходяться вісім кнопок, за допомогою яких задається байт (8 біт) даних, який передається через модуль УАПП.

Кнопки підключено до восьми ліній порту А мікроконтролера: PA.0, PA.1, … , PA.7. Якщо якась із кнопок відпущена, то через резистори R1...R8 на відповідну лінію порту А подається логічна 1. Якщо кнопка нажата, то вводиться логічний нуль. Праворуч зображено вісім світлодіодів, які підключено до ліній порту С: PC.0, PC.1, … , PC.7. Катоди світлодіодів підключено до спільного провода (землі), а на аноди через резистори R9…R16, які обмежують струм, із виходів порту С подаються логічні одиниці, коли треба засвітити відповідний світлодіод.

До ліній TxD – вихід передавача УАПП та RxD – вхід приймача підключено осцилограф, та Virtual Terminal, який вибрано з віртуальних інструментів програми Proteus. На них відображаються повідомлення, які вводиться та виводяться через модуль УАПП в/з мікроконтролера.

До виводів XTAL1 та XTAL2 в реальних схемах підключають кварцовий резонатор частотою, яка визначає тактову частоту генератора мікроконтролера. В нашому прикладі вона дорівнює f_{BQ} = 8МГц. Також підключають конденсатори С1 та С2, ємністю 30пФ, які призначені для підвищення стабільності роботи системного генератора. Резонатор та конденсатори потрібні в практичній схемі. В модель Proteus їх можна не вводити. Для задання частоти треба двічі лівою кнопкою миші клацнути на мікроконтролері та в опції Clock Frequency вказати значення частоти.

В якості мікроконтролера в моделі використано AVR-мікроконтролер ATmega8515.

9.4 Порядок моделювання

1. Перш за все, потрібно пересвідчитись, що Terminal налаштовано правильно. Для цього треба двічі натиснути на нього лівою кнопкою миші, та зробити відповідні налаштування. Нижче наведено приклад для випадку, коли потрібно передавати інформацію зі швидкістю 9600 біт/сек: 8 біт даних та один стоп-біт з перевірки на не парність (рисунок 9.6). Налаштовувати перевірку на парність або непарність треба в опції Parity.

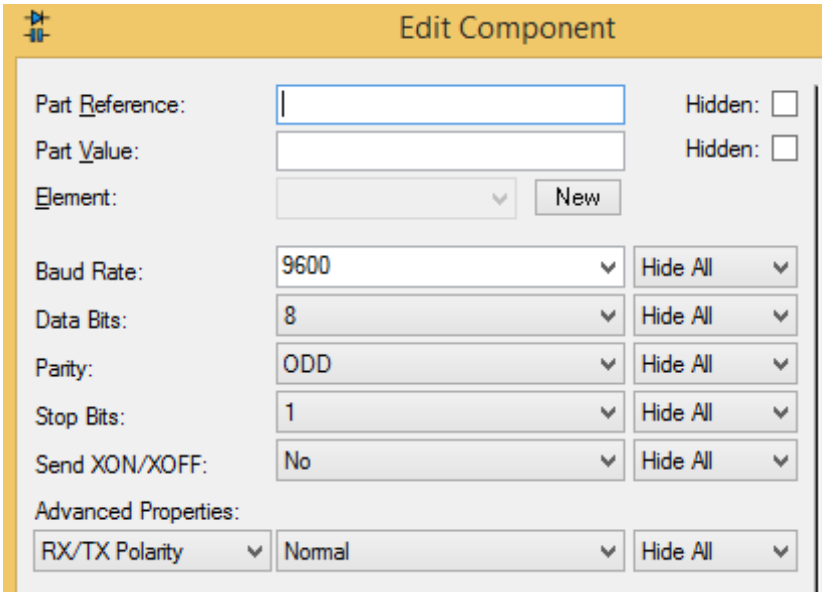


Рисунок 9.6 – Налаштування віртуального термінала

2. Натисканням Start VSM Debugging запускається процес моделювання (рисунок 9.7).

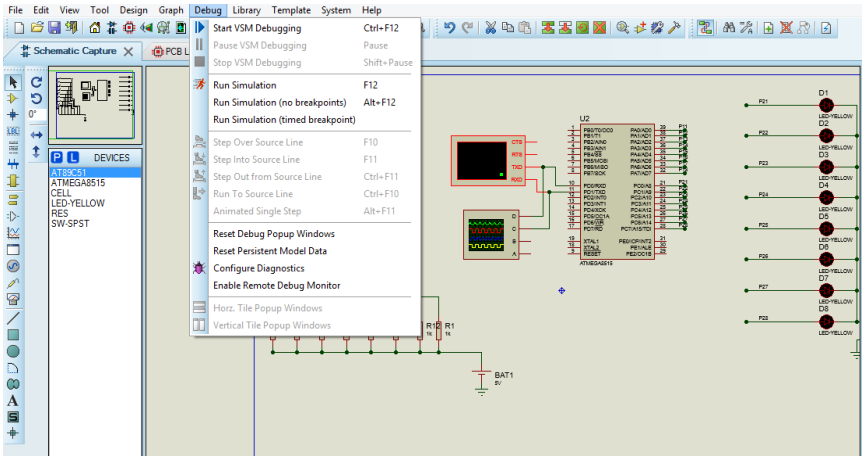


Рисунок 9.7 – Запуск процесу моделювання

3. Для встановлення двох точок зупинки при виконанні програми натискається кнопка Pause VSM Debugging.

4. У вкладці Source Code встановлюються дві точки зупинки. Для цього потрібно лівою кнопкою миші двічі клацнути з лівої сторони двох команд, як показано на рисунку 9.8.

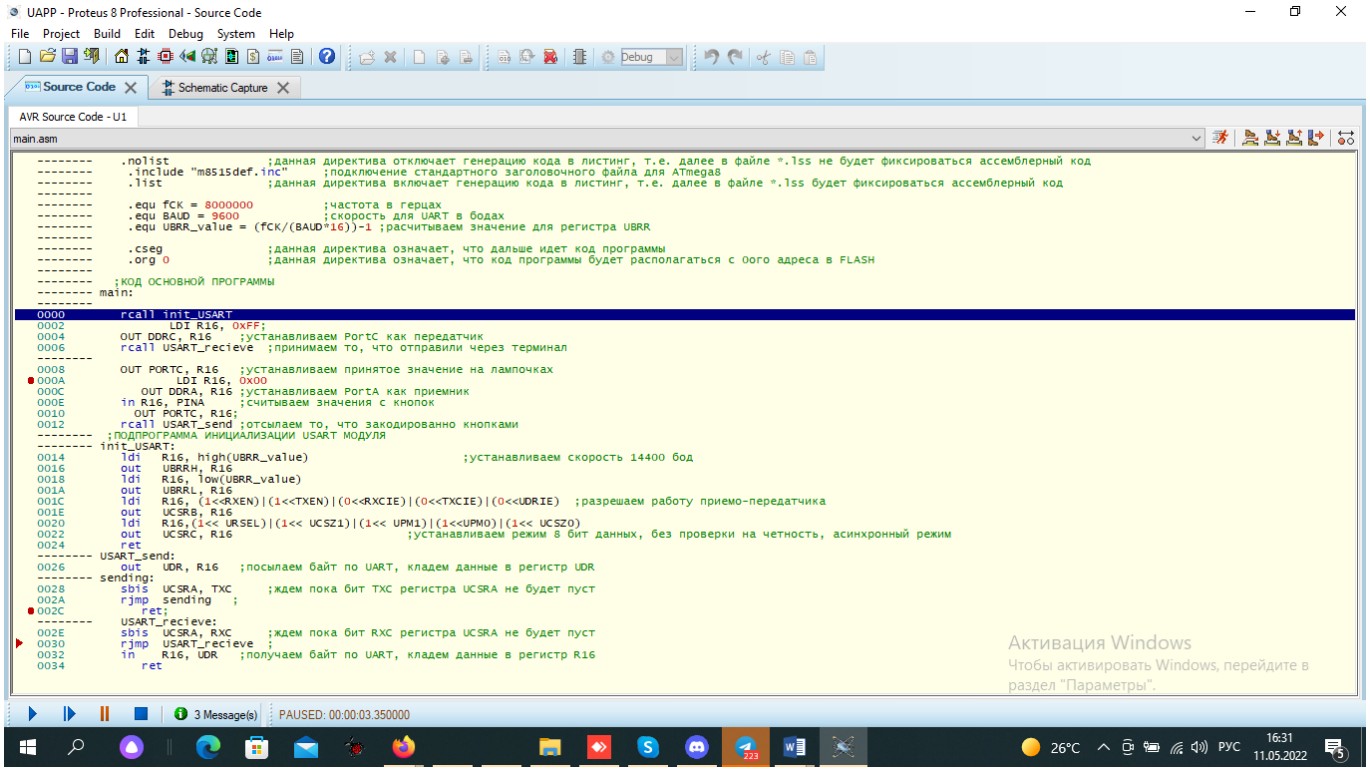


Рисунок 9.8 – Встановлення двох точок зупинки

5. Натискається Run Simulation (F12).

6. У вікні Virtual Terminal натисканням правої кнопки обираються: Hex Display Mode та Echo Typed Characters (рисунок 9.9).

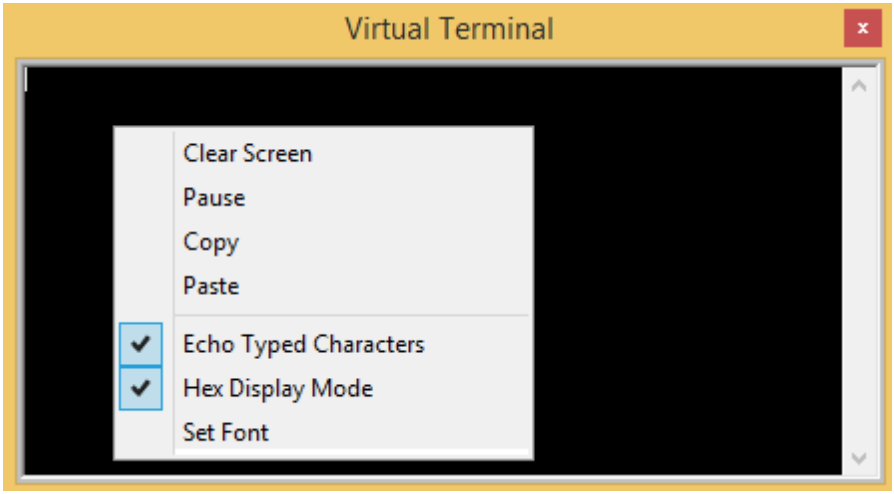


Рисунок 9.9 – Налаштування віртуального термінала

Якщо вікно Virtual Terminal закрито, то його можна відкрити із вкладки Debug.

7. Вводиться символ, який надсилається від віртуального терміналу до мікроконтролера через УАПП. При введенні символу у вікні терміналу повинен

стояти курсор. Символ, який вводиться з клавіатури, віртуальний термінал перетворює в ASCII-код згідно таблиці [4].

Наприклад, вводиться цифра 7, ASCII код якої 37 (hex) = 0011 0111 (bin). Програма перейде на рядок, де встановлено першу точку зупину.

У вкладці зі схемою Schematic Capture на світлодіодах та осцилографі відображається символ, який прийнято від термінала через виходи порту C (рисунок 9.10).

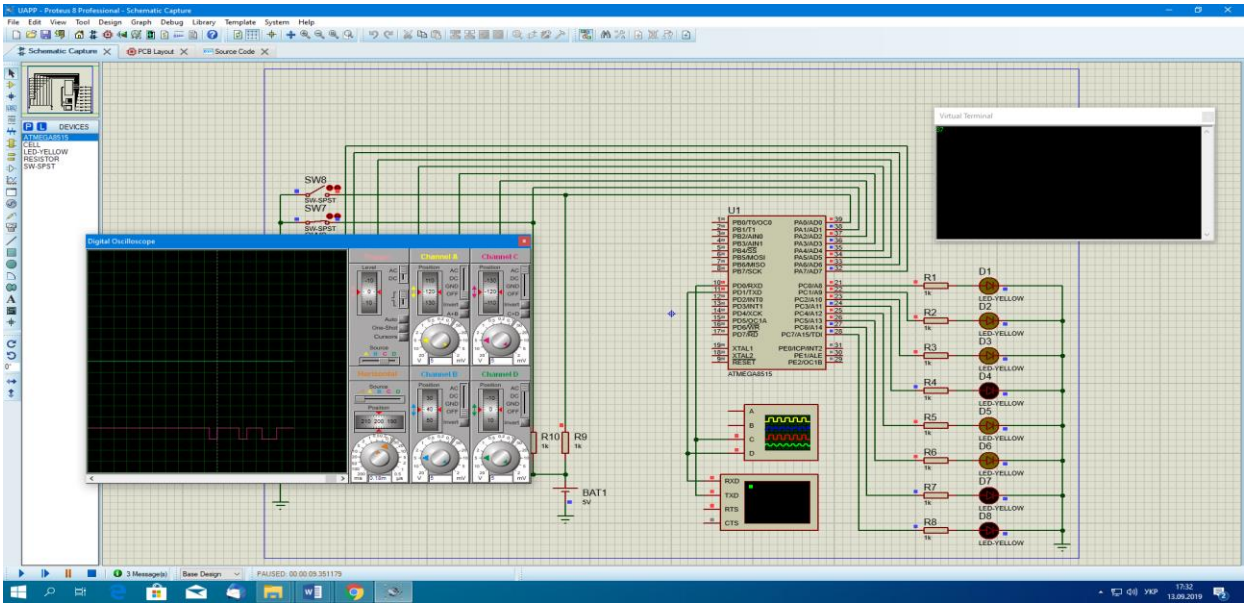


Рисунок 9.10 – Відображення на світлодіодах та осцилографі символу, який прийнято від термінала

Світлодіоди точно відображають бінарний код символу, який ми прийняли від термінала (якщо дивитися знизу уверх, а бінарне число читати зліва на право), нуль – світлодіод вимкнено, одиниця – світлодіод увімкнено (прийнято символ 00110111в = 37h). Нижній (червоний) луч осцилографа також вірно відображає прийнятий символ, який було передано від термінала в асинхронному старт-стопному режимі. Після останнього другого нульового біта перед одиничним стоп-бітом іде одиничний біт, тому що сума одиниць у символі непарна та згідно формулою 9.4: $P_{0DD} = 1$. Якщо ми від термінала передамо цифру 9, ASCII-код якої 39 (hex) = 00111001 (bin), то після другого нульового біта перед одиничним стоп-бітом іде третій нульовий біт, тому що сума одиниць у символі парна та згідно 9.4: $P_{0DD} = 0$.

8. Символ для передачі мікроконтролером через УАПП кодується кнопками: замкнена кнопка – нуль, а розімкнена – одиниця (рисунок 9.11).

Якщо дивитися на кнопки знизу уверх, то на рисунку 9.11 набрано число 1110 0111 (bin) = E7 (hex).

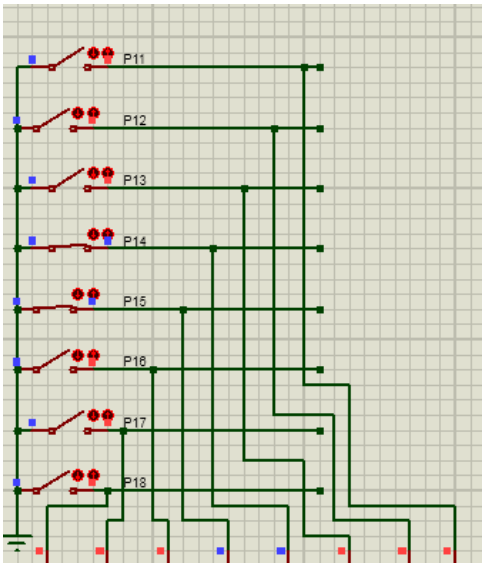


Рисунок 9.11 – Кодування кнопками символу для передачі

9. Щоб побачити переданий сигнал на осцилографі, потрібно натиснути кнопку на опції One-Shot (один кадр) (рисунок 9.12).

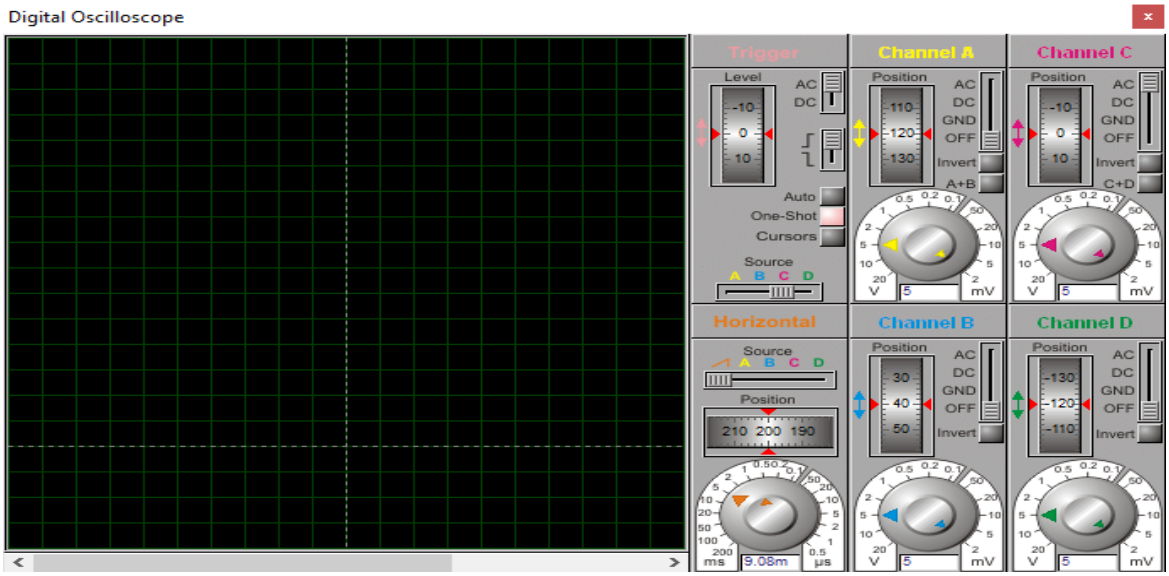


Рисунок 9.12 – Налаштування осцилографа

10. Для продовження виконання програми знову натискається Run Simulation Програма переходить до другої точки зупину.

11. Нижче пояснюються сигнали, що відображаються на осцилографі (рисунок 9.13).

Осцилограф відображає сигнали за двома промінями: верхній (зелений) – сигнал, який передається від кнопок, нижній (червоний) – сигнал, який приймається від терміналу: 37(hex).

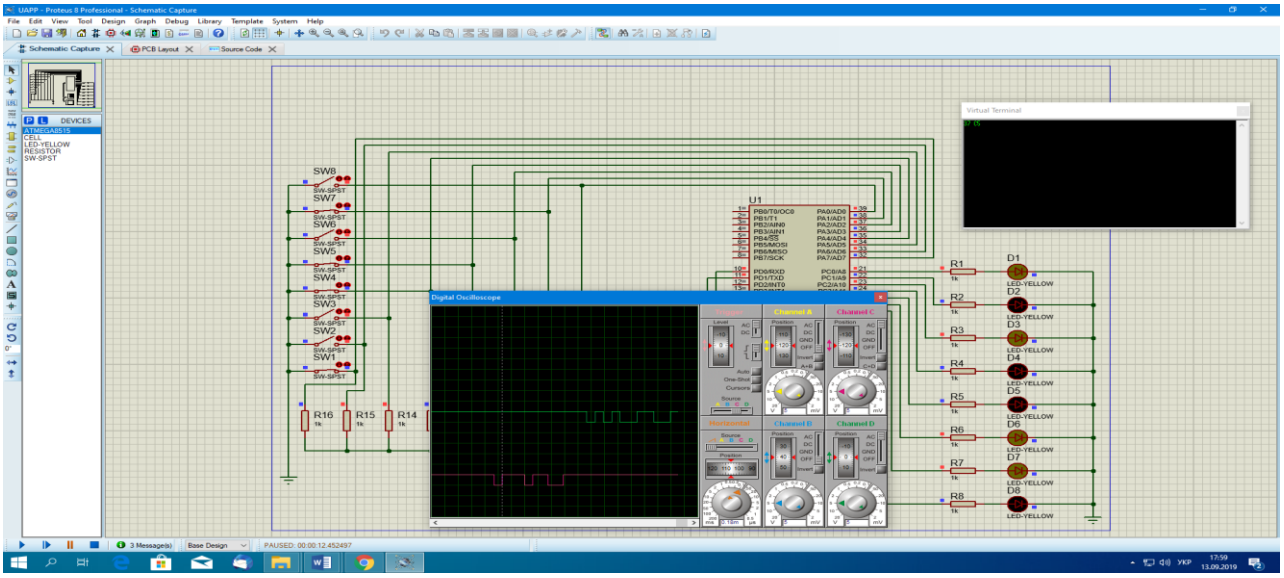


Рисунок 9.13 – Відображення сигналів на осцилографі: від терміналу – червоний; від МК (кнопок) – зелений

Розглянемо сигнал, який передається від МК (кнопок).

Перший перепад із високого рівня у низький – старт-біт.

Потім ідуть вісім інформаційних бітів, починаючи з молодшого розряду, які будуть відображені на віртуальному терміналі та осцилографі, як 01100101(bin) = 65 (hex). Далі іде 9-й біт, який дорівнює нулю, тому що число одиниць в байті, який було передано – парне (P_{0DD} = 0, формула 9.4). Останній 10-й біт, який дорівнює одиниці, це стоп-біт.

12. Для визначення отриманої швидкості передачі, яку в даному прикладі було обрано – 9600 біт/сек = 9600 пос/сек = 9600 бод, підбирається така розгортка сигналу на осцилографі, щоб сумістити бічні сторони імпульсу із вертикальними лініями сітки осцилографа (рисунок 9.14).

Як видно з рисунку, тривалість одного імпульсу дорівнює приблизно 0,104 мс. Якщо розділити один біт на цей час, та помножити на 1000, отримується задана швидкість:

$$V = \frac{1}{0.104} * 1000 = 9600 \frac{\text{біт}}{\text{сек}}.$$

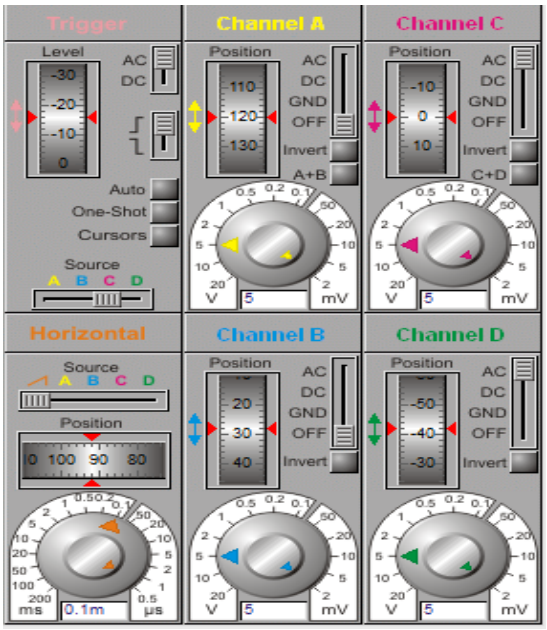


Рисунок 9.14 – Визначення отриманої швидкості передачі на осцилографі

Далі нижче розглянуто ще один приклад, коли на кнопках було набрано число 67(hex) = 01100111(bin) (рисунок 9.15).

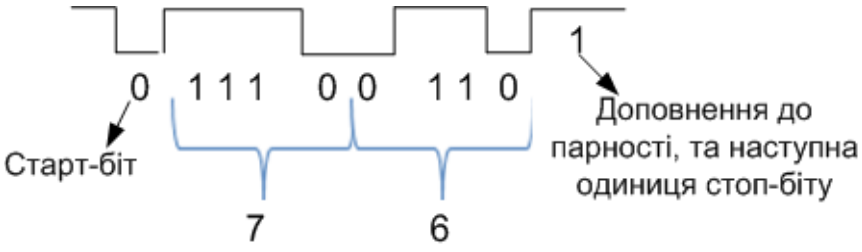


Рисунок 9.15 – Передача числа: 67 (hex) = 01100111 (bin)

На верхньому промені осцилографа відображено: перший нуль – старт-біт, потім три одиниці та один нуль – це число 7 на кнопках, далі йде нуль, дві одиниці та нуль – це число 6 на кнопках, потім біт, який відображає непарну кількість одиниць у повідомленні (доповнення до парності) – одиниця, та стоп-біт – одиниця (рисунок 9.16).

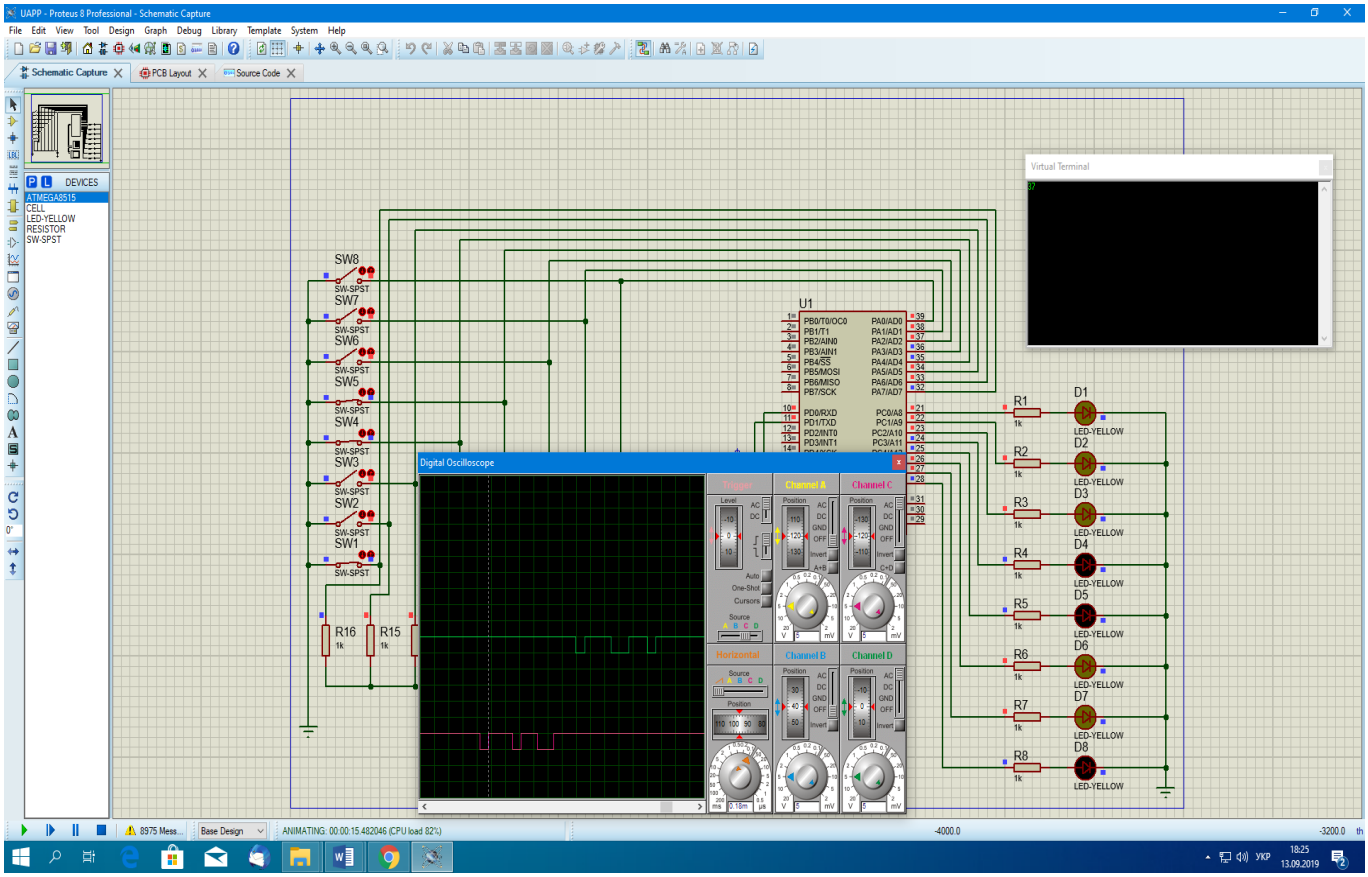


Рисунок 9.16 – Відображення сигналів на осцилографі: від терміналу – червоний, від МК (кнопок) – зелений

9.5 Схема алгоритму роботи моделі

На рисунку 9.17 наведено схему алгоритму роботи моделі.

Далі представлено лістинг робочої програми, яку розроблено згідно з цим алгоритмом. В дужках праворуч біля кожної команди цифрою відмічено номер блоку схеми алгоритму, який реалізує ця команда.

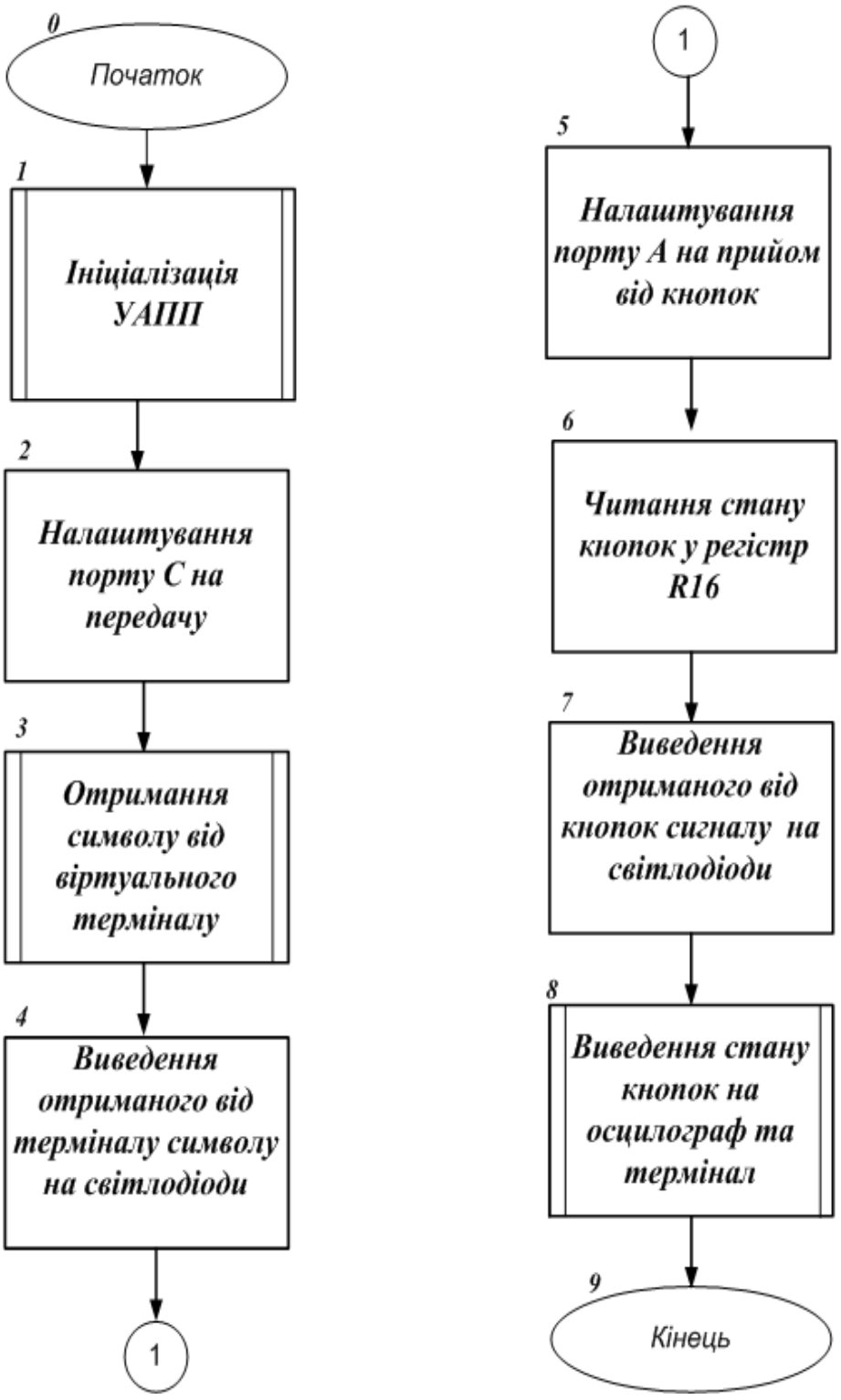


Рисунок 9.17 – Схема алгоритму роботи моделі

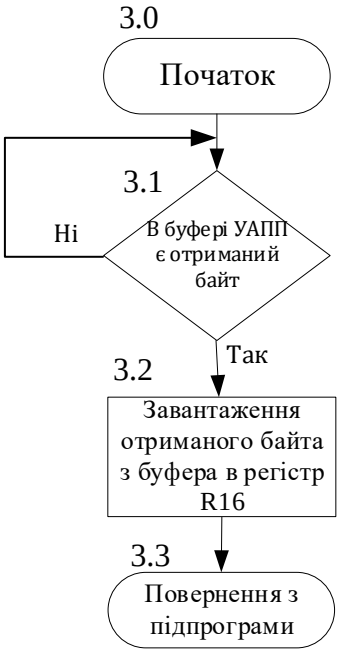


Схема алгоритму отримання символу від віртуального терміналу

Схема алгоритму ініціалізація модуля

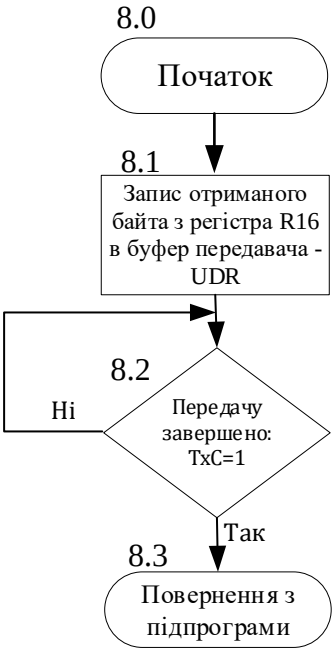


Схема алгоритму стану кнопок на осцилограф та термінал

Закінчення рисунку 9.17 (стор.255)

9.6 Робоча програма

```
.nolist                 ; директива відключає генерацію коду у лістинг,
        ;тобто далі у файлі *.lss не буде фіксуватися асемблерний код
.include "m8515def.inc" ;підключення стандартного заголовочного
                        ;файлу для ATmega8
.list                   ;директива включає генерацію коду у лістинг,
        ;тобто далі у файлі *.lss буде фіксуватися асемблерний код
.equ f_CK = 8000000     ;частота в герцах
.equ BAUD = 9600        ;швидкість для УСАПП у бодах
.equ UBRR_value = (f_CK/(BAUD*16))-1; розрахунок значення для регістра
                        ;UBRR
.cseg                   ;директива визначення, що далі іде код програми
.org 0                  ;директива визначення, що  код програми у FLASH
                        ; буде розміщено з нульової адреси
; КОД ОСНОВНОЇ ПРОГРАМИ
Start: ; (блок 0)
;ініціалізації з мітки init_USART
rcall init_USART        ; (блок 1) відносний виклик підпрограми
;налаштування порту С на передачу
LDI R16, 0xFF           ; (блок 2) R16← 0xFF
OUT DDRC, R16           ; (блок 2) DDRC← R16
; (блок 3) отримання того, що відправив віртуальний термінал
rcall USART_recieve     ; (блок 3) відносний виклик підпрограми з мітки
                        ; USART_recieve
; (блок 4) виведення отриманого від терміналу символу на світлодіоди
OUT PORTC, R16          ; (блок 4) PORTC←R16
```

```
; (блок 5) налаштування порту А на прийом від кнопок
LDI R16, 0x00          ; (блок 5) R16←0x00
OUT DDRA, R16          ; (блок 5) DDRA← R16
; (блок 6) читання стану кнопок у регістр R16
in R16, PINA           ; (блок 6) R16← PINA
; (блок 7) виведення отриманого від кнопок на світлодіоди
OUT PORTC, R16         ; (блок 7) PORTC←R16
; (блок 8)  виведення стану кнопок на осцилограф та термінал
; (підпрограма з міткою USART_send)
rcall USART_send       ; (блок 8) відносний виклик підпрограми з мітки
                       ; USART_ send
; (блок 1) ПІДПРОГРАМА ІНІЦІАЛІЗАЦІЇ МОДУЛЯ USART
init_USART:
; програмування швидкості обміну 9600бод
ldi   R16, high(UBRR_value)    ; (блок 1) R16←старший байт UBRR_value
out   UBRRH, R16               ; (блок 1) UBRRH←R16
ldi   R16, low(UBRR_value)     ; (блок 1) R16←молодший байт UBRR_value
out   UBRRL, R16               ; (блок 1) UBRRL←R16
; дозвіл роботи приймача-передавача модуля USART
ldi   R16, (1<<RXEN)|(1<<TXEN)|(0<<RXCIE)|(0<<TXCIE)|(0<<UDRIE)
out   UCSRB, R16               ; (блок 1) UCSRB←R16
; програмування передачі восьми біт з перевіркою на непарність,
; у асинхроному режимі, з одним стоп-бітом
ldi   R16, (1<< URSEL)|(1<<UPM1)|(1<<UPM0)|(1<< UCSZ1)|(1<< UCSZ0)
out   UCSRC, R16               ; (блок 1) UCSRC← R16
ret                            ; (блок 1) повернення з підпрограми ініціалізації
; передача через модуль USART
USART_send:
```

```
out   UDR, R16                    ; (блок 8) регістр даних UDR← R16
sending:
; чекання завершення передачі
sbis  UCSRA, TXC        ; (блок 8) якщо біт TXC = 1, то наступна ;команда
пропускається, інакше  –  наступна команда
rjmp  sending             ; (блок 8) безумовний перехід на мітку sending
ret                       ; (блок 8) повернення з підпрограми
USART_recieve:
sbis  UCSRA, RXC          ; (блок 3), якщо біт RXC = 1 (в буфері ПРМ є
; отриманий байт) , то пропускається наступна команда,
; інакше наступна команда
rjmp  USART_recieve    ; (блок 3) безумовний перехід на мітку
                          ;USART_recieve
in       R16, UDR         ; (блок 3) R16←UDR (завантаження в
                          ; регістр  R16 отриманого байта від термінала
ret                       ; (блок 3) повернення з підпрограми
END Start                 ; (блок 9) 7.7 Зміст звіту
```

9.7 Зміст звіту

Звіт по роботі повинен містити:

- схему моделі;
- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

Контрольні запитання та завдання

1) Що таке УСАПП (USART) та УАПП (UART)?
2) З яких трьох частин складається структура модуля УСАПП/УАПП? Які елементи містить кожна з них?

3) Які регістри використовуються для керування модулями УАПП та УСАПП?
4) Чим в асинхронному режимі задається швидкість прийому та передачі даних?
5) Поясніть структурну схему блока синхронізації модуля УСАПП для мікроконтролера Mega 128.
6) Які режими синхронізації підтримує УСАПП?
7) Як визначається швидкість обміну в модулі УСАПП?
8) Як визначається розмір слова даних у модулях УСАПП?
9) Як працює схема контролю парності?
10) Як проходить передача даних в модулях УСАПП?
11) Як проходить прийом даних в модулях УСАПП?
12) Що таке режим мультиконтролерного обміну?
13) Опишіть послідовність дій для здійснення обміну даними у мікроконтролерній мережі.
14) Як визначити швидкість передачі даних для асинхронного звичайного режиму?
15) Як визначити тривалість передачі кадру для асинхронного прискоренного режиму?
16) Як в моделі підключено кнопки, які задають байт для передачі?
17) Як в моделі підключено світлодіоди, які відображають передану в моделі інформацію?
18) Як в моделі визначається тривалість переданого біта?
19) Опишіть формат та призначення розрядів регістра UCSRC.
20) Як в програмі відбувається налаштування порту А на передачу?
21) Як в програмі визначається час завершення передачі та прийому чергового байта?
22) Дайти характеристику ASCII-коду.
23) Поясніть як в моделі підключено світлодіоди та в якому випадку вони будуть світитися?
24) Як працюють схеми відновлення тактового сигналу і даних при асинхронному прийомі?

10 ЛАБОРАТОРНА РОБОТА №7. ДОСЛІДЖЕННЯ МОДЕЛІ ГОДИННИКА РЕАЛЬНОГО ЧАСУ

Тема: Моделювання модуля годинника реального часу

Мета: Користуючись пакетом Proteus 8.6 дослідити моделювання годинника реального часу

10.1 Порядок виконання роботи

1) Створити модель пристрою в пакеті Proteus 8.6.
2) Розробити схему алгоритму роботи моделі та робочу програму.
3) Створити hex-файл та підключити його до мікроконтролера.
4) Запустити модель та виконати її дослідження згідно методичних вказівок.
5) Зробити відповідні висновки.

10.2 Загальна характеристика

Для розробки годинника реального часу треба використовувати таймери мікроконтролерів, наприклад ATmega64x і ATmega128x, які можуть працюватиу асинхронному режиму роботи (мають блок RTC). Таймер з таким блоком при відповідному налаштуванні викликає переривання з періодом, який дорівнює 1 секунді. На відміну від основного кварцу мікроконтролера, блок RTC використовує додатковий кварц, який має частоту 32768 Гц. Це дозволяє при діленні 32768/128/256 отримати рівно 1с. В моделях ATmega64x і ATmega128x восьмирозрядний таймер/лічильник T0 має блок RTC та може працювати в асинхронному режимі. В цьому режимі на вхід попереднього дільника надходить сигнал від кварцового генератора, що дозволяє використовувати таймер/лічильник як годинник реального часу. Задавачем частоти сигналу може бути як кварцовий резонатор, що підключається до виводів TOSC1 та TOSC2 мікроконтролера, так і сигнал від зовнішньої схеми, що подається на вивід TOSC1. Незважаючи на те що тактовий генератор таймера/лічильника налаштований на частоту 32768 Гц,

частота кварцового резонатора або сигналу від зовнішньої схеми може лежати в межах 0...256 кГц. При цьому вона повинна бути в чотири рази менше частоти тактового сигналу мікроконтролера.

10.3 Моделювання годинника реального часу

10.3.1 Опис моделі

Робочу модель годинника реального часу у пакеті PROTEUS 8.6 показано на рисунку 10.1.

Основним компонентом моделі є мікроконтролер ATmega128 (U1). Мікроконтролер виконує всі обчислення і керування зовнішніми пристроями. Як задаючий елемент для відліку часу використовується годинниковий кварцовий резонатор із частотою 32768 Гц (X1). Кварц підключено до виводів TOSC1, TOSC2 мікроконтролера. У мікроконтролері сигнал з цих виводів проходить блок генератора, передподільника частоти та надходить у таймер/лічильник 0, де використовується в якості опорної частоти. Крім годинникового кварцу в схемі є основний кварцовий резонатор (X2), з частотою 4МГц. Сигнал з цього кварцу використовується для тактування ядра мікроконтролера та його периферії.

Для індикації значення годин, хвилин і секунд застосовано семисегментний світлодіодний індикатор на 6 розрядів, що має спільний катод. Індикатор відображає час по 2 розряди: 1 і 2 – години, 3 і 4 – хвилини, 5 і 6 – секунди, якщо рахувати зліва, між якими ставиться крапка, наприклад: "12.45.17".

Для управління індикатором потрібно подавати керуючі сигнали на лінії включення сегментів (A…G, DP) і обирати активний розряд за допомогою ліній 1…6. Оскільки виходи мікроконтролера мають не достатню здатність навантаження за струмом, перед індикатором використовуються дві додаткові мікросхеми: TC4468 (U2, U3), які виконують функцію буфера-защіпки з трьома станами. Ці мікросхеми мають 4 входи і 4 виходи, логічні стани яких ідентичні входам, але здатні віддавати у навантаження струм до 35 мА.

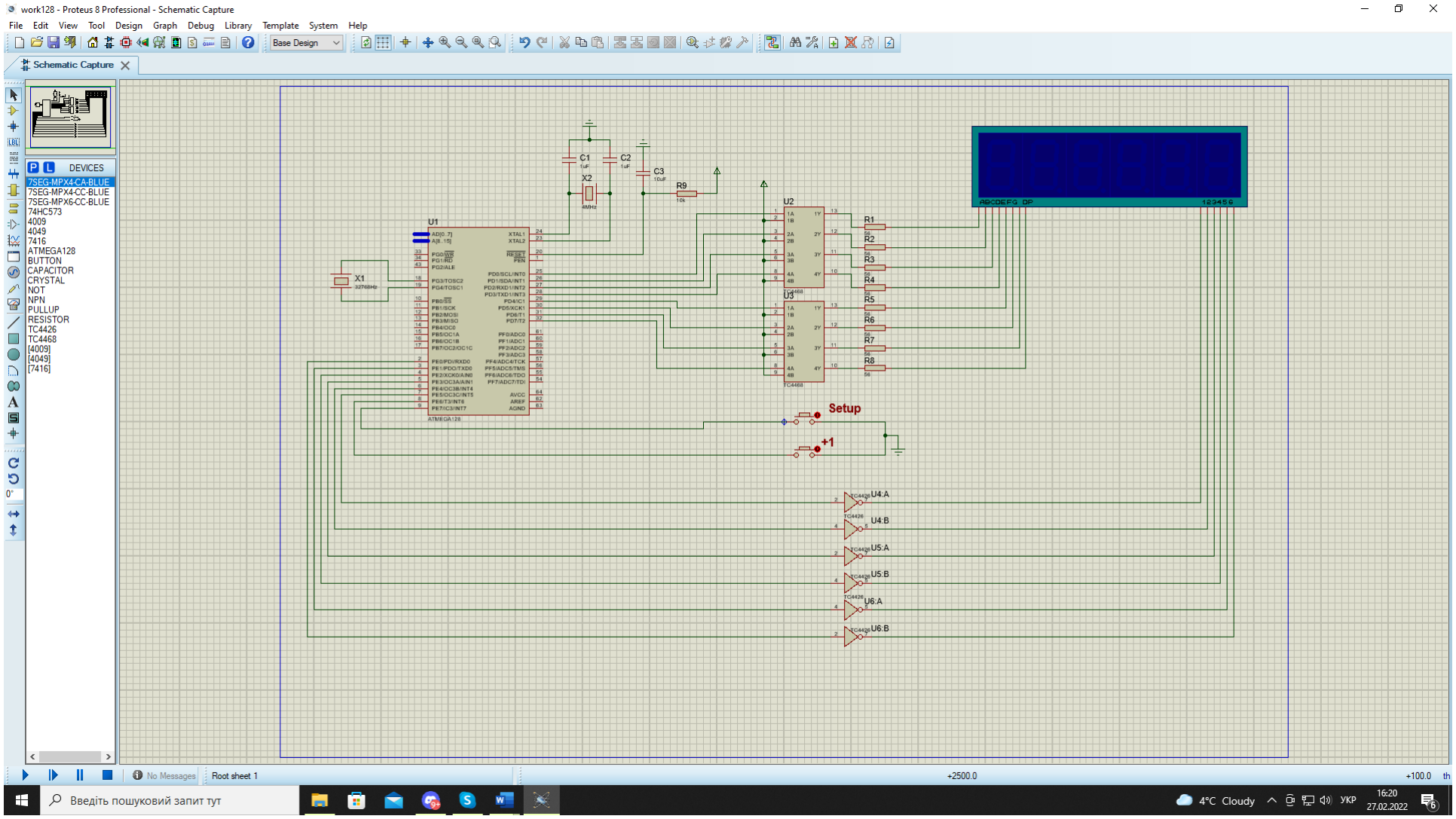


Рисунок 10.1 – Робоча модель годинника реального часу

Крім цих мікросхем для керування індикатором використовується також мікросхеми TC4426 (U4…U6), які виконують функцію буфера з відкритим стоком, який інвертує. Цей буфер здатний віддавати струм у навантаження до 1,5А, коли його вихід знаходиться в стані логічного нуля.

Наявність інвертора дозволяє вибирати потрібний розряд на індикаторі за допомогою логічної одиниці на відповідному виході мікроконтролера. Для обмеження струму через світлодіодний індикатор використовуються резистори R1…R8.

Для виведення цифр в програмі застосовано метод динамічної індикації. Кожна цифра виводиться окремо одна за одною. В певний момент часу на індикаторі горить лише одна цифра, але за рахунок високої швидкості виведення для ока цей процес не помітний, і все виглядає так, ніби все цифри горять одночасно.

Для виведення бітового образу числа на відповідний розряд індикатора використовується порт D, а для вибору активного розряду індикатора – шість молодших розрядів порту E мікроконтролера.

Старші два біти порту E використовуються як входи зовнішніх переривань INT6 і INT7. До цих входів підключено кнопки: +1 і Setup. Кнопка Setup виконує циклічне переключення режимів (робочий режим -> налаштування годин -> налаштування хвилин -> налаштування секунд -> робочий режим). Кнопка +1 працює тільки в режимах налаштування і виконує інкремент значення часу (в залежності від режиму налаштування – годин, хвилин або секунд). При натисканні та утриманні цієї кнопки відбувається автоматичний інкремент відповідного значення.Для живлення схеми в реальному пристрої може використовуватися стабілізатор напруги LM7805, з вихідним напругою 5В і максимальним струмом в 1А. Для додаткової стабілізації напруги застосовуться конденсатори на вході і виході стабілізатора. Для фільтрації високочастотних завад використовується конденсатор C5 ємністю 100нФ, розміщений в безпосередній близькості від мікроконтролера.

Для автоматичного формування сигналу RESET в реальному пристрої на однойменному вході мікроконтролера при включенні живлення застосовується RC-ланцюжок (C3, R9).

Роботу годинника засновано на використанні RTC-блоку 8-розрядного таймера T0 мікроконтролера. Цей блок при відповідному налаштуванні викликає переривання з періодом, який дорівнює 1 секунді. На відміну від основного кварцу мікроконтролера, блок RTC використовує додатковий кварц, який має частоту 32768 Гц. Це дозволяє при діленні 32768/128/256 отримати рівно 1с. Для керування роботою годинника використовуються дві кнопки, контроль стану яких виконується за допомогою зовнішніх переривань INT6 та INT7.

Мікросхема TC4468

Ця мікросхема є чотирьохканальним буфером-защіпкою з третім станом. Завдяки наявності цього стану і високої здатності навантаження виходів, дана мікросхема підходить для використання в якості шинного формувача.

Регістр-защіпка керується за допомогою входу 1B/2B/3B/4B (активація засувки). Коли на цьому вході присутня логічна одиниця, значення на виходах Y будуть повторювати значення з входів A. Якщо на вході 1B/2B/3B/4B логічний нуль, то виходи будуть зберігати своє значення до тих пір поки на цьому вході знову не з`явиться логічна одиниця.

Мікросхема TC4426

Дана мікросхема містить два інвертуючих буфери з виходами типу "відкритий сток". У пристрої є три мікросхеми, які виконують функцію шести незалежних інверторів. Кожен буфер здатний віддавати струм у навантаження до 1,5А, коли його вихід знаходиться в стані логічного нуля. Наявність інвертора дозволяє вибирати потрібний розряд на індикаторі за допомогою логічної одиниці на відповідному виході мікроконтролера.

10.3.2 Схема алгоритму роботи моделі

Схему алгоритму роботи моделі годинника наведено на рисунку 10.2.

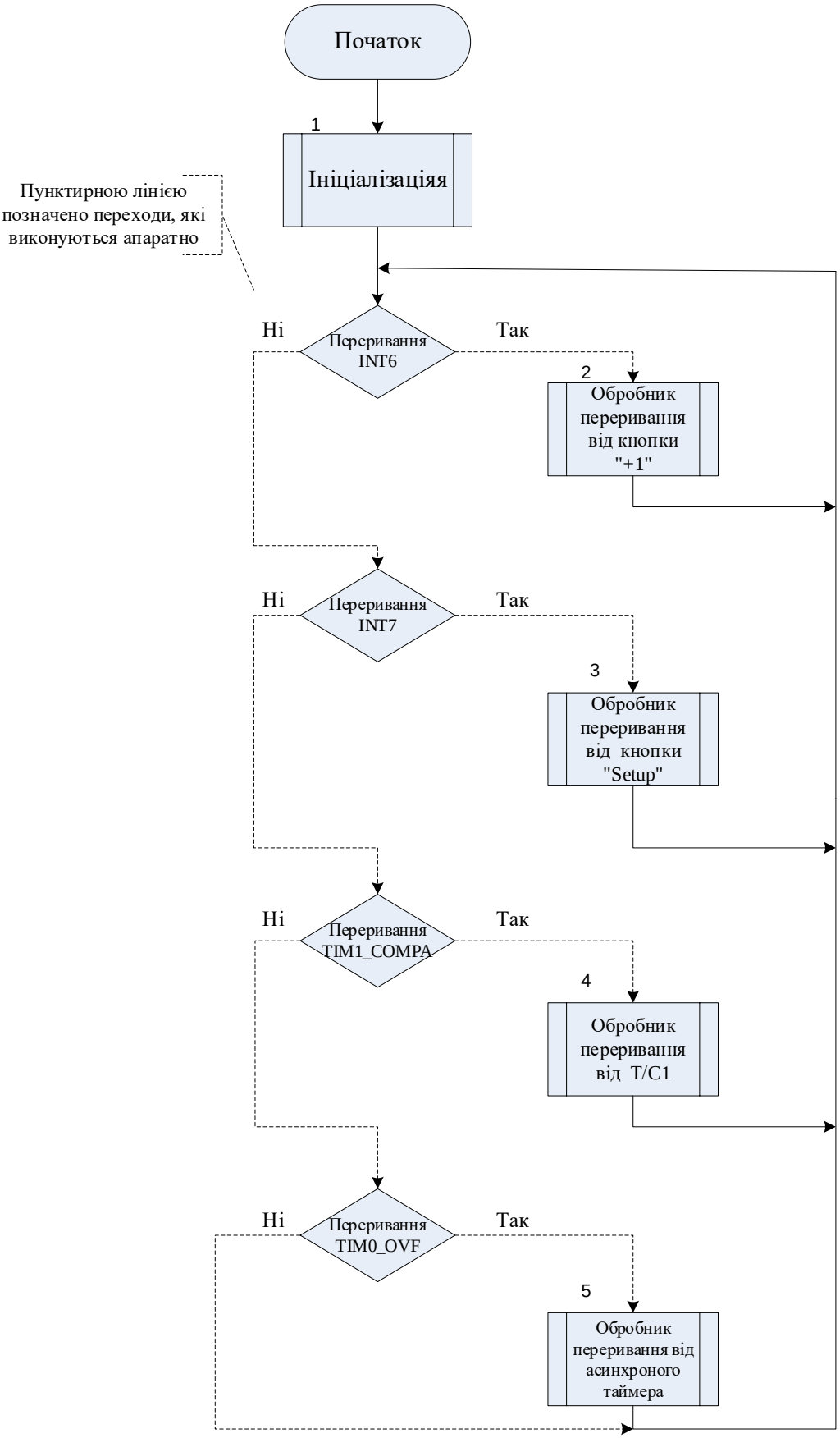
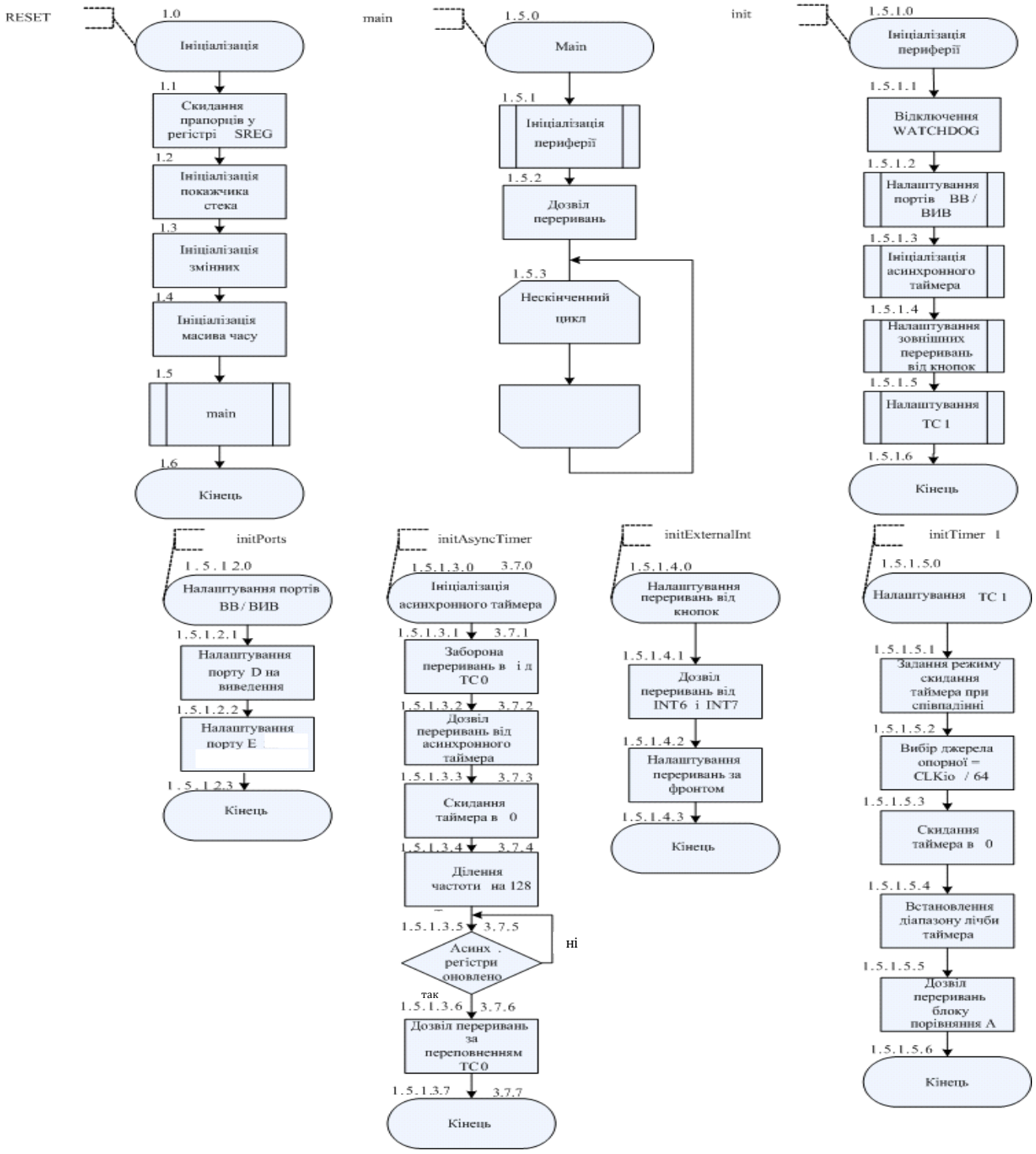
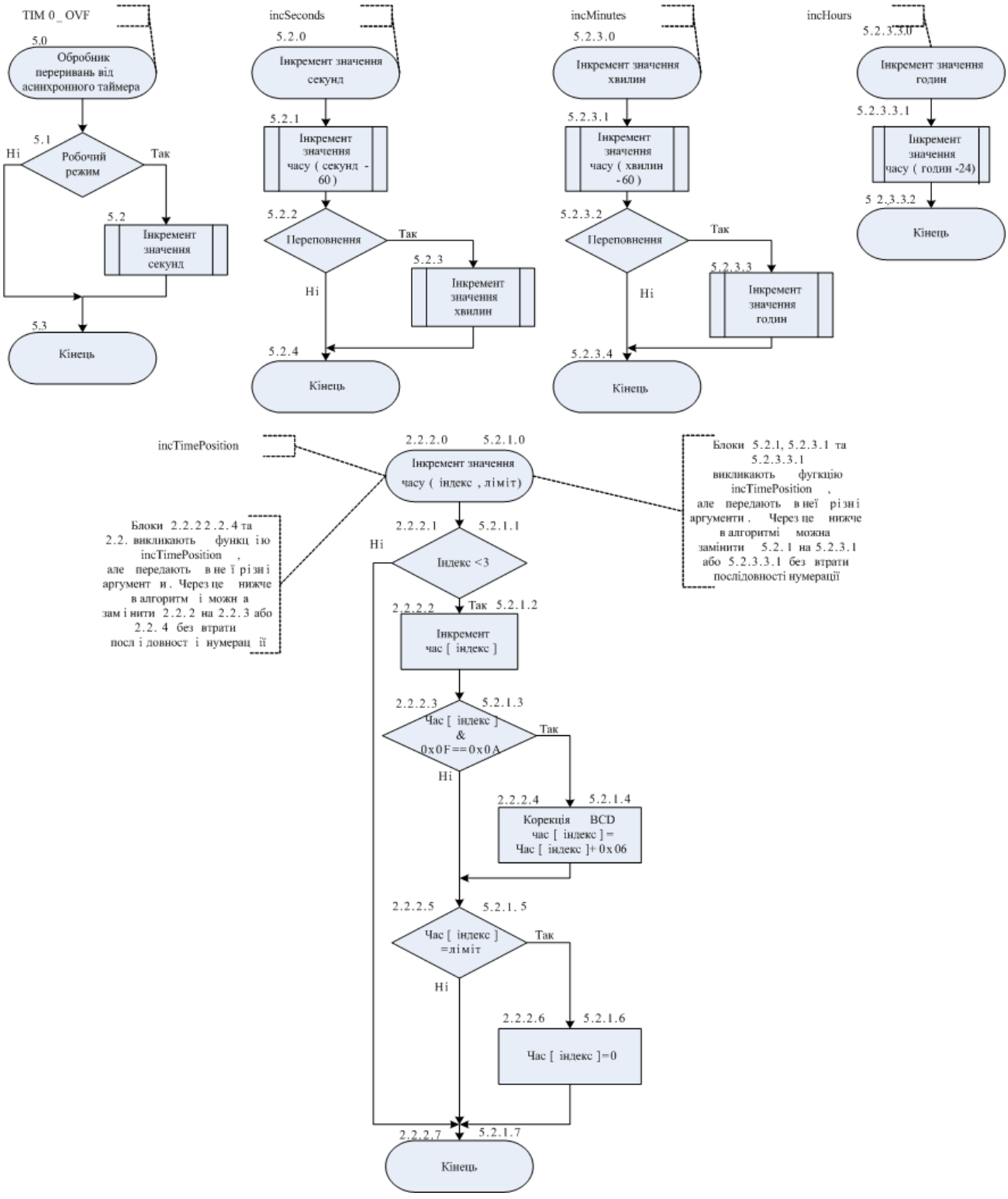


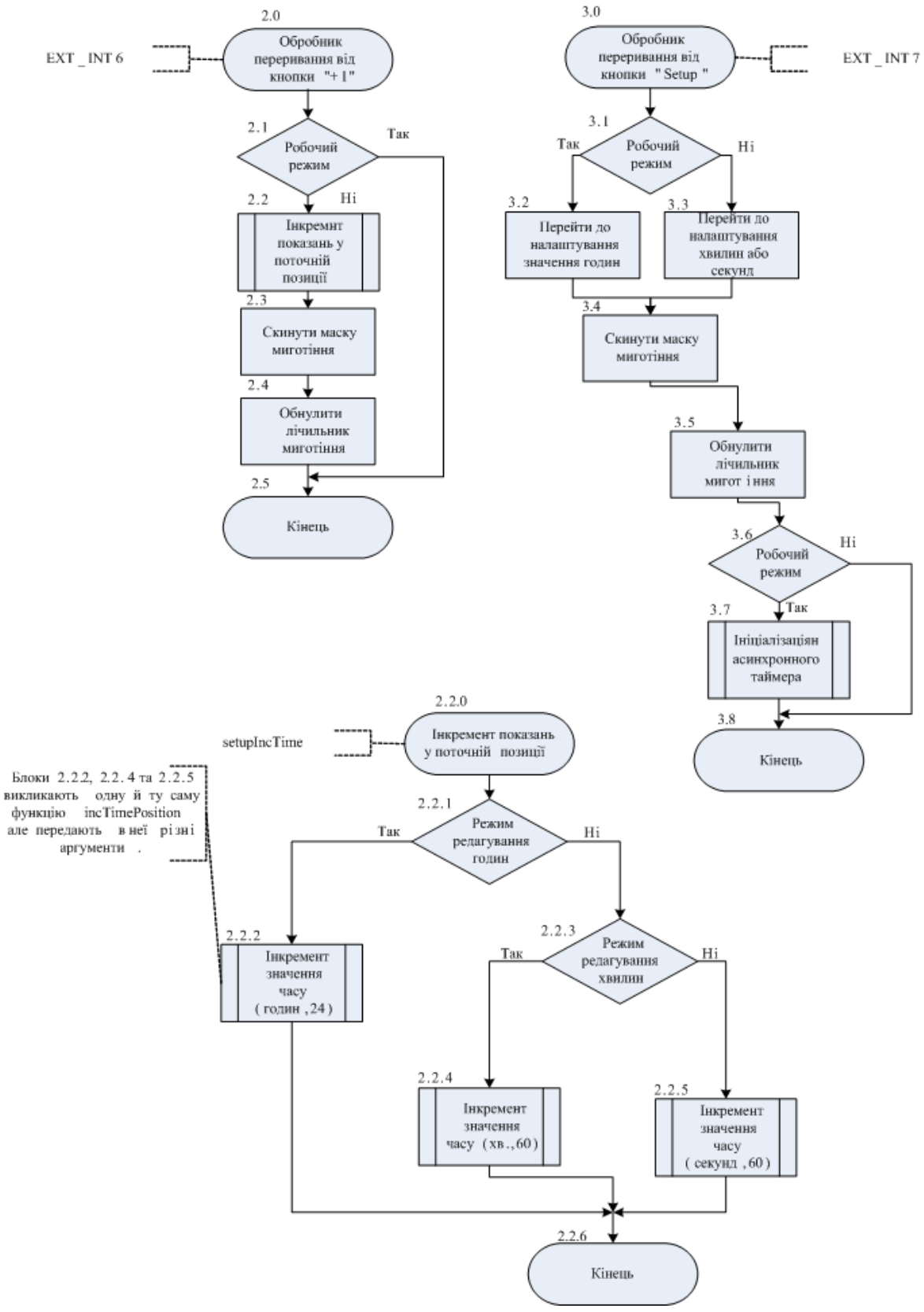
Рисунок 10.2 – Схема алгоритму роботи моделі



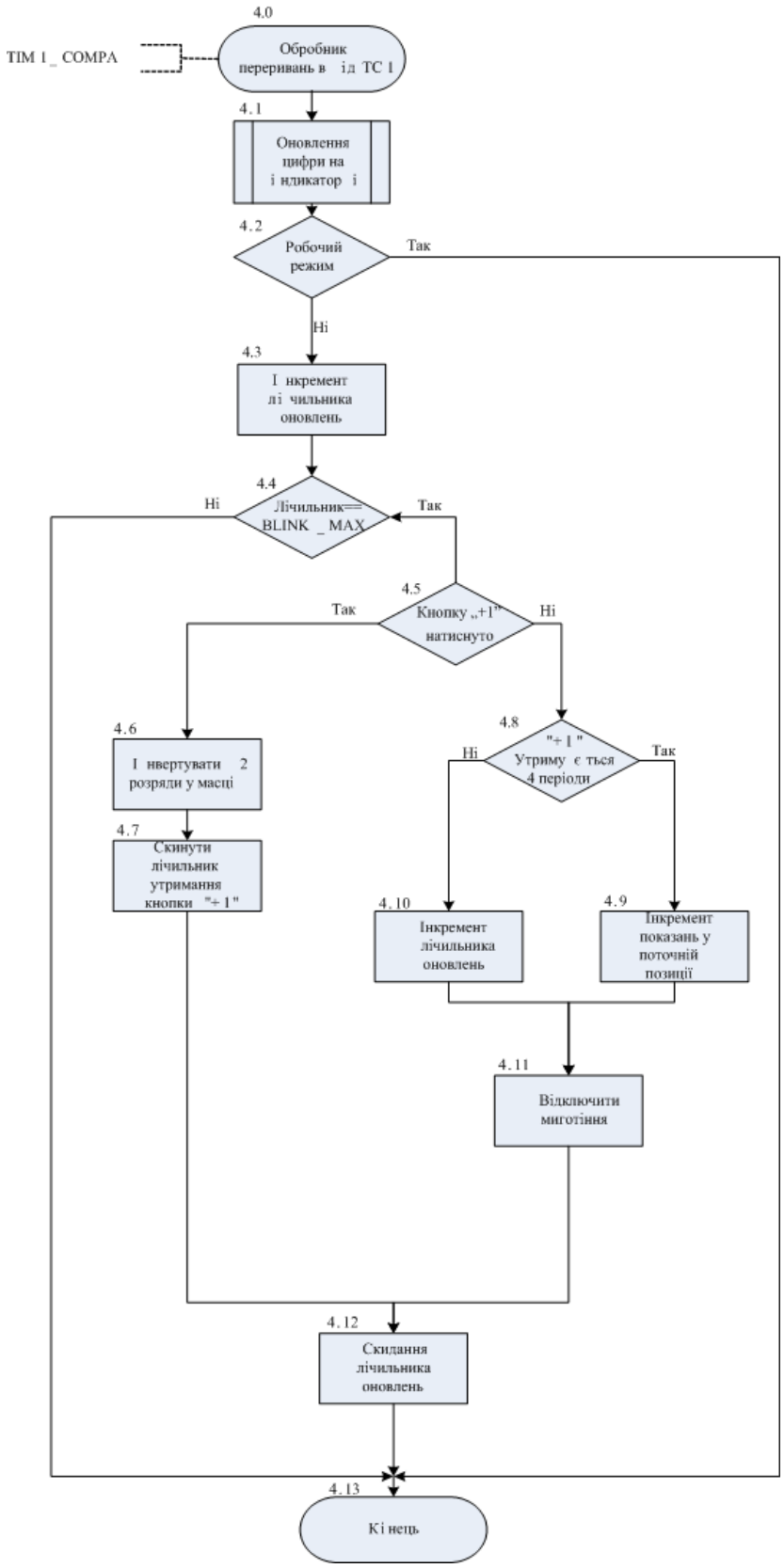
Продовження рисунка 10.2 (стор. 266)



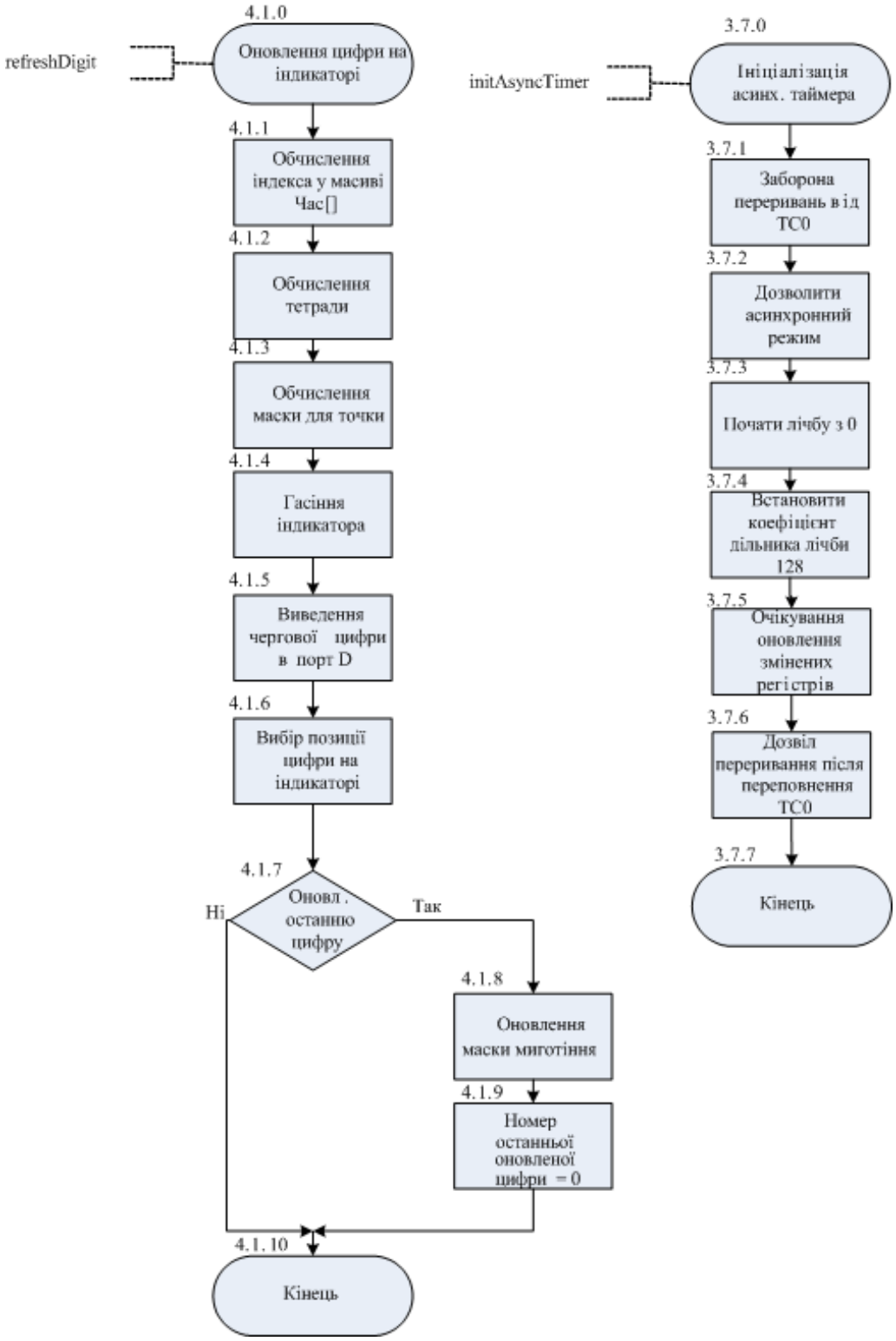
Продовження рисунка 10.2 (стор. 266, 267)



Продовження рисунка 10.2 (стор. 266 …268)



Продовження рисунка 8.2 (стор. 266…269)



Закінчення рисунка 8.2 (стор. 266…270)

10.3.3 Робоча програма

Початок роботи програми

Робота програми починається одразу після скидання мікроконтролера. Обробник переривання скидання за адресою 0x0000 пам'яті програм передає керування процедурі ініціалізації – RESET (блок 1.0 схеми алгоритму). Вона скидає вміст регістра прапорців, заповнює покажчик стека адресою кінця СПД, ініціалізує всі змінні їх початковими значеннями, використовуючи оголошені за допомогою .EQU константи. Крім того, виконується заповнення масиву LED_CODE, бітових образів цифр, призначених для виведення на індикатор. Після завершення ініціалізації керування передається процедурі main (блок 1.5.0 схеми алгоритму). Ця процедура виконує ініціалізацію всіх периферійних модулів, які задіяні в роботі, викликаючи процедуру init (блок 1.5.1.0 схеми алгоритму). Init виконує виклики процедур відповідальних за ініціалізацію портів введення/виведення (initPorts – блок 1.5.1.2.0 схеми алгоритму), асинхронного таймера (initAsyncTimer – блок 1.5.1.3.0 схеми алгоритму), зовнішніх переривань (initExternalInt – блок 1.5.1.4.0 схеми алгоритму), таймера-лічильника 1 (initTimer1 – блок 1.5.1.5.0 схеми алгоритму).

Ініціалізація портів введення-виведення

Порт D налаштовується на виведення, початкове значення 0xFF. Порт E –на виведення, крім двох старших біт, які налаштовуються на введення.

Ініціалізація асинхронного таймера

Таймер-лічильник 0 програмується в асинхронний режим. Початкове значення лічильника таймера – нуль. Дільник частоти налаштовується на коефіцієнт 128, що з урахуванням максимального значення лічильника 255 і частоти кварцу 32768, дає в результаті період в 1 с. Після налаштування дозволяються переривання від таймера-лічильника 0.

Ініціалізація зовнішніх переривань

В підпрограмі initExternalInt дозвіляються зовнішні переривання INT6, INT7 та налаштовується режим спрацювання за фронтом.

Ініціалізація таймера/лічильника 1

Для таймера/лічильника 1 (T/C1) програмується режим скидання за збігом. В якості опорної використовується робоча частота підсистеми введення-виведення мікроконтролера, яка поділено на 64. Початкове значення лічильника таймера встановлюється в нуль, а максимальне налаштовується таким чином, щоб забезпечити виклик переривання 600 раз в секунду, що при 6 розрядах дає швидкість відновлення індикатора в 100 гц. Завершується налаштування дозволом відповідного переривання.

Процедура init закінчується встановленням прапорця глобального дозволу переривань та переходом програми в нескінченний цикл.

Годинник знаходиться в режимі налаштування значення годин, про що свідчать миготливі розряди годин.

З цього моменту виконання будь-якої процедури викликається лише за перериваннями.

У програмі передбачені обробники наступних переривань:

– EXT_INT6 (блок 2 схеми алгоритму) – обробник натискання на кнопку +1 (викликається при її відпусканні). Цей обробник перевіряє роботу програми. Якщо це не режим налаштування, подальші дії не виконуються і відбувається вихід з обробника. Якщо годинник знаходиться в одному з режимів налаштування, то викликається процедура setupIncTime (блок 2.2.0 схеми алгоритму). Ця процедура перевіряє режим налаштування і в залежності від нього інкрементує відповідне значення (годин, хвилин або секунд). На завершення обробник скидає маску миготіння, що викликає негайне відображення "миготливих" цифр на індикаторі. Це означає, що якщо миготливі цифри в даний момент миготіння були невидимі, то не чекаючи кінця циклу миготіння вони стануть видимі;
– EXT_INT7 (блок 3 схеми алгоритму) – обробник натискання на кнопку Setup (викликається при її відпусканні). Виконує циклічну зміну режимів: робочий режим -> налаштування годин -> налаштування хвилин ->

налаштування секунд -> робочий режим. Виконується скидання маски миготіння, щоб при переході від налаштування одного розряду часу до налаштування іншого, попередній миготливий розряд не залишився невидимим. На завершення процедура перевіряє чи відбувся перехід в робочий режим, і, якщо це так, то виконується повторна ініціалізація асинхронного таймера. Це виконується для того щоб перша секунда після переходу в робочий режим була повною, інакше можливо що перехід в робочий режим відбудеться перед переповненням таймера і перша секунда буде короткою. Такий підхід дозволяє запускати відлік часу при переході в робочий режим з 00 секунд;

– TIM1_COMPA (блок 4 схеми алгоритму) – обробник переривання за переповненням таймера/лічильника 1 (канал порівняння А). Обробник починає свою роботу з виклику процедури refreshDigit (блок 4.1 схеми алгоритму), що виконує оновлення чергової цифри на індикаторі. Далі виконується інкремент і перевірка на досягнення максимального значення лічильника миготіння. Цей лічильник призначено для того щоб виконувати миготіння певним розрядом індикатора. Максимальне значення лічильника розраховано для досягнення прийнятної частоти миготіння, кратною частоті оновлення. При переповненні лічильника миготіння виконується інверсія бітів в масці миготіння відповідно до режиму роботи. Для прикладу, в режимі налаштування секунд в масці будуть циклічно інвертуватися нульовий та перший біти, приймаючи значення 00 і 11. У процедурі оновлення індикатора refreshDigit (блок 4.1.0 схеми алгоритму) виконується операція AND маски миготіння і маски вибору розряду для оновлення. Це дозволяє відключати оновлення певних розрядів, роблячи їх невидимими половину циклу миготіння, і видимими іншу половину циклу. Наприклад, маска вибору розряду для оновлення розраховується циклічним зсувом одиниці на n позицій вліво, і приймає значення від 00000001b до 00100000b для вибору від першого до шостого розряду відповідно. Наприклад, для налаштування секунд (маска миготіння 11111100b – розряд секунд

приховується) виконується операція: 00000001b AND 11111100b = 00000000b. Аналогічно для маски вибору розряду 00000010b, та на інші маски вибору розрядів, маска миготіння не впливає. Якщо обробник виявляє переповнення лічильника миготіння він формує нову маску миготіння, і зберігає її у змінній newBlinkMask, яка використовується в процедурі refreshDigit для виконання описаної вище операції. Маска не змінюється безпосередньо в обробнику переривання, оскільки це може викликати несинхронне миготіння розрядів на індикаторі, коли один з розрядів використовує стару маску і його бачимо, а інший нову і невидимий, та навпаки. Маска оновлюється тільки після оновлення всього вмісту індикатора. Крім операцій з маскою обробник також виконує облік часу утримання кнопки +1 і при перевищенні певного ліміту виконує автоматичний інкремент часу;

– TIM0_OVF (блок 5 схеми алгоритму) – обробник переривання за переповненням асинхронного таймера. Виконує перевірку чи знаходиться годинник в робочому режимі роботи. Якщо це так, виконує інкремент часу на одну секунду. Це робиться шляхом виклику процедури інкременту секунд (incSeconds блок 5.2.0 схеми алгоритму), яка при переповненні значення секунд (60) викличе процедуру інкременту хвилин (incMinutes блок 5.2.3.0 схеми алгоритму), яка в свою чергу при переповненні (60) викличе інкремент годин (incHours блок 5.2.3.3 схеми алгоритму). Всі ці процедури використовують процедуру incTimePosition (блок 2.2.2.0 схеми алгоритму), передаючи в неї відповідне значення ліміту і індекс в масиві значення часу, за який необхідно провести інкремент.

Лістінг робочої програми

Нижче наведено робочу програму мовою С.

```
#include <avr/io.h>                          //1.0

                                             //1.1
#include <avr/interrupt.h>

#define TRUE       1
                                                        //1.2
#define FALSE      0

/* Режими роботи годинника. */
#define MODE_WORK        0
#define MODE_EDIT_SECS   1
#define MODE_EDIT_MINS   2
#define MODE_EDIT_HOURS  3

/* Ліміти для годин, хвилин і секунд у BCD-коді*/
#define LIMIT_HOURS      0x24
#define LIMIT_MINUTES    0x60
#define LIMIT_SECONDS    0x60

/* Позиції у масиві, де зберігаються відповідні компоненти поточного часу */
#define I_SECONDS   0
#define I_MINUTES   1
#define I_HOURS     2
/* Кількість цифр на індикаторі*/
#define N_DIGITS    6
/*
* Частота процесора
* UL - беззнаковий Long [0, +4 294 967 295]
*/
#define F_CPU (4000000UL)

/* кількість зсувів праворуч для переміщення старшої тетради у молодшу */
#define HI_NIBBLE_SHIFT      0x04

/* Маска для виділення молодшої тетради */
#define LO_NIBBLE_MASK       0x0F    //0b00001111 == 0x0F

/* Константа для виконання корекції BCD-числа при досягенні межі 0xXA*/
#define BCD_CORRECTION       0x06
```

```
/* Маска для виділення стану вивода, до якого підключено кнопку +1*/
#define PLUS_BUTTON_MASK      0x40

/* Затримка для утримання кнопки +1 перед початком автоінкременту*/
#define AUTOINC_DELAY         2

/* Маска для ініціалізації порту E */
#define PE_INIT_MASK          0x3F    //0b00111111 == 0x3F
/*
* Маска для гасіння індикатора, враховано два старших біта,
* біти що використовуються для включения резисторів, що підтягують,  на входах PE6 і PE7
*/
#define INDICATOR_DESELECT_MASK    0xC0    // 0b11000000 == 0xC0

/* Частота оновлення індикатора у Гц. */
#define    REFRESH_RATE    100

/* Коефіцієнт передподільника T/C1. */
#define    PRESCALER       64

/* Сегменти індикатора. */
#define A        1     // 0b00000001 == 0x01
#define B        2     // 0b00000010 == 0x02
#define C        4     // 0b00000100 == 0x04
#define D        8     // 0b00001000 == 0x08
#define E        16    // 0b00010000 == 0x10
#define F        32    // 0b00100000 == 0x20
#define G        64    // 0b01000000 == 0x40
#define DP       128   // 0b10000000 == 0x80

/* Таблиця перекодування у "семисегментний" код. */
const unsigned char LED_CODE[] = {                                          //1.3
   A|B|C|D|E|F,    // 0x3F -> '0'
   B|C,            // 0x06 -> '1'
   A|B|D|E|G,      // 0x5B -> '2'
   A|B|C|D|G,      // 0x4F -> '3'
   B|C|F|G,        // 0x66 -> '4'
   A|C|D|F|G,      // 0x6D -> '5'
   A|C|D|E|F|G,    // 0x7D -> '6'
   A|B|C,          // 0x07 -> '7'
   A|B|C|D|E|F|G,  // 0x7F -> '8'
   A|B|C|D|F|G     // 0x6F -> '9'
```

```
};

/* Граничне значення T/C1, за якого індикатор (повністю) оновлюється:
REFRESH_RATE разів у секунду. */
const unsigned int TIMER_MAX = F_CPU / PRESCALER / REFRESH_RATE /
N_DIGITS;

/*
 * Число оновлень індикатора, после якого відбувається оновлення маски
 * миготіння, простіше – період миготіння.
 */
const unsigned int BLINK_MAX = 150;

/* Лічильник оновлень. */
unsigned char updateCounter = 0;                                    //1.4

/* Маска миготіння. */
unsigned char blinkMask = 0xFF;

/* Нова маска миготіння. */
unsigned char newBlinkMask = 0xFF;

/* Масив для зберігання поточного часу у BCD-коді. */
unsigned char time[3] = {
    0,    // секунди
    0,     // хвилини
    0    // години
};

/* Остання оновлена цифра. */
unsigned char lastRefreshedDigit = 0;

/* Режим. */
unsigned char mode = MODE_EDIT_HOURS;

/* Лічильник утримання кнопки. */
unsigned char holdCounter = 0;

/*
 * Функція інкремента заданого значения у масиві.
 *     index – індекс у масиві часу
 *     limit – максимальне значення (у BCD-коді).
 * Повертає TRUE, якщо у поточній позиції відбулося переповнення, яке може
 * означати, що необхідно оновити також інші позиції.
```

```
 */
unsigned char incTimePosition(unsigned char index, unsigned char limit)       //2.2.2.0
{                                                                             //5.2.1.0

   if (index < N_DIGITS >> 1) {                                               //2.2.2.1; 5.2.1.1

      time[index]++;                                                          //2.2.2.2; 5.2.1.2

      /*
       * Лічба відбувається в BCD-коді. Якщо досягли значення 0xXA, то
необхідно
       * додати 0x06, щоб отримати 0xY0, де Y == X + 1.
       */

      if ((time[index] & LO_NIBBLE_MASK) == 0x0A) {                           //2.2.2.3; 5.2.1.3
         /* Наприклад, 0x39 + 0x01 = 0x3A, а потрібно 0x40, тому
          * 0x3A + 0x06 = 0x40.
          */

         time[index] += BCD_CORRECTION;                                       //2.2.2.4; 5.2.1.4
      }

       /* У випадку досягнення граничного значення лічби -> почати з 0. */
      if (time[index] == limit) {                                             //2.2.2.5; 5.2.1.5

         time[index] = 0;                                                     //2.2.2.6; 5.2.1.6

         return TRUE;
      }
   }

   return FALSE;                                                              //2.2.2.7; 5.2.1.7

}

/*
 * Оновлює цифри на індикаторі.
 */
void refreshDigit()                                                           //4.1.0
{

   /* lastRefreshedDigit / 2 -> індекс у масиві time[]. */
   unsigned char index = lastRefreshedDigit >> 1;                             //4.1.1
```

```
    /* Старша чи молодша тетрада?
     * Якщо старша тетрада, то виконати зсув на 4 біта праворуч,
     * щоб правильно виконати виведення у порт E.
     * За допомогою конструкції (lastRefreshedDigit & 0x01) виконується перевірка на парність.
     */

    unsigned char shift = (lastRefreshedDigit & 0x01) ? HI_NIBBLE_SHIFT : 0x00;  //4.1.2

    /* Визначаємо, чи потрібна крапка (у крайній правій позиції – не потрібна). */

    unsigned char point = (shift == 0 && index > 0) ? DP : 0x00;                 //4.1.3

    /* Гасимо індикатор. */

    PORTE &= INDICATOR_DESELECT_MASK;                                            //4.1.4

     /* Виведення чергової цифри в порт D. */

    PORTD = LED_CODE[(time[index] >> shift) & LO_NIBBLE_MASK] | point;  //4.1.5

    /* Вибір позиції на індикаторі з урахуванням маски миготіння. */

    PORTE |= (1 << lastRefreshedDigit) & blinkMask;                              //4.1.6

    /* Після оновлення останньої цифри -> почати з початку. */

    if (++lastRefreshedDigit == N_DIGITS) {                                      //4.1.7

        /*
         * Оновлення маски миготіння на останній цифрі, інакше можливе
         * несинхронне миготіння цифр, коли одну із них
         * ще не видно, а інша використовує нову маску і її видно,
         * та навпаки.
         */

        blinkMask = newBlinkMask;                                                //4.1.8

        lastRefreshedDigit = 0;                                                  //4.1.9
    }
}                                                                                //4.1.10
```

```
/*
 * Функція інкременту значення годин.
 */
void incHours()                                                    //5.2.3.3.0
{

   incTimePosition(I_HOURS, LIMIT_HOURS);                          //5.2.3.3.1
}

/*
 * Функція інкременту значення хвилин.
 */
void incMinutes()                                                  //5.2.3.0
{

   /* Переповнено значеннння хвилин → інкремент годин. */
   if (incTimePosition(I_MINUTES, LIMIT_MINUTES)) {                //5.2.3.1
     //5.2.3.2

      incHours();                                                  //5.2.3.3
   }
}                                                                  //5.2.3.4


/*
 * Функція інкременту значення секунд.
 */
void incSeconds()
                                                                   //5.2.0
{

   /* Переповнено значеннння секунд → інкремент хвилин. */
   if (incTimePosition(I_SECONDS, LIMIT_SECONDS)) {                //5.2.1 - 5.2.2
      incMinutes();                                                //5.2.3
   }
}

/*
 * Функція інкременту показань годин у режимі налаштування.
 */
void setupIncTime()                                                //2.2.0
{

   if (mode == MODE_EDIT_HOURS) {                                  //2.2.1
```

```
      incTimePosition(I_HOURS, LIMIT_HOURS);                    //2.2.2
   } else if (mode == MODE_EDIT_MINS) {                         //2.2.3
      incTimePosition(I_MINUTES, LIMIT_MINUTES);                //2.2.4
   } else {
      incTimePosition(I_SECONDS, LIMIT_SECONDS);                //2.2.5
   }
}                                                               //2.2.6

/*
 * Обробник переривань від асинхронного таймера T/C0.
 */
ISR(TIMER0_OVF_vect)                                            //5.0
{
   if (!mode) {                                                 //5.1
      /* Якщо не у режимі налаштування → інкрементувати час. */
      incSeconds();                                             //5.2
   }
}                                                               //5.3

/*
 * Обробник переривань від T/C1.
 */
ISR(TIMER1_COMPA_vect)                                          //4.0
{

   /* Оновити цифру на індикаторі. */
   refreshDigit();                                              //4.1

   if (mode == 0) {                                             //4.2
      /* Якщо в рабочому режимі → більше нічого не робити. */
      return;
   }
/* Інкремент лічильника оновлень */
   updateCounter++;                                             //4.3

   if (updateCounter == BLINK_MAX) {                            //4.4
      /*
       * Якщо лічильник оновлень досяг значенння BLINK_MAX ->
       * оновити маску видимості.
       */

      /* Кнопку  "+" натиснуто? */
      if (PINE & PLUS_BUTTON_MASK) {                            //4.5
         /* Не натиснуто. */
```

```
        /*
         * Інвертувати 2 розряди у масці => 00110000b – години,
         * 00001100b – хвилини, 00000011b – секунди.
         */
        newBlinkMask ^= ((3 << ((mode - 1)<<1)));                    //4.6

        /* Скинути лічильник утримання кнопки "+1". */
        holdCounter = 0;                                             //4.7
      } else {
        /* Натиснуто. */

        /*
         * Якщо кнопка "+1" утримується 4 періоди миготіння →
         * почати інкремент часу.
         */
        if (holdCounter == AUTOINC_DELAY) {                          //4.8
          setupIncTime();                                            //4.9
        } else {
          holdCounter++;                                             //4.10
        }

        /* У режимі автоінкременту відключити миготіння. */
        newBlinkMask = 0xFF;                                         //4.11
      }

      /* Скидання лічильника оновлень. */
      updateCounter = 0;                                             //4.12
    }
}                                                                    //4.13

/*
 * Обробник переривань від кнопки "Setup".
 */
ISR(INT7_vect)                                                       //3.0
{

  if (mode == MODE_WORK) {                                           //3.1
    /* Перейти до налаштування значення годин. */
    mode = MODE_EDIT_HOURS;                                          //3.2
  } else {
    /* Перейти до налаштування хвилин або секунд. */
    mode--;                                                          //3.3
  }
```

```
   /*
    * Скинути маску миготіння, щоб минулий налаштований розряд не
    * залишився невидимим.
    */
   newBlinkMask = 0xFF;                                          //3.4
        /* Обнулити лічильник миготіння. */
   updateCounter = 0x00;                                         //3.5
   /*
    * Після завершення налаштування реініціалізуємо таймер,
    * щоб перша секунда була повною, інакше можливий варіант,
    * коли після виходу із режиму налаштування значення секунд
    * інкрементується одразу.
    */
   if(mode == MODE_WORK)                                         //3.6
   {
      initAsyncTimer();                                          //3.7
   }
}                                                                //3.8

/*
 * Обробник переривань від кнопки "+1".
 */
ISR(INT6_vect)                                                   //2.0
{

   if (mode == MODE_WORK) {                                      //2.1
      /* Якщо у робочому режимі → нічого не робити. */
      return;
   }

   /* Інкрементувати показання у поточній позиції. */
   setupIncTime();                                               //2.2
   /* Не блимаємо. */
   newBlinkMask = 0xFF;                                          //2.3
   /* Якщо кнопку натиснуто → обнулити лічильник миготіння. */
   updateCounter = 0x00;                                         //2.4
}                                                                //2.5
/* Налаштування портів введення/виведення */
void initPorts()                                                 //1.5.1.2.0
{
```

```
    /* Налаштування порту D на виведення. */
    DDRD  = 0xFF;                                          //1.5.1.2.1
    PORTD = 0xFF;

    /* Налаштування порту E: PE0…PE5 – на виведення, PE6, PE7 – на введення. */
    DDRE  = PE_INIT_MASK;                                  //1.5.1.2.2
    PORTE = 0xFF;
}                                                          //1.5.1.2.3
/* Ініціалізація асинхронного таймера */
void initAsyncTimer()                                      //1.5.1.3.0; 3.7.0
{

    /* Заборона переривань від T/C0. */
    TIMSK &= ~((1 << TOIE0) | (1 << OCIE0));               //1.5.1.3.1; 3.7.1

    /* Дозволити асинхронний режим. */
    ASSR |= (1 << AS0);                                    //1.5.1.3.2; 3.7.2

    /* Почати лічбу з нуля. */
    TCNT0 = 0x00;                                          //1.5.1.3.3; 3.7.3

    /* Встановити коефіцієнт дільника частоти 128. */
    TCCR0 |= (1<<CS00)|(1<<CS02);                          //1.5.1.3.4; 3.7.4

    /* Очікування оновлення регістрів. */
    while (ASSR & 0x07)                                    //1.5.1.3.5; 3.7.5
       ;

    /* Дозволити переривання після переповнення T/C0. */
    TIMSK |= (1 << TOIE0);                                 //1.5.1.3.6; 3.7.6
}                                                          //1.5.1.3.7; 3.7.7
/* Налаштування зовнішніх переривань для кнопок */
void initExternalInt()                                     //1.5.1.4.0
{

    /* Кнопки пов'язано з перериваннями  INT6 та INT7. */
    EIMSK |= (1 << INT7) | (1 << INT6);                    //1.5.1.4.1

    /* Переривання за фронтом. */
    EICRB |= (1<<ISC71)|(1<<ISC70)|(1<<ISC61)|(1<<ISC60);  //1.5.1.4.2
}                                                          //1.5.1.4.3
/* Налаштування T/C1 */
void initTimer1()                                          //1.5.1.5.0
{
```

```
    /* WGM11, WGM10 <- 0: режим скидання лічильника T/C1 при співпадінні
   //1.5.1.5.1

    /* WGM13 <- 0, WGM12 <- 1, CS <- 3: джерело опорної частоти == CLKio/64. */
    TCCR1B |= 0x0B;                                          //1.5.1.5.2

    /* Почати лічбу з нуля. */
    TCNT1 = 0x00;                                            //1.5.1.5.3

    /* Граничне значення лічби таймера. */
    OCR1A = TIMER_MAX;                                       //1.5.1.5.4

    /* Дозволити переривання для каналу порівняння A T/C1. */
    TIMSK |= (1 << OCIE1A);                                  //1.5.1.5.5
}                                                            //1.5.1.5.6
/* Ініціалізація периферії */
void init()                                                  //1.5.1.0
{

    /* Вимкнути Watchdog (можливо не знадобиться). */
    WDTCR = 0x10;                                            //1.5.1.1

    /* Налаштування портів ВВ/ВИВ. */
    initPorts();                                             //1.5.1.2

    /* Переведення T/C0 в асинхронний режим. */
    initAsyncTimer();                                        //1.5.1.3

    /* Налаштування зовнішніх переривань для кнопок. */
    initExternalInt();                                       //1.5.1.4

    /* Налаштування T/C1. */
    initTimer1();                                            //1.5.1.5
}
/* Головна  програма*/
int main()                                                   //1.5.0
{

    /* Ініціалізація периферії */
    init();                                                  //1.5.1
    /* Дозволити глобальні переривання. */
    sei();                                                   //1.5.2
```

```
 /* Нескінченний цикл, доки не виникає одне з чотирьох переривань. */
   while (1)                                                    //1.5.3
      ;

   return 0;
}
```

10.4 Зміст звіту

Звіт по роботі повинен містити:

- схему моделі;
- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

Контрольні запитання та завдання

1) Для чого може використовуватись таймер/лічильник?
2) Які регістри відповідають за дозвіл/заборону переривань від таймерів/лічильників?
3) Яким чином задається режим роботи таймера/лічильника T0 (T2)?
4) Чим відрізняється режим роботи CTC від режиму Normal?
5) Які таймери/лічильники можуть працювати в асинхронному режимі?
6) Що може виступати в якості задавача частоти при роботі таймера/лічильника в асинхронному режимі?
7) При виникненні яких подій таймери/лічильники T1, T3, T4, T5 можуть генерувати переривання?
8) За допомогою яких регістрів виконується керування таймером/лічильником?
9) Яким чином здійснюється вибір джерела тактового сигналу, а також запуск і зупинка таймерів/лічильників?
10) У чому полягає різниця між роботою 8-розрядних та 16-розрядних таймерів/лічильників у режимі Fast PWM?

11 ЛАБОРАТОРНА РОБОТА №8. ДОСЛІДЖЕННЯ МОДЕЛІ ЦАП

Тема: Моделювання модуля ЦАП

Мета: Користуючись пакетом Proteus 8.6 дослідити роботу модуля ЦАП

11.1 Порядок виконання роботи

1) Створити модель ЦАП в пакеті Proteus 8.6.
2) Розробити схему алгоритму роботи моделі та робочу програму.
3) Створити hex-файл та підключити його до мікроконтролера.
4) Запустити модель та виконати її дослідження згідно методичних вказівок.
5) Зробити відповідні висновки.

11.2 Опис роботи та розрахунок цифро-аналогових перетворювачів на основі резисторної матриці R-2R з підсумовуванням напруг

Загальна характеристика

У даному підрозділі буде розглянуто принцип роботи та розрахунок ЦАП на основі резисторної матриці R-2R з підсумовуванням напруг, який використовується в моделі ЦАП мікроконтролерів сімействах Mega [12; 19].

ЦАП з підсумовуванням напруг (рисунок 11.1) використовує режим роботи підсумовуючого елемента, близький до холостого ходу (операційний підсилювач підсумовує напруги).

Принцип дії ЦАП з матрицею R-2R та підсумовуванням напруг

ЦАП, з підсумовуванням напруг, використовує зворотне включення входу і виходу матриці R-2R (рисунок 11.1).

На входи $a_0, a_1, a_2 ... a_{n-1}$ надходять цифрові сигнали, які відповідають значенню і-го розряду вхідного двійкового коду (i = 0, 1,…, n-1). Якщо на вході і-го розряду присутня логічна одиниця, то відповідний ключ КЛі переключається у верхнє положення та опорна напруга $U_{\text{ОП}}$ через резистори матриці R-2R з визначеним

коефіцієнтом ділення подається на не інвертуючий вхід операційного підсилювача (ОП) DA1, де відбувається підсумовування напруг.

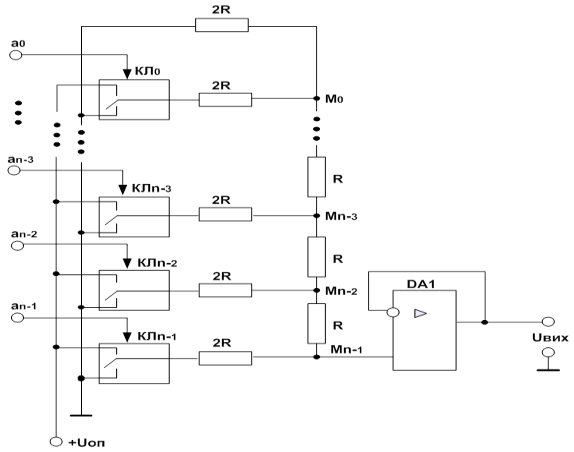


Рисунок. 11.1 – n-розрядний ЦАП з матрицею R-2R та з підсумовуванням напруг

Якщо на вхід і-го розряду надходить логічний нуль, то ключ переключається в нижнє положення, і дана гілка матриці R-2R підключається до спільної шини (логічний нуль).

Оскільки матриця резисторів є лінійним ланцюгом, її роботу можна проаналізувати методом суперпозиції, тобто внесок у вихідну напругу від кожного джерела (розряду) розрахувати незалежно один від одного. Внески від кожного розряду підсумовуються на не інвертуючому вході ОП і на виході отримуємо результат у вигляді напруги.

Розрахунок ЦАП

Якщо в старшому розряді вхідного ДК присутня логічна одиниця, а в інших розрядах – логічні нулі, то ключ $Кл_{n-1}$ знаходиться у верхньому положенні і підключає гілку резисторної матриці (РМ) з резистором 2R до джерела опорної напруги $U_{\text{ОП}}$. Інші ключі знаходяться у нижньому положенні і підключають інші гілки РМ (резистори 2R) до спільної шини.

Еквівалентну схему ЦАП для цього випадку наведено на рисунок 11.2, а.

Еквівалентний опір РМ вище вузла M_{n-1} дорівнює 2R. Вхідний опір ОП великий та він працює в режимі, близькому до холостого ходу. Тому струм, створюваний джерелом $U_{\text{ОП}}$ протікає через два однакових резистори 2R, що

утворюють дільник напруги $U_{\text{ОП}}$. У цьому випадку напруга на виході дільника визначається з виразу:

$$U_{діл} = U_{\text{н}} = \frac{U_{\text{оп}} \cdot 2R}{2R + 2R} = \frac{U_{\text{оп}}}{2}. \qquad (11.1)$$

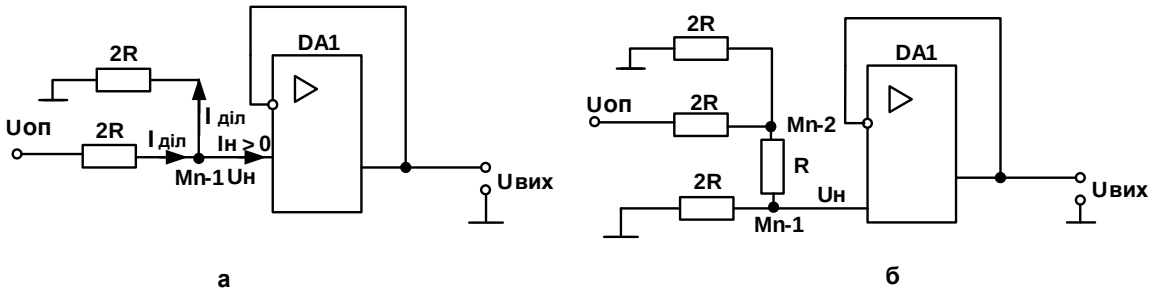


Рисунок 11.2 – Еквівалентні схеми ЦАП: а – при перетворенні коду 100…0В; б – при перетворенні коду 010…0В

Якщо на вхід схеми надходить комбінація ДК: 010…0В, то в цьому випадку ключ $Кл_{n-2}$ увімкнений у верхнє положення, а інші ключі – у нижнє. Еквівалентна схема ЦАП, прийме вигляд, представлений на рисунок 11.2. б.

Резистори R і 2R, які розташовано нижче вузла M_{n-2}, включено послідовно $(R_{\text{ВХ.DA1}} \to \infty)$. Тоді замінюємо їх еквівалентним опором:

$$\text{R} + 2\text{R} = 3\text{R}. \qquad (11.2)$$

Напруга в точці M_{n-2} визначається виразом:

$$U_{M_{n-2}} = \frac{U_{\text{оп}} \cdot 2\text{R} \| 3\text{R}}{2\text{R} + 2\text{R} \| 3\text{R}} = \frac{U_{\text{оп}} \cdot \frac{6}{5} \cdot R}{2\text{R} + \frac{6}{5} \cdot R} = \frac{U_{\text{оп}} \cdot 3}{8}. \qquad (11.3)$$

Напруга у вузлі M_{n-1}:

$$U_{M_{n-1}} = U_{\text{н}} = \frac{U_{M_{n-2}} \cdot 2\text{R}}{\text{R} + 2\text{R}} = \frac{U_{\text{оп}}}{4}. \qquad (11.4)$$

При подачі на вхід ЦАП ДК: 001…0В напруга на не інвертуючому вході ОП буде дорівнювати:

$$U_{\text{Н}} = \frac{U_{\text{ОП}}}{8}. \qquad (11.5)$$

При надходженні коду: 00...01 В напруга:

$$U_{\text{Н}} = \frac{U_{\text{ОП}}}{2^n}. \qquad (11.6)$$

Оскільки коефіцієнт передачі не інвертуючого операційного підсилювача $K_{\text{U.ІМС ОП}} = 1$ [12], то вираз для визначення сумарної вихідної напруги від дії одиниць у всіх розрядах вхідного ДК прийме вигляд:

$$U_{\text{ВИХ max}} = U_{\text{ОП}}\left(\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + ... + \frac{1}{2^n}\right) = \frac{\text{U}_{\text{ОП}}}{2^n}\sum_{i=0}^{n-1} 2^i. \qquad (11.7)$$

Якщо позначити значення і-х розрядів вхідного ДК як a_i, де a_i дорівнює 0 чи 1, то останній вираз перетвориться до вигляду:

$$U_{\text{ВИХ}} = \frac{\text{U}_{\text{ОП}}}{2^n}\sum_{i=0}^{n-1} a_i \cdot 2^i. \qquad (11.8)$$

Співмножник $\sum_{\text{i=0}}^{\text{n–1}} \text{a}_\text{i} \cdot 2^\text{i}$ є десятковим еквівалентом вхідного двійкового (цифрового) коду.

Розглянутий перетворювач називають помножуючим, тому що вихідна напруга пропорційна добутку значення опорного сигналу $U_{\text{ОП}}$ на значення вхідного цифрового коду.

Коефіцієнт передачі, тобто розрахункове збільшення вихідної напруги при зміні вхідного коду на одиницю молодшого розряду (ціна молодшого значущого розряду) складає:

$$K_{\text{ЦАП}} = \frac{U_{\text{ОП}}}{2^{\text{n}}}\left[\frac{\text{В}}{\text{МЗР}}\right]. \qquad (11.9)$$

11.3 Архітектура модуля ЦАП у складі мікроконтролерів XMega

Загальна характеристика

Частина AVR-мікроконтролерів, наприклад, XMega мають модуль цифро-аналогового перетворювача (ЦАП). Нижче наведено основні характеристики цього модуля [1; 12]:

– 12-розрядна роздільна здатність;
– частота перетворення до 1 МГц;
– гнучкий діапазон перетворення;
– кілька джерел запуску;
– в якості виходу ЦАП може бути один неперервний вихід для одноканального режиму роботи, або два окремих виходи зі схемою вибірки-зберігання для двоканального режим роботи.
– вбудоване калібрування зміщення і коефіцієнта передачі;
– внутрішнє або зовнішнє джерело опорної напруги (ДОН);
– можливість використання в якості входу для аналогового компаратора та АЦП;
– наявність енергозберігаючого режиму роботи;
– висока навантажувальна здатність – внутрішній опір становить 300 Ом для ATXMegaA3, ATXMegaA4 та 850 Ом для ATXMegaA1.

Принцип роботи модуля

Модуль ЦАП призначено для перетворення цифрового коду в аналогову напругу. Безпосередньо цифро-аналоговий перетворювач цього модуля виконано на основі резисторної матриці R-2R з підсумовуванням напруг (підрозділ 11.2).

Розмах вихідного сигналу залежить від опорної напруги U_{REF}.

Вихідна напруга та коефіцієнт передачі каналу ЦАП обчислюються за наступними виразами:

$$U_{\text{цап}} = \frac{CHnDATA}{4096} \cdot U_{ref}, \qquad (11.10)$$

$$K_{\text{цап}} = U_{ref}/4096 \text{ [мВ/МЗР]}, \qquad (11.11$$

де $CHnDATA$ – десятковий еквівалент числа, яке зберігається в регістрі даних,

n = 0/1 – номер регістра, U_{ref} – опорна напруга.

На рисунку 11.3 зображено функціональну схему модуля ЦАП.

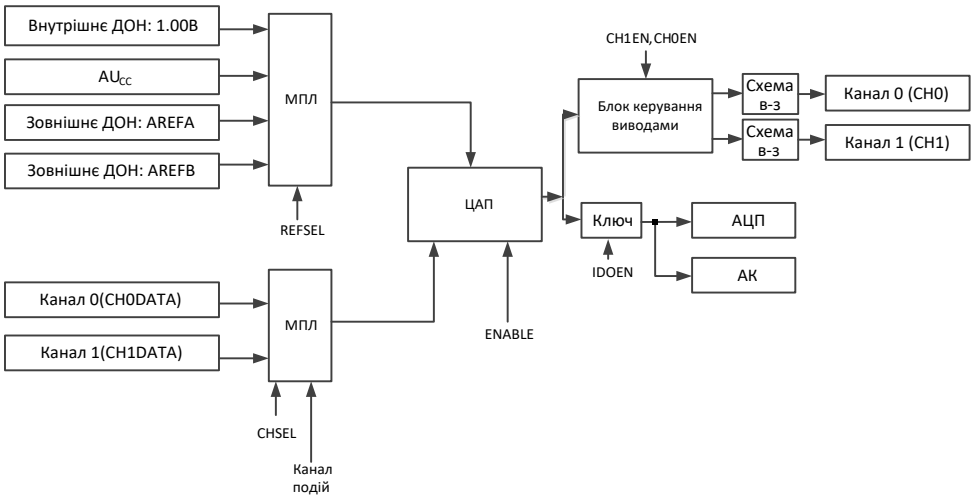


Рисунок 11.3 – Функціональна схема модуля ЦАП

Схема функціонує наступним чином. Вхідні дані для перетворення записуються у вхідні регістри даних CH0DATA (в одноканальному режимі роботи) або CH0DATA та CH1DATA (у двоканальному режимі роботи). Після початку перетворення, дані з регістрів крізь мультиплексор потрапляють безпосередньо на схему ЦАП, де за допомогою джерела опорної напруги (ДОН) – U_{ref}, обраного відповідними бітами регістра керування, виконується перетворення. Регістри керування задають також режим живлення, вибір каналу, джерело запуску, дані калібрування тощо. Аналоговий сигнал, отриманий під час перетворення, в залежності від обраного режиму роботи подається через блок керування виводами в канал 0 та канал 1, та при встановленні розряду IDOEN регістра CTRLA на вхід аналогового компаратора та АЦП.

Джерела опорної напруги

Для ЦАП у складі XMega можуть використовуватися наступні джерела опорної напруги:

- внутрішнє ДОН зі значенням 1,00 В;
- зовнішнє джерело AU_{cc};
- зовнішня напруга, що подається на вивід AREF порту A та/або B.

Якщо в якості $U_{ОП}$ для модуля ЦАП мікроконтролера AVR використати зовнішню напругу, що подається на вивід AREF порту A або B та дорівнює 4,096В = 4096mВ а число розрядів дорівнює 12, то коефіцієнт передачі $К_{ЦАП} = 4096/2^{12} = 4096/4096 = 1$mВ/МЗР, а максимальне значення напруги на виході ЦАП, коли у всіх розрядах вхідного коду присутні одиниці: $U_{ВИХ.max} = 4095*1 = 4095$mВ = 4,095В.

Вихідні канали

В якості виходу ЦАП можуть служити або один аналоговий вихід (канал 0), або два окремих виходи зі схемами вибірки-зберігання (в-з). Виходи схеми вибірки-зберігання можуть працювати повністю незалежно, дозволяючи генерувати два аналогових сигнали, що розрізняються як за амплітудою, так і за частотою.

Для кожного з виходів схем вибірки-зберігання передбачено окремі регістри даних та регістри керування. Вихідна напруга ЦАП може бути подана на вхід аналогового компаратора та АЦП. Ця напруга на АЦП та АК може подаватися з виходу безпосередньо ЦАП, а не з виходу схеми вибірки зберігання (рисунок 11.3).

Режими роботи

Модуль ЦАП може працювати в одноканальному, або двоканальному режимах роботи.

Одноканальний режим роботи

В одноканальному режимі роботи регістр даних CH0DATA через мультиплексор з'єднується з входом безпосередньо ЦАП (рисунок 11.4).

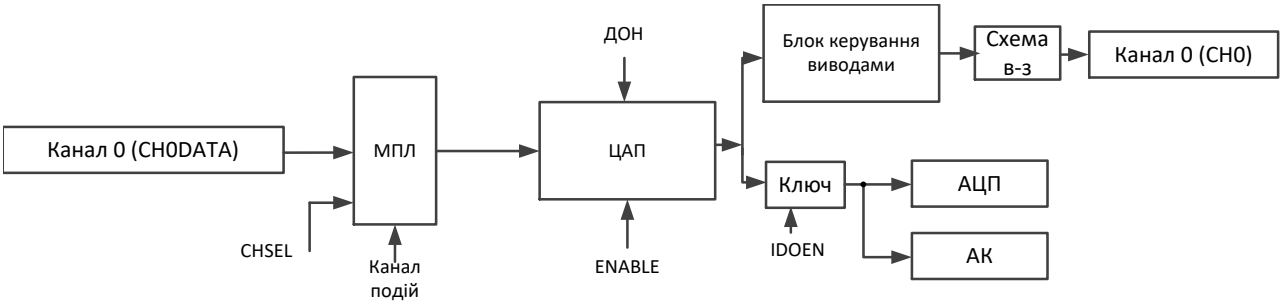


Рисунок 11.4 – Функціональна схема роботи ЦАП в одноканальному режимі

Нижче наведено послідовність програмування ЦАП для цього режиму роботи:

– записом 0x00 в розряди CHSEL[1:0] регістра керування CTRLB, встановити одноканальний режим роботи;

– для запуску перетворення за записом у регістр даних скинути розряд CH0TRIG в регістрі керування CTRLB;

– записом 01 в розряди REFSEL[1:0] регістра керування CTRLC обрати в якості ДОН аналогову напругу AU_{cc};

– встановити в регістрі керування CTRLA розряд дозволу роботи каналу 0 – CH0EN;

– встановити в регістрі керування CTRLA розряд дозволу роботи модуля ЦАП − ENABLE;

– для запуску перетворення записати 12-розрядне значення (з правим вирівнюванням) в регістр даних каналу 0 – CH0DATA. Молодші біти записуються в першу чергу.

Двоканальний режим роботи

У двоканальному режимі роботи (рисунок 11.5) ЦАП по черзі перетворює дані з каналів CH0DATA та CH1DATA. Схеми вибірки-зберігання використовуються для збереження значень між перетвореннями.

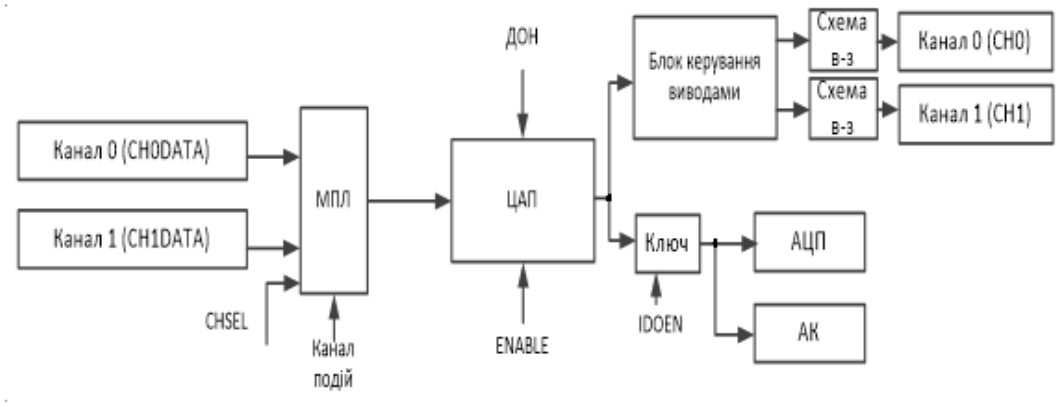


Рисунок 11.5 – Функціональна схема роботи ЦАП у двоканальному режимі

Для коректної видачі вихідного значення на два виходи ЦАП повинен регулярно оновлювати канали.

Нижче наведено послідовність програмування ЦАП для двоканального режиму роботи:

– записом 0x01 в розряди CHSEL[1:0] регістра керування CTRLB встановити двоканальний режим роботи;

– для запуску перетворення за записом в регістр даних, а не за подією скинути розряд CH0TRIG в регістрі керування CTRLB;

– для того, щоб обрати аналогове ДОН встановити в регістрі керування CTRLC розряди REFSEL в 0x01;

– встановити в регістрі керування CTRLA розряди дозволу роботи каналу 0 (CH0EN) та каналу 1 (CH1EN);

– встановити розряди вибірки каналу CONINTCAL[2:0] регістра керування таймером TIMCTRL в 0x04 для встановлення інтервалу між перетвореннями в 24 такти (інтервал, що дорівнює 3 мкс при периферійній частоті 8 МГц);

– встановити розряди REFRESH[3:0] регістра керування часом TIMCTRL в 0x06 (інтервал оновлення величиною в 1024 такти, що дорівнює 128 мкс при периферійній частоті 8 МГц);

– для дозволу роботи модуля ЦАП встановити розряд ENABLE в регістрі керування CTRLA;

– для запуску перетворення записати 12-розрядне значення (з правим вирівнюванням) в регістр даних відповідного каналу. При відсутності змін в регістрах даних виходи будуть оновлюватися кожні 128 мкс.

Тактування модуля

Модуль ЦАП тактується сигналом синхронізації від зовнішнього тактового генератора. Інтервал перетворень і частота оновлення в двоканальному режимі задаються кратними періоду сигналу синхронізації.

Обмеження часових характеристик

Для коректної роботи модуля ЦАП необхідно дотримуватися ряду часових обмежень, які задаються кратними періоду сигналу синхронізації. Недотримання обмежень може погіршити точність перетворення.

Час оновлення ЦАП – це інтервал часу між оновленнями каналів у двоканальному режимі. Величина цього інтервалу не повинна перевищувати 30 мкс. Цей параметр впливає на те, щоб забезпечувати підтримку стабільного вихідного сигналу на два виходи у двоканальному режимі роботи. Ця необхідність виникає тому що, схема буде втрачати амплітуду сигналу з часом, як конденсатор втрачає напругу завдяки під'єднаному до нього резистору. Більш висока частота оновлення спричиняє більш високе енергоспоживання.

Інтервал вибірки та перетворення ЦАП – це проміжок часу від моменту початку перетворення в каналі до запуску нового перетворення. Цей проміжок не повинен бути менше 1 мкс в одноканальному режимі і 1.5 мкс у двоканальному режимі. Фактично це час, необхідний для перетворення та зберігання значення для аналогового виходу. Це аналогічно процесу зарядки конденсатора, якщо часовий інтервал завеликий, ви можете втратити інформацію з сигналу з більш високою швидкістю наростання вхідної напруги. Якщо інтервал вибірки та перетворення менший ніж інтервал оновлення, то канали будуть оновлені у визначений час, навіть якщо зайві (повторні) перетворення було виконано між інтервалами, що викликано ручним оновленням даних в регістрі.

На рисунку 11.6 наведено часові діаграми оновлення та запитів перетворення каналів 0 та 1, якщо ЦАП запрограмовано на роботу у двоканальному режимі.

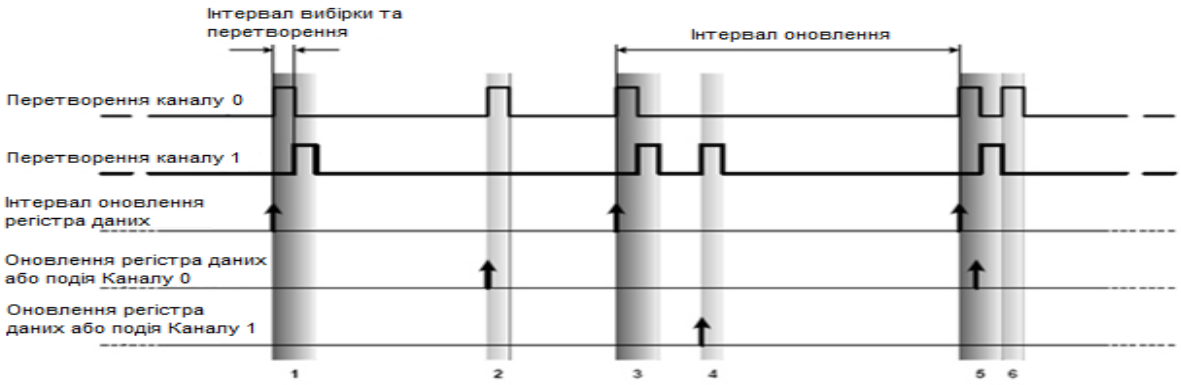


Рисунок 11.6 – Оновлення та запити перетворення каналів

Цифрами від 1 до 6 відмічено окремі моменти часу, в які відбуваються події які описано нижче:

1) Одночасно з інтервалом вибірки і перетворення починається інтервал оновлення каналу 0. Після закінчення першого інтервалу вибірки та перетворення настає інтервал вибірки та перетворення каналу 1.
2) Запит на перетворення (оновлення регістра даних або подія) для каналу 0, ініціює вибірку і перетворення тільки каналу 0, навіть якщо сигнал з'являється в середині інтервалу оновлень.
3) Початок чергового інтервалу оновлення, аналогічно п.1.
4) Запит перетворення (оновлення регістра даних або подія) для каналу 1, ініціює вибірку та перетворення тільки каналу 1, навіть якщо обидва канали були тільки що оновлені.
5) Початок чергового інтервалу оновлення, аналогічно п.1. Запит перетворення каналу 0 відкладається до моменту описаному в п.6.
6) Відкладений запит перетворення в п.5 ініціює чергову вибірку та перетворення по каналу 0 відразу після закінчення вибірки та перетворення каналу 1.

Зверніть увагу на те, що якщо частота запитів перетворення досягає частоти оновлення, частота перетворення може стати неточною, оскільки в момент надходження запиту перетворення може виконуватись звичайне оновлення. Такий запит відкладається до завершення оновлення обох каналів. При надходженні запитів перетворення для декількох каналів до виконання першого з них всі, крім першого запиту, буде проігноровано.

Якщо для конкретної програми це є проблемою, можна вимкнути автоматичне оновлення і натомість виконувати оновлення з достатньо високою частотою вручну. Зауважимо, що вимкнути автоматичне оновлення тільки для одного каналу неможливо.

Режим енергозбереження

При необхідності зниження споживаного струму модулем ЦАП під час перетворень, його можна перевести в економічний режим роботи. У цьому режимі між виконанням перетворень модуль переходить у відключений стан.

Робота в цьому режимі супроводжується збільшенням часу перетворення при запуску нового перетворення.

Система подій

Система подій мікроконтролерів XMEGA − це набір функцій, який дозволяє периферійним модулям взаємодіяти один з одним без втручання центрального процесора. Деякі периферійні модулі можуть генерувати події, часто за тими ж умовами, що і переривання. Ці події проходять через систему маршрутизації подій до споживачів подій, де споживачами подій можуть бути ініційовані певні дії. Центральний процесор не бере участі в цьому процесі, за винятком етапу налаштування. Наприклад, можна ініціювати захоплення входу таймера/лічильника при натисканні користувачем на кнопку, або почати аналого-цифрове перетворення при переповненні таймера/лічильника.

Програмування модуля

Регістри даних

На рисунках 11.7…11.10 наведено опис двох регістрів даних: CHnDATAH і CHnDATAL (де n = 0/1) − відповідно старша і молодша частини 12-розрядного значення CHnDATA, яке перетворюється в аналоговий сигнал. За замовчуванням 12-розрядів поділяються на 8 розрядів в CHnDATAL і 4 CHnDATAH з позиції молодшого значущого розряду − вирівнювання вправо. Для деяких програм краще використовувати ліво-спрямовані розряди. Для вибору даних з вирівнюванням вліво треба встановити розряд LEFTADJ у регістрі CTRLC.

	розряд		7	6	5	4	3	2	1	0
Вирівнювання вправо		+0x18	CHDATA[7:0]							
Вирівнювання вліво		+0x18	CHDATA[3:0]				-	-	-	-
Вирівнювання вправо	Читання/Запис		R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Вирівнювання вліво	Читання/Запис		R/W	R/W	R/W	R/W	R	R	R	R
Вирівнювання вправо	Початкове значення		0	0	0	0	0	0	0	0
Вирівнювання вліво	Початкове значення		0	0	0	0	0	0	0	0

Рисунок 11.7 – CH0DATAL − молодший байт регістра даних 0

	розряд	7	6	5	4	3	2	1	0
Вирівнювання вправо	+0x19	-	-	-	-	CHDATA[11:8]			
Вирівнювання вліво	+0x19	CHDATA[11:4]							
Вирівнювання вправо	Читання/Запис	R	R	R	R	R/W	R/W	R/W	R/W
Вирівнювання вліво	Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Вирівнювання вправо	Початкове значення	0	0	0	0	0	0	0	0
Вирівнювання вліво	Початкове значення	0	0	0	0	0	0	0	0

Рисунок 11.8 – CH0DATAH − старший байт регістра даних 0

	розряд	7	6	5	4	3	2	1	0
Вирівнювання вправо	+0x1A	CHDATA[7:0]							
Вирівнювання вліво	+0x1A	CHDATA[3:0]				-	-	-	-
Вирівнювання вправо	Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Вирівнювання вліво	Читання/Запис	R/W	R/W	R/W	R/W	R	R	R	R
Вирівнювання вправо	Початкове значення	0	0	0	0	0	0	0	0
Вирівнювання вліво	Початкове значення	0	0	0	0	0	0	0	0

Рисунок 11.9 – CH1DATAL − молодший байт регістра даних 1

	розряд	7	6	5	4	3	2	1	0
Вирівнювання вправо	+0x1B	-	-	-	-	CHDATA[11:8]			
Вирівнювання вліво	+0x1B	CHDATA[11:4]							
Вирівнювання вправо	Читання/Запис	R	R	R	R	R/W	R/W	R/W	R/W
Вирівнювання вліво	Читання/Запис	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Вирівнювання вправо	Початкове значення	0	0	0	0	0	0	0	0
Вирівнювання вліво	Початкове значення	0	0	0	0	0	0	0	0

Рисунок 11.10 – CH1DATAH − старший байт регістра даних 1

Регістри керування

CTRLA-регістр керування А (рисунок 11.11)

Bit	7	6	5	4	3	2	1	0	
+0x00	-	-	-	IDOEN	CH1EN	CH0EN	.	ENABLE	CTRLA
Read/Write	R	R	R	R/W	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 11.11 – CTRLA-регістр керування А

– Bits 7:5 – зарезервовано;

– Bit 4 − IDOEN: дозвіл виведення в АЦП та АК. Встановлення цього розряду в одиницю направляє внутрішній вихід ЦАП також до АЦП та АК;

– Bit 3 − CH1EN: дозвіл виведення каналу 1. Встановлення цього розряду в одиницю виводить результат перетворення на вихід мікросхеми, інакше канал доступний тільки для внутрішнього використання.

– Bit 2 − CH0EN: дозвіл виведення каналу 0. Встановлення цього розряду в одиницю виводить результат перетворення на вихід мікросхеми, інакше канал доступний тільки для внутрішнього використання;

– Bit 1 – зарезервовано;

– Bit 0 − ENABLE: дозвіл ЦАП. Встановлення цього розряду в одиницю дозволяє функціонування ЦАП.

CTRLB-регістр керування B (рисунок 11.12)

Bit	7	6	5	4	3	2	1	0	
+0x01	-	CHSEL[1:0]		-	-	-	CH1TRIG	CH0TRIG	CTRLB
Read/Write	R	R/W	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 11.12 – CTRLB-регістр керування B

– Bit 7 – зарезервовано;

– Bits 6:5 – CHSEL [1:0]: вибір режиму роботи ЦАП (таблиця 11.1). Ці розряди програмують режими роботи ЦАП: одно- або двоканальний;

– Bits 4:2 – зарезервовано;

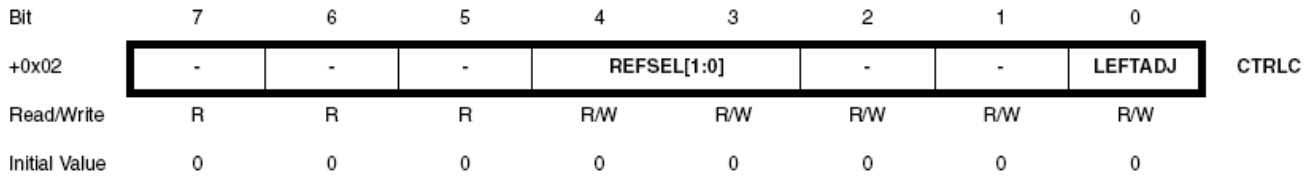
– Bit 1 – CH1TRIG: режим автоматичного перетворення каналу 1. Якщо розряд встановлено в одиницю, то подія, яку встановлено у регістрі EVCTRL, ініціює перетворення за умови, що дані у регістрі CH1DATA ще не перетворено;

– Bit 0 – CH0TRIG: режим автоматичного перетворення каналу 0. Якщо розряд встановлено в одиницю, то подія, яку встановлено у регістрі EVCTRL, ініціює перетворення за умови, що дані у регістрі CH0DATA ще не перетворено.

Таблиця 11.1 – Вибір режиму роботи ЦАП

CHSEL [1:0]	Режим роботи
00	Одноканальний (працює тільки канал 0)
01	Зарезервовано
10	Двоканальний (вибірка/зберігання для каналів 0 та 1)
11	Зарезервовано

CTRLC-регістр керування С (рисунок 11.13)



Bit	7	6	5	4	3	2	1	0	
+0x02	-	-	-	REFSEL[1:0]		-	-	LEFTADJ	CTRLC
Read/Write	R	R	R	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 11.13 – CTRLC-регістр керування С

– Bits 7:5 – зарезервовано;

– Bits 4:3 − REFSEL [1:0]: вибір ДОН. Джерело опорної напруги і, відповідно, діапазон перетворення ЦАП вибирається згідно таблиці 11.2;

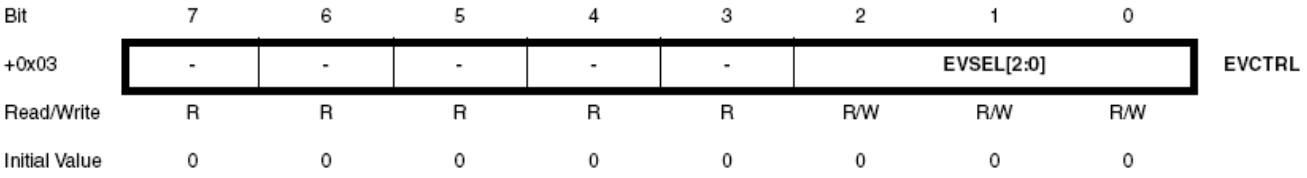
– Bit 2:1 – зарезервовано;

– Bit 0 – LEFTADJ: лівоспрямовані дані; якщо розряд встановлено в одиницю, то регістри CH0DATA і CH1DATA – лівоспрямовані.

Таблиця 11.2 – Вибір джерела опорної напруги ЦАП

REFSEL [1:0]	Джерело опорної напруги
00	Вбудоване джерело 1,00 В
01	AU_{cc}
10	Вивід AREF порту А
11	Вивід AREF порту В

Регістр керування подіями (рисунок 11.14)



Bit	7	6	5	4	3	2	1	0	
+0x03	-	-	-	-	-	EVSEL[2:0]			EVCTRL
Read/Write	R	R	R	R	R	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 11.14 – EVCTRL-регістр керування подіями

Bits 7:3 – зарезервовано;

– Bits 2:0 – обирають маршрут подій (таблиця 11.3).

Таблиця 11.3 – Вибір маршруту подій ЦАП

EVSEL[2:0]	Маршрут подій
000	0
001	1
010	2

Закінчення таблиці 11.3

011	3
100	4
101	5
110	6
111	7

Регістр керування часовими інтервалами (рисунок 11.15)

Bit	7	6	5	4	3	2	1	0	
+0x04	-	CONINTVAL[2:0]			REFRESH[3:0]				TIMCTRL
Read/Write	R	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 9.15 – TIMCTRL-регістр керування часовими інтервалами

– Bit 7 – зарезервовано;

– Bits 6:4 – CONINTVAL[2:0]: інтервал між двома завершеними перетвореннями. Інтервал повинно бути встановлено відповідно периферійному тактовому генератору, щоб переконатися, що нові перетворення не почнуться до тих пір, поки результат попереднього перетворення не запишеться. Інтервал перетворення ЦАП не повинен бути менше ніж 1 мкс в одноканальному режимі, і не менше ніж 1,5 мкс у двоканальному режимі (таблиця 11.4);

– Bits 3:0 – REFRESH [3:0]: керування часом оновлення результатів. Ці розряди контролюють часовий інтервал між оновленням результату в двоканальному режимі. Інтервал повинно бути встановлено відповідно з периферійним тактовим генератором, щоб уникнути втрати точності конвертованого значення (таблиця 11.5).

Таблиця 11.4 – Доступні установки інтервалу між перетвореннями

CONINTVAL[2:0]	Кількість тактів між перетвореннями в одноканальному режимі	Кількість тактів між перетвореннями у двоканальному режимі
0	1	1
1	2	3
10	4	6
11	8	12
100	16	24
101	32	48
110	64	96
111	128	192

Таблиця 11.5 – Частота оновлення результатів

REFRESH[3:0]	Кількість тактів між оновленням результату
0	16
1	32
10	64
11	128
100	256
101	512
110	1024
111	2048
1000	4096
1001	8192
1010	16384
1011	32768
1100	65536
1101	Зарезервовано
1110	Зарезервовано
1111	Оновлення вимкнуто

Регістр статусу ЦАП (рисунок 11.16)

Bit	7	6	5	4	3	2	1	0	
+0x05	-	-	-	-	-	-	CH1DRE	CH0DRE	STATUS
Read/Write	R	R	R	R	R	R	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Рисунок 11.16 – STATUS-регістр

– Bits 7:2 – зарезервовано;

– Bit 1 – CH1DRE: регістр даних для каналу 1 порожній. Якщо розряд скинуто у нуль, запис у регістр даних може призвести до втрати результату перетворення. Цей розряд може використовуватись для запитів DMA [1];

– Bit 0 – CH0DRE: регістр даних для каналу 0 порожній. Якщо розряд скинуто у нуль, запис у регістр даних може призвести до втрати результату перетворення. Цей розряд може використовуватись для запитів DMA.

Регістри калібрування

Для досягнення високої точності перетворення, треба калібрувати коефіцієнти передачі і зміщення модуля ЦАП. Калібрувальні значення для корегування коефіцієнту передачі і зміщення є 7-розрядними.

Найкращі результати досягаються при калібруванні в тих же умовах, в яких планується використовувати ЦАП, тобто за одних і тих же U_{REF}, вихідному каналі, часі перетворення і інтервалі оновлення.

З урахуванням похибок, теоретичну передатну функцію ЦАП можна записати наступним чином:

$$U_{DACn} = gain \cdot U_{ref} \cdot \frac{CHnDATA}{4096} + offset,$$

де *gain* - калібрування коефіцієнту передачі, *offset* - калібрування коефіцієнту зсуву.

У ідеального ЦАП коефіцієнт передачі дорівнює 1, а коефіцієнт зміщення дорівнює 0. Калібрування відбувається за допомогою регістрів GAINCAL та OFFSETCAL (рисунок 11.17, 11.18) і не залежить від вибору режиму.

Регістр калібрування коефіцієнту передачі ЦАП (рисунок 11.17)

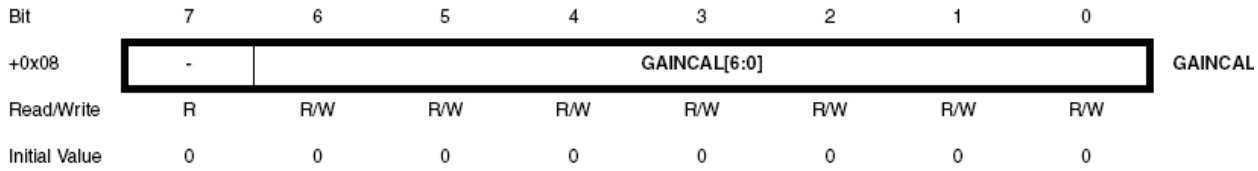


Рисунок 11.17 – GAINCAL-регістр калібрування підсилення ЦАП

– Bit 7 – зарезервований;

– Bits 6:0 – GAINCAL [6:0] – корегування коефіцієнту передачі ЦАП. Ці розряди використовуються, щоб компенсувати помилку підсилення ЦАП.

Регістр калібрування коефіцієнту зсуву ЦАП (рисунок 11.18)

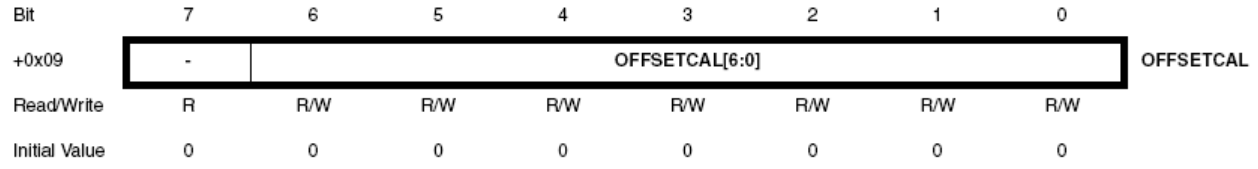


Рисунок 11.18 – OFFSETCAL-регістр калібрування зсуву ЦАП

– Bit 7 – зарезервований;

– Bits 6:0 – OFFSETCAL [6:0]: корегування коефіцієнта зсуву. Ці розряди використовуються щоб компенсувати помилку зсуву ЦАП.

Нижче у таблиці 9.6 наведено основні регістри модуля ЦАП

Характеристики ЦАП у складі деяких мікроконтролерів AVR

Характеристики ЦАП у складі дяких мікроконтролерів AVR наведено у таблиці 11.7.

Таблиця 11.6 – Основні регістри модуля ЦАП

Адреса	Регістр	Розряд 7	Розряд 6	Розряд 5	Розряд 4	Розряд 3	Розряд 2	Розряд 1	Розряд 0
+0x00	**CTRLA**	-	-	-	IDOEN	CH1EN	CH0EN	-	ENABLE
+0x01	**CTRLB**	-	CHSEL[1:0]		-	-	-	CH1TRIG	CH0TRIG
+0x02	**CTRLC**	-	-	-	REFSEL[1:0]		-	-	LEFTADJ
+0x03	**EVCTRL**	-	-	-	-	-	EVSEL[2:0]		
+0x04	**TIMCTRL**	-	CONINTVAL[2:0]			REFRESH[3:0]			
+0x05	**STATUS**	-	-	-	-	-	-	CH1DRE	CH0DRE
+0x06	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x07	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x08	**GAINCAL**	-	GAINCAL[6:0]						
+0x09	**OFFSETCAL**	-	OFFSETCAL[6:0]						
Адреса	**Регістр**	Розряд 7	Розряд 6	Розряд 5	Розряд 4	Розряд 3	Розряд 2	Розряд 1	Розряд 0
+0x10	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x11	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x12	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x13	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x14	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x15	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x16	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x17	**Зарезервовано**	-	-	-	-	-	-	-	-
+0x18	**CH0DATAL**	CHDATA[7:0]							
+0x19	**CH0DATAH**	-	-	-	-	CHDATA[11:8]			
+0x1A	**CH1DATAL**	CHDATA[7:0]							
+0x1B	**CH1DATAH**	-	-	-	-	CHDATA[11:8]			

Таблиця 11.7 – Характеристики ЦАП у складі деяких мікроконтролерів AVR XMEGA

МК	Каналів	Розрядність	МК	Каналів	Розрядність
ATxmega64A1	4	12bit	ATxmega32A4	2	12bit
ATxmega128A1	4	12bit	ATxmega64A4	2	12bit
ATxmega192A1	4	12bit	ATxmega128A4	2	12bit
ATxmega256A1	4	12bit	AT90PWM3B	1	10bit
ATxmega384A1	4	12bit	AT90PWM2B	1	10bit
ATxmega64A3	2	12bit	AT90PWM81	1	10bit
ATxmega128A3	2	12bit	AT90PWM216	1	10bit
ATxmega192A3	2	12bit	AT90PWM316	1	10bit
ATxmega256A3	2	12bit	ATmega16M1	1	10bit
ATxmega256A3B	2	12bit	ATmega32M1	1	10bit
ATxmega16A4	2	12bit	ATmega64M1	1	10bit

11.4 Моделювання ЦАП у програмному пакеті Proteus

11.4.1 Опис моделі

Оскільки в програмному пакеті Proteus поки ще не має мікроконтролера AVR з модулем ЦАП, тому ЦАП з матрицею R-2R та підсумовуванням напруг промоделюємо в пакеті PROTEUS 8.6 (рисунок 11.19).

На входи матриці ЦАП подається 7-розрядний дійковий код. Кожен вхід має свою «вагу». Входи розташовані в порядку зменшення ваги згори вниз. Тобто верхній вхід надає найбільший вплив на вихідний сигнал. Наступний за ним вдвічі менше і т. д. (підрозділ 9.2). Останній (нижній) вхід змінює вихідний сигнал в мілівольтах згідно значенню коефіцієнта передачі. Якщо комбінація біт, яка надходить на вхід ЦАП відома, то вихідна напруга моделі обчислюється згідно формули

$$U\text{вих} = \frac{U\text{оп} * D}{2^N}, \qquad (11.10)$$

де D – десятковий еквівалент двійкового коду, N = 7 – число розрядів ЦАП,
Uоп = 5 В.

Припустимо, що на вході у нас число 10010101. Тоді вихідну напругу можна розрахувати як:

$$U_{вих} = U_{дж} * (\mathbf{1} *1/2 + \mathbf{0} *1/4 + \mathbf{0} *1/8 + \mathbf{1}*1/16 + \mathbf{0}*1/32 + \mathbf{1}*1/64 + \mathbf{0}*1/128 + 1*\mathbf{1}/256). \qquad (11.11)$$

11.4.2 Дослідження моделі

Нижче наведено результати моделювання ЦАП у пакеті PROTEUS 8.6 при ДК = 1111 111b (рисунок 11.19) та ДК = 0111100 (рисунок 11.20).

Наприклад, при подачі на вхід ЦАП кодової комбінації – 01111111 (рисунок 11.19), D = 127, а

$$U\text{вих} = \frac{5 * 127}{128} = 4.96 \text{ В}.$$

Якщо подамо, наприклад, кодову комбінацію 0111100 (рисунок 9.20), D = 60, а

$$U\text{вих} = \frac{5 * 60}{128} = 2.34 \text{ В}.$$

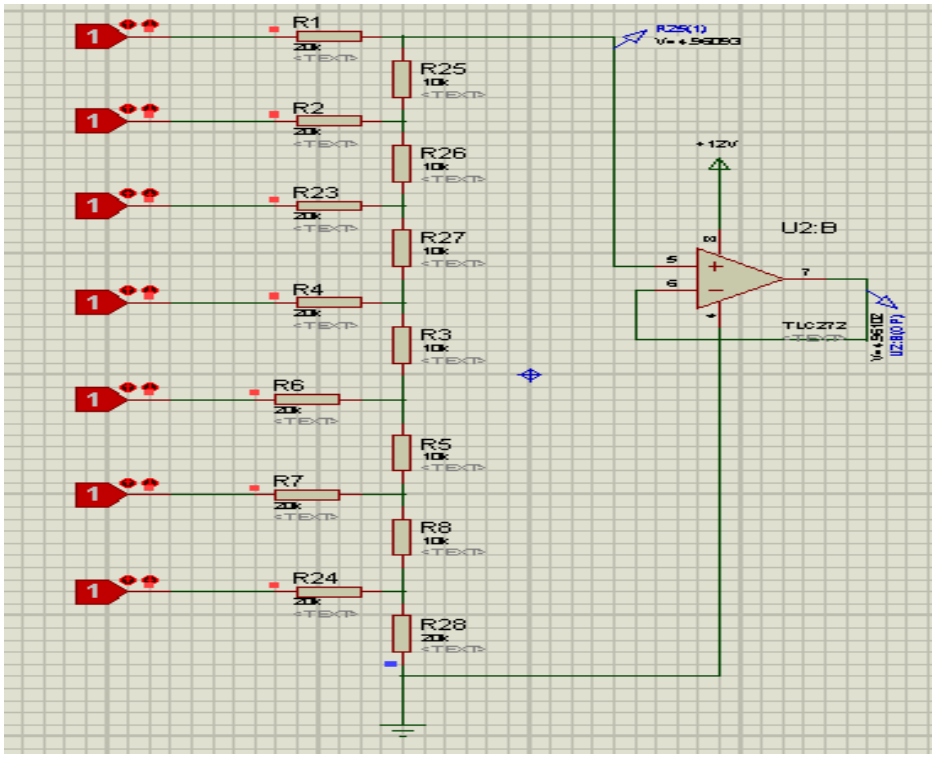


Рисунок 11.19 – Схема моделі ЦАП з матрицею R-2R та підсумовуванням напруг при ДК = 1111 111b

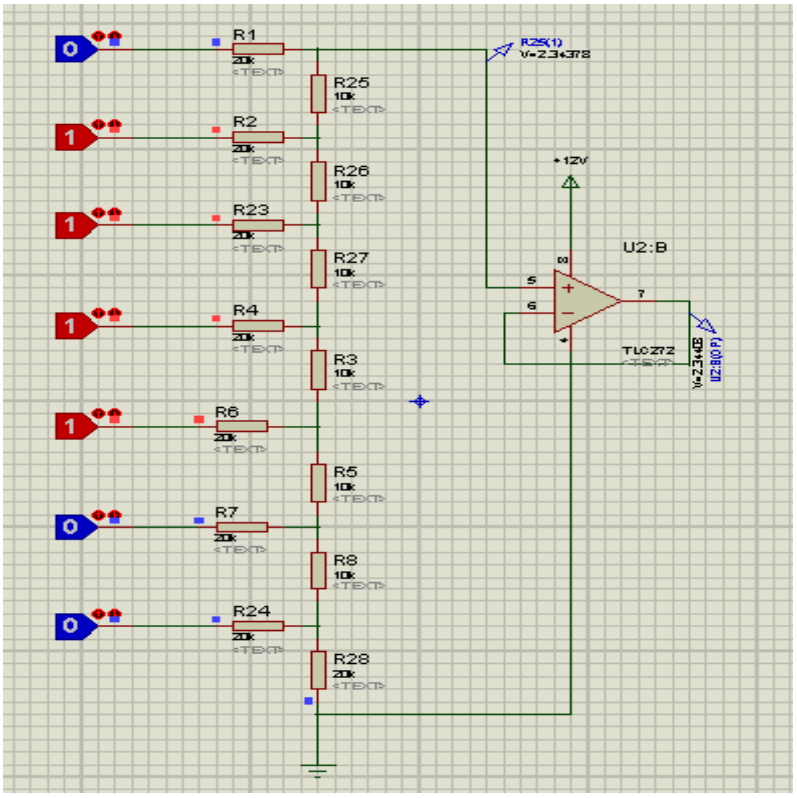


Рисунок 11.20 – Робота моделі при ДК = 0111100

11.4.3 Моделювання модуля ЦАП мікроконтролера LPC2138

Нижче (рисунок 11.21) наведено приклад моделювання в середовищі Proteus генератора синусоїдального сигналу з управлінням за частотою та виведенням через модуль ЦАП. Регулювання частоти сигналу зав'язано на векторні переривання (IRQ) від зовнішнього джерела. Формування періоду квантування виконується за допомогою таймера T/C0.

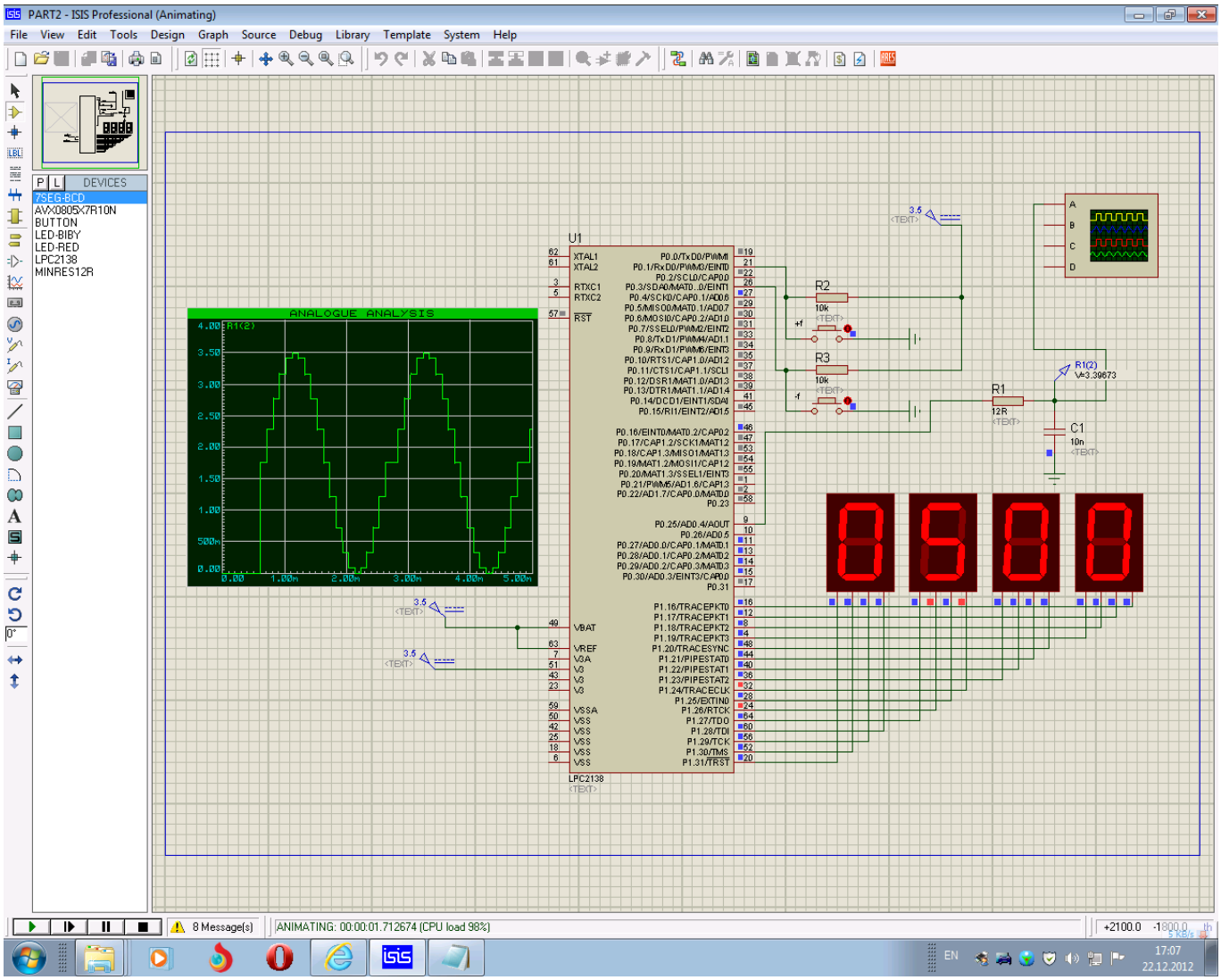


Рисунок 11.21 – Модель використання ЦАП на МК LPC2138

Якщо на діаграмі натиснути правою кнопкою миші на зображенні осцилографа і вибрати з випадаючого меню пункт [Digital Oscilloscope], то можна отримати зображення на осцилографі (рисунок 11.22). У нас задіяний лише канал A (жовтий графік).

При натисканні на кнопки [+ f] і [– f] частота синусоїдних коливань буде змінюватися, що відображається на рідкокристалічному дисплеї та екрані осцилографа.

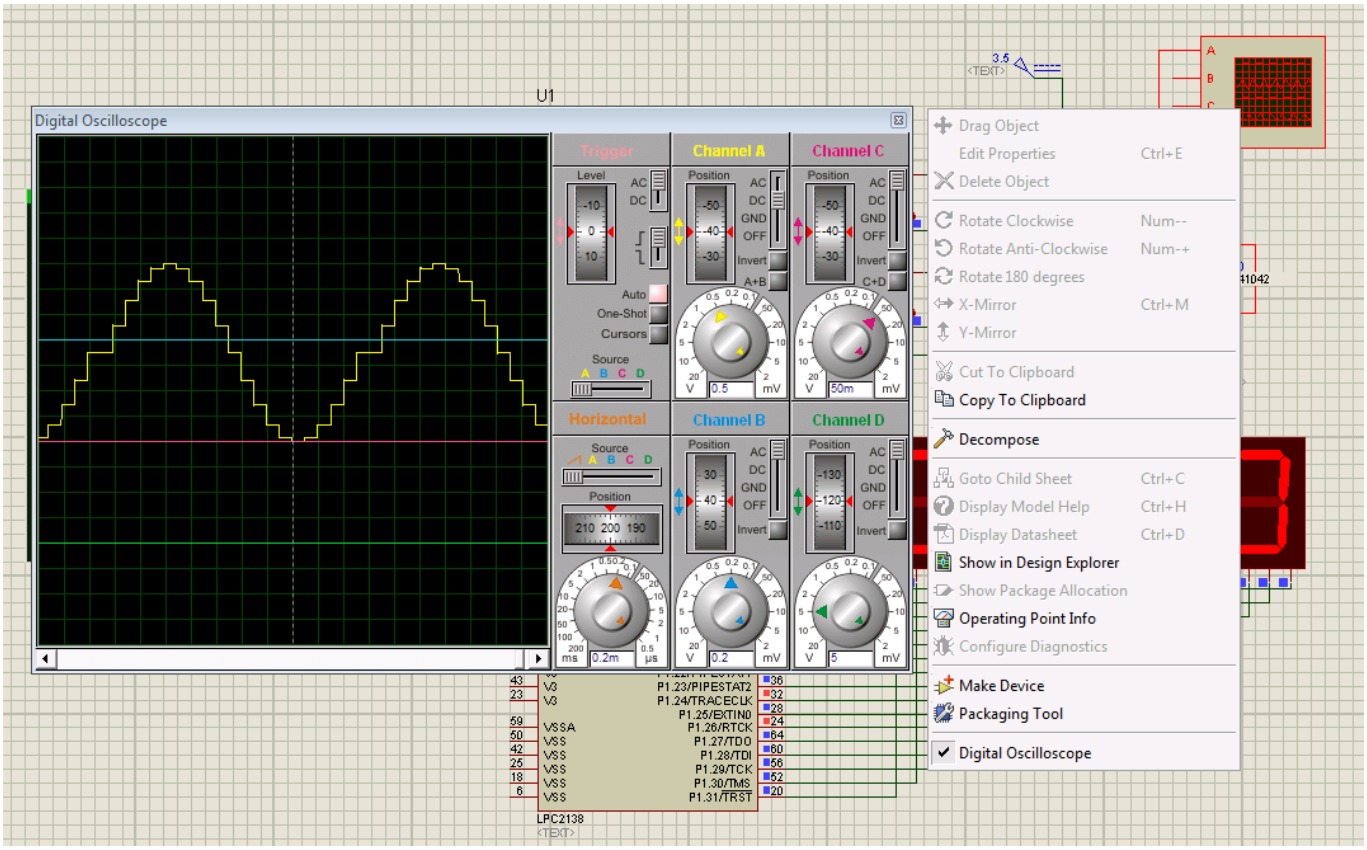


Рисунок 11.22 – Вихідний сигнал ЦАП на осцилографі

11.4.4 Розрахунки, що підтверджують працездатність моделі

Згідно з зображенням на екрані рідкокристалічного дисплея (рисунок 11.21)

частота промодельованої синусоїди – $f_{вих}$ = 500 Гц .

Тоді період синусоїди $T_{вих} = \frac{1}{f_{вих}}$ = 1/500 = 2 мс, що відповідає отриманому на осцилографі.

Значення періоду дискретизації синусоїдального сигналу обчислюється формулою: $\Delta t = \frac{1}{f_{дискр}}$, де $f_{дискр}$ = 10000 Гц. Тоді Δt = 0,1 мс.

Період синусоїдального сигналу можна представити у вигляді періоду дискретизації, помноженого на величину, яка показує, скільки разів змінюється вихідна напруга за один період синусоїди: $T_{вих} = N \cdot \Delta t$.

Тоді N = $T_{ВИХ}/\Delta t$ = 2 мс/0,1 мс = 20.

Отже за один період синусоїди вихідна напруга змінюється 20 разів. Подивившись на осцилограф, можна зробити висновок, що розрахунок зроблено вірно.

Робоча програма

Функція ind() керує виводом поточного значення частоти синусоїдальних коливань на РК-дисплей.

Функції IRQ_EINT0() і IRQ_EINT1() виконують зменшення/збільшення несучої частоти синусоїдальних коливань.

Функція IRQ_TC0() обчислює значення синусоїди.

```
#include <LPC213x.h>
#include <math.h>

static int f_max = 1000;        // Максимальне робоче значення частоти
static int f;                   // Робоче значення частоти
static int df;                  // Величина зміни частоти при натисканні на кнопки
static int n_max;               // Число квантів в періоді
static int n;                   // Номер кванта в періоді

void IRQ_EINT0(void) __irq;
void IRQ_EINT1(void) __irq;
void IRQ_TC0(void) __irq;
void ind( int f );              // Візуалізація значення частоти на індикаторі

void main(void) {

        f = f_max/2;                    // Ініціалізація робочої частоти

// Нехай частота прирощення таймера = 10000 Гц

        n_max = (10000/f);              // Визначаємо число квантів в періоді робочої частоти

// Ініціалізація портів введення-виведення

        PINSEL0 = 0x000CC;      // Встановити виводи: P0.1 в EINT0, P0.3 в EINT1
        PINSEL1 = 0x80000;      // Встановити виводи: P0.25 в AOUT
        IODIR1  = 0xffff0000;   // Встановити виводи: P1.16…P1.31 на виведення
```

```
// Налаштування таймера T/C0

        T0CTCR = 0x00;                  // Вибрати для T/C0 режим таймера

// Регістр керування джерелом лічильних імпульсів таймера/лічильника 0
// Кварц 10МГц, множення 6, ділення 1, тоді периферійний синхросигнал 15 МГц

        T0PR = 14;                      // Попередній дільник таймеру 0
        T0MR0 = 100;
        T0MCR = 0x3;          // Дозволяємо скидання T/C0 і переривання при збігу його з MR0

// Регістр керування порівнянням. Керує самим T/C0 і перериванням від схеми порівняння.
// D0 – дозвіл переривання від MR0; D1 – дозвіл скидання T/C0 від MR0; D2 – дозвіл останову

        T0TCR = 1;                      // Дозволяємо роботу T/C0

// Ініціалізація зовнішнього переривання

        EXTMODE = 0x3;              // Переривання за фронтом
        EXTPOLAR = 0x00;            // Для EINT0, EINT1 – переривання за зрізом

// Ініціалізація контролера переривання

        VICVectCntl0 = 0x00000020+14;  // Дозволити Слот 0 + Задати номер переривання IENT0
        VICVectCntl1 = 0x00000020+15;  // Дозволити Слот 1 + Задати номер переривання IENT1
        VICVectCntl2 = 0x00000020+04;  // Дозволити Слот 2 + Задати номер переривання T0

// біти D0…D4 – номер джерела переривання
//00 – WDT;   01 – не вик.; 02 – ARMCore1;  03 – ARMCore2
//04 – Timer0;  05 – Timer1;   06 – UART0;     07 – UART1
//08 – PWM0;    09 – I2C0;    10 – SPI0;      11 – SPI1
//12 – PLL;     13 – RTC;      14 – EINT0;    15 – EINT1
//16 – EINT2;   17 – EINT3;    18 – AD0;       19 – I2C1
//20 – BOD;     21 – I2C1;      22 – AD1;       23 – не вик.
// біт D5 – дозвіл переривання для даного слоту; біт = 1 – переривання дозволено

        VICIntSelect = 0x00;                    // Не включити FIQ переривань
        VICVectAddr0 = (unsigned)IRQ_EINT0;    // Задати адресу переривання від EINT0
        VICVectAddr1 = (unsigned)IRQ_EINT1;    // Задати адресу переривання від EINT0
        VICVectAddr2 = (unsigned)IRQ_TC0;       // Задати адресу переривання від EINT0
        VICIntEnable = (1<<14)+(1<<15)+(1<<4); // Дозволити переривання EINT0, EINT1 та T0

        while (1) { ind(f);};                   // Нескінченний цикл
}

void ind( int i )                       // Візуалізація значення частоти на індикатор
{
  int var1, var2, var3;
        var1 = i/1000;
        i = i-var1*1000;
        var2 = i/100;
        i = i-var2*100;
```

```
        var3 = i/10;
        i = i-var3*10;
        i = i+(var3<<4)+(var2<<8)+(var1<<12);
        i = i<<16;
        IOSET1 = i;
        IOCLR1 = i^0xffff0000;
}

void IRQ_EINT0(void) __arm __irq
{
        df = f/10;                      // Змінити девіацію частоти
        f = f+df;                       // Змінити частоту
    if (f>f_max) f = f_max;             // Перевірка границь
        n_max = (10000/f);              // Визначення числа квантів в періоді
        EXTINT = 0x01;                  // Скинути прапорець зовнішнього переривання EINT0
        VICVectAddr = 0x00000000;       // Скинути контролер переривань
}

void IRQ_EINT1(void) __arm __irq
{
        df = f/11;                      // Змінити девіацію частоти
        f = f-df;                       // Змінити частоту
    if (f<1) f = 1;                     // Перевірка границь
        n_max = (10000/f);              // Визначення числа квантів в періоді
        EXTINT = 0x02;                  // Скинути прапорець зовнішнього переривання EINT1
        VICVectAddr = 0x00000000;       // Скинути контролер переривань
}

void IRQ_TC0(void) __arm __irq
{       double a;
        a = 0x7ff0+0x7ff0*sin(2*3.141592*n/n_max);      // Розрахунок значень синусоїди
        DACR = a;
        n = n+1;
    if (n>n_max) n = 0;
        T0IR = 0x1;                     //Скинути прапорець переривань таймера T/C0 від модуля
                                        //порівняння MAT0
        VICVectAddr = 0x00000000;       //Скинути контролер переривань
}
```

11.4.5 Дослідження моделі ЦАП з використанням мікроконтролера mega8

Нижче наведено схему моделювання з зовнішним ЦАП, виконаним з використанням матриці R-2R та інтегральної мікросхеми операційного підсилювача на МК mega8 (рисунок 11.23).

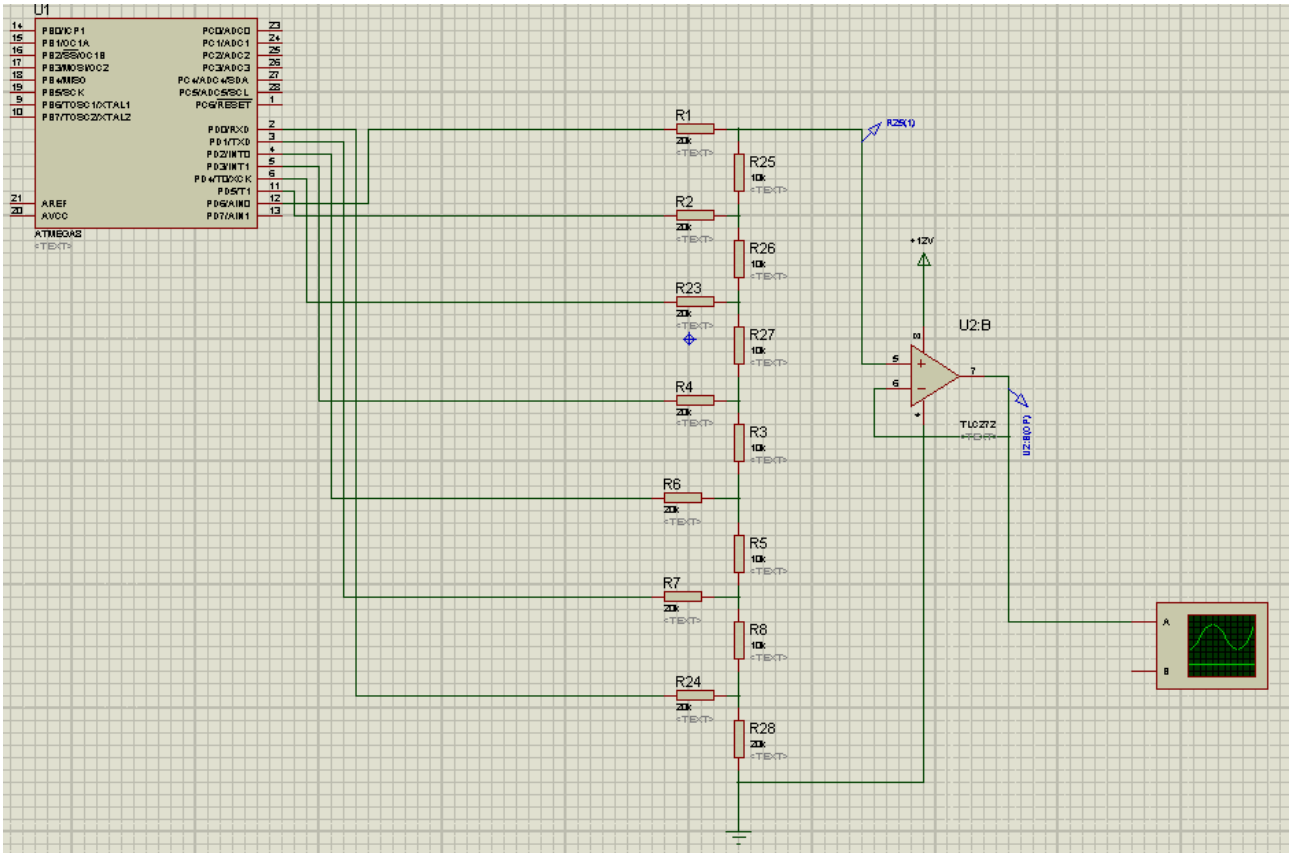


Рисунок 11.23 – Модель ЦАП в Proteus 8.6 на мікроконтролері mega8

Робоча програма

```
.include "m8def.inc"
.MACRO              OUTI
                    LDI         R16, @1
                    OUT         @0, R16
                    .ENDMACRO
.MACRO      LDPA
                    LDI         ZL, low(@0*2)
                    LDI         ZH, High(@0*2)
                    .ENDMACRO
.def Counter = R1
.def OSRG = R2

.DSEG
SinCT:              .byte 1
.CSEG
                    .ORG  0x0000
                    RJMP  Reset
Reset:          OUT SPL, low(RAMEND)
                OUT SPH, High(RAMEND)
Init:
                OUT DDRD, 0xFF
                OUT PORTD, 0xFF

Sine:               SetTask  TS_Sine            ; Повторний виклик через  API RTOS.
```

```
                LDS     Counter,SinCT    ; Завантажити з пам'яті поточне значення
    ;вказівника
Loop:           WDR                      ;
                LDPA    Sinus    ; Макрос, який завантажує  в Z адресу з таблиці з   синусом

                CLR     OSRG             ; Скидаємо робочий регистр
                ADD     ZL,Counter       ; Обчислюємо  зсув за адресою таблиці
                ADC     ZH,OSRG
                LPM     OSRG,Z           ; Завантажуємо в робочий регістр з таблиці
                OUT     PORTD,OSRG       ; Виводимо його в порт

                INC     Counter          ; Збільшуємо лічільник
                STS     SinCT,Counter    ; Зберігаємо значення лічільника в пам'яті
                RJMP    Loop             ; Перехід на мітку Loop
                RET
; Через те, що таблиця на 256 значень, то перевірку лічильника на переповнення робити не
потрібно, лічильник дорахує до 255 та скинеться.
; Для формування синусоїдальної напруги використовується схема паралельного ЦАП. Кожен
вивід порту мікроконтролера, до якого підключено паралельний ЦАП, має свою власну  вагу в
вольтах. За таблицею значень,що зберігається в пам'яті мікроконтролера, по черзі в порт
видається відповідна комбінація, активні значення складаються, утворюючи сумарну напругу
на виході ЦАП.
; 8-бітна таблиця синуса, використовується для формування на виході сумарної напруги у
формі синусоїди.
Sinus:  .DB 64,65,67,68,70,72,73,75
        .DB 76,78,79,81,82,84,85,87
        .DB 88,90,91,92,94,95,97,98
        .DB 99,100,102,103,104,105,107,108
        .DB 109,110,111,112,113,114,115,116
        .DB 117,118,118,119,120,121,121,122
        .DB 123,123,124,124,125,125,126,126
        .DB 126,127,127,127,127,127,127,127
        .DB 128,127,127,127,127,127,127,127
        .DB 126,126,126,125,125,124,124,123
        .DB 123,122,121,121,120,119,118,118
        .DB 117,116,115,114,113,112,111,110
        .DB 109,108,107,105,104,103,102,100
        .DB 99,98,97,95,94,92,91,90
        .DB 88,87,85,84,82,81,79,78
        .DB 76,75,73,72,70,68,67,65
        .DB 64,62,61,59,58,56,54,53
        .DB 51,50,48,47,45,44,42,41
        .DB 39,38,36,35,34,32,31,30
        .DB 28,27,26,25,23,22,21,20
        .DB 19,18,17,15,14,13,13,12
        .DB 11,10,9,8,8,7,6,5
        .DB 5,4,4,3,3,2,2,2
        .DB 1,1,1,0,0,0,0,0
        .DB 0,0,0,0,0,0,1,1
        .DB 1,2,2,2,3,3,4,4
        .DB 5,5,6,7,8,8,9,10
        .DB 11,12,13,13,14,15,17,18
```

```
.DB 19,20,21,22,23,25,26,27
.DB 28,30,31,32,34,35,36,38
.DB 39,41,42,44,45,47,48,50
.DB 51,53,54,56,58,59,61,62
```

11.5 Зміст звіту

Звіт по роботі повинен містити:

- схему моделі;
- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

Контрольні запитання та завдання

1) Наведіть та поясніть формули розрахунку напруги на виході ЦАП, та коефіцієнту передачі.
2) Чому дорівнює максимальне значення напруги на виході ЦАП, коли у всіх розрядах вхідного коду присутні одиниці?
3) Чому дорівнює коефіцієнт передачі операційного підсилювача у схемі ЦАП з підсумовуванням напруг?
4) Яку роздільну здатність має ЦАП у складі мікроконтролерів XMega?
5) Які джерела опорної напруги можуть бути використані для модуля ЦАП?
6) Наведіть вираз, що пов'язує вихідну напругу ЦАП зі значенням, яке записано у регістр даних.
7) Де можнв використати аналоговий сигнал, отриманий у результаті перетворення?
8) Для чого може бути використаний режим запуску мікроконтролера за настанням події?
9) Завдяки якому пристрою забезпечується двоканальне функціонування ЦАП?
10) Які часові обмеження накладаються на роботу ЦАП?
11) Як досягається режим енергозберігання модулю ЦАП і які наслідки має використання цього режиму?
12) Навіщо і як виконується калібрування ЦАП?

12 ЛАБОРАТОРНА РОБОТА №9. МОДЕЛЮВАННЯ МІКРОКОНТРОЛЕРНОЇ МЕРЕЖІ 1-WIRE

Тема: Моделювання мікроконтролерної мережі 1-Wire з використанням мікроконтролера сім'ї AVR.

Мета: Користуючись пакетом Proteus 8.6 дослідити роботу мікроконтролерної мережі 1-Wire.

12.1 Порядок виконання роботи

1) Створити модель мікроконтролерної мережі в пакеті Proteus 8.6.
2) Розробити схему алгоритму роботи мережі та робочу програму.
3) Створити hex-файл та підключити його до мікроконтролера.
4) Запустити модель та виконати її дослідження згідно методичних вказівок.
5) Зробити відповідні висновки.

12.2 Опис інтерфейсу 1-WIRE

12.2.1 Загальна характеристика

Однопровідний інтерфейс 1-Wire розроблений фірмою Dallas Semiconductor та може застосуватися в таких сферах:

- прилади в спеціальних корпусах MicroCAN для вирішення проблем ідентифікації, перенесення або перетворення інформації (технологія iButton);
- програмування вбудованої пам'яті інтегральних компонентів;
- ідентифікація елементів обладнання та захист доступу до ресурсів електронної апаратури;
- системи автоматизації.

Перший з цих напрямів широко відомий у світі та вже давно користується популярністю. Другий забезпечує можливість легкої перебудови функцій напівпровідникових компонентів, які вироблено фірмою Dallas Semiconductor, та мають малу кількість зовнішніх виводів.

Третій дозволяє забезпечити дешеву, але достатньо ефективну ідентифікацію та надійний захист найрізноманітнішого обладнання.

Щодо четвертого застосування, то реалізація локальних розподілених систем на базі 1-Wire-шини є на сьогоднішній день дуже ефективним рішенням для великої кількості практичних задач автоматизації [6].

В області систем автоматизації 1-Wire-мережі застосовуються для обслуговування високотехнологічних галузей машинобудування та хімічної промисловості, виготовлення електроніки для унікального експериментального і наукового обладнання. Випускаються різноманітні зонди для вимірювання високих і низьких температур, датчики вологості, тиску і кислотності з особливими функціями, спеціальні оптичні сенсори, плати збору інформації, пристрої сполучення з різним аналітичним обладнанням та багато іншого.

Проте справді революційну роль у розвитку однопровідних систем зіграла поява технології LINK. Завдяки використанню алгоритмів цифрової фільтрації ця технологія поліпшує механізм активної підтяжки 1-Wire-лінії, що дозволяє отримувати майже ідеальні сигнали обміну на довжинах кабелю більше 300 метрів з використанням до 250 ведених пристроїв. Крім того, це рішення забезпечує можливість вільної топології і значну стійкість систем до електромагнітних завад. Зараз випускається широка номенклатура LINK-адаптерів, включаючи модифікації для COM-порту, для вбудованих рішень, USB-1-Wire-Hub на 4 окремих лініях і т. ін. Таким чином, застосовуючи досягнення iBUTTONLINK, розробники 1-Wire-систем придбали надійний невибагливий інструмент, за допомогою якого можна успішно вирішувати непрості завдання автоматизації в умовах промислового виробництва [1; 6].

Іншим прикладом, що наочно демонструє на практиці можливості технології шини 1-Wire, є проект побудови повністю автоматичних метеорологічних станцій (1-Wire Weather Station (1-WWS)). Було створено кілька експериментальних систем 1-WWS, побудованих на базі персонального комп'ютера з адаптером DS9097U, керуючого комплексом з трьох термометрів DS18S20 для контролю температури, мікросхеми DS2438 для обслуговування датчика вологості повітря, компоненти

DS2423 для визначення швидкості вітру і 16-ти електронних міток DS2401, що визначають його напрям. Причому окремі 1-WWS-системи комплектувалися додатковими однопровідними рішеннями, які забезпечували контроль сигналів від датчиків: барометричного тиску, розрядів блискавки, кількості опадів на поверхні, сонячної активності, вологості ґрунту і т. ін. Дані з усіх сенсорів, що реєструються кожною з подібних систем, надходять до персонального комп'ютера і через Інтернет транслюються в режимі реального часу на центральний операторський пульт. Там виконується прийом, обробка та архівація результатів про погоду всього регіону, які було отримано завдяки аналізу інформації від кількох територіально розподілених станцій.

До одного з найбільш затребуваних типових застосувань 1-Wire відноситься маркування та безпека картриджів для принтерів і копіювальних апаратів, а також будь-яких електронних виробів, що вимагають цього. Однією з причин затребуваності є вбудований в чіпи механізм шифрування SHA. Однопровідні 1-Wire-чіпи DS2401, DS2411, DS2431, DS2432, DS28E01-100, DS2433, DS250x та інші від Dallas Semiconductor надзвичайно широко поширені для реалізації захисту CRUM (Customer Replaceable Unit Monitor – змінний користувачем блок моніторингу), що входять до складу більшості копіювальних і розмножувальних пристроїв. При цьому вони часто виконують відразу кілька функцій, використовуючи мінімум контактів для обміну інформацією (тільки шини DATA і RET). Наприклад, роль ідентифікатора устаткування, аутентифікатора легальності доступу, лічильника копій (або витрати тонера) і т. ін. Однопровідні 1-Wire-чіпи сьогодні використовуються в картриджах величезного числа моделей копіювальних апаратів від таких фірм-виробників, як: Xerox, Sharp, Fujitsu, Minolta і т. ін. Причому багато компаній, що виробляють копіювальне обладнання, закуповують ці мікросхеми в промислових масштабах саме через вбудований SHA-механізм. Таке рішення є в даний час найбільш оптимальним у співвідношенні ефективність/ціна в порівнянні з будь-якими іншими варіантами апаратного захисту.

Досить перспективною галуззю, в якій повною мірою використовуються переваги технології 1-Wire-мереж і якій особливо багато уваги приділяє компанія

Dallas Semiconductor, є менеджмент автономних хімічних джерел струму – акумуляторних батарей. Під менеджментом тут розуміється, перш за все, сувора і повна ідентифікація джерел енергії, збереження в пам'яті вбудованого в батарею електронного пристрою особливостей її виготовлення та індивідуальних технічних характеристик, найбільш повний моніторинг їх основних експлуатаційних параметрів протягом усього терміну використання, а також формування коректного керуючого впливу, пов'язаного з відновленням заряду акумулятора, який обслуговується. Від правильного менеджменту і знання історії експлуатації батареї багато в чому залежить вибір алгоритму її повторного заряду, що безпосередньо пов'язано з ефективністю використання і терміном служби багатьох типів акумуляторів. Для цього кожна з батарей багатоелементних енергетичних конструкцій (особливо для мобільних або бездротових пристроїв і джерел безперебійного живлення) забезпечується індивідуальним однопровідним 1-Wire-компонентом, перетворюючись по суті в інтелектуальний системний елемент автономного живлення. При цьому сповідується комплексний підхід до проблеми менеджменту енергетичних елементів, коли 1-Wire-рішення дозволяють організувати недорогу багатоточкову шину комплексного обслуговування пристроїв менеджменту і керування зарядом, що дає можливість супроводжувати як окремі автономні джерела енергії, так і цілі батареї, складені з багатьох одиничних енергетичних складових.

На даний час Dallas Semiconductor постачає широку номенклатуру різних за функціональним призначенням однопровідних компонентів для реалізації найрізноманітніших мережевих задач. Тому є велике число конкретних прикладів застосування 1-Wire-інтерфейса для автоматизації у найрізноманітніших областях і все більше розробників проявляють інтерес до цієї технології.

У якості середовища для передачі інформації по однопровідній лінії найчастіше використовується звичайний телефонний кабель і, як наслідок, швидкість обміну в цьому випадку невелика.

Але, якщо ретельно проаналізувати більшість об’єктів, які потребують автоматизації, то для більшості з них гранична швидкість обслуговування в 16,4 Кбіт/с є достатньою. А інші переваги 1-Wire-технології, такі як:

- просте вирішення адресування;
- нескладний протокол;
- проста структура лінії зв'язку;
- проста зміна конфігурації мережі;
- дешевизна всієї технології в цілому

показують раціональність та високу ефективність цього інструмента при вирішенні задач комплексної автоматизації у різних областях діяльності.

12.2.2 Основні характеристики інтерфейсу та мережі 1-WIRE

Загальний опис інтерфейсу

Мережа 1-Wire являє собою, по суті, мережу з централізованим керуванням та шинною топологією. Для здійснення зв'язку використовується одна лінія даних (DATA) та один зворотний провід. Таким чином, для реалізації середовища обміну цієї мережі можуть бути застосовані доступні кабелі, які мають неекрановану виту пару будь-якої категорії, а також, наприклад, звичайний телефонний провід. Такі кабелі при їх прокладці не потребують наявності спеціального обладнання, а обмеження максимальної довжини однопровідної лінії регламентовано розробниками на рівні 300 м.

Основою архітектури 1-Wire-мереж є топологія спільної шини, коли кожен з пристроїв підключено безпосередньо до єдиної магістралі без каскадних з'єднань або розгалужень. При цьому у якості базової використовується структура мережі з одним ведучим та багатьма веденими.

Конфігурація будь-якої 1-Wire-мережі може довільно змінюватися в процесі її роботи, при цьому не виникає перешкод подальшої експлуатації та працездатності всієї системи в цілому, якщо при цих змінах дотримуються основні принципи організації однопровідної шини. Ця можливість досягається завдяки наявності в протоколі 1-Wire-інтерфейсу спеціальної команди пошуку ведених пристроїв, яка

дозволяє швидко визначити нових учасників інформаційного обміну. Стандартна швидкість виконання такої команди дозволяє обробити приблизно 75 вузлів мережі за секунду.

Завдяки наявності у складі будь-якого пристрою, який має 1-Wire-інтерфейс, унікальної індивідуальної адреси (відсутність збігу адрес для пристроїв, які виробляє Dallas Semiconductor, гарантується самою фірмою-виробником) така мережа має практично необмежений адресний простір. При цьому кожен з однопровідних приладів одразу готовий до використання у складі 1-Wire-мережі, без будь-яких додаткових апаратно-програмних модифікацій.

Однопровідні компоненти є самотактуючими напівпровідниковими пристроями, в основі обміну інформацією між якими лежить керування зміною тривалості часових інтервалів імпульсних сигналів в однопровідному середовищі та їх вимірювання. Передача сигналів для 1-Wire-інтерфейсу асинхронна та напівдуплексна, а вся інформація, що циркулює в мережі, сприймається абонентами або як команди, або як дані. Команди мережі генеруються ведучим та забезпечують різні варіанти пошуку та адресації ведених пристроїв, визначають активність на лінії окремих компонентів, керують обміном даними в мережі і т. ін.

Стандартну швидкість роботи 1-Wire-мережі, яка складає 16,4 Кбіт/с, було обрано, по-перше, з урахуванням забезпечення максимальної надійності передачі даних на великі відстані, та, по-друге, з урахуванням швидкодії найбільш широко розповсюджених типів мікроконтролерів, які в основному повинні використовуватися при реалізації ведучих пристроїв однопровідної шини. Це значення швидкості обміну може бути зменшено до будь-якого можливого значення завдяки введенню примусової затримки між передачею в лінію окремих бітів даних (розтягуванню часових слотів протоколу). Швидкість обміну також може бути збільшено за рахунок переходу на спеціальний прискорений режим обміну (швидкість Overdrive – до 125 Кбіт/с), який допускається для окремих типів однопровідних 1-Wire-компонентів на невеликій по відстані якісній лінії, яка не перевантажена іншими пристроями лінії зв'язку.

При реалізації однопровідного інтерфейсу використовуються стандартні КМОН/ТТЛШ логічні рівні сигналів, а живлення більшості однопровідних компонентів може здійснюватися від зовнішнього джерела з робочою напругою в діапазоні від 2,8 до 6,0 В. Альтернативою застосуванню зовнішнього живлення є, так званий, механізм «паразитного живлення», дія якого полягає у використанні кожним з ведених компонентів 1-Wire-лінії електричної енергії імпульсів, які передаються по шині даних. Ця енергія акумулюється спеціальною ємністю, яку вбудовано в пристрій. Окрім того, окремі компоненти однопровідних мереж можуть використовувати режим живлення по шині даних, коли енергія до приймача поступає безпосередньо від ведучого пристрою по лінії зв'язку. При цьому обмін інформацією в мережі примусово припиняється.

Фізична реалізація

Фізична реалізація інтерфейсу 1-Wire достатньо проста. На рисунку 12.1 показано спрощену схему апаратної реалізації інтерфейсу 1-Wire.

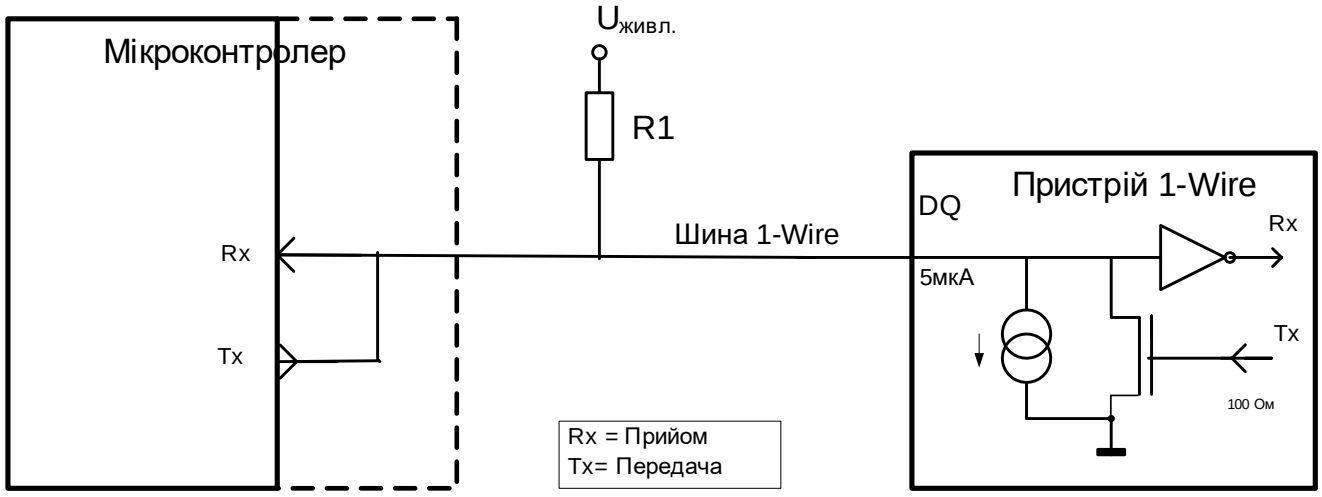


Рисунок 12.1 – Схема апаратної реалізації інтерфейсу 1-Wire

Вивід DQ пристрою 1-Wire є входом КМОН-логічного елементу, який може бути зашунтований польовим транзистором. Опір каналу цього транзистора у відкритому стані – близько 100 Ом. Коли транзистор закрито – є невеликий струм витоку (приблизно 5 мкА) на спільний провід.

Шина 1-Wire повинна бути підтягнута окремим резистором R1 до напруги живлення пристроїв. Опір цього резистора 4,7 кОм, однак це значення рекомендовано лише для достатньо коротких ліній. Якщо шина 1-Wire використовується для підключення віддалених на велику відстань пристроїв, то опір цього резистора слід зменшити. Мінімально допустиме його значення – 300 Ом, а максимальне – близько 20…30 кОм. Дані величини орієнтовні та завжди уточнюються за характеристиками конкретного 1-Wire-пристрою.

Підключення шини 1-Wire до мікроконтролера на рисунку 12.1 показано умовно в двох варіантах: з використанням або 2-х окремих виводів мікроконтролера (один у якості входу, інший – виходу), так й одного, який працює як на вхід, так і на вихід. Розділення цих способів показано за допомогою пунктирної лінії, яка умовно позначає межу корпуса мікроконтролера. З деяким допущенням можна уявити собі логічну будову шини 1-Wire, як відоме з'єднання виводів мікросхем з відкритим колектором за схемою «монтажне АБО» для логічних сигналів низького рівня. Очевидно, що передача будь-якої інформації можлива тільки видачею низького рівня в лінію замиканням її на спільний провід, а у високий логічний рівень лінія повернеться сама, завдяки наявності зовнішнього резистора, який підтягує. Залежно від способу сполучення з веденими пристроями і матеріалів, що при цьому використовуються, розрізняють декілька варіантів організації 1-Wire-мереж, кожен з яких передбачає використання особливої технології та аксесуарів при реалізації лінії. Класифікацію та особливості 1-Wire мереж в залежності від довжини лінії, кількості ведених пристроїв у мережі, типу кабелю, що використовується, топології та підключення ведучого пристрою до лінії наведено у [6].

Передача даних в 1-Wire-мережі

Основні правила передачі даних в 1-Wire-мережі [6]:

- обмін завжди починається за ініціативою одного ведучого пристрою, яким в більшості випадків є мікроконтролер (МК);
- будь-який обмін інформацією починається з подачі ведучим пристроєм імпульсу скидання (“Reset Pulse” або просто RESET) в лінію 1-Wire;

– для інтерфейсу 1-Wire в загальному випадку передбачається «гаряче» підключення (тобто підключення до системи без відключення її живлення, зупинки системи та відключення пристроїв);
– будь-який пристрій, що під'єднаний до 1-Wire, після отримання живлення видає в лінію DQ імпульс присутності "Presence Pulse" (PRESENCE). Цей імпульс пристрій завжди видає в лінію, якщо отримає сигнал RESET;
– поява в шині 1-Wire імпульсу PRESENCE після видачі RESET однозначно свідчить про наявність хоча б одного підключеного пристрою;
– обмін інформацією ведеться так званими тайм-слотами: один тайм-слот служить для обміну одним бітом інформації;
– дані передаються побайтно, біт за бітом, починаючи з молодшого біта. Достовірність переданих/прийнятих даних (перевірка відсутності спотворень) гарантується шляхом підрахунку циклічної контрольної суми.

Таким чином, процедура ініціалізації інтерфейсу, з якої починається будь-який обмін даними між пристроями, має довжину мінімум 960 мікросекунд та складається з передачі від МК сигналу RESET та прийому від пристрою сигналу PRESENCE. Якщо сигнал PRESENCE не було виявлено, це означає, що на шині 1-Wire немає готових до обміну пристроїв. Вищезазначені правила визначають логічний низькорівневий протокол обміну даними. В [6] показано діаграму сигналів RESET та PRESENCE, з яких завжди починається будь-який обмін даними. Видача імпульсу RESET в процесі обміну слугує також для дострокового завершення процедури обміну інформацією.

Обмін бітами інформації здійснюються певними тайм-слотами [6]. Тайм-слот – це по суті певна, досить жорстко лімітована за часом послідовність зміни рівнів сигналу в лінії 1-Wire. Розрізняють 4 типи тайм-слотів: передача "логічної 1" від МК, передача "логічного 0" від МК, прийом "логічної 1" від пристрою і прийом "логічного 0" від пристрою.

Передачу будь-якого тайм-слота завжди починає МК шляхом переведення шини 1-Wire в низький логічний рівень. Тривалість будь-якого тайм-слота повинна знаходитися в межах від 60 до 120 мікросекунд. Між окремими тайм-слотами

завжди повинен передбачатися інтервал не менше 1 мікросекунди (конкретне значення визначається параметрами веденого пристрою).

Тайм-слоти, що передаються, відрізняються від тайм-слотів, що приймаються, поведінкою МК: при передачі він тільки формує сигнали, при прийомі, крім того, ще й опитує (тобто приймає) рівень сигналу в лінії 1-Wire.

В [6] показано часові діаграми тайм-слотів всіх 4-х типів та наведено їх опис.

Важливо розуміти, що слід дуже ретельно підходити до забезпечення в шині 1-Wire необхідних часових інтервалів, тому що, наприклад, збільшення тривалості тайм-слота виведення "логічного 0" понад рекомендованого значення може призвести до помилкового сприйняття цього тайм-слота, як сигналу RESET, і, зрозуміло, після цього вся процедура обміну припиниться. Але так само слід враховувати вплив самої лінії на тривалість фронтів імпульсів. Тому в загальному випадку, це не просте завдання. Але виконання нескладних рекомендацій дозволить її вирішити досить простими засобами: по-перше, всі сигнали, які повинен формувати МК, слід формувати за принципом необхідного мінімуму тривалості (тобто трохи більше, ніж зазначена мінімальна тривалість), а від пристрою слід очікувати сигналів за принципом найгіршого (тобто орієнтуватися на найгірші варіанти часових параметрів сигналу).

Обмеження застосування 1-Wire-мереж

Безумовно, 1-Wire-мережі мають свою нішу для застосування при побудові систем автоматизації. Безглуздо всерйоз використовувати їх для передачі великих масивів інформації, при побудові, наприклад, систем відеоспостереження або швидкісного обміну, пов'язаних з обслуговуванням швидких процесів, або ж порівнювати можливості однопровідних мереж з такими потужними мережевими промисловими інтерфейсами, як ProfiBus, FeldBus, LonWorks, Industrial Internet і т. ін. Застосування однопровідної технології, особливо при автоматизації розподіленого технологічного обладнання, високому рівні завад і жорстких вимогах до надійності (коли важливим є кожне показання і кожний керуючий вплив, а програмне забезпечення не може фільтрувати випадкові збої), безумовно, не виправдано.

Надійність 1-Wire-інтерфейсу апріорі набагато гірше промислових шин RS-485 і CAN, які мають спеціальні апаратні методи придушення завад і виключення помилок.

Можна сформулювати основні на сьогоднішній день обмеження для застосування систем, побудованих на базі 1-Wire-мереж в області автоматизації:

- необхідність в деяких випадках безперервної часової синхронізації або синхронної роботи окремих пристроїв у мережі (ця проблема може бути вирішена введенням в структуру шини мережі додаткової лінії для передачі сигналу загальної синхронізації);
- не дуже висока швидкість обміну інформацією і, як наслідок, неможливість високої динаміки при обслуговуванні швидких процесів в режимі реального часу (якщо контрольований швидкий процес є відносно нетривалим, локальний мікроконтролер в складі однопровідної шини може обслужити його, зберігши результуючі дані в буферній пам'яті, а потім вже здійснити їх передачу безпосередньо до ведучого);
- не дуже висока завадозахищеність (може бути значно поліпшена при ретельній прокладці магістралі, її екрануванні, гальванічному поділі абонентів, програмній фільтрації помилок і зниженні швидкості обміну);
- складність у реалізації мультимайстерного режиму роботи мережі (спеціалізований розгалужувач 1-Wire-мереж DS2409 вирішує проблему конфліктів між кількома ведучими на одній однопровідній шині).

Як видно із реакції на обмеження, які наведено у дужках, навіть очевидні для 1-Wire-мереж труднощі не є нездоланними. Більше того, існують підходи, що дозволяють органічно інтегрувати повільні однопровідні територіально розподілені структури до складу таких продуктивних мереж як CAN або Industrial Ethernet. Це можна досягти завдяки застосуванню спеціальних апаратно-програмних рішень, реалізованих на базі сучасних мікроконтролерів, а також унікального інструменту компанії Dallas Semiconductor – пристроїв TINI (Tiny Inter Net Interface).

TINI це програмно-апаратна платформа, розроблена компанією Dallas Semiconductor.

Апаратне забезпечення платформи TINI являє собою простий і недорогий мікроконтролерний модуль, який може застосовуватися для вирішення широкого кола завдань, і інтерфейсну плату, що забезпечує з'єднання модуля з зовнішніми пристроями. На базі багатофункціонального модуля TINI можуть бути побудовані конвертори мережевих протоколів, а також інтелектуальні контролери з широким набором інтерфейсів – 1-Wire, CAN, RS-232, Ethernet. Процесорне ядро з підтримкою мови Java забезпечує обробку даних, контроль, сполучення з іншими апаратними засобами і мережевими ресурсами.

Крім технології TINI існують і інші варіанти сполучення 1-Wire-мереж з Ethernet. Наприклад, модуль фірми M-TECH зі Словаччини, який дозволяє легко реалізовувати найрізноманітніші рішення з організації однопровідних розподілених систем дистанційного кліматичного контролю через Ethernet.

12.2.3 Особливості застосування 1-WIRE-мереж на прикладі системи моніторингу температури

Загальна характеристика та структура системи

Структурну схему системи вимірювання температури наведено на рисунку 12.2.

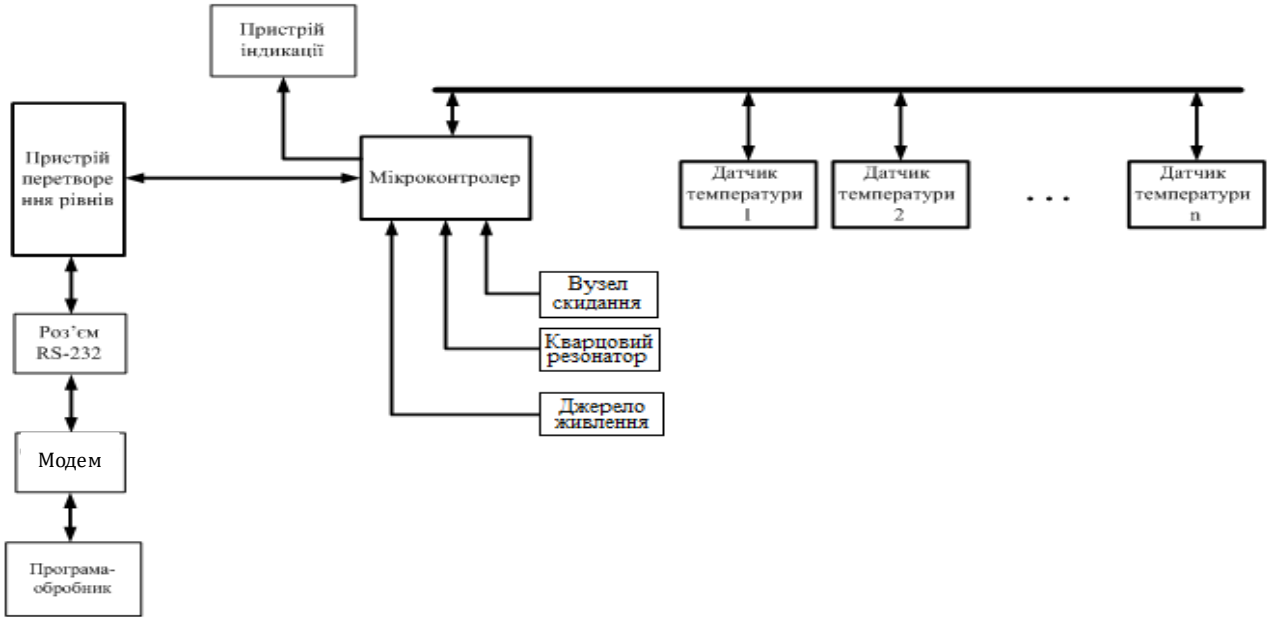


Рисунок 12.2 – Структурна схема системи вимірювання температури

Система призначена для вимірювання температури в приміщенні, де необхідно вимірювати температуру в декількох точках. Система являє собою набір датчиків, які можуть бути розміщено на відстані до 300 метрів. Система надає індикацію температури в самому приміщенні (виведення на LCD-дисплей) і передачу даних на керуючий комп'ютер для подальшої обробки.

В даній системі у якості ведучого в мережі 1-Wire може використовуватися, наприклад, мікроконтролер ATmega8535. Останній є 8-розрядним КМОН-мікроконтролером, виконаним за розширеною AVR-RISC-архітектурою. За рахунок виконання більшості інструкцій за один машинний цикл ATmega8535 досягає продуктивності 1 млн. операцій за секунду, що дозволяє проектувальникам систем оптимізувати співвідношення енергоспоживання і швидкодії [1; 6].

Загальна характеристика датчика температури

В якості ведених пристроїв 1-Wire-мережі (рисунок 12.2) виступають датчики температури DS18B20 з програмованим значенням розрядності вихідного коду від 9 до 12 біт, яку можна зберігати в EEPROM-пам'яті приладу. Датчик DS18B20 обмінюється даними по 1-Wire-шині і при цьому може бути, як єдиним пристроєм на лінії, так і працювати в групі. Всіма процесами на шині керує ведучий мікроконтролер.

Діапазон вимірювань температури: від -55 ° С до +125 ° С із точністю 0,5 ° С в діапазоні від -10 ° С до +85 ° С. На додаток, датчик DS18B20 може живитися напругою лінії даних ("паразитне живлення") за відсутності зовнішнього джерела напруги.

На рисунку 12.3 показано розташування контактів датчика.

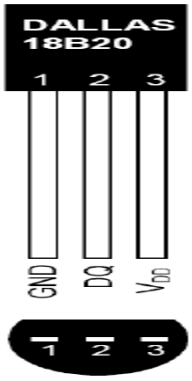


Рисунок 12.3 – Розташування контактів датчика

В таблиці 12.1 наведено призначення виводів датчика DS18B20.

Таблиця 12.1 – Призначення виводів датчика DS18B20

№	Назва виводу	Опис
1	GND	Спільний
2	DQ	Вивід введення/виведення даних (Input/Output pin). За цією лінією може подаватися живлення в режимі роботи з «паразитним живленням»
3	VDD	Вивід живлення. Для режиму роботи із паразитним живленням VDD необхідно з'єднати зі спільним проводом

Кожен датчик DS18B20 має унікальний 64-бітовий послідовний двійковий код адреси, який дозволяє спілкуватися з багатьма іншими датчиками DS18B20, які встановлено на одній шині. Такий принцип дозволяє використовувати один ведучий мікроконтролер, щоб контролювати багато датчиків DS18B20, розташованих на деякій відстані. Ця особливість дозволяє проектувати системи контролю температури в будівлях, устаткуванні чи машинах і т. ін.

На рисунку 12.4 наведено структурну схему датчика DS18B20.

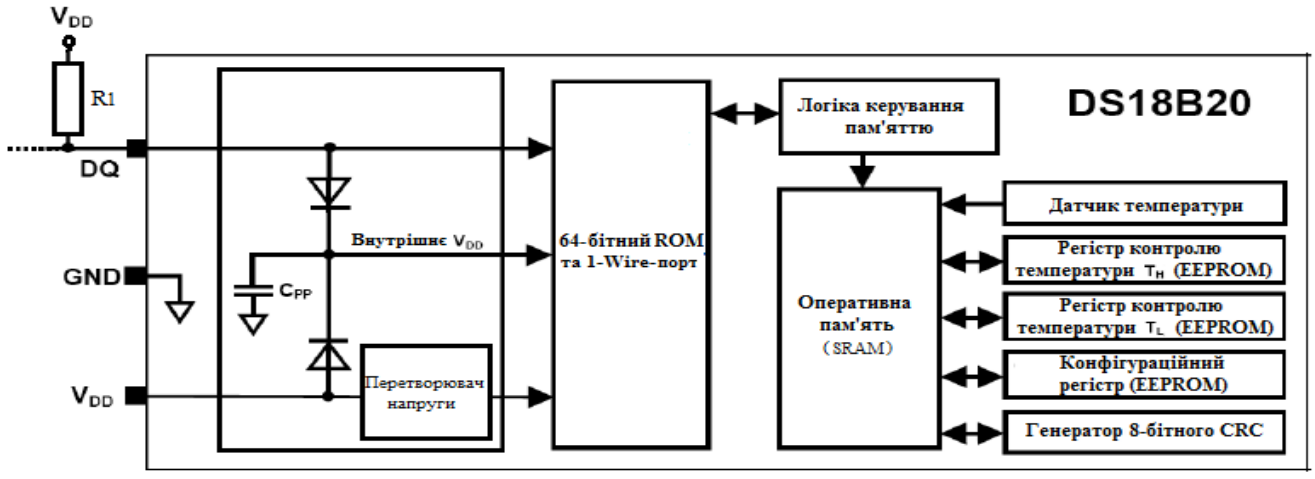


Рисунок 12.4 – Структурна схема датчика DS18B20

Схема містить 64-бітовий постійний запам’ятовуючий пристрій (ROM), в який записано унікальний двійковий код приладу.

В оперативній пам'яті (SRAM) знаходиться 2-байтовий регістр температури, який зберігає значення температури по закінченню процесу її вимірювання. EEPROM-пам’ять має два однобайтових регістра контролю температури: TH, TL і регістр конфігурації. Регістр конфігурації дозволяє користувачеві встановлювати розрядність цифрового перетворювача температури: 9, 10, 11, або 12 біт, що впливає на час вимірювання температури. Оскільки регістри TH, TL та регістр конфігурації є комірками енергонезалежної пам'яті даних – EEPROM, то вони зберігають дані, навіть коли прилад вимкнено.

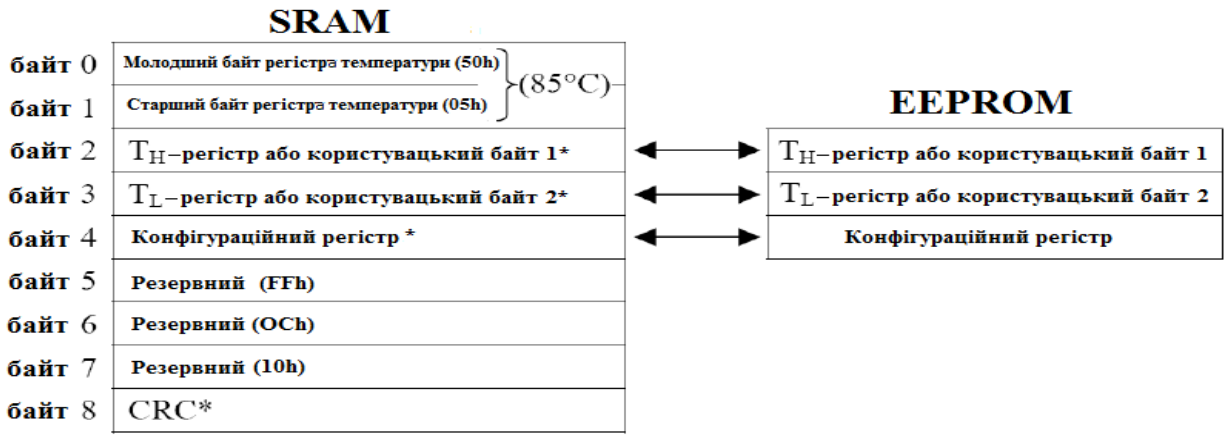
Датчик DS18B20 використовує 1-Wire-протокол. Спільна шина повинна бути підключена до джерела живлення через резистор, що підтягує, тому що всі пристрої, які зв'язані з шиною, використовують з'єднання через Z-стан або вивід відкритого стоку. Використовуючи цю шину, ведучий мікроконтролер (пристрій керування) ідентифікує і звертається до датчиків температури, використовуючи 64-бітовий код приладу. Оскільки кожен прилад має унікальний код, число приладів, до яких можна звернутися на одній шині, фактично необмежено.

Інша особливість датчика DS18B20 – здатність працювати без зовнішнього живлення. Ця можливість надається через резистор, що підтягує. Високий сигнал шини заряджає внутрішній конденсатор (Cpp), який живить прилад, коли на шині низький рівень. Цей метод носить назву «паразитне живлення». При цьому максимальна температура, яка вимірюється, становить: +100 ° C. Для розширення діапазону температур до +125 ° C необхідно використовувати зовнішнє живлення.

Організація пам'яті

Організацію пам'яті датчика DS18B20 показано на рисунку 12.5.

Пам'ять складається з оперативної енергозалежної пам'яті (SRAM) і енергонезалежної пам'яті EEPROM. Перші два байти пам'яті SRAM – це регістр температури, далі йдуть регістри стану «Аварія», які відображають верхню та нижню межу температури: TH і TL і регістр конфігурації. Якщо стан «Аварія» не використовується, то регістри TH і TL можуть бути комірками статичної оперативної пам'яті – SRAM.



(* - Стан після включення живлення залежить від значень, збережених в EEPROM)

Рисунок 12.5 – Організація пам’яті датчика DS18B20

Регістр температури

Перші два байти регістра температури зберігають поточну вимірювану температуру. При подачі живлення початкове значення температури: +85 градусів. Формат регістра представлено на рисунку 12.6, де S – знак температури (логічний 0 – додатна температура, логічна 1 – від’ємна), біти від 11-го до 4-го – ціла частина значення температури, біти від 3-го до 0-го – дробова частина.

	біт 7	біт 6	біт 5	біт 4	біт 3	біт 2	біт 1	біт 0
Молодший байт	2^3	2^2	2^1	2^0	2^{-1}	2^{-2}	2^{-3}	2^{-4}
	біт 15	**біт 14**	**біт 13**	**біт 12**	**біт 11**	**біт 10**	**біт 9**	**біт 8**
Старший байт	S	S	S	S	2^7	2^6	2^5	2^4

Рисунок 12.6 – Формат регістра температури

Формування стану «Аварія»

Після того, як датчик DS18B20 виконає температурне перетворення, поточне температурне значення порівнюється зі значеннями, які записано в регістрах TH і TL (рисунок 12.7).

біт 7	біт 6	біт 5	біт 4	біт 3	біт 2	біт 1	біт 0
S	2^6	2^5	2^5	2^5	2^2	2^1	2^0

Рисунок 12.7 – Формат регістрів TH та TL

Біт знака (S) вказує на додатне чи від’ємне значення температури: для додатних чисел S = 0 і для від’ємних чисел S = 1.

Регістри TH і TL є комірками енергонезалежної пам’яті даних EEPROM. Таким чином, вони зберігають дані, коли пристрій знеструмлено. Звернутися до TH і TL можна через байти 2 і 3 SRAM.

Для порівняння використовується тільки ціле значення регістра температури: біти 4…10.

Якщо вимірювана температура нижче або дорівнює TL або вище або дорівнює TH, формується стан «Аварія» і встановлюється прапорець «Аварія». Цей прапорець оновлюється після кожного наступного температурного перетворення. Якщо при наступному перетворенні стан «Аварія» пропаде, то прапорець буде скинуто.

Ведучий пристрій може перевірити стан «Аварія» всіх датчиків DS18B20 на шині, подаючи команду «Пошук Аварії» [ECh]. Будь-який датчик DS18B20 з встановленим прапорцем «Аварія» відповість на цю команду. Таким чином, ведучий пристрій може визначити, які датчики DS18B20 знаходяться в стані «Аварія». Якщо змінюються значення регістрів TH або TL, то необхідно запустити нове температурне перетворення, щоб виконалась перевірка умови контролю температури, що задано в регістрах TH або TL.

Конфігураційний регістр

Четвертий байт SRAM-пам'яті виконує функцію конфігураційного регістра. В ньому задається температурна розрядність датчика. Початкове значення регістра залежить від вмісту EEPROM-пам'яті. Формат конфігураційного регістра представлено на рисунку 12.8, де R1, R0 – біти, що задають температурну розрядність термометра, від якої залежить максимальний час перетворення температури (таблиця 12.2).

біт 7	біт 6	біт 5	біт 4	біт 3	біт 2	біт 1	біт 0
0	R1	R0	1	1	1	1	1

Рисунок 12.8 – Формат конфігураційного регістра

Таблиця 12.2 – Залежність розрядності та максимального часу перетворення від значень бітів R1, R0

R1	R0	Розрядність	Максимальний час перетворення
0	0	9 біт	93,75 мс
0	1	10 біт	187,5 мс
1	0	11 біт	375 мс
1	1	12 біт	750 мс

Контрольна сума циклічного надлишкового коду (CRC)

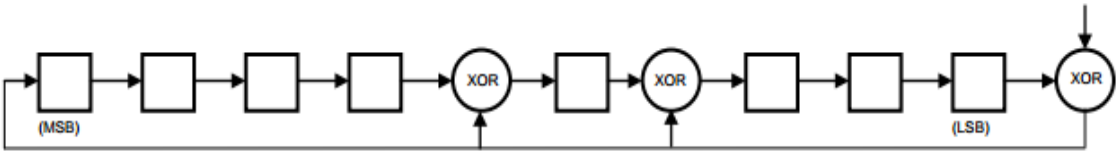
Контрольна сума або CRC-сума – це байт, значення якого зберігається останнім в SRAM-пам'яті і обчислюється за спеціальним алгоритмом на основі значення всіх 7-ми попередніх байтів. Алгоритм підрахунку такий, що якщо всі байти передані-прийняті без спотворень (а спотворення цілком можливі, якщо згадати характер апаратної реалізації інтерфейсу), прийнятий байт контрольної суми обов'язково збігається з розрахованим в пристрої значенням. Тобто при реалізації програмного алгоритму обміну інформацією треба при передачі та прийомі байтів підраховувати їх контрольну суму за встановленим алгоритмом, а потім або передати отримане значення (якщо ми вели передачу адреси/даних), або порівняти розраховане значення з прийнятим значенням CRC.

Тільки при збігу обох CRC мікроконтролер або пристрій вважають прийняті дані достовірними. В іншому випадку продовження обміну неможливе. Очевидно, що алгоритм підрахунку CRC повинен бути однаковим як для МК, так і для будь-якого пристрою.

В системі 1-Wire проходить перевірка CRC, використовуючи утворюючий многочлен:

$$x^8 + x^5 + x^4 + 1 .$$

Схема формування CRC при цьому многочлені наступна:



Наприклад, реальна адреса датчика DS18B20 є «2830C5B80000008E». Останній байт адреси (8E) – контрольна сума.

Примітка: DallasSemiconductor розраховує CRC для оберненої послідовності. Тобто байти розташовуються у тому порядку, у якому вони були прийняті: перший байт стає останнім, останній – першим. Для даного випадку маємо комбінацію:

8E 00 00 00 B8 C5 30 28.

У двійковій системі комбінація виглядає так:

10001110 000000000000000000000000101110001100010100110000000101000.

Дзеркально відображаємо цю комбінацію, щоб полегшити обчислення, інакше, нам довелося б ділити комбінацію з кінця, зсуваючи недостаючі біти вліво (а не вправо, як при звичайному діленні). Відкидаємо останній байт та записуємо адресу датчика у двійковій системі:

000101000000110010100011000111010000000000000000000000000000000.

Дописуємо в кінець 8 нулів (кількість останніх відповідає ступеню утворюючого многочлена). Утворюючий многочлен можна представити як $2^8 + 2^5 + 2^4 + 1 = 305$, що у двійковому вигляді дорівнює 100110001.

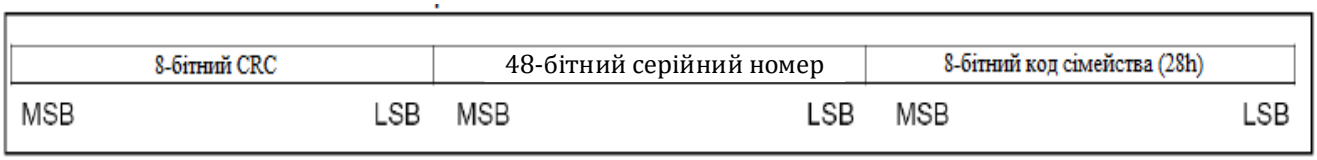
Виконуємо ділення адреси датчика (з вісьма нулями в кінці) на утворюючий многочлен:

000101000000110010100011000111010000000000000000000000000000000	100110001
…	
01110001	

Перевертаємо комбінацію 01110001та отримаємо10001110, що дорівнює 8E у 16-ковій системі числення (збігається з останнім контрольним байтом нашої адреси). Приклад розрахунку CRC для «28 30 C5 B8 00 00 00 8E» наведено у [6].

Формат коду датчика

Кожен датчик DS18B20 містить унікальний 64-бітовий код, збережений в ROM-пам'яті. Формат коду подано на рисунку 12.9.



Примітка: LSB – Least Significant Bit (МЗР – молодший значущий розряд)

MSB – Most Significant Bit (СЗР – старший значущий розряд)

Рисунок 12.9 – Формат коду датчика

Молодші 8 біт коду містять код сімейства 1-Wire. Для датчика DS18B20 це 28h. Наступні 48 біт містять унікальний серійний номер датчика. Старші 8 біт містять CRC-байт, який обчислено від перших 56 біт коду ROM. Саме завдяки унікальності коду забезпечується адресація пристроїв 1-Wire.

Принципова схема пристрою вимірювання температури

На рисунку 12.10 наведено принципову схему пристрою вимірювання температури, в якій використано датчик DS18B20. Принципова схема включає в себе мікроконтролер ATmega8535 (DD3), який підключено до шини з двома датчиками DS18B20 (DD1, DD2), перетворювачі рівнів MAX232 (DD4, DD5) і LCD-індикатор типу HD44780 (DD6). При старті системи контролер опитує шину на предмет наявності підключених пристроїв. Якщо замкнений хоча б один з ключів SB1 або SB2, мікроконтролер починає опитування датчиків, виводить значення температури для кожного з них на LCD-індикатор і подає сигнал на перетворювачі рівнів. Останні виконано на мікросхемах MAX232 (DD4, DD5). Вони перетворюють цифрові сигнали від модуля УАПП мікроконтролера в рівні інтерфейсу RS-232, які через його дев`ятиконтактний роз`єм (XS2) подаються, наприклад, до модему. Останній може зв`язувати систему з віддаленим керуючим пристроєм, де відбувається подальша обробка значень датчиків програмою високого рівня.

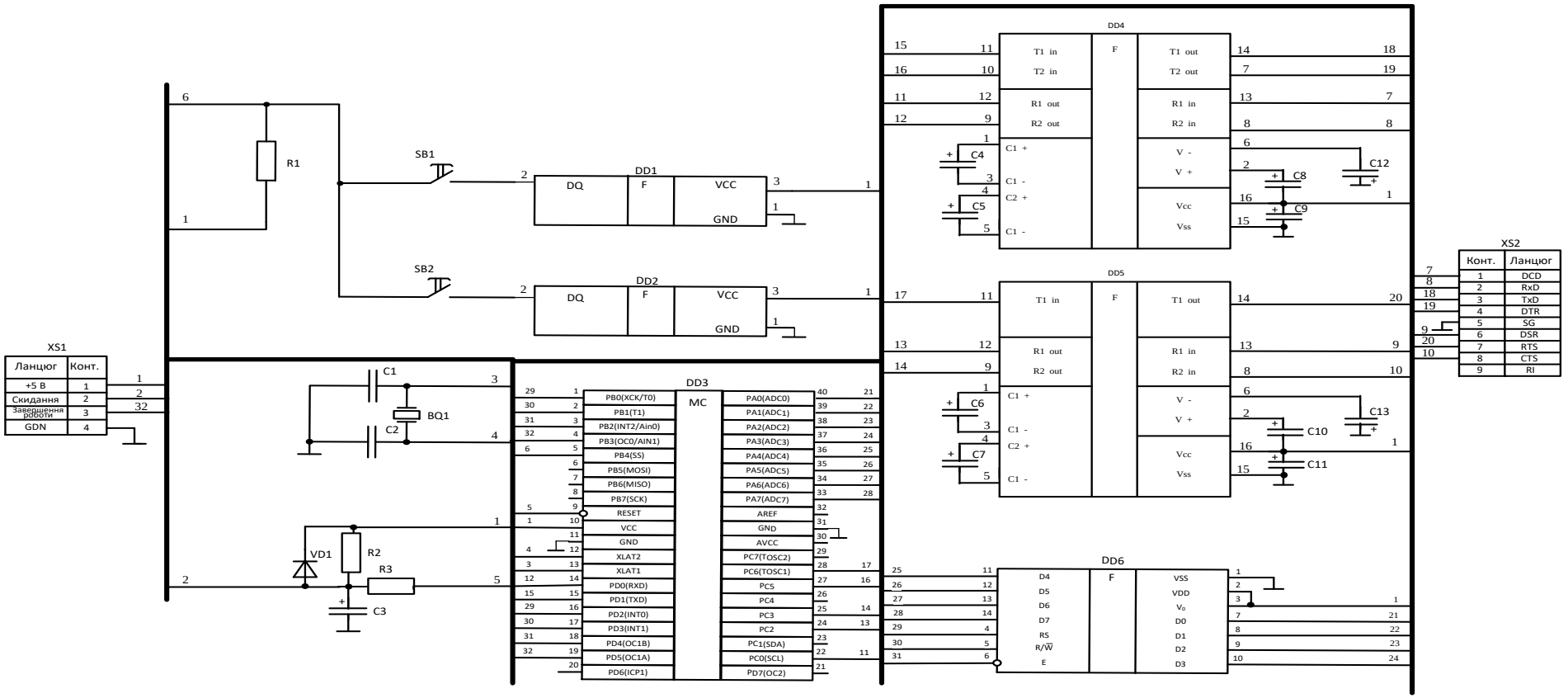


Рисунок 12.10 – Принципова схема пристрою вимірювання температури

12.2.4 Команди керування пристроями 1-WIRE

Нижче розглянуто принципи адресації пристроїв 1-WIRE та керування ними. Кожен пристрій 1-Wire має унікальний ідентифікаційний 64-бітовий номер, який був запрограмований на етапі виробництва мікросхеми. Унікальний – це означає, що фірма-виробник гарантує, що не знайдеться двох мікросхем з однаковим ідентифікаційним номером (принаймні протягом кількох десятків років при існуючих темпах виробництва).

При розгляді протоколу обміну будемо виходити з принципу, що на шині 1-Wire є більше одного пристрою. У цьому випадку перед мікроконтролера, який керує мережею, постають 2 проблеми: визначення кількості наявних пристроїв і вибір (адресація) одного конкретного з них для обміну даними.

Команди керування пристроями діляться на дві групи:

– ROM-команди – команди для роботи з адресами пристроїв (пошук адреси, зчитування адреси, вибір адреси, тощо);

– функціональні команди – команди, наявність яких вимагає виконання відповідних дій з боку кінцевого пристрою.

Послідовність операцій для доступу до датчика DS18B20 наступна:

– ініціалізація;

– подача ROM-команди;

– подача функціональної команди DS18B20.

ROM-команди

Деякі з важливих загальних ROM-команд наведено в таблиці 12.3.

Команда SEARCH ROM

Для того, щоб дізнатися адресу 1-Wire пристроїв, підключених до шини, використовується команда SEARCH ROM, яка має код – 0xF0.

Будь-який 1-Wire пристрій має 64-розрядний унікальний код. Цей код зберігається в ROM-пам’яті і використовується для адресації пристрою на шині. Молодший байт коду визначає сімейство, до якого належить 1-Wire-пристрій

(наприклад, код датчиків DS18B20 – 0x28). Наступні 6 байт – серійний номер пристрою. Старший байт – байт CRC, який обчислено згідно значень перших 7 байтів.

Таблиця 12.3 – ROM-команди пристроїв 1-Wire

Команда	Значення байта	Опис
SEARCH ROM	0xF0	Пошук адрес – використовується при універсальному алгоритмі визначення кількості та адрес підключених пристроїв
READ ROM	0x33	Зчитування адреси пристрою – використовується для визначення адреси єдиного пристрою на шині
MATCH ROM	0x55	Вибір адреси – використовується для звернення до конкретної адреси пристрою з багатьох підключених
SKIP ROM	0xCC	Ігнорування адреси – використовується для звернення до всіх або єдиного пристрою на шині, при цьому адреса пристрою ігнорується (можна звертатися до невідомого пристрою)

Отже, в загальному випадку є шина 1-Wire, на якій присутній один ведучий пристрій і невідома кількість ведених пристроїв (не обов'язково однакових). Згідно до схеми включення, шина підтягується до плюса живлення резистором, так, що на ній при простої завжди присутня «логічна 1».

Весь обмін даними з пристроями починається з того, що ведучий пристрій відсилає в шину імпульс скидання RESET. Ведені пристрої, отримавши імпульс скидання, відповідають в шину імпульсом присутності PRESENCE. Оскільки всі пристрої присутні на одній шині, всі вони відповідають в шину одночасно. Але на даному етапі їх відповідь (PRESENCE) збігається.

Після того, як сигнал PRESENCE був отриманий, ведучий пристрій починає пошук потрібного веденого, відправляючи в шину команду 0xF0 – SEARCH ROM.

Отримавши команду SEARCH ROM, кожен ведений пристрій передає в шину перший біт свого 64-бітного коду з ROM-пам’яті і інверсію цього біта, так що ведучий отримує логічне «І» перших бітів кодів і логічне «І» їх інверсій.

На підставі отриманого логічного «І» перших бітів та логічного «І» інверсій проводиться аналіз:

- якщо отримане логічне «І» перших бітів дорівнює «1» та логічне «І» інверсій дорівнює «1», це означає, що на лінії немає готових пристроїв, тобто ніхто не відповів;
- якщо логічне «І» перших бітів дорівнює «1», логічне «І» інверсій дорівнює «0» або навпаки – це означає, що у всіх пристроїв на шині цей біт збігається;
- якщо логічне «І» перших бітів дорівнює «0» та логічне «І» інверсій дорівнює «0» – це означає, що у пристроїв перший біт коду не збігається, тобто у деяких пристроїв він дорівнює «1», а у деяких – «0».

В першому випадку пошук припиняється тому, що немає активних пристроїв, які під’єднано до шини.

В другому випадку перший біт однаковий у всіх ведених пристроїв, тому він записується ведучим в якості першого біта коду і відправляється назад в шину для підтвердження.

В третьому випадку перший біт у ведених пристроїв на шині не збігається, і ведучий відправляє в шину «0». Всі ведені пристрої, у яких перший біт дорівнює «1», відключаються від шини і більше не реагують на команди в шині до наступного скидання, тобто більше не беруть участі в пошуку. Залишаються ті, у яких перший біт дорівнював «0». Ведучий записує «0» в якості першого біта коду пристрою, а також записує позицію, де сталося це розходження. Аналогічно аналізуються наступні біти адрес.

При цьому запам`ятовується позиція (номер) біта, де було останне розходження. Це буде використано ведучим при наступному пошуку. Тобто при наступному пошуку ведучий, аналізуючи дані, бачить позицію (номер) біта останнього розходження, де було записано та повернуто «0», та видає в шину «1», тим самим, відкидаючи попередні пристрої, адреси яких вже було записано. Ця зміна «напряму» з «0» на «1» відбувається саме на останній відмінності попереднього циклу пошуку. При цьому відмінності, які відбулися перед останньою, ігноруються – обирається той же «напрям» що й раніше. Це потрібно, для того, щоб не загубити жодного пристрою. Алгоритм пошуку адрес та приклад пошуку адрес для чотирьох пристроїв з 4-бітовою адресою, якщо ведучим є мікроконтролер, наведено в [6].

Отже, весь пошук зводиться до послідовності:

- прийняти біт коду пристрою і його інверсію;
- на підставі цього і останньої відмінності попереднього пошуку прийняти рішення і видати в шину «напрям» пошуку;
- запам'ятати отриманий біт та повернути в шину його значення;
- якщо була відмінність – запам'ятати позицію біта, на якій вона відбулася;
- перейти до прийому наступного біта.

Повторюється ця послідовність для кожного з 64-х біт. В результаті отримується 64-бітний код одного з пристроїв. Потім пошук починається ще раз, вибравши інший «напрям». Отримується ще один 64-бітний код пристрою. Ця процедура виконується до тих пір, поки на шині не залишиться невідомих пристроїв [6].

Команда READ ROM

Для зчитування 64-бітового коду веденого пристрою використовується команда READ ROM, яка має код – 0x33.

Команда може використовуватися також, якщо до шини підключений тільки один ведений пристрій.

Послідовність роботи:

– ініціалізація;
– ведучий видає в шину команду READ ROM;
– ведучий приймає вісім байт коду веденого пристрою.

Команда SKIP ROM

Для звернення до єдиного або всіх 1-Wire-пристроїв, що підключено до шини, використовується команда SKIP ROM, яка має код – 0xCC. Наприклад, ця команда застосовується, щоб запустити температурне перетворення відразу всіх датчиків DS18B20.

Послідовність роботи:

– ініціалізація;
– ведучий видає в шину команду SKIP ROM;
– ведучий видає в шину команду CONVERT T.

Ця команда використовується також, якщо до шини підключений тільки один ведений пристрій (не потрібна адресація).

Команда MATCH ROM

Для звернення до конкретного веденого пристрою застосовується команда MATCH ROM, яка має код – 0x55.

Ведучий видає в шину команду MATCH ROM, а потім 64-бітний код пристрою, до якого він звертається. Відповідати на функціональну команду буде тільки той пристрій, який «розпізнає» свою адресу (тобто отриманий з шини 64-бітний код збігається з 64-бітним кодом, що зберігається в ROM-пам`яті пристрою). Решта пристроїв не будуть відсилати в шину дані, поки не отримають імпульс скидання.

Послідовність роботи:

– ініціалізація;
– ведучий видає в шину команду MATCH ROM;
– ведучий видає в шину вісім байт коду веденого пристрою;
– ведучий видає в шину функціональну команду.

Функціональні команди

Функціональні команди дозволяють мікроконтролеру читати і записувати інформацію в оперативну пам’ять датчика DS18B20, запускати температурне перетворення датчика, визначати його режим живлення.

Команда CONVERT T

Команда має код – 0x44, та призначена для запуска процесу температурного перетворення датчика. Значення температури зберігається в перших двох байтах статичної оперативної пам’яті датчика. Час виконання перетворення залежить від встановленої в конфігураційному регістрі розрядності.

Після запуску температурного перетворення ведучий відстежує, коли воно буде виконано. Для цього він формує в шині тайм-слоти зчитування і чекає, коли датчик передасть «логічну 1» на підтвердження. Доки операція виконується, в шині буде логічний нуль.

Послідовність роботи у випадку одного датчика:

- ініціалізація;
- ведучий видає в шину команду SKIP ROM;
- ведучий видає в шину команду CONVERT T;
- ведучий формує тайм-слоти зчитування і очікує, коли з шини надійде у відповідь «логічна 1».

Команда WRITE SCRATCHPAD

Команда має код – 0x4E, та призначена для запису в статичну оперативну пам'ять датчика DS18B20 трьох байтів: T_H, T_L та конфігураційного байта (2-й, 3-й, 4-й байти статичної оперативної пам’яті відповідно).

Послідовність роботи у випадку одного датчика:

- ініціалізація;
- ведучий видає в шину команду SKIP ROM;
- ведучий видає в шину команду WRITE SCRATCHPAD;
- ведучий передає в шину по черзі 3 байти – TH, TL і конфігураційний байт.

Команда READ SCRATCHPAD

Команда має код – 0xBE, та призначена для зчитування вмісту оперативної пам’яті датчика DS18B20. Дані передаються з 0-го по 8-й байт, починаючи з молодшого біта. Ведучий може в будь-який момент видати в шину сигнал скидання та припинити прийом даних. Наприклад, він це робить якщо потрібні тільки перші два байти оперативної пам’яті.

Послідовність роботи у випадку одного датчика:

– ініціалізація;
– ведучий видає в шину команду SKIP ROM;
– ведучий видає в шину команду READ SCRATCHPAD;
– ведучий приймає з шини 9 байт статичної оперативної пам’яті датчика.

Команда COPY SCRATCHPAD

Команда має код – 0x48 та призначена для копіювання 2-го, 3-го і 4-го байтів статичної оперативної пам’яті датчика (T_L, T_H і конфігураційний байт відповідно) в EEPROM-пам`ять датчика.

Послідовність роботи у випадку одного датчика:

– ініціалізація;
– ведучий видає в шину команду SKIP ROM;
– ведучий видає в шину команду COPY SCRATCHPAD.

Команда RECALL E2

Команда має код – 0xB8, та призначена для запису у 2-й, 3-й та 4-й байти статичної оперативної пам’яті датчика (T_L, T_H і конфігураційний байт відповідно) відповідних значень, які збережено в EEPROM-пам`яті датчика.

Ведучий відслідковує процес виконання цієї операції, формуючи в шині тайм-слоти зчитування. Доки операція виконується, в шині буде логічний нуль, коли операція завершиться – логічна одиниця.

Послідовність роботи у випадку одного датчика:

– ініціалізація;
– ведучий видає в шину команду SKIP ROM;

– ведучий видає в шину команду RECALL E2;
– ведучий формує тайм-слоти зчитування і очікує, коли з шини надійде у відповідь «логічна 1».

Команда READ POWER SUPPLY

Команда має код – 0xB4 та призначена для визначення, як живляться датчики DS18B20. Якщо вони живляться від сигнальної лінії (режим «паразитного живлення»), то під час тайм-слота зчитування шина буде утримуватися в нулі. Якщо використовується зовнішнє джерело живлення, шина буде підтягнута до одиниці.

Послідовність роботи у випадку одного датчика:

– ініціалізація;
– ведучий видає в шину команду SKIP ROM;
– ведучий видає в шину команду READ POWER SUPPLY;
– ведучий формує тайм-слоти зчитування і зчитує стан шини, з якого визначає вид живлення.

12.3 Схема алгоритму роботи системи вимірювання температури

Схему алгоритму роботи системи вимірювання температури при використанні датчиків 1-WIRE наведено у [6]. Ця схема включає: схему алгоритму ініціалізації мікроконтролера; схему алгоритму ініціалізації LCD; схему алгоритму ініціалізації УАПП; схему алгоритму роботи головної підпрограми, що виконується під час роботи системи; схему алгоритму пошуку пристроїв на шині; схему алгоритму пошуку адреси пристрою; схему алгоритму процедури отримання значень температури від датчиків; схеми алгоритмів перевірки CRC та розрахунку контрольної суми.

Також у [6] наведено програму, яка реалізує алгоритм роботи мовою С.

12.4 Моделювання мережі 1-WIRE

12.4.1 Опис моделі

Для моделювання мережі 1-Wire може використовуватися пакет програмного забезпечення Proteus 8.6. Нижче на рисунку 12.11 наведено схему моделювання мережі 1-Wire у цьому пакеті.

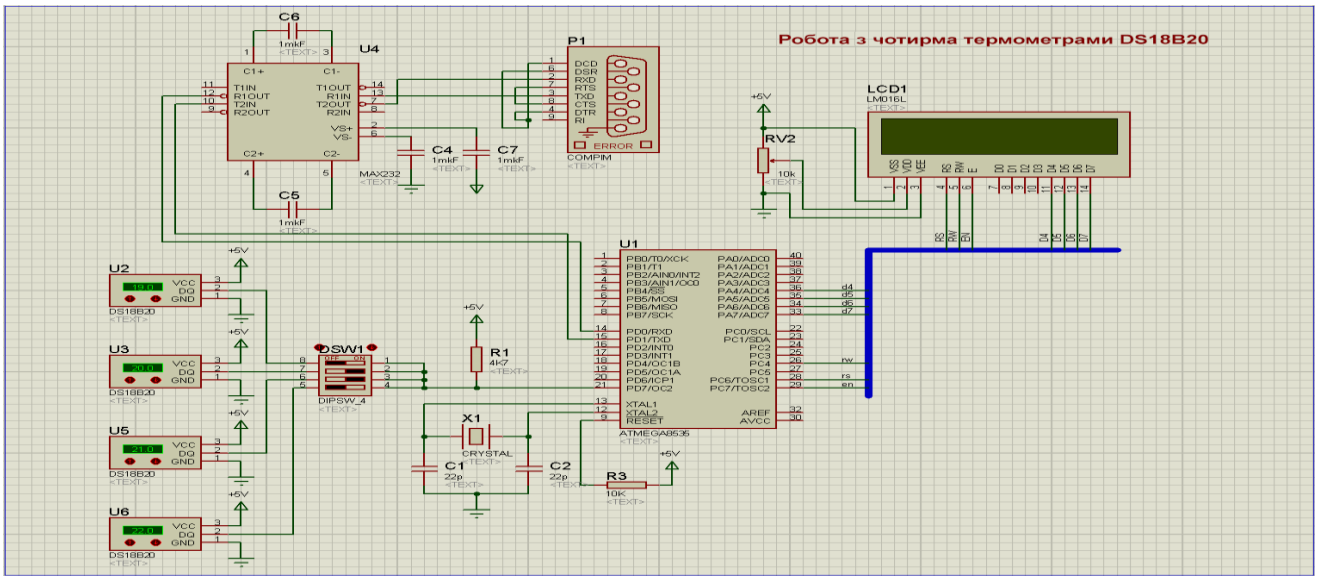


Рисунок 12.11 – Схема моделювання мережі 1-Wire

В моделі використано чотири датчики температури – DS18B20 (U2… U6), які підключено до входу PD7 мікроконтролера. Цей вхід підтянуто до живлення: +5В за допомогою резистора R1. Чотирьохпозиційний перемикач виконує динамічне підключення датчиків до шини.

Для зміни температури на кожному з датчиків, використовуються кнопки під дисплеєм встановлення температури (рисунок 12.12).

Для режиму роботи із паразитним живленням необхідно вивід VCC з'єднати зі спільним проводом GND (рисунок 12.13).

Через те, що на схемі можуть бути присутні кілька датчиків, то їх підключення або відключення здійснюється через «світч», який має 2 положення – «on» та «off» відповідно. На рисунку 12.14 наводиться «світч», де всі 4 датчики не підключено.

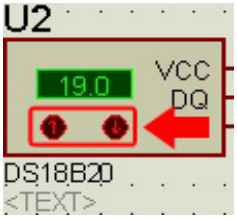


Рисунок 12.12 – Кнопки встановлення температури

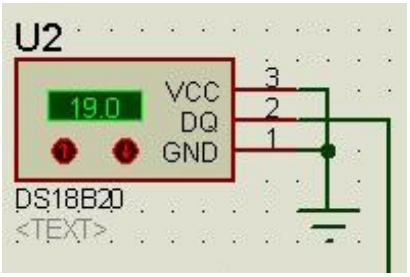


Рисунок 12.13 – Приклад датчика з паразитним живленням

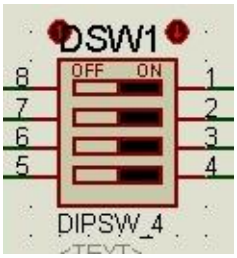


Рисунок 12.14 – Приклад «світча» з вимкненими датчиками

На старті роботи системи контролер опитує шину на наявність підключених датчиків. Якщо підключено хоча б один датчик, контролер починає опитувати датчики, виводити значення температури від кожного з них на LCD-дисплей та подавати сигнал на перетворювач рівнів MAX232.

Після перетворювача сигнал поступає на 9-ти контактний роз`єм RS232 та далі може подаватися за допомогою модема до віддаленого керуючого пристрою, де виконується подальша обробка значень температури програмою високого рівня.

Для запуску моделювання необхідно користуватися панеллю моделювання (рисунок 12.15).

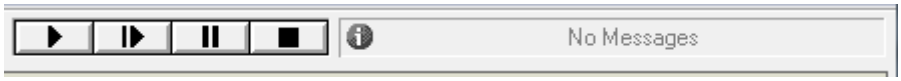


Рисунок 12.15 – Панель для запуску моделювання

12.4.2 Схема алгоритму роботи моделі

Нижче наведено схему алгоритму роботу моделі (рисунок 12.16).

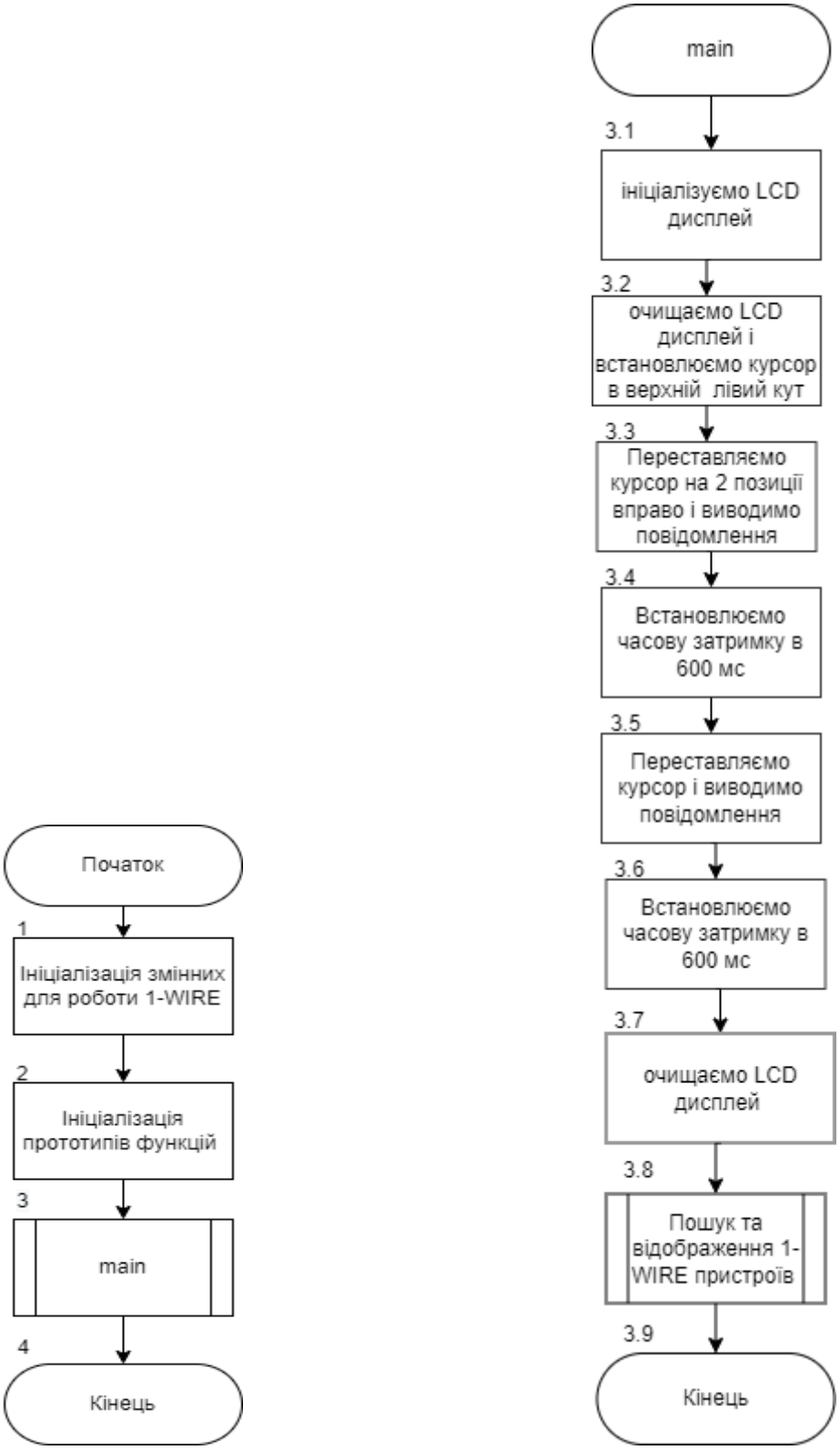
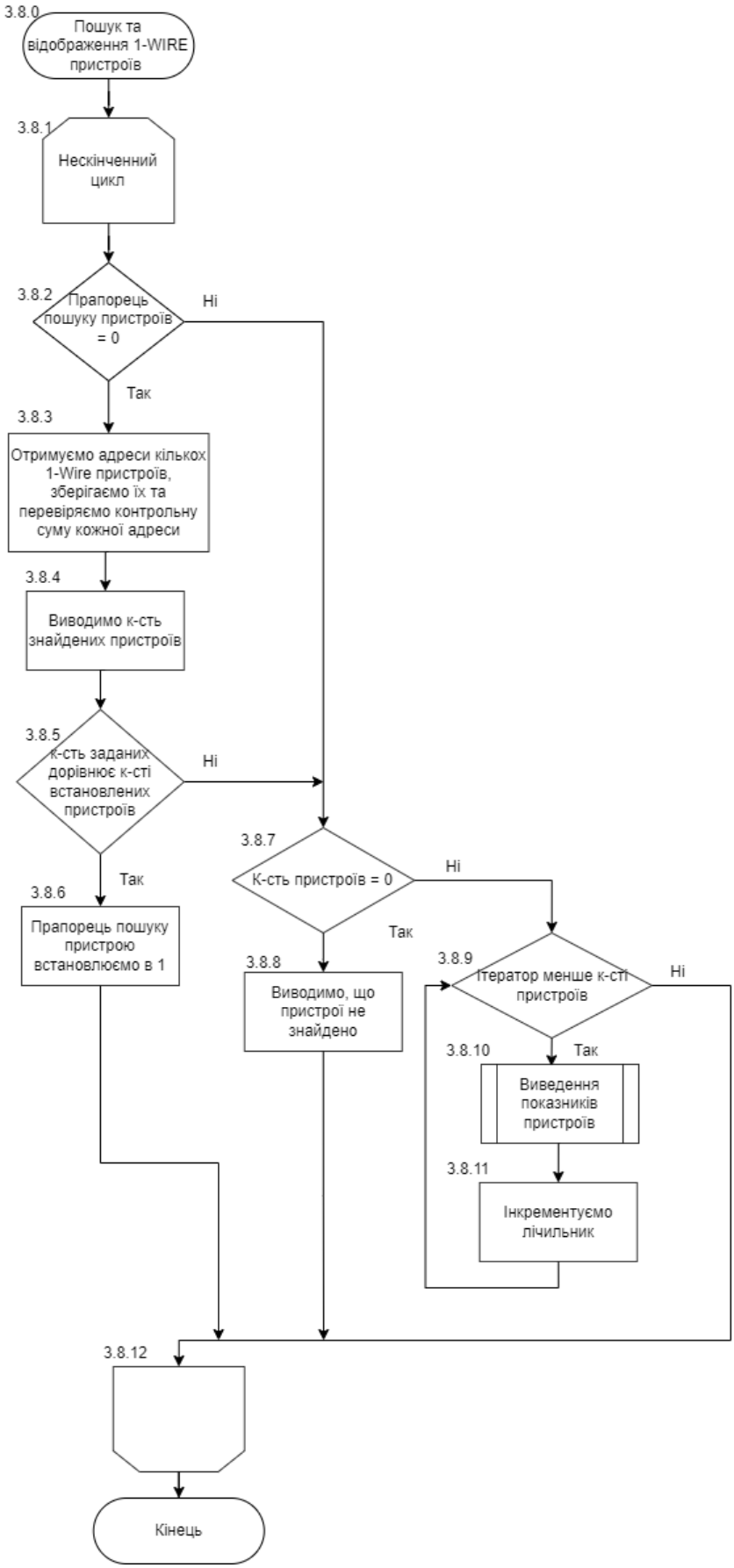
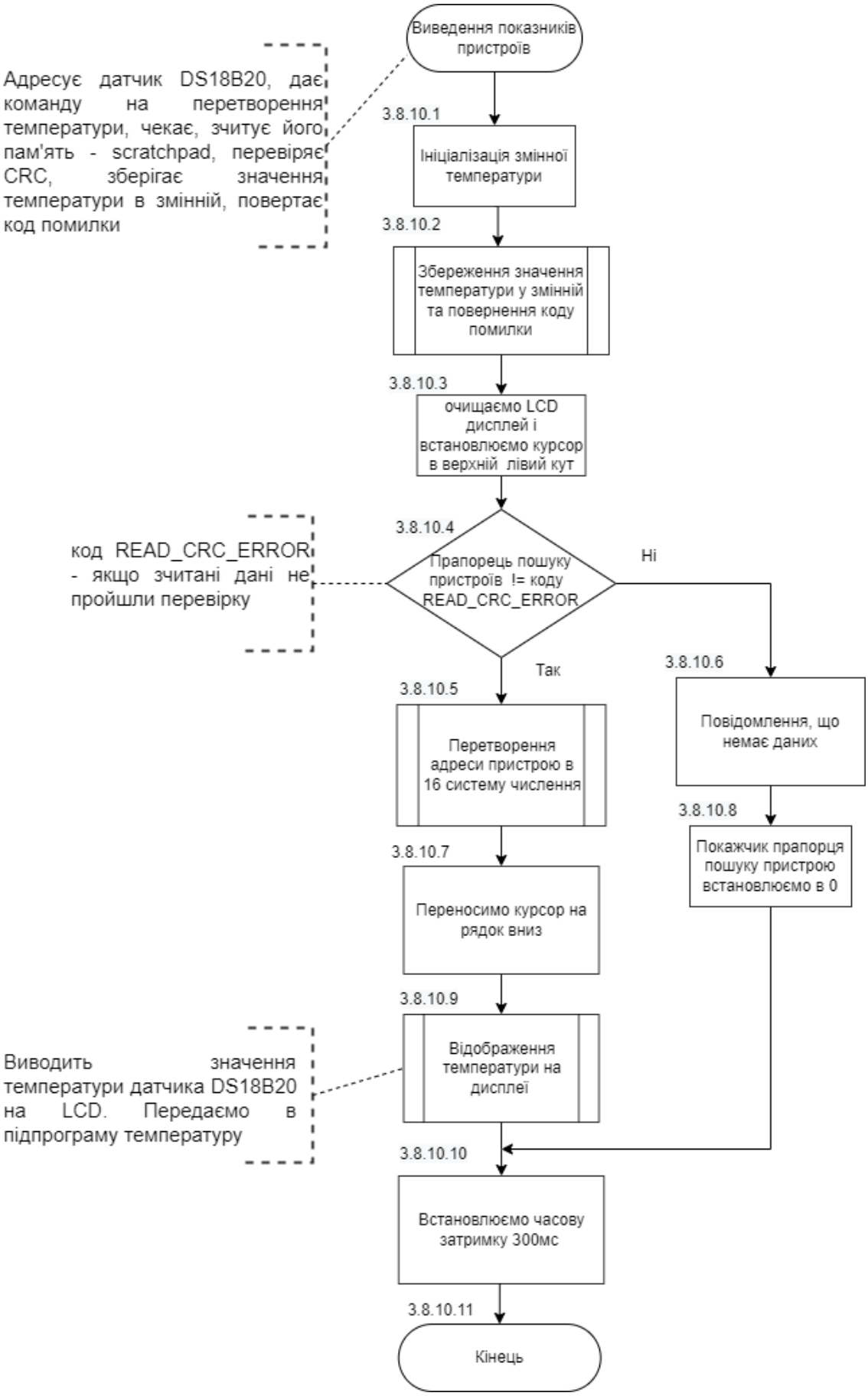


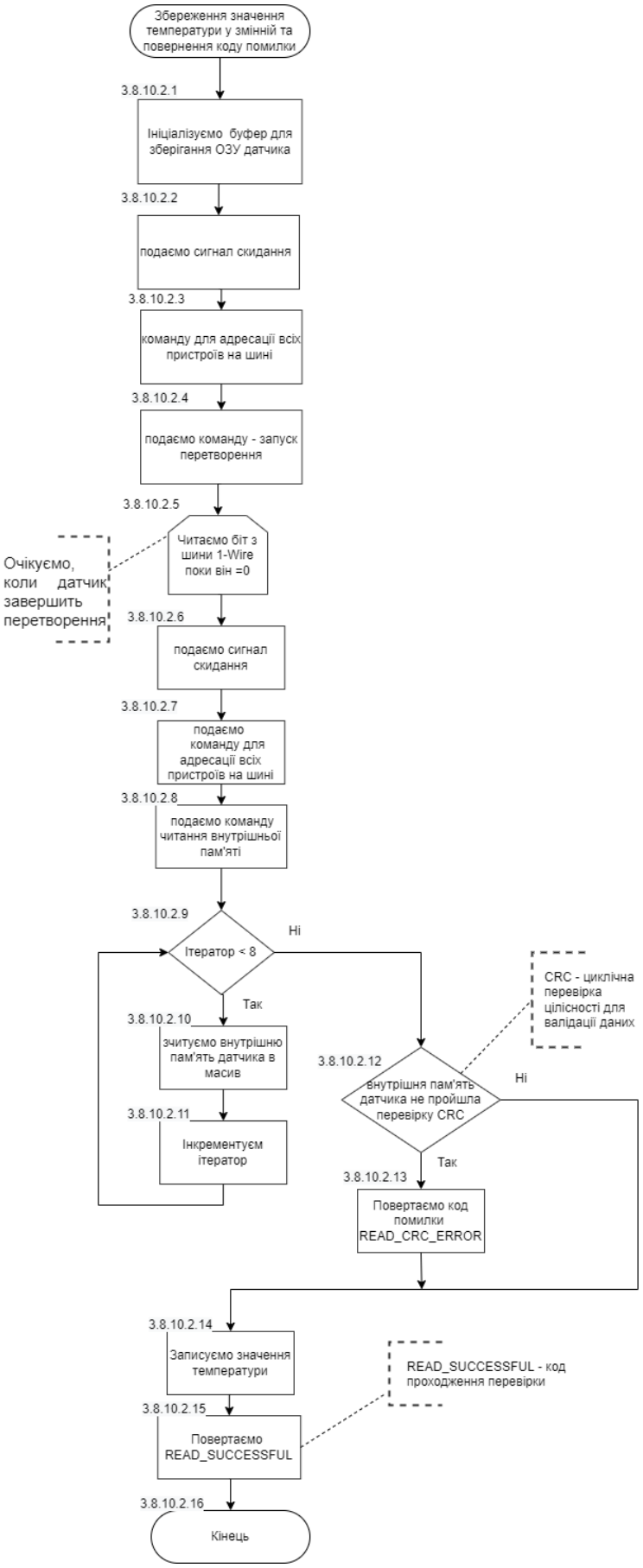
Рисунок 12.16 – Схема алгоритму роботи моделі



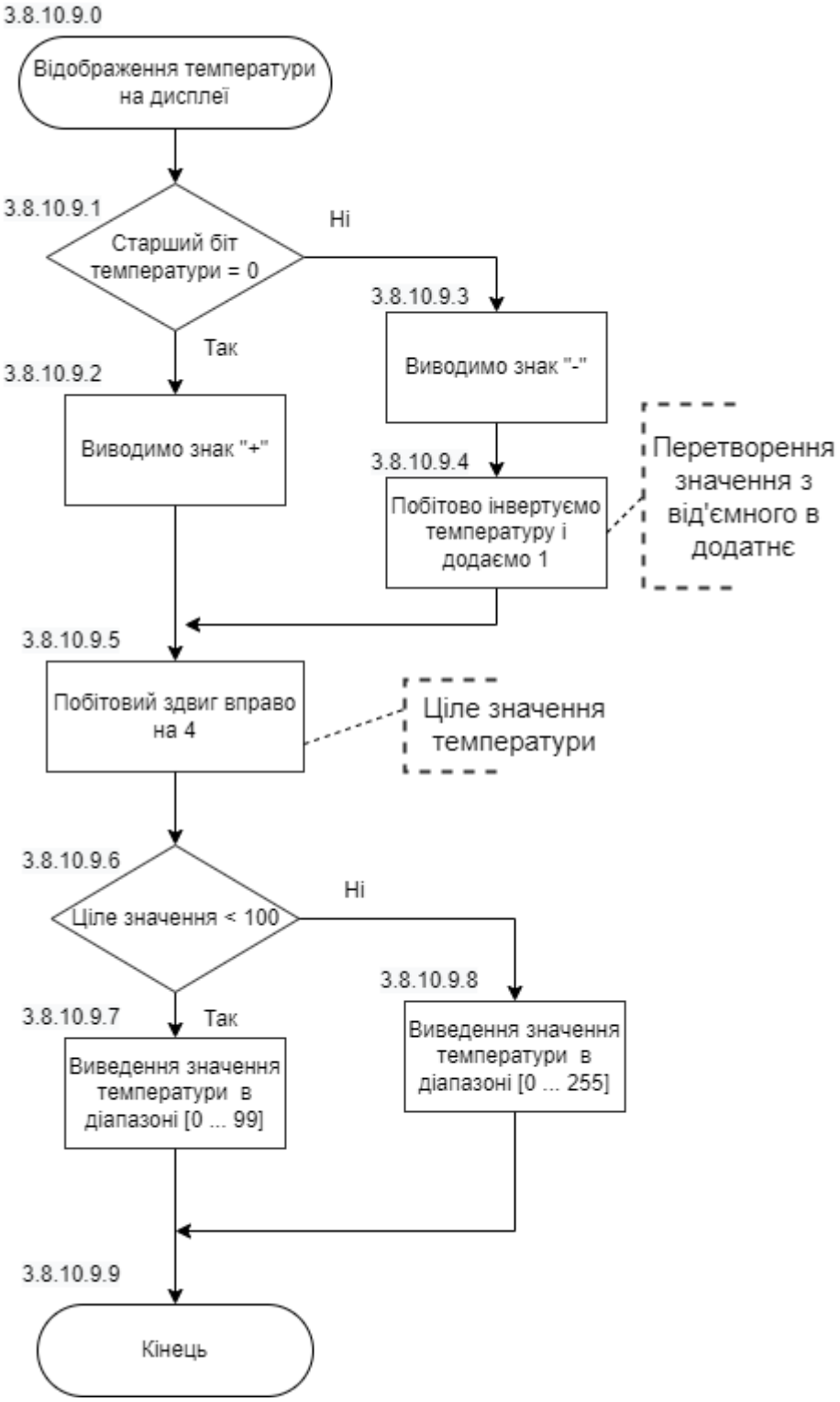
Продовження рисунку 12.16 (стор. 349)



Продовження рисунку 12.16 (стор. 349, 350)

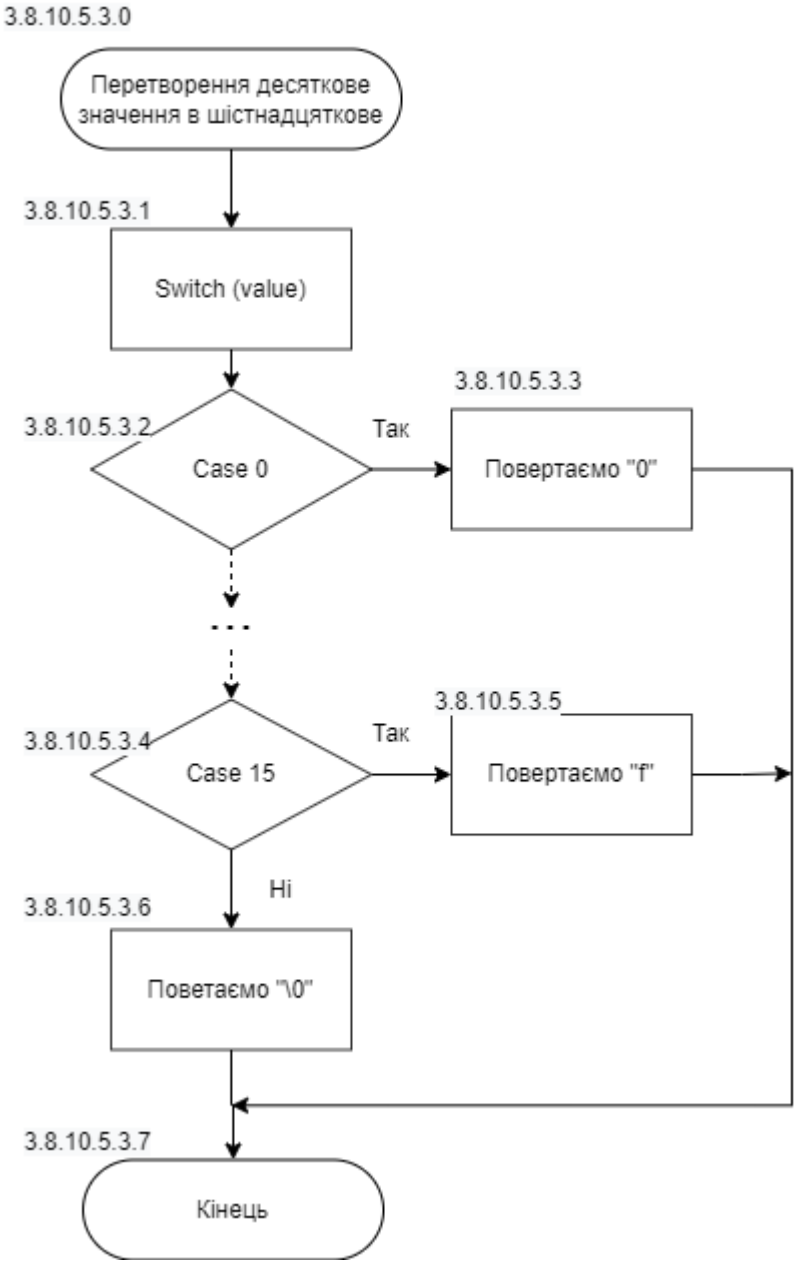


Продовження рисунку 12.16 (стор. 349…351)



Продовження рисунку 12.16 (стор. 349… 352)

Продовження рисунку 12.16 (стор. 349…353)



Закінчення рисунку 12.16 (стор. 349…354)

12.4.3 Робоча програма

Нижче наведено робочу програму, яка реалізує наведений вище алгоритм мовою С. В ній для полегшення вивчення програми проти окремих команд проставлено номера частин алгоритму, які вони реалізують у мові програмування.

```
//*****************************************************************
//
//  Author(s)...: Pashgan    http://ChipEnable.Ru
//
//  Target(s)...: ATMega8535
//
//  Compiler....: IAR 5.11
//
//  Description.: Робота з чотирма температурними давачами DS18B20, що
// знаходяться на одній шині. Функція пошуку адреси 1Wire-пристроїв.
//
//*****************************************************************
#include <ioavr.h>
#include <intrinsics.h>
#include "lcd_lib.h"
#include "bcd.h"
#include "OWIPolled.h"
#include "OWIHighLevelFunctions.h"
#include "OWIBitFunctions.h"
#include "OWIcrc.h"
//код сімейства та коди команд давача DS18B20 // БЛОК 1
#define DS18B20_FAMILY_ID               0x28
#define DS18B20_CONVERT_T               0x44
#define DS18B20_READ_SCRATCHPAD         0xbe
```

```
#define DS18B20_WRITE_SCRATCHPAD        0x4e
#define DS18B20_COPY_SCRATCHPAD         0x48
#define DS18B20_RECALL_E                0xb8
#define DS18B20_READ_POWER_SUPPLY       0xb4

//вивід, до якого під'єднені  1Wire-пристрої
#define BUS   OWI_PIN_7
#define _delay_ms(ms)   __delay_cycles((F_CPU / 1000) * (ms));


//кількість пристроїв  на шині 1Wire
#define MAX_DEVICES       0x05

//коди помилок для функції читання  температури
#define READ_SUCCESSFUL   0x00
#define READ_CRC_ERROR    0x01              // БЛОК 1 – кінець

//прототипи функцій                       // БЛОК 2
unsigned char DS18B20_ReadTemperature(unsigned char bus, unsigned char * id,
unsigned int* temperature);
void DS18B20_PrintTemperature(unsigned int temperature); // БЛОК 2 – кінець

OWI_device allDevices[MAX_DEVICES];
unsigned char rom[8];

unsigned char dec2hex(unsigned char value) {       // 3.8.10.5.3.0
   switch(value) {                          // 3.8.10.5.3.1
      case 0  : return '0';                 // 3.8.10.5.3.2; 3.8.10.5.3.3
      case 1  : return '1';                 // .....
```

```
        case 2  : return '2';                   // .....
        case 3  : return '3';                   // .....
        case 4  : return '4';                   // .....
        case 5  : return '5';                   // .....
        case 6  : return '6';                   // .....
        case 7  : return '7';                   // .....
        case 8  : return '8';                   // .....
        case 9  : return '9';                   // .....
        case 10 : return 'a';                    // .....
        case 11 : return 'b';                    // .....
        case 12 : return 'c';                    // .....
        case 13 : return 'd';                    // .....
        case 14 : return 'e';                    // .....
        case 15 : return 'f';                   // 3.8.10.5.3.4; 3.8.10.5.3.5
        default : return '\0';                  // 3.8.10.5.3.6
    }
}

void id2hex(unsigned char * value) {              // 3.8.10.5.0
    int i;
    for (i = 0; i < 8; i++) {                   // 3.8.10.5.1
        LCD_WriteData(dec2hex(value[i]/16));  // 3.8.10.5.2; 3.8.10.5.3; 3.8.10.5.4
        LCD_WriteData(dec2hex(value[i]%16)); //3.8.10.5.5; 3.8.10.5.3; .8.10.5.7
    }
}                                       // 3.8.10.5.8
void display_sensor(int i, unsigned char * searchFlag) {   // 3.8.10.0
    unsigned int temperature = 0;                       // 3.8.10.1
*searchFlag = DS18B20_ReadTemperature(BUS, allDevices[i].id, &temperature);     //
3.8.10.2
```

```
    LCD_Clear();                                    // 3.8.10.3
    LCD_Goto(0,0);                                  // 3.8.10.3
    if (*searchFlag != READ_CRC_ERROR){             // 3.8.10.4
      id2hex(allDevices[i].id);                     // 3.8.10.5
      LCD_Goto(0,1);                                // 3.8.10.7
      DS18B20_PrintTemperature(temperature);        // 3.8.10.9
    }
    else{
      LCD_SendString("No Data");                    // 3.8.10.6
      *searchFlag = 0;                              // 3.8.10.8
    }
     _delay_ms(300);              // 3.8.10.10
}
int main( void )      // БЛОК 3
{
  unsigned char searchFlag = 0;
  unsigned char num = 0;
   LCD_Init();         // 3.1
  LCD_Clear();        // 3.2
  LCD_Goto(2,0);      // 3.3
  LCD_SendString("1-WIRE METEO");
  LCD_Goto(2,1);
  LCD_SendString("*  SYSTEM  *");        // 3.3 – кінець
 _delay_ms(600);        // 3.4
  LCD_Goto(0,0);        // 3.5
  LCD_SendString("by KPI Students");
  LCD_Goto(0,1);
  LCD_SendString(" course project ");    // 3.5 – кінець
 _delay_ms(600);        // 3.6
```

```
 LCD_Clear();          // 3.7
 while(1)              // 3.8
 {                     // 3.8.1
  if (searchFlag == 0){               // 3.8.2
    num = 0;
    OWI_SearchDevices(allDevices, MAX_DEVICES, BUS, &num);  // 3.8.3
    LCD_Clear();                  // 3.8.4
    LCD_Goto(0,0);
    LCD_SendString("Devices Found: ");
    LCD_Goto(0,1);
    BCD_1Lcd(num);                // 3.8.4 – кінець
    if (num == MAX_DEVICES) searchFlag = 0xff;              // 3.8.5 – 3.8.6
  }
  if (num == 0) {                 // 3.8.7
     LCD_Clear();                 // 3.8.8
     LCD_Goto(0,0);
     LCD_SendString("No Devices Found");
     _delay_ms(1000);             // 3.8.8 – кінець
  } else {
     int i;
     for (i = 0; i < num; i++) {         // 3.8.9; 3.8.11
        display_sensor(i, &searchFlag); // 3.8.10
     }
  }
 };                      // 3.8 – кінець; 3.8.12
 return 0;
}                        // БЛОК 3 - кінець
/*****************************************************************
```

```
*   Function name :   DS18B20_ReadTemperature
*    Returns :   коди – READ_CRC_ERROR, якщо дані які зчитались не пройшли перевірку
*       READ_SUCCESSFUL, якщо дані, які зчитались, пройшли перевірку

*   Parameters :    bus – вивід мікроконтролера, який виконює роль 1WIRE-шини
*      *id – ім’я масиву з 8-ми елементів, в якому міститься
*                    адреса давача DS18B20
*    *temperature – вказівник на шістнадцати розрядну змінну,
*                         в якій буде зберігатися зчитане значення температури
*   Purpose :   Адресує давач DS18B20, дає команду на перетворення температури,
*               чекає, зчитує його пам’ять – scratchpad, перевіряє CRC,
*               зберігає значення температури у змінній, повертає код помилки
**********************************************************************/
unsigned char DS18B20_ReadTemperature(unsigned char bus, unsigned char * id, unsigned int* temperature) // 3.8.10.2.0
{
   unsigned char scratchpad[9];                  // 3.8.10.2.1
   unsigned char i;
  /*подаємо сигнал скидання,
   команду для адресації всіх пристроїв на шині,
   подаемо команду – запук перетворення */
   OWI_DetectPresence(bus);                // 3.8.10.2.2
   OWI_MatchRom(id, bus);                  // 3.8.10.2.3
   OWI_SendByte(DS18B20_CONVERT_T ,bus);     // 3.8.10.2.4
   /*чекаємо, коли давач закінчить перетворення*/
   while (!OWI_ReadBit(bus));              // 3.8.10.2.5
   /*подаємо сигнал скидання,
   команду для адресації всіх пристроїв на шині,
```

```
   команду – читання  внутрішній пам’яті,
   потім зчитуємо внутрішню пам’ять давача в масив
   */
   OWI_DetectPresence(bus);                  // 3.8.10.2.6
   OWI_MatchRom(id, bus);                    // 3.8.10.2.7
   OWI_SendByte(DS18B20_READ_SCRATCHPAD, bus); // 3.8.10.2.8
   for (i = 0; i<=8; i++){                   // 3.8.10.2.9; 3.8.10.2.11
    scratchpad[i] = OWI_ReceiveByte(bus);    // 3.8.10.2.10
   }
   if(OWI_CheckScratchPadCRC(scratchpad) != OWI_CRC_OK){ // 3.8.10.2.12
    return READ_CRC_ERROR;                   // 3.8.10.2.13
   }
   *temperature = (unsigned int)scratchpad[0];          // 3.8.10.2.14
   *temperature |= ((unsigned int)scratchpad[1] << 8);   // 3.8.10.2.14
    return READ_SUCCESSFUL;       // 3.8.10.2.15
}// 3.8.10.2.16
/*****************************************************************
*   Function name :  DS18B20_PrintTemperature
*   Returns :        ні
*   Parameters :     temperature – температура давача DS18B20
*   Purpose :        Виведення значення температури давача DS18B20
*                    на LCD. Адресу знакоміста треба виставляти раніше.
*****************************************************************/
void DS18B20_PrintTemperature(unsigned int temperature)      // 3.8.10.9.0
        {
         unsigned char tmp = 0;
         /*виводимо знак температури
         *якщо вона від’ємна,
         *робимо перетворення*/
```

```
  if ((temperature & 0x8000) == 0){               // 3.8.10.9.1
    LCD_WriteData('+');                           // 3.8.10.9.2
  }
  else{
    LCD_WriteData('-');                   // 3.8.10.9.3
    temperature = ~temperature + 1;       // 3.8.10.9.4
  }
//виводимо значення цілої частини температури
  tmp = (unsigned char)(temperature>>4);    // 3.8.10.9.5
  if (tmp<100){                           // 3.8.10.9.6
    BCD_2Lcd(tmp);                        // 3.8.10.9.7
  }
  else{
    BCD_3Lcd(tmp);                        // 3.8.10.9.8
  }                                   // 3.8.10.9.9
  }
```

12.5 Зміст звіту

Звіт по роботі повинен містити:

- схему моделі;
- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

Контрольні запитання та завдання

1) Ким та коли було розроблено інтерфейс 1-WIRE?
2) Де може застосовуватись інтерфейс 1-WIRE?
3) Назвіть основні переваги інтерфейсу 1-WIRE.
4) Назвіть основні характеристи інтерфейсу 1-WIRE.
5) Наведіть та опишіть схему фізичної реалізації інтерфейсу 1-WIRE.
6) Опишіть основні правила передачі даних в мережі 1-WIRE.

7) Назвіть обмеження застосування 1-WIRE-мереж.
8) Наведіть та опишіть структуру системи для вимірювання температури в приміщенні з використанням давача 1-WIRE.
9) Дайте загальну характеристику датчику температури DS18B20.
10) Назвіть та опишіть команди мікроконтролера для керування мережею 1-WIRE
11) Наведіть та опишіть модель мережі 1-WIRE в PROTEUS 8.6.
12) Наведіть та опишіть схему алгоритму роботи моделі мережі 1-WIRE в PROTEUS 8.6.
13) Наведіть та опишіть робочу програму моделювання мережі 1-WIRE в PROTEUS 8.6.

13 ЛАБОРАТОРНА РОБОТА №10. ДОСЛІДЖЕННЯ МОДЕЛІ ПРИСТРОЮ КЕРУВАННЯ LCD-ДИСПЛЕЄМ

Тема: Моделювання пристрою керування LCD-дисплеєм

Мета: Користуючись пакетом Proteus 8.6 дослідити роботу модуля керування LCD-дисплеєм

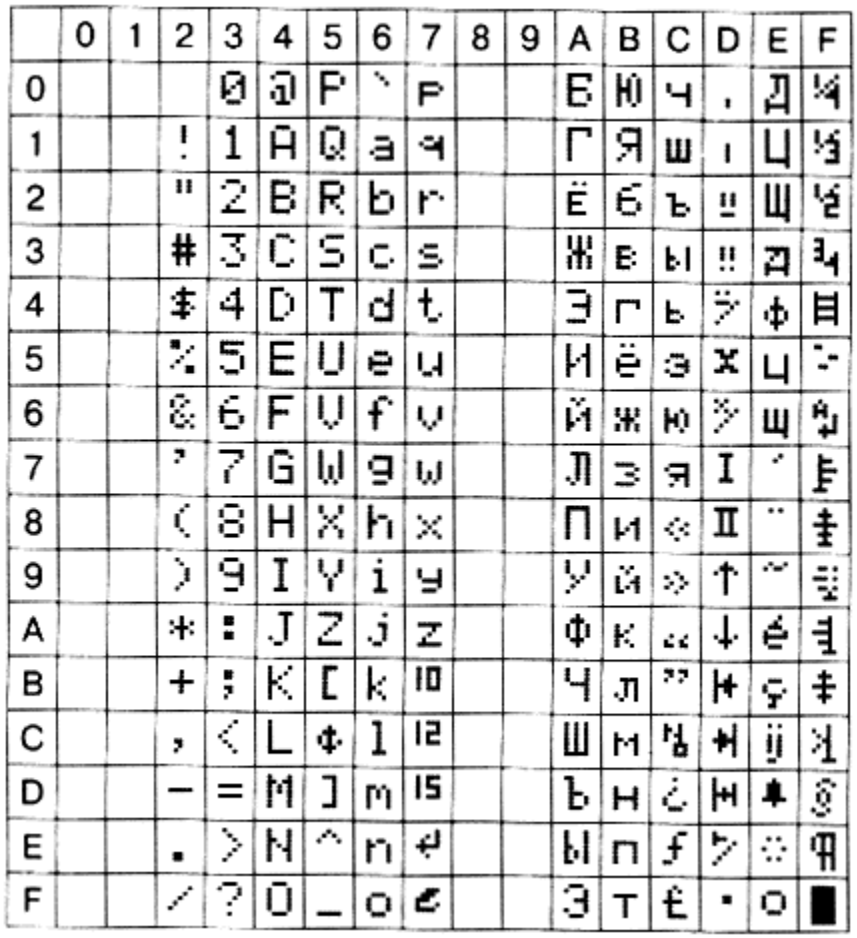
13.1 Моделювання пристрою керування LCD-дисплеєм

В роботі розглядаються LCD-дисплеї на рідких кристалах (рідкокристаличні індикатори – РКІ), які використовуються для відображення графічної або текстової інформації в ноутбуках, телефонах, цифрових фотоапаратах, електронних книгах, навігаторах, планшетах, електронних перекладачах, калькуляторах, годинниках та у вбудованих системах для відображення інформації про результати контролю.

Сьогодні одним з найбільш поширених пристроїв керування відображенням алфавітно-цифрової інформації у вбудованих мікроконтролерних системах є контролер HD44780 виробництва Hitachi. Майже всі провідні виробники РКІ – Epson, Sanyo, Toshiba, Samsung, Philips випускають аналоги цього контролера або сумісні з ним за інтерфейсом і командною мовою – мікросхеми або РКІ на базі цих контролерів. Тобто практично можна говорити про HD44780 як про промисловий стандарт. Модулі з цим контролером застосовуються в найрізноманітніших пристроях: вимірювальні прилади, промислове, технологічне та медичне обладнання, офісна техніка і т. ін.

Мікросхема HD44780 має наперед визначену таблицю символів, розміщену в оперативному запам`ятовуючому пристрої – RAM знакогенератора CGRAM (Character Generator RAM). Для відображення будь-якого з них програма мікроконтролера повинна передати координати позиції і безпосередньо за ними саму адресу символу з CGRAM. Нижче, як приклад, наведено таблицю кодів символів контролера HD44780 (таблиця 13.1).

Таблица 13.1 – Таблиця кодів символів контролера HD44780



Літери, числові знаки, а також більшість розділових знаків збігаються в ній з кодами ASCII. Набір символів, розміщених за адресами 0xA0…0xFF, містить національний алфавіт (в даному випадку кирилицю) регіону, де передбачається його використання. Перші 16 комірок CGRAM мають особливе значення. За бажання, в них можуть бути записані будь-які символи користувача, яких немає в таблиці (відразу після включення модуля в них знаходиться випадкова інформація). Спростити перетворення рядка, що складається з літер російської та англійської абетки, на набір кодів HD44780, можна за допомогою утиліти "HD44780" (зовнішній вигляд на рисунку 13.1). Все, що робить ця програма – приводить у відповідність набір введених символів з їх відображенням у таблиці CGRAM. Результатом перетворення є набір байтів (з нульовим значенням наприкінці), що починаються з директиви резервування FLASH-пам'яті програм .db.

Рисунок 13.1 – Зовнішній вигляд індикатора типу HD44780

Перш ніж переходити до програмування індикатора, треба знати конфігурацію контактів РК-дисплею. В основному РК-дисплей має 16 контактів. Призначення виводів цього індикатора наведено таблиці 13.2.

Таблиця 13.2 – Назви та призначення виводів індикатора

Pin	Symbol	Purpose
1	Vss	Ground
2	Vcc	+5V
3	Vee	Contrast
4	RS	Register select
5	RW	Read/Write
6	E	Enable
7	D0	Data
8	D1	Data
9	D2	Data
10	D3	Data
11	D4	Data
12	D5	Data
13	D6	Data
14	D7	Data
15	LED Vcc	LED backlight
16	LED Vss	LED backlight

Вивод VE (Vee) призначений для регулювання контрастності дисплея, тому ми його підключають до заземлення через змінний резистор (рисунок 13.2).

Контакти 15 і 16 використовуються для регулювання яскравості, контакт 5 використовується для операції читання/запису.

Якщо сигнал на вході вхід дорівнює логічній 1, то виконується операція читання, а якщо вхід дорівнює логічному 0, то виконується операція запису. Контакт 4 (RS) призначений для вибору регістра. Якщо вхід дорівнює логічній 1, це очищає РК-дисплей та відбувається запис команди, а якщо вхід дорівнює логічному 0, то відображаються дані. На вхід E від мікроконтролера подається стробуючий імпульс (активний фронт – задній), яким байт даних переписується у внутрішній регістр для подальшої обробки. Напрямок передачі визначає рівень лінії R/W (R/W =1 – читання з індикатора, R/W =0 – запис в індикатор). У реальних додатках, як правило, немає потреби в читанні даних. Тому вивід R/W завжди з'єднують із загальним дротом.

У таблиці 13.3 наведено команди керування індикатором

Таблиця 13.3 – Команди керування індикатором

Код	Команди до регістра команд LCD
1	Очистити екран дисплея
2	Повернення назад
4	Курсор повернути назад
6	Курсор переставити далі
8	Дисплей «ВИМКНЕНО» і курсор «ВИМКНЕНО»
0c	Дисплей «ON» і курсор «OFF»
E/F	Дисплей «УВІМКНЕНО» і курсор «Моргає»
80	Примусово перевести курсор на початок 1-го рядка
c0	Примусово перевести курсор на початок 2-го рядка
38	Дворядкова матриця 5x7

Схему підключення індикатора в складі працюючої моделі цифрового вольтметра, яку реалізовано з використанням мікроконтролера ATmega128, наведено на рисунку 13.2.

З правого боку рисунка бачимо LCD-дисплей 16x2, який з’єднаний з портами C та D мікроконтролера ATmega128 відповідними лініями зв’язку. У верхній частині рисунку знаходиться батарея та резистор RV1, напругу з виходу якого можна змінювати у режимі реального часу. Ця напруга подається на порт PF0 мікроконтролера ATMega128. Цю напругу модуль АЦП мікроконтролера перетворює у десятковий еквівалент двійкового коду [1].

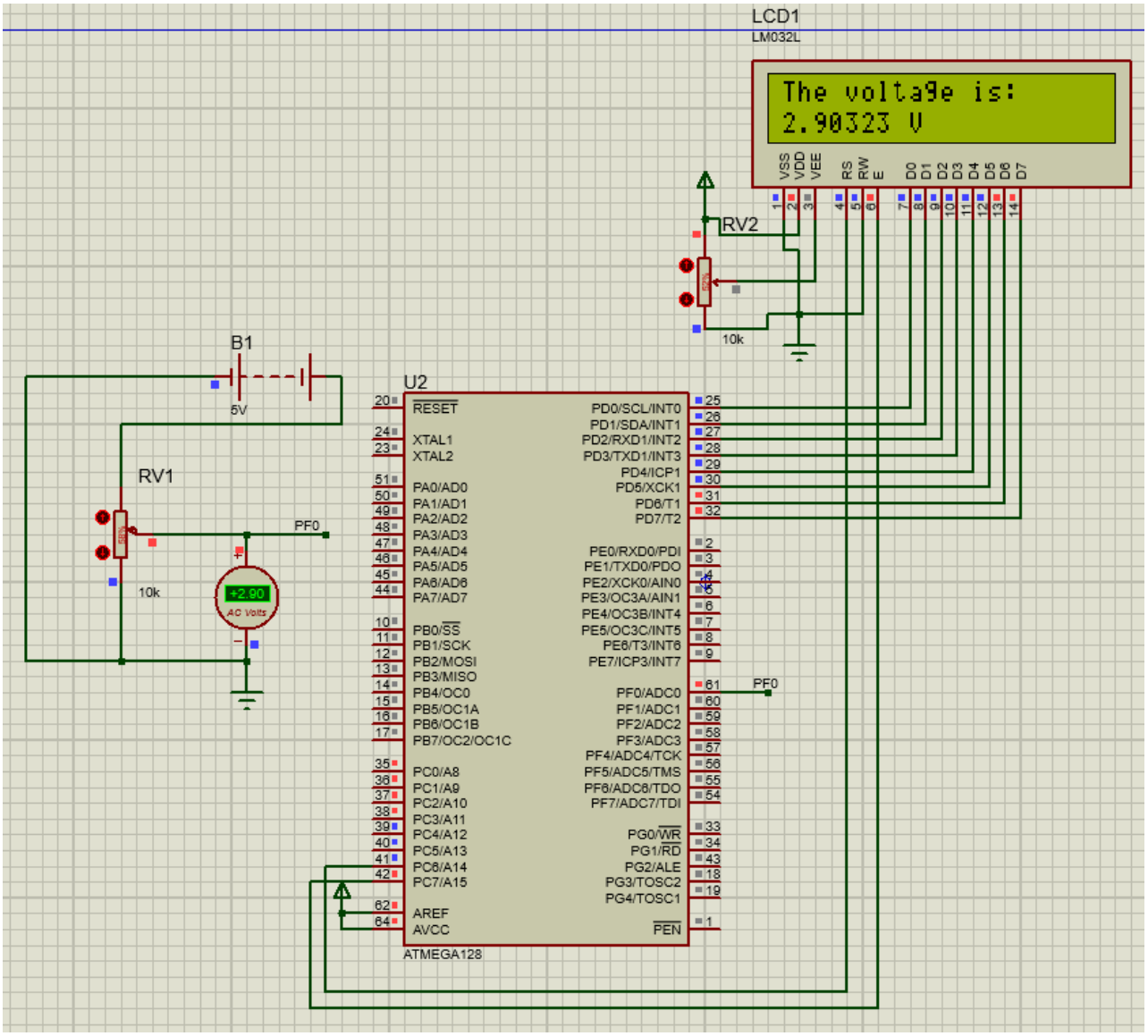


Рисунок 13.2 – Схема працюючої моделі цифрового вольтметра

Різниця між виведенням результату перетворення АЦП, який описано далі в цій роботі, полягає в тому, що в ній розглянуто виведення на чотири свілодіодні індикатори цифрової інформація у чотирьохрозрядному десятковому коді, а в моделі цифрового вольтметра, яка описується в роботі, відображається абсолютне значення вхідної напруги у вольтах з точністю до сотих долей вольта.

При підключенні LCD-дисплея в моделі використовуються його наступні входи: VSS-земля, VDD – живлення 5В, VEE – керування контрастністю, RS – вибір регістра, RW – вибір режиму: читання/запис, E – увімкнення, D0…D7 – виводи для передачі даних.

Дисплей може працювати в 2-ох режимах: 4-бітний та 8-бітний. В 4-бітному режимі до МК залишаються непідключеними виводи D0…D3. Команди та дані спочатку передаються послідовно – спочатку старші 4 біти, а потім молодші. Команди є попередньовизначеними та вказуються в специфікації LCD.

У мовах програмування Сі і С++ часто використовуються заголовні файли, які є одним з основних способів підключити до програми типи даних, структури, прототипи функцій, типи і макроси, що перераховуються, використовувані в іншому модулі. За замочуванням в цих файлах використовується розширення .h. Іноді для заголовних файлів мови С++ використовують розширення .hpp.

Щоб уникнути повторного включення одного й того ж коду, використовуються директиви #ifndef, #define, #endif.

Заголовний файл в загальному випадку може містити будь-які конструкції мови програмування, але на практиці виконуваний код (за винятком inline-функцій С++) в заголовні файли не поміщають. Наприклад, ідентифікатори, які повинні бути оголошені більш ніж в одному файлі, зручно описати в заголовному файлі, а потім підключати його при необхідності. Подібним чином працює модульність і в більшості асемблерів.

За традицією, що склалася, в заголовних файлах оголошують функції стандартної бібліотеки Сі і Сі++.

У програмуванні заголовний файл (англ. header file) або файл, що підключається – файл, вміст якого автоматично додається препроцесором у вихідний текст в тому місці, де розташовується директива: #include <file.h>.

13.2 Схема алгоритму роботи моделі

Схему алгоритму роботи моделі наведено на рисунках 13.3…13.6.

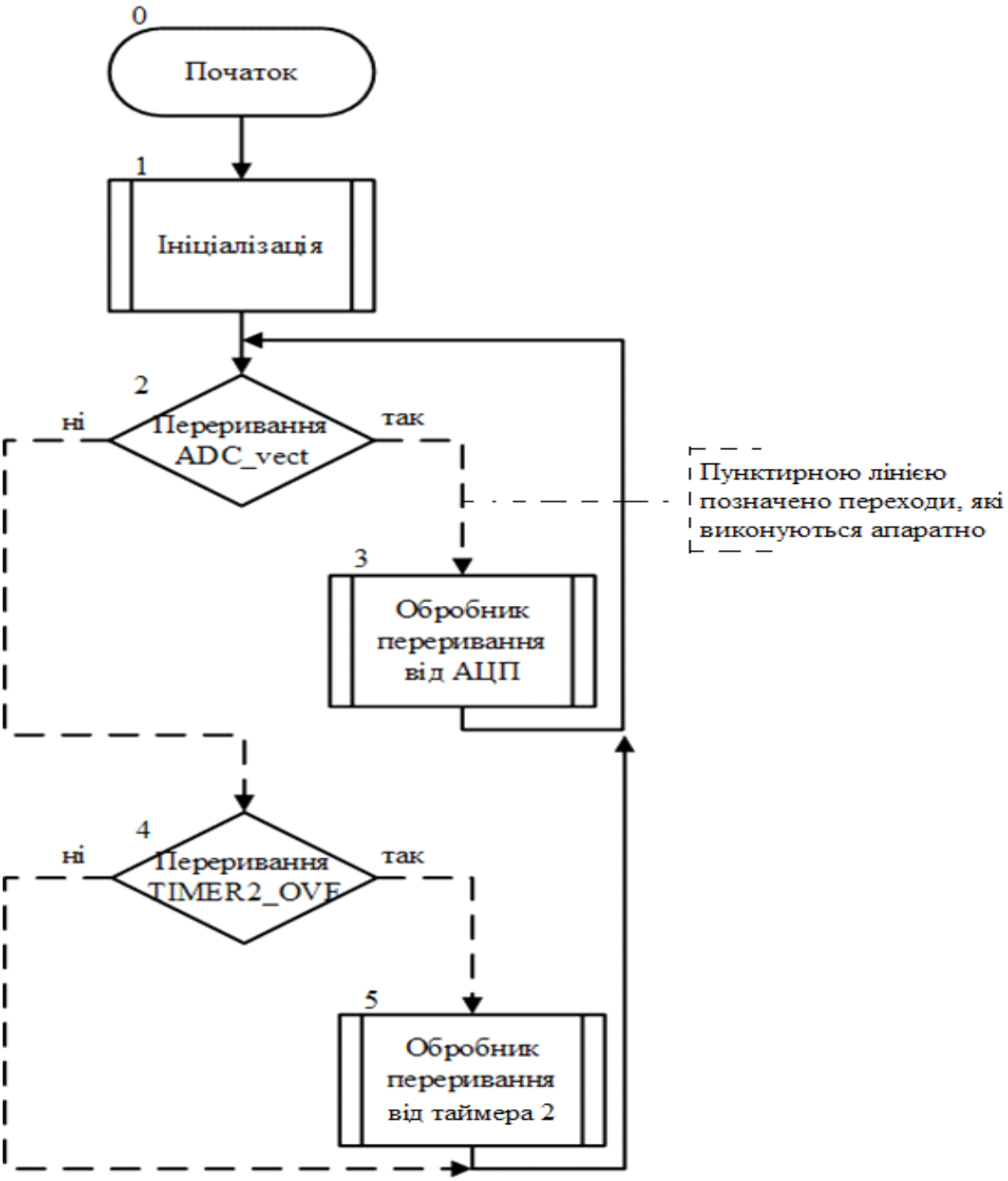


Рисунок 13.3 – Схема загального алгоритму роботи модуля

Схему алгоритму ініціалізації наведено на рисунку 13.4.

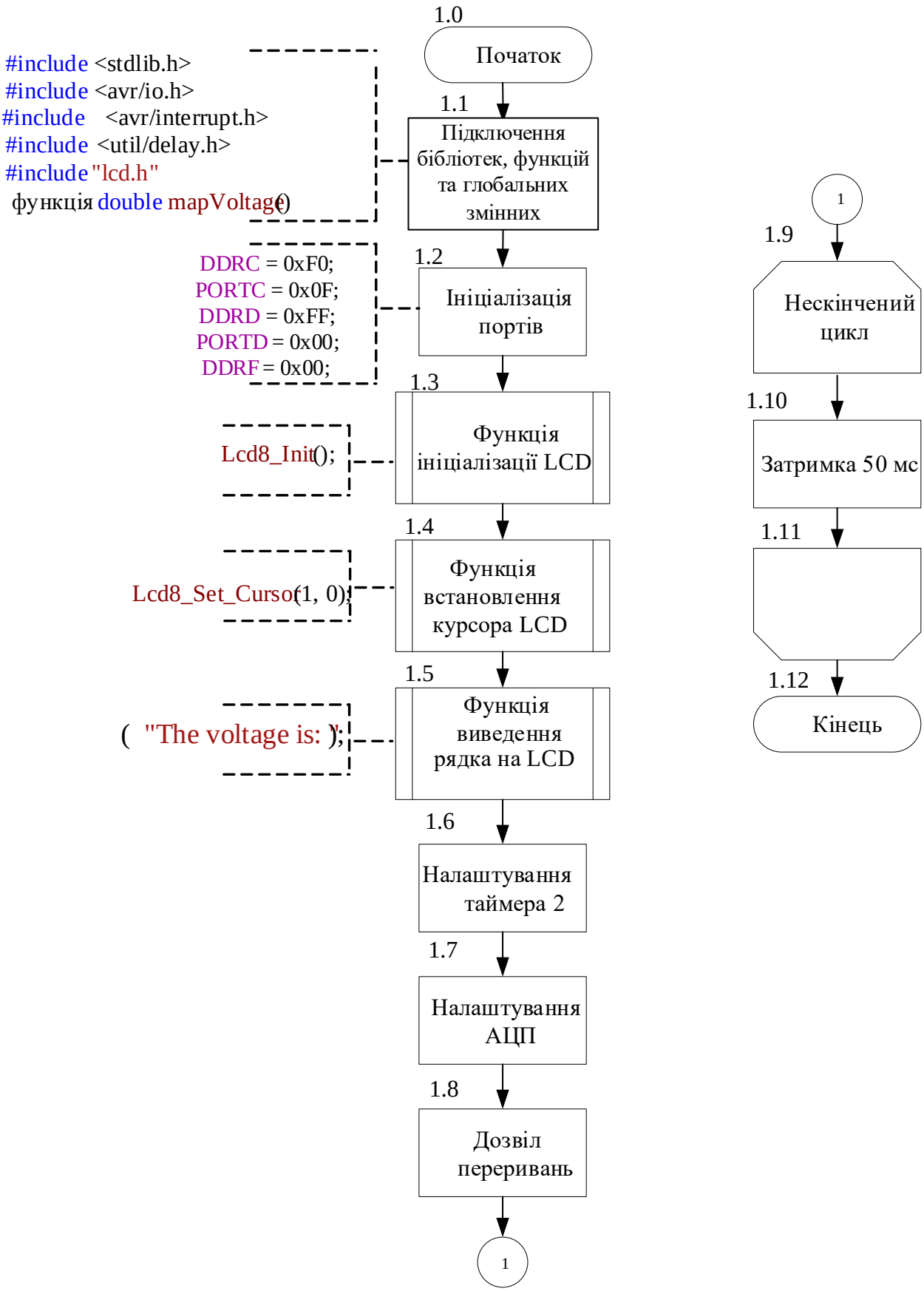


Рисунок 13.4 – Схема алгоритму ініціалізації

Схему алгоритму обробки переривання від таймера 2 наведено на рисунку 13.5.

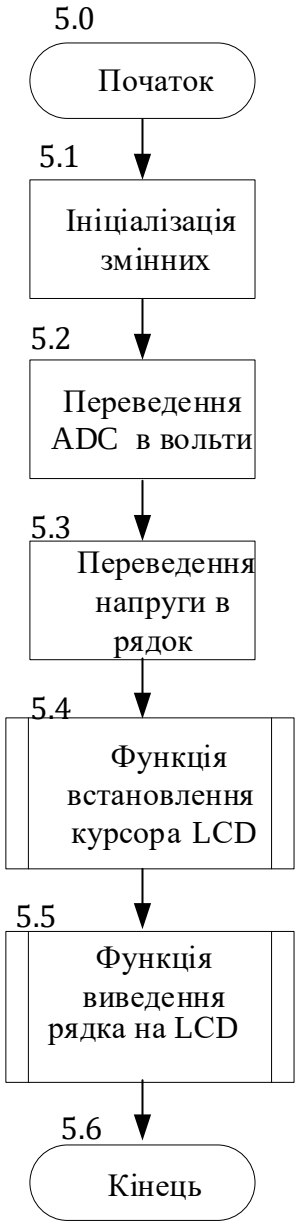


Рисунок 13.5 – Схема алгоритму обробки переривання від таймера 2

Схему алгоритму обробки переривання завершення перетворення АЦП наведено на рисунку 13.6.

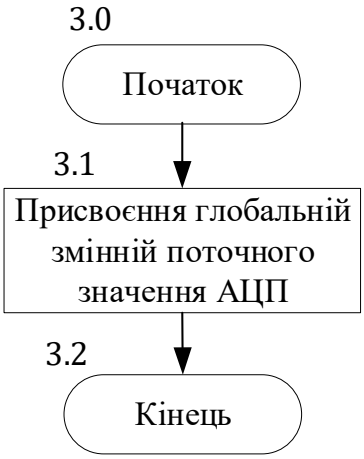


Рисунок 13.6 – Схема алгоритму обробки переривання завершення перетворення АЦП

Після виконання блоку ініціалізації (блок 1) АЦП запускається в режим безперервного перетворення. Паралельно починає роботу таймер/лічильник Т2, який при кожному переповненні викликає підпрограму його обробки. Задачею цієї підпрограми є виведення на LCD-дисплей результату перетворення від АЦП, який формується у десятковому коді. До завершення першого перетворення АЦП на дисплей виводиться нульове значення. Коли АЦП закінчує перше перетворення, встановлюється відповідний прапорець, виконання програми переривається та викликається підпрограма обробки цього переривання.

При черговому переповненні таймер отримує результат від АЦП та відповідна підпрограма виводить його на дисплей.

Далі АЦП буде безперервно виконувати наступні перетворення, результат яких за допомогою таймера також буде виводитись на дисплей.

Схему загального алгоритму роботи модуля наведено на рисунку 13.3.

13.3 Робоча керуюча програма

Нижче наведено робочу керуючу програму

```
  #ifndef F_CPU
#define F_CPU 1600000UL//16000000UL // 16 MHz clock speed
#endif

// 1.1 – Підключення бібліотек, функцій та глобальних змінних
#include <stdlib.h>
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>
#include "lcd.h" // Бібліотека з функціями для роботи з LCD
  Ініціалізації глобальних змінних
volatile unsigned char current_indicator = 0;
volatile double display = 0.0;
volatile unsigned int ADC_value;
double mapVoltage()
{
        return ADC_value*5.0/1023;
}
int main (void)
{
        // 1.2 – Ініціалізація портів
        DDRC = 0xF0;
        PORTC = 0x0F;
        DDRD = 0xFF;
        PORTD = 0x00;
        DDRF = 0x00;

       Lcd8_Init();                  //1.3   Виклик функції ініціалізації LCD
       Lcd8_Set_Cursor(1, 0); // 1.4   Виклик функції встановлення
                              // курсора LCD в рядок 1 і положення 0
       Lcd8_Write_String("The voltage is: "); // 1.5   Виклик функції виведення
                                              // рядка на LCD

       // 1.6 – Налаштування таймера 2
       TIMSK |= (1 << TOIE2); // Дозвіл переривання за переповненням таймера 2
```

```
        TCCR2 |= (1 << CS21); // Програмування коефіцієнта ділення системного
        // тактового сигналу на 8

        // 1.7 – Налаштування АЦП
        ADCSRA |= (1<<ADEN) // Дозвіл АЦП
        | (1<<ADPS2) // Програмування попереднього дільника, що формує тактовий сигнал
        // для АЦП: передільник дорівнює 64
        | (1<<ADPS1)
        | (1<<ADIE) // Дозвіл переривання від завершення АЦП
        | (1<<ADFR) // Вибір режиму роботи АЦП: 1 – безперервне перетворення
        | (1<<ADSC); // Запуск перетворення (1 – почати перетворення)

        // 1.8 – Глобальний дозвіл переривань
        sei();

        // Основний цикл
        while(1)
        {                       // 1.9
          _delay_ms(50);        // 1.10   Затримка 50 мс
        }                       // 1.11
}                               // 1.12

      // 2          Умова переривання від АЦП
ISR (ADC_vect)
{     // 3.0        Обробник перериваннь від АЦП
  ADC_value = ADC; // 3.1  Присвоюємо глобальній змінній поточне
                   // значення АЦП
}     //3.2

// 4          Умова переривання від таймера Т2
ISR (TIMER2_OVF_vect)
{     // 5.0              Обробник переривань від таймера Т2
        char voltageString[15];         //5.1
        char doubleString[10];
// 5.2  Обраховуємо значення напруги  за формулою: ADC_value*5.0/1023;
        display = mapVoltage();         //5.2
        // Конвертуємо значення напруги
        // Перетворюємо дані з плаваючою точкою в масив символів
        dtostrf(display,1,5, doubleString);  //5.3
        // Функція повертає рядок, створений за допомогою doubleString
        sprintf(voltageString, "%7s V", doubleString);          //5.3
```

```
        // Виклик функції встановлення курсору в задане положення LCD
        Lcd8_Set_Cursor(2, 0);        //5.4
        // Виклик функції виведення рядка на LCD
        Lcd8_Write_String(voltageString);  //5.5
}               //5.6
```

Лістинг коду .h файлу для керування LCD-дисплеєм:

```
#define   eS_PORTA0   0
#define   eS_PORTA1   1
#define   eS_PORTA2   2
#define   eS_PORTA3   3
#define   eS_PORTA4   4
#define   eS_PORTA5   5
#define   eS_PORTA6   6
#define   eS_PORTA7   7
#define   eS_PORTB0   10
#define   eS_PORTB1   11
#define   eS_PORTB2   12
#define   eS_PORTB3   13
#define   eS_PORTB4   14
#define   eS_PORTB5   15
#define   eS_PORTB6   16
#define   eS_PORTB7   17
#define   eS_PORTC0   20
#define   eS_PORTC1   21
#define   eS_PORTC2   22
#define   eS_PORTC3   23
#define   eS_PORTC4   24
#define   eS_PORTC5   25
#define   eS_PORTC6   26
#define   eS_PORTC7   27
#define   eS_PORTD0   30
#define   eS_PORTD1   31
#define   eS_PORTD2   32
```

```
#define eS_PORTD3 33
#define eS_PORTD4 34
#define eS_PORTD5 35
#define eS_PORTD6 36
#define eS_PORTD7 37

#ifndef D0
#define D0 eS_PORTD0
#define  D1  eS_PORTD1
#define  D2  eS_PORTD2
#define  D3  eS_PORTD3
#define  D4  eS_PORTD4
#define  D5  eS_PORTD5
#define  D6  eS_PORTD6
#define  D7  eS_PORTD7
#define  RS  eS_PORTC6
#define  EN  eS_PORTC7
#endif

#include<util/delay.h>

void pinChange(int a, int b)
{
        if(b == 0)
        {
                if(a == eS_PORTA0)
                  PORTA &= ~(1<<PA0);
                else if(a == eS_PORTA1)
                  PORTA &= ~(1<<PA1);
                else if(a == eS_PORTA2)
                  PORTA &= ~(1<<PA2);
                else if(a == eS_PORTA3)
                  PORTA &= ~(1<<PA3);
                else if(a == eS_PORTA4)
                  PORTA &= ~(1<<PA4);
                else if(a == eS_PORTA5)
                  PORTA &= ~(1<<PA5);
                else if(a == eS_PORTA6)
                  PORTA &= ~(1<<PA6);
                else if(a == eS_PORTA7)
                  PORTA &= ~(1<<PA7);
```

```
else if(a == eS_PORTB0)
  PORTB &= ~(1<<PB0);
else if(a == eS_PORTB1)
  PORTB &= ~(1<<PB1);
else if(a == eS_PORTB2)
  PORTB &= ~(1<<PB2);
else if(a == eS_PORTB3)
  PORTB &= ~(1<<PB3);
else if(a == eS_PORTB4)
  PORTB &= ~(1<<PB4);
else if(a == eS_PORTB5)
  PORTB &= ~(1<<PB5);
else if(a == eS_PORTB6)
  PORTB &= ~(1<<PB6);
else if(a == eS_PORTB7)
  PORTB &= ~(1<<PB7);
else if(a == eS_PORTC0)
  PORTC &= ~(1<<PC0);
else if(a == eS_PORTC1)
  PORTC &= ~(1<<PC1);
else if(a == eS_PORTC2)
  PORTC &= ~(1<<PC2);
else if(a == eS_PORTC3)
  PORTC &= ~(1<<PC3);
else if(a == eS_PORTC4)
  PORTC &= ~(1<<PC4);
else if(a == eS_PORTC5)
  PORTC &= ~(1<<PC5);
else if(a == eS_PORTC6)
  PORTC &= ~(1<<PC6);
else if(a == eS_PORTC7)
  PORTC &= ~(1<<PC7);
else if(a == eS_PORTD0)
  PORTD &= ~(1<<PD0);
else if(a == eS_PORTD1)
  PORTD &= ~(1<<PD1);
else if(a == eS_PORTD2)
  PORTD &= ~(1<<PD2);
else if(a == eS_PORTD3)
  PORTD &= ~(1<<PD3);
else if(a == eS_PORTD4)
  PORTD &= ~(1<<PD4);
```

```
  else if(a == eS_PORTD5)
    PORTD &= ~(1<<PD5);
  else if(a == eS_PORTD6)
    PORTD &= ~(1<<PD6);
  else if(a == eS_PORTD7)
    PORTD &= ~(1<<PD7);
  }
  else
  {
   if(a == eS_PORTA0)
  PORTA |= 1<<PA0);
  else if(a == eS_PORTA1)
    PORTA |= (1<<PA1);
  else if(a == eS_PORTA2)
    PORTA |= (1<<PA2);
else if(a == eS_PORTA3)
    PORTA |= (1<<PA3);
else if(a == eS_PORTA4)
    PORTA |= (1<<PA4);
else if(a == eS_PORTA5)
    PORTA |= (1<<PA5);
else if(a == eS_PORTA6)
    PORTA |= (1<<PA6);
else  if(a  ==  eS_PORTA7)
    PORTA |= (1<<PA7);
else if(a == eS_PORTB0)
    PORTB |= (1<<PB0);
else if(a == eS_PORTB1)
    PORTB |= (1<<PB1);
else if(a == eS_PORTB2)
    PORTB |= (1<<PB2);
else if(a == eS_PORTB3)
    PORTB |= (1<<PB3);
else if(a == eS_PORTB4)
    PORTB |= (1<<PB4);
else if(a == eS_PORTB5)
    PORTB |= (1<<PB5);
else if(a == eS_PORTB6)
    PORTB |= (1<<PB6);
```

```
else if(a == eS_PORTB7)
    PORTB |= (1<<PB7);
else if(a == eS_PORTC0)
    PORTC |= (1<<PC0);
else if(a == eS_PORTC1)
    PORTC |= (1<<PC1);
else if(a == eS_PORTC2)
    PORTC |= (1<<PC2);
else if(a == eS_PORTC3)
    PORTC |= (1<<PC3);
else if(a == eS_PORTC4)
    PORTC |= (1<<PC4);
else if(a == eS_PORTC5)
    PORTC |= (1<<PC5);
else if(a == eS_PORTC6)
    PORTC |= (1<<PC6);
else if(a == eS_PORTC7)
    PORTC |= (1<<PC7);
else if(a == eS_PORTD0)
    PORTD |= (1<<PD0);
else if(a == eS_PORTD1)
    PORTD |= (1<<PD1);
else if(a == eS_PORTD2)
    PORTD |= (1<<PD2);
else if(a == eS_PORTD3)
    PORTD |= (1<<PD3);
else if(a == eS_PORTD4)
    PORTD |= (1<<PD4);
else if(a == eS_PORTD5)
    PORTD |= (1<<PD5);
else if(a == eS_PORTD6)
    PORTD |= (1<<PD6);
else if(a == eS_PORTD7)
    PORTD |= (1<<PD7);
}
}

//LCD 8 Bit Interfacing Functions void
Lcd8_Port(char a)
{
```

```
        if(a & 1)
        pinChange(D0,1); else
        pinChange(D0,0);

        if(a & 2)
        pinChange(D1,1); else
        pinChange(D1,0);

        if(a & 4)
        pinChange(D2,1); else
        pinChange(D2,0);

        if(a & 8)
        pinChange(D3,1); else
        pinChange(D3,0);
        if(a & 16)
        pinChange(D4,1); else
        pinChange(D4,0);

        if(a & 32)
        pinChange(D5,1); else
        pinChange(D5,0);

        if(a & 64)
        pinChange(D6,1); else
        pinChange(D6,0);

        if(a & 128)
        inChange(D7,1); else
        pinChange(D7,0);
        }

void Lcd8_Cmd(char a)
{
        pinChange(RS,0);          // => RS = 0
        Lcd8_Port(a);             //Data transfer
        pinChange(EN,1);          // => E = 1
        _delay_ms(1);
        pinChange(EN,0);          // => E = 0
        _delay_ms(1);
}
```

```
void Lcd8_Clear() // Функція очищення LCD
{
        Lcd8_Cmd(1);
}

// Функція встановлення курсора (номер рядка, номер клітинки)
void Lcd8_Set_Cursor(char a, char b)
{
        if(a == 1)
        Lcd8_Cmd(0x80 + b);
        else if(a == 2)
        Lcd8_Cmd(0xC0 + b);
}

void Lcd8_Init()
{
        pinChange(RS,0);
        pinChange(EN,0);
        _delay_ms(20);
        ////////////// Reset process from datasheet /////////
        Lcd8_Cmd(0x30);
        _delay_ms(5);
        Lcd8_Cmd(0x30);
        _delay_ms(1);
        Lcd8_Cmd(0x30);
        _delay_ms(10);
        ////////////////////////////////////////////////////
        //Lcd8_Cmd(0xDF);                   //function set
        Lcd8_Cmd(0x38);  //function set
        Lcd8_Cmd(0x0C);  //display on,cursor off,blink off
        Lcd8_Cmd(0x01);  //clear display
        Lcd8_Cmd(0x06);  //entry mode, set increment
}

void Lcd8_Write_Char(char a) // Функція виведення символа на LCD
{
        pinChange(RS,1);        // => RS = 1
        Lcd8_Port(a);     //Data transfer
        pinChange(EN,1);        // => E = 1
        _delay_ms(1);
        pinChange(EN,0);        // => E = 04
        _delay_ms(1);
}
```

```
void Lcd8_Write_String(char *a) // Функція виведення рядка на LCD
{
	int i;
	for(i=0;a[i]!='\0';i++) // Посимвольне виводення через цикл
	Lcd8_Write_Char(a[i]);
}

void Lcd8_Shift_Right() // Функція, що зсуває текст LCD праворуч
{
	Lcd8_Cmd(0x1C);
}


void Lcd8_Shift_Left() // Функція, що зсуває текст LCD ліворуч
{
	Lcd8_Cmd(0x18);
}
```

13.4 Використання LCD-індикатора в моделі мережі 1-WIRE

У попередньому розділі цього посібника при моделювання мережі 1-WIRE було використано LCD-індикатор. При описанні алгоритму роботи моделі та керуючої програми було наведено пояснення керування виведенням стану 4-х давачів температури на LCD-індикатор. Нижче наведено декілька фрагментів роботи цього індикатора.

Після запуску моделі на індикаторі з`являється повідомлення з назвою моделі (рисунок 13.7).

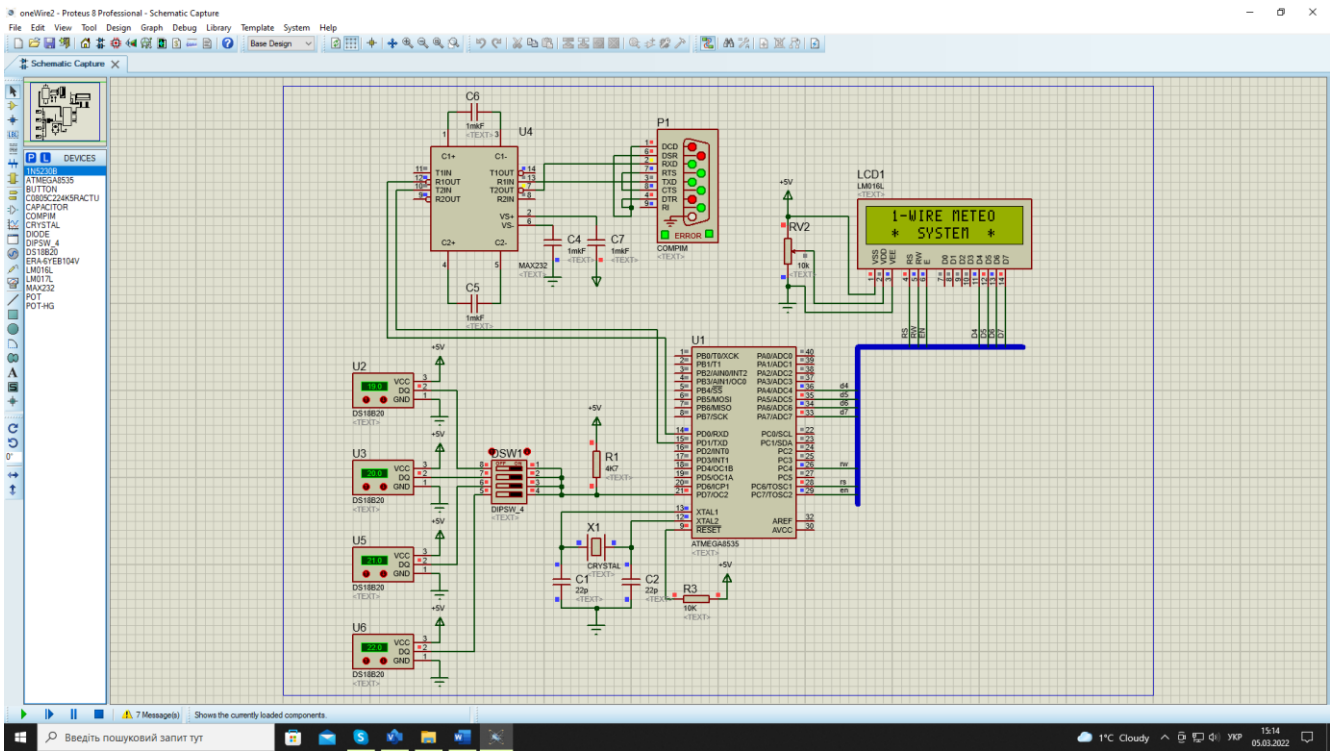


Рисунок 13.7 – Повідомлення про назву моделі

Якщо після запуску моделі жоден з давачів не був підключений, то на індикаторі з`явиться відповідне повідомлення (рисунок 13.8).

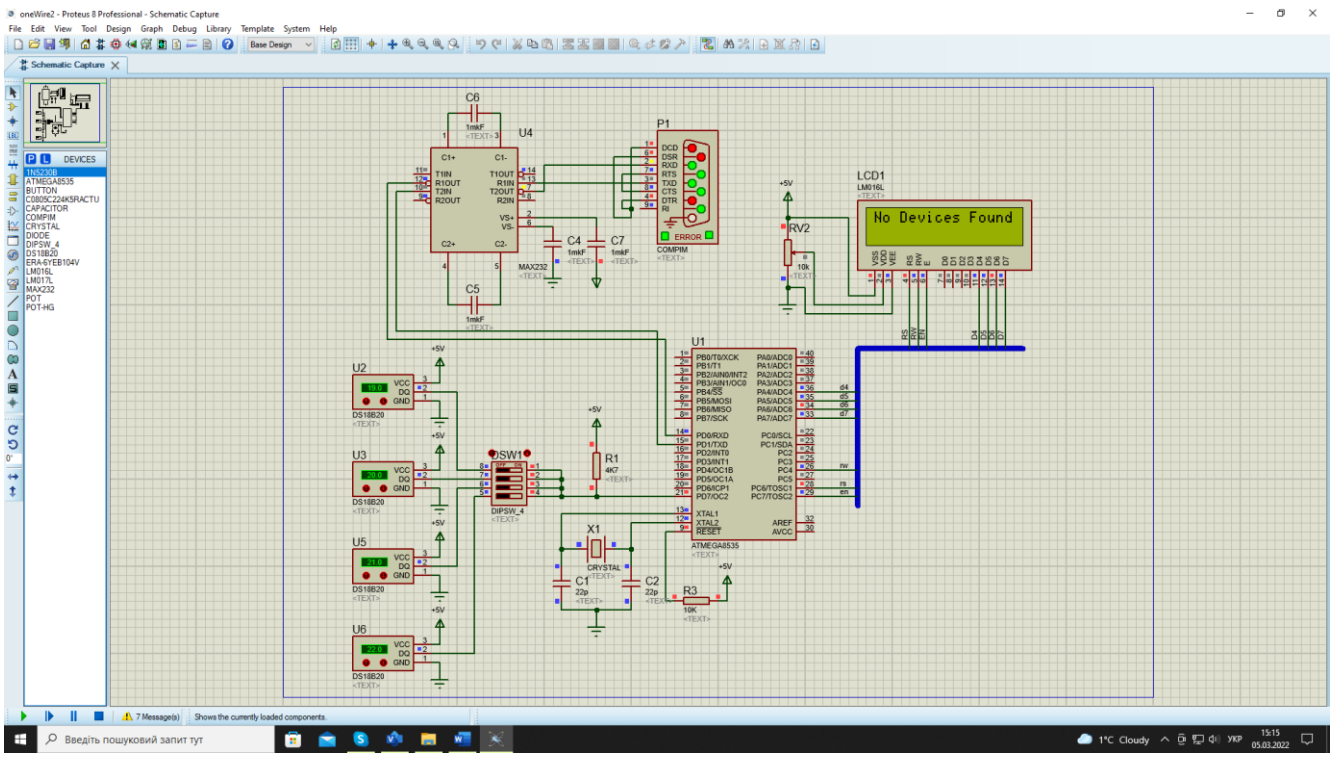


Рисунок 13.8 – Стан моделі при відсутності підключених давачів до моделі

Якщо після запуску моделі був підключений один з чотирьох давачів, який виміряв температуру 19 градусів, то на індикаторі з’явиться відповідне повідомлення (рисунок 13.9).

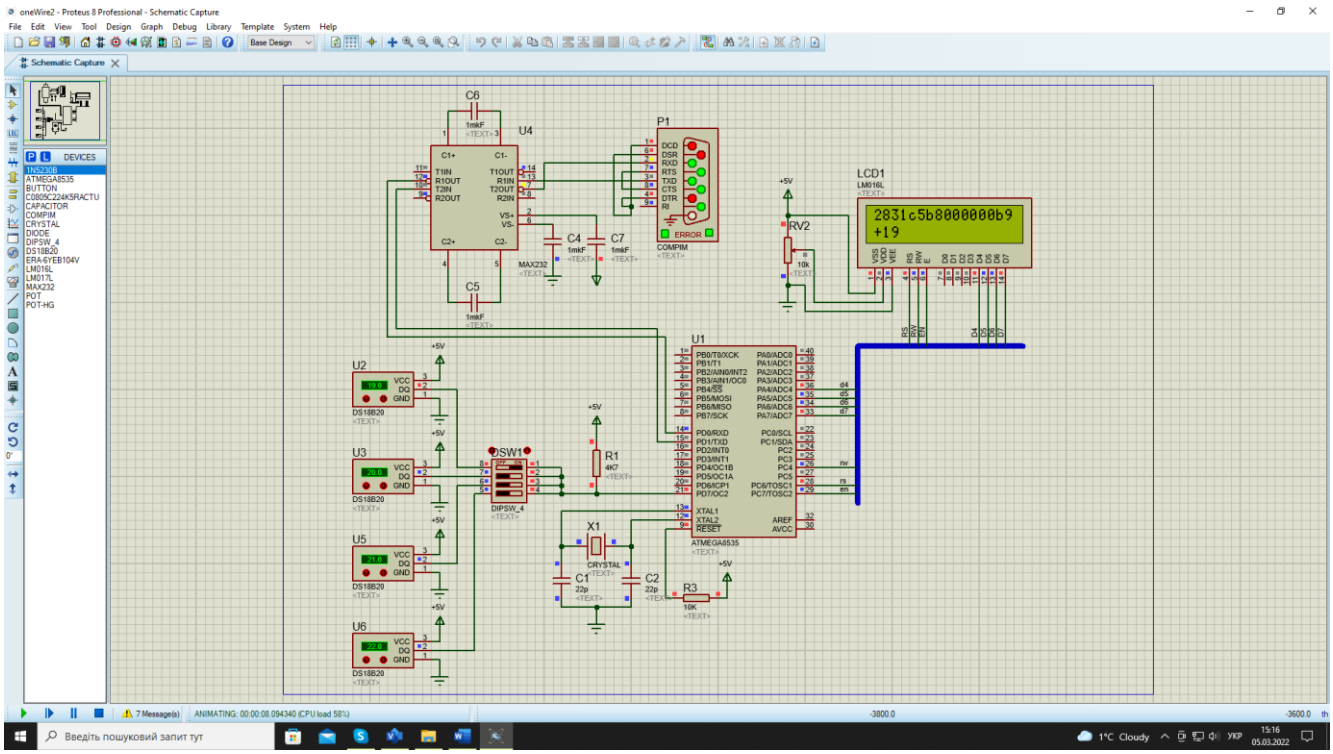


Рисунок 13.9 – Стан моделі при підключенні одного давача, який вимірює температуру 19 град

13.5 Висновки

LCD-дисплеї використовуються в різних системах для показів параметрів їх роботи та різної інформації. Був використаний LCD-дисплей, який може відображати 2 рядки по 16 символів. Кожен символ є матрицею 5 на 8 пікселів. Керування та передача даних на дисплей здійснюється через піни RS, RW, E та D0…D7 за допомогою подачі визначених кодів команд. На відміну від 7-ми сегментного дисплея, LCD дозволяє відображати більшу кількість інформації, символи задані користувачем і взаємодія з ним є простішою.

13.6 Зміст звіту

Звіт по роботі повинен містити:

– схему моделі;

- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

Контрольні запитання та завдання

1) Де використовуються рідкокристалічні індикатори?
2) Назвіть основні характеристики мікросхеми HD44780.
3) Назвіть призначення виводів індикатора типу HD44780.
4) Назвіть та опишіть команди керування індикатором типу HD44780.
5) Наведіть та опишіть схему підключення індикатора типу HD44780 в складі моделі цифрового вольтметра.
6) Наведіть та опишіть схему алгоритму роботи пристрою керування LCD-дисплеєм.
7) Наведіть та опишіть робочу програму пристрою керування LCD-дисплеєм.
8) Поясненіть використання LCD-індикатора в моделі мережі 1-WIRE.

14 ЛАБОРАТОРНА РОБОТА №11. ДОСЛІДЖЕННЯ МОДЕЛІ МОДУЛЯ АЦП ТА ЦИФРОВОГО ВОЛЬТМЕТРА

Тема: Моделювання модуля АЦП мікроконтролерів сім'ї AVR. Моделювання цифрового вольтметра

Мета: Користуючись пакетом Proteus 8.6 дослідити роботу модуля АЦП та дослідити моделювання цифрового вольтметра

14.1 Порядок виконання роботи

1) Створити модель модуля АЦП та модель цифрового вольтметра в пакеті Proteus 8.6.
2) Розробити схеми алгоритмів роботи цих моделей та робочі програми.
3) Створити відповідні hex-файли та підключити їх до мікроконтролера.
4) Запустити моделі та виконати їх дослідження згідно методичних вказівок.
5) Зробити відповідні висновки.

14.2 Стислі теоретичні відомості

14.2.1 Особливості модуля АЦП у складі AVR-мікроконтролера

До складу більшості моделей AVR-мікроконтролерів входить модуль 10-розрядного аналого-цифрового перетворювача (АЦП) послідовного наближення [1; 10]. Нижче, наприклад, наведено скорочений опис архітектури модуля АЦП у складі мікроконтролера ATmega32, який використано у моделі.

Основні параметри модуля [1; 2]:

- абсолютна похибка: ±2 молодшого значущого розряду (МЗР);
- інтегральна нелінійність: ±0.5 МЗР;
- швидкодія: до 15 тис. вибірок/с.

В зарубіжній літературі МЗР позначається як LSB (Least Significant Bit – молодший значущий розряд). Аналогічно, MSB (Most Significant Bit – старший значущий розряд) відповідає позначенню СЗР.

На вході модуля є восьмиканальний аналоговий мультиплексор, що визначає кількість аналогових каналів перетворення.

Входи АЦП можуть також об`єднуватися попарно для формування каналів з диференціальним входом.

Для них є можливість 10- та 200-кратного попереднього підсилення вхідного сигналу. При коефіцієнтах підсилення 1х та 10х діюча роздільна здатність АЦП складає 8 розрядів, а при коефіцієнті підсилення 200х – 7 розрядів.

Як джерело опорної напруги для АЦП може використовуватись напруга живлення мікроконтролера та внутрішнє або зовнішнє джерело опорної напруги.

Модуль АЦП може функціонувати у двох режимах:

– режим одиночного перетворення, коли запуск кожного перетворення ініціюється користувачем;

– режим безперервного перетворення, коли запуск перетворень виконується безперервно через певні інтервали часу.

Модуль АЦП містить також пристрій вибірки-зберігання (ПВЗ,) що під час перетворення підтримує напругу на вході АЦП на постійному рівні.

Опис функціональної схеми модуля АЦП

Функціональну схему модуля АЦП наведено в [1; 10].

Вхідний аналоговий сигнал перед перетворюванням зберігається у ПВЗ. АЦП працює за принципом послідовного наближення [1; 12]. В кінці перетворення у регістрі даних АЦП (ADCH та ADCL) буде десятирозрядний двійковий код, який еквівалентний вхідній напрузі.

АЦП може обробляти вхідні сигнали у двох режимах: однополярному та диференціальному.

При однополярному режимі вхідний сигнал поступає на один із входів ADC0…ADC7. При диференціальному режимі використовуються входів мультиплексора: ADC0…ADC2. Різниця сигналів між цими входами підсилюється за допомогою диференціального підсилювача і поступає на вхід ПВЗ для подальшого перетворення.

Програмне керування модулем АЦП

Як приклад для деяких моделей сімейства AVR у таблиці 14.1 наведено регістри, що використовуються для керування модулем АЦП.

Таблиця 14.1 – Регістри керування модулем АЦП

Регістр	Адреса	ATmega8535x	ATmega8x	ATmega16x\32x	ATmega163x	ATmega323x	ATmega48x\88x\168x	ATmega64x	ATmega164x\324x\644x	ATmega165x	ATmega325x/3250x, ATmega645x/6450x	ATmega640x, ATmegal280x/1281x, ATmega2560x/2561x	ATmega128x	Опис
ADCSR	$06($26)		♦		♦	♦								Регістр керування і стану
ADCSRA	$06 ($26)	♦		♦				♦					♦	Регістр керування і стану А
	($7A)						♦		♦	♦	♦	♦		
ADCSRB	($8E)							♦						Регістр керування і стану В
	($7B)						♦		♦	♦	♦	♦		
ADMUX	$07 ($27)	♦	♦	♦	♦	♦		♦					♦	Регістр керування мультиплексором
	(7C)						♦		♦	♦	♦	♦		
SFIOR	$30 ($50)	♦	♦	♦										Регістр спеціальних функцій
	$20 ($40)												♦	

Формат регістрів ADCSRA (ADCSR) і ADMUX наведено на рисунках 14.2 і 14.3 а короткий опис функцій їх розрядів наведено у таблицях 14.2 і 14.3 відповідно. Формат регістра SFIOR наведено на рисунку 14.4 (не використовувані розряди регістра позначено як «–»).

.

	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADFR	ADIF	ADIÉ	ADPS2	ADPS1	ADPSO	ATmega8x ATmega128x
Зчитування(R)/Запис(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	
	7	6	5	4	3	2	1	0	
	ADEN	ADSC	ADATE	ADIF	ADIÉ	ADPS2	ADPS1	ADPSO	Інші моделі
Зчитування(R)/Запис(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

ATmega 8x – ADCSR
Інші моделі – ADCSRA

Рисунок 14.2 – Формат регістра ADCSRA (ADCSR)

	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	–	MUX3	MUX2	MUX1	MUX0	ATmega8x ATmega48x/88x/168x
Зчитування(R)/Запис(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	
	7	6	5	4	3	2	1	0	
	REFS1	REFS0	ADLAR	MUX4	MUX3	MUX2	MUX1	MUX0	Інші моделі
Зчитування(R)/Запис(W)	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Початкове значення	0	0	0	0	0	0	0	0	

Рисунок 14.3 – Формат регістра ADMUX

Таблиця 14.2 – Розряди регістра ADCSRA (ADCSR*)

Розряд	Назва	Опис
7	ADEN	Дозвіл АЦП (1 – увімкнено, 0 – вимкнено)
6	ADSC	Запуск перетворення (1 – почати перетворення)
5	ADATE (ADFR **)	Вибір режиму роботи АЦП (0 – одиночне, 1 – безперервне перетворення)
4	ADIF	Прапорець переривання завершення АЦП
3	ADIE	Дозвіл переривання від завершення АЦП
2..0	ADPS2:ADPS0	Вибір частоти перетворення (див. таблицю 12.6)
* В моделі ATmega8x ** В моделях ATmega8x, ATmega128x		

Таблиця 14.3 – Розряди регістра ADMUX

Розряд	Назва	Опис	Модель
7..6	REFS1:REFS0	Вибір джерела опорної напруги (див. таблицю 12.5)	Всі моделі
5	ADLAR	Ліве вирівнювання результату (див. таблицю 12.9)	Всі моделі
4	–	Зарезервовано	ATmega8x
	MUX4	Вибір вхідних каналів (див. таблицю 21.8)	Всі моделі крім ATmega8x
3..0	MUX3…MUX0	Вибір вхідних каналів і частоти перетворення (див. таблицю 12.8)	Всі моделі

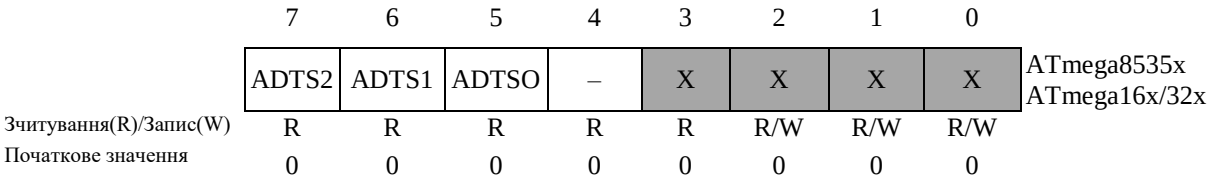


Рисунок 14.4 – Формат регістра SFIOR

Для дозволу робити АЦП необхідно записати одиницю у розряд ADEN регістра ADCSRA (ADCSR), а для заборони – відповідно нуль. Якщо АЦП буде вимкнено під час циклу перетворення, то перетворення не буде завершено (в регістрі даних АЦП залишиться результат попереднього перетворення).

Запуск АЦП можливий не тільки командою користувача, але й перериванням від деяких периферійних пристроїв, наявних у складі мікроконтролера. Для вибору режиму роботи в цих моделях використовується розряд ADATE регістра ADCSRA і розряди ADTS2...0 регістра SFIOR.

Якщо розряд ADATE скинуто в "0", АЦП працює в режимі одиночного перетворення. Якщо ж розряд ADATE встановлено в "1", функціонування АЦП визначається вмістом розрядів ADTS2...0 відповідно до таблиці 14.4.

Таблиця 14.4 – Джерело сигналу для запуску перетворення

ADTS2	ADTS1	ADTS0	Джерело стартового сигналу
0	0	0	Режим безперервного перетворення
0	0	1	Переривання від аналогового компаратора
0	1	0	Зовнішнє переривання INT0
0	1	1	Переривання за подією "Збіг А" таймера/лічильника Т0
1	0	0	Переривання за переповненням таймера/лічильника Т0
1	0	1	Переривання за подією "Збіг В" таймера/лічильника Т1
1	1	0	Переривання за переповненням таймера/лічильника Т1
1	1	1	Переривання за подією "Захоплення" таймера/лічильника Т1

Запуск кожного перетворення в режимі одиночного перетворення, а також запуск першого перетворення в режимі безперервного перетворення здійснюється встановленням в "1" розряду ADSC регістра ADCSRA (ADCSR).

Запуск перетворення за перериванням здійснюється при встановлені в "1" прапорця обраного переривання. Розряд ADSC регістра ADCSRA при цьому апаратно встановлюється в "1". Запуск перетворення в цих режимах також може бути здійснений встановленням в "1" розряду ADSC регістра ADCSRA.

Модуль АЦП може використовувати різні джерела опорної напруги (ДОН). Вибір конкретного джерела опорної напруги здійснюється за допомогою розрядів REFS1:REFS0 регістра ADMUX (таблиця 14.5).

Таблиця 14.5 – Вибір джерела опорної напруги

REFS1	REFS0	Джерело опорної напруги[1)]	Модель
0	0	Зовнішнє ДОН, підключено до виводу AREF; внутрішнє ДОН відключено	Всі моделі
0	1	Напруга живлення AVcc[2)]	Всі моделі
1	0	Внутрішнє ДОН напругою 1.1 В[2)]	ATmega164x/324x/644x ATmega640x/1280x/1281x ATmega2560x/2561x
		Зарезервовано	Решта моделей
1	1	Внутрішнє ДОН напругою 2.56 В[2]	ATmega48x/88x/168x, mega32 ATmega165/325x/3250x ATmega645x/6450x
		Внутрішнє ДОН напругою 1.1 В[2)]	Решта моделей

[1)] При роботі з підсиленням 10x чи 200x в якості внутрішнього ДОН можна використовувати тільки ДОН яке, підключено при REFS1:0 = 11.
[2)] Якщо до виводу AREF підключено зовнішнє джерело напруги, дані варіанти використовуватись не можуть.

АЦП перетворює вхідну аналогову напругу в 10-розрядний код методом послідовного наближення [12]. Мінімальне значення відповідає рівню GND, а максимальне рівню AREF мінус рівень, який відповідає одиниці молодшого розряду.

Робота АЦП дозволяється шляхом встановлення розряду ADEN у регістрі ADCSRA. Вибір опорного джерела та каналу перетворення неможливо виконати до встановлення ADEN. Якщо ADEN = 0, то АЦП не споживає струм, тому при переході в економічні режими сну рекомендується попередньо відключити АЦП.

АЦП генерує 10-розрядний результат, що міститься в парі регістрів даних АЦП: ADCH і ADCL. У початковому стані результат перетворення розміщується в молодших 10-ти розрядах 16-розрядного слова (вирівнювання вправо), але

може бути розміщений у старших 10-ти розрядах (вирівнювання вліво) шляхом встановлення розряду ADLAR у регістрі ADMUX (таблиці 14.3).

Практична корисність подання результату з вирівнюванням вліво існує, коли досить точності 8-розрядного значення. В цьому випадку необхідно читати тільки регістр ADCH. В іншому ж випадку необхідно першим читати вміст регістра ADCL, а потім ADCH, чим гарантується, що обидва байти є результатом того самого перетворення. Як тільки виконано читання ADCL блокується доступ до регістрів даних з боку АЦП. Це означає, що якщо зчитано ADCL і перетворення завершується перед читанням регістра ADCH, то жоден з регістрів не може модифікуватися й результат перетворення губиться. Після читання ADCH доступ до регістрів ADCH і ADCL з боку АЦП знову дозволяється.

АЦП генерує власний запит на переривання за завершенням перетворення. Якщо між читанням регістрів ADCH і ADCL доступ до даних для АЦП заборонено, то переривання виникне, навіть якщо результат перетворення буде загублено.

Одиночне перетворення запускається шляхом запису логічної одиниці у розряд запуску перетворення АЦП – ADSC. Даний розряд залишається у високому стані в процесі перетворення й скидається за завершенням перетворення. Якщо в процесі перетворення перемикається канал аналогового введення, то АЦП автоматично завершить поточне перетворення, перш ніж перемкне канал.

У режимі автоматичного перезапуску АЦП безупинно оцифровує аналоговий сигнал і оновлює регістр даних АЦП. Даний режим задається шляхом запису логічної одиниці у розряд ADFR (ADATE) регістра ADCSR (ADCSRA). Перше перетворення ініціюється шляхом запису логічної одиниці у розряд ADSC регістра ADCSR (ADCSRA). У даному режимі АЦП виконує послідовні перетворення, незалежно від того скидається прапорець переривання АЦП: ADIF, чи ні.

Формування тактової частоти АЦП

Для формування тактового сигналу для АЦП використовується попередній дільник, схему якого наведено на рисунку 14.5.

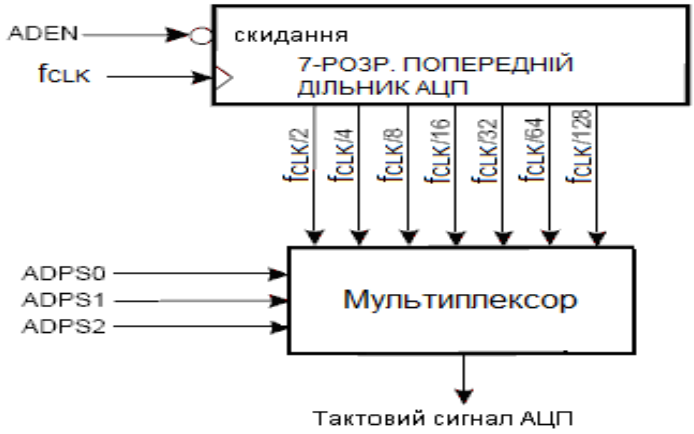


Рисунок 14.5 – Схема попереднього дільника АЦП

Попередній дільник формує похідні частоти відносно частоти синхронізації мікроконтролера. Коефіцієнт ділення встановлюється за допомогою розрядів ADPSn у регістрі ADCSRA (таблиця 14.6).

Попередній дільник починає лічбу з моменту включення АЦП встановленням розряду ADEN у регістрі ADCSRA. Він працює доки розряд ADEN = 1 та скидається, коли ADEN = 0.

Якщо потрібно забезпечити максимальну роздільну здатність (10 розрядів), то частота на вході схеми послідовного наближення повинна бути в діапазоні 50...200 кГц [1]. Якщо достатньо точності менше 10 розрядів, але потрібна більш висока частота перетворення, то частота на вході АЦП може бути встановлена понад 200 кГц.

Таблиця 14.6 – Задання коефіцієнта ділення попереднього дільника АЦП

ADPS2	ADPS1	ADPS0	Коефіцієнт ділення
0	0	0	2
0	0	1	2
0	1	0	4
0	1	1	8
1	0	0	16
1	0	1	32
1	1	0	64
1	1	1	128

Часові діаграми роботи АЦП

На рисунках 14.6…14.9 наведено часові діаграми роботи АЦП у різних режимах.

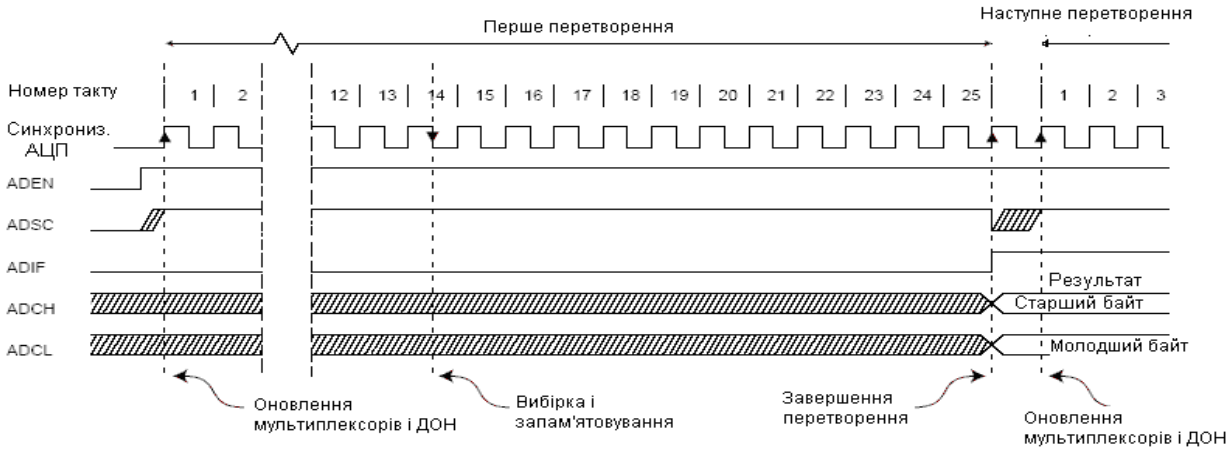


Рисунок 14.6 – Часові діаграми роботи АЦП при першому перетворенні в режимі одиночного перетворення

Рисунок 14.7 – Часові діаграми роботи АЦП у режимі одиночного перетворення

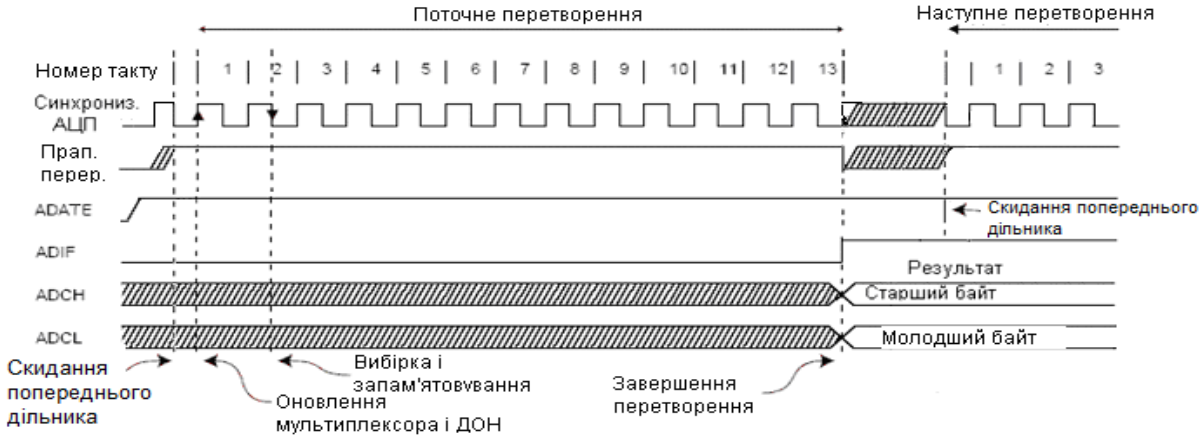


Рисунок 14.8 – Часові діаграми роботи АЦП у режимі запуску за перериванням

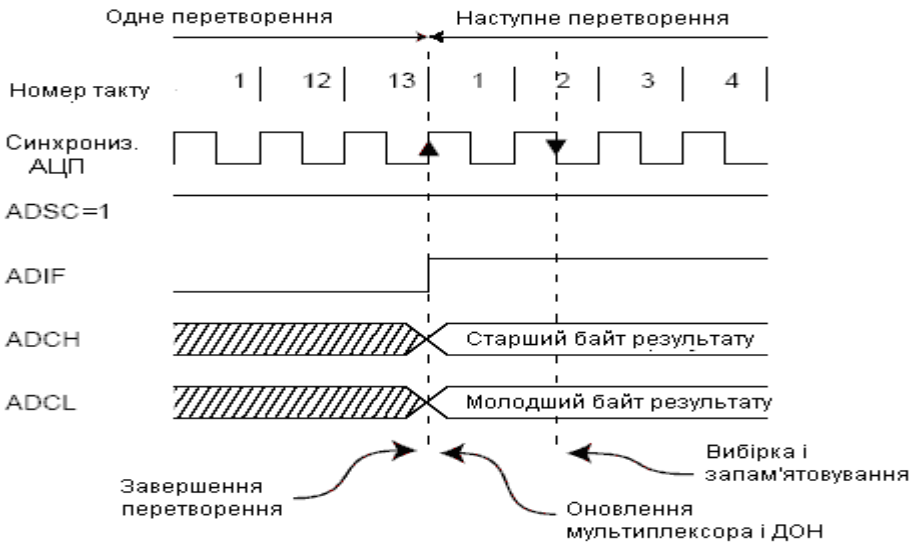


Рисунок 14.9 – Часові діаграми роботи АЦП у режимі автоматичного перезапуску для однополярного перетворення

Якщо ініціюється одиночне однополярне перетворення встановленням розряду ADSC у регістрі ADCSRA, то перетворення починається з наступного наростаючого фронту тактового (синхро) сигналу АЦП.

У режимі одиночного перетворення нове перетворення може бути запущено відразу ж після скидання розряду ADSC (до збереження результату поточного перетворення). Однак реально цикл перетворення почнеться не раніше ніж через один такт після закінчення поточного перетворення.

Нормальне перетворення вимагає 13 тактів синхронізації АЦП. Перше перетворення після включення АЦП (встановлення ADEN в регістрі ADCSRA) вимагає 25 тактів синхронізації АЦП за рахунок необхідності ініціалізації модуля.

Після початку нормального перетворення на вибірку-зберігання затрачується 1,5 такти синхронізації АЦП, а після початку першого перетворення – 13,5 тактів. По завершенні перетворення результат зберігається в регістрах даних АЦП і встановлюється прапорець ADIF. У режимі одиночного перетворення одночасно скидається розряд ADSC. Програмно розряд ADSC може бути знову встановлено і нове перетворення буде ініційовано першим наростаючим фронтом тактового сигналу АЦП.

У режимі безперервного перетворення (автоматичного перезапуску) нове перетворення починається відразу по завершенні попереднього, при цьому ADSC залишається у високому стані. Час перетворення для різних режимів перетворення представлено в таблиці 14.7.

Таблиця 14.7 – Час перетворення АЦП

Тип перетворення	Тривалість вибірки–зберігання (у тактах з моменту початку перетворення)	Час перетворення (у тактах)
Перше перетворення	14,5	25
Нормальне однополярне перетворення	1,5	13
Нормальне диференціальне перетворення	1,5/2,5	13/14

Керування вхідним мультиплексором

Виводи мікроконтролера, які підключені до входу АЦП, визначаються станом розрядів MUX4...MUX0 регістра ADMUX (рисунок 14.3). Для каналів з диференціальним входом зазначені розряди визначають також коефіцієнт попереднього підсилення вхідного сигналу. Керування вхідним мультиплексором для деяких мікроконтролерів відображає таблиця 14.8.

Програмування розрядів MUXn (n = 0,1,…,4) і REFS1:0 у регістрах ADMUX та ADCSRB підтримується буферизацією через тимчасовий регістр. Цим гарантується, що нові налаштування каналу перетворення та опорного джерела набудуть чинності в безпечний момент для перетворення.

До початку перетворення будь-які зміни каналу та опорного джерела набувають чинності відразу після їхньої модифікації. Як тільки починається процес перетворення доступ до зміни каналу та опорного джерела блокується, чим гарантується достатність часу на перетворення для АЦП. Безперервність модифікації повертається на останньому такті АЦП перед завершенням перетворення (перед встановленням прапорця ADIF у регістрі ADCSRA). Зверніть увагу, що перетворення починається наступним наростаючим фронтом тактового сигналу АЦП після встановлення ADSC.

Таким чином, користувачеві не рекомендовано записувати нове значення каналу або опорного джерела в ADMUX до 1-го такту синхронізації АЦП після встановлення ADSC.

Таблиця 14.8 – Керування вхідним мультиплексором у моделях ATmega16x/164x/32x/8535x/64x/128x/164x/165x/325x/3250x/645x/6450x/ 1281x/2561x

MUX4…MUX0	Однополярний вхід	Диференціальний вхід		Попереднє підсилення
		(додатний)	(від'ємний)	
00000	ADC0	Не застосовується		
00001	ADC1			
00010	ADC2			
00011	ADC3			
00100	ADC4			
00101	ADC5			
00110	ADC6			
00111	ADC7			
01000[1]	Не застосовується	ADC0	ADC0	10x
01001[1]		ADC1	ADC0	10x
01010[1]		ADC0	ADC0	200x
01011[1]		ADC1	ADC0	200x
01100[1]		ADC2	ADC2	10x
01101[1]		ADC3	ADC2	10x
01110[1]		ADC2	ADC2	200x
01111[1]		ADC3	ADC2	200x
10000[1]		ADC0	ADC1	1x
10001[1]		ADC1	ADC1	1x
10010[1]		ADC2	ADC1	1x
10011[1]		ADC3	ADC1	1x
10100[1]		ADC4	ADC1	1x
10101[1]		ADC5	ADC1	1x
10110[1]		ADC6	ADC1	1x
10111[1]		ADC7	ADC1	1x
11000[1]		ADC0	ADC2	1x
11001[1]		ADC1	ADC2	1x
11010[1]		ADC2	ADC2	1x
11011[1]		ADC3	ADC2	1x
11100[1]	Не застосовується	ADC4	ADC2	1x
11101[1]		ADC5	ADC2	1x
11110	1.22 В (1,1 В[1])	Не застосовується		
11111	0В (GND)			
[1] У моделях ATmega165x/325x/3250x/645x/6450x/1251x/2561x				

Збереження результату перетворення

Після завершення перетворення (при встановленні в "1" прапорця ADIF регістра ADCSR) його результат зберігається в регістрі даних АЦП. Оскільки АЦП має 10 розрядів, цей регістр фізично розміщено у двох регістрах

введення/виведення ADCH:ADCL, доступних тільки для читання. Ці регістри розташовані за адресами $05:$04 і при включенні мікроконтролера містять значення "$0000".

У початковому стані результат перетворення вирівнюється вправо (старші 6 розрядів регістра ADCH – не є значущими).

Однак він може вирівнюватися також вліво (молодші 6 розрядів регістра ADCL – не є значущими). Для керування вирівнюванням результату перетворення призначено розряд ADLAR регістра ADMUX. Якщо цей розряд встановлено в "1", результат перетворення вирівнюється за лівою границею 16-розрядного слова, якщо скинутий в "0" – за правою границею.

При використанні диференціального режиму перетворення результат представляється в коді двійкового доповнення до двох (в додатковому коді).

У таблиці 14.9 наведено приклади вирівнювання результату вліво та вправо.

Таблиця 14.9 – Вирівнювання результату АЦП

ADLAR	Розряд	15	14	13	12	11	10	9	8	ADCH
0		–	–	–	–	–	–	ADC9	ADC8	
		ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADC1	ADC0	ADCL
	Розряд	7	6	5	4	3	2	1	0	
	R/$\overline{W}$	R	R	R	R	R	R	R	R	ADCH
		R	R	R	R	R	R	R	R	ADCL
	Поч. зн.	0	0	0	0	0	0	0	0	ADCH
		0	0	0	0	0	0	0	0	ADCL
1	Розряд	15	14	13	12	11	10	9	8	
		ADC9	ADC8	ADC7	ADC6	ADC5	ADC4	ADC3	ADC2	ADCH
		ADC1	ADC0	–	–	–	–	–	–	ADCL
	Розряд	7	6	5	4	3	2	1	0	
	R/$\overline{W}$	R	R	R	R	R	R	R	R	ADCH
		R	R	R	R	R	R	R	R	ADCL
	Поч. зн.	0	0	0	0	0	0	0	0	ADCH
		0	0	0	0	0	0	0	0	ADCL

Звернення до регістрів ADCH і ADCL для отримання результату перетворення повинно виконуватися в певній послідовності: спочатку необхідно прочитати регістр ADCL, а потім ADCH. Ця вимога пов’язана з тим, що після звернення до регістра ADCL процесор блокує доступ до регістрів даних з боку

АЦП доти, поки не буде прочитано регістр ADCH. Завдяки цьому можна бути впевненим, що при читанні регістрів ADCH, ADCL у них будуть перебувати складові того самого результату. Відповідно, якщо чергове перетворення завершиться до звернення до регістра ADCH, результат перетворення буде загублено. З іншого боку, якщо результат перетворення вирівнюється вліво й досить точності 8-розрядного значення, для отримання результату можна прочитати тільки вміст регістра ADCH.

Особливості підключення джерела опорної напруги

Джерело опорної напруги (ДОН) для АЦП ($U_{ДОН}$) визначає діапазон перетворення АЦП. Якщо рівень однополярного сигналу понад $U_{ДОН}$, то результатом перетворення буде 0x3FF. В якості $U_{ДОН}$ можуть виступати AVCC, внутрішнє ДОН 2,56 В, або зовнішнє ДОН, що підключено до виводу AREF. AVCC підключається до АЦП через пасивний ключ. Внутрішня опорна напруга 2,56 В генерується внутрішнім еталонним джерелом VBG, що буферизовано внутрішнім підсилювачем. Зовнішній вивід AREF зв’язаний безпосередньо з АЦП. Щоб знизити вплив шумів на опорне джерело треба підключати конденсатор між виводом AREF і спільним виводом [1; 10].

Якщо користувач використовує зовнішнє опорне джерело, що підключено до виводу AREF, то не допускається використання іншої опції опорного джерела, тому що це приведе до шунтування зовнішньої опорної напруги. Якщо до виводу AREF не прикладена напруга, то користувач може обрати AVCC і 2,56 В як опорне джерело. Результат першого перетворення після перемикання опорного джерела може характеризуватися низькою точністю, тому користувачеві рекомендується його ігнорувати.

Результат перетворення АЦП

Для каналів з однополярним (несиметричним) входом результат перетворення визначається виразом:

$$ADC = \frac{1023 U_{IN}}{U_{REF}}, \qquad (14.1)$$

де U_{IN} – значення вхідної напруги, U_{REF} – величина опорної напруги, ADC – десятковий еквівалент двійкового коду на виході АЦП.

Коефіцієнт передачі АЦП визначається формулою:

$$K_{ПЕР} = 1023/U_{REF} \text{ [МЗР/мВ]}. \quad (14.2)$$

На рисунку 14.10 представлено функцію перетворення АЦП в однополярному режимі.

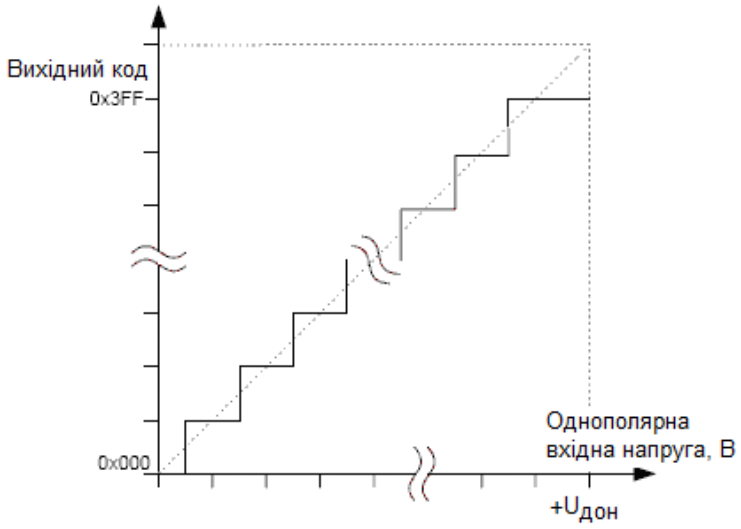


Рисунок 14.10 – Функція перетворення АЦП при зміні однополярного сигналу

Код 0x000 відповідає рівню аналогової землі, а 0x3FF – рівню напруги ДОН мінус рівень, який відповідає одному кроку квантування за напругою.

Зв’язок між вхідним сигналом та вихідними кодами для однополярного режиму відображає таблиця 14.10.

Таблиця 14.10 – Зв’язок між вхідним и вихідним кодами

$U_{АЦПn}$*	Зчитаний код	Відповідне десяткове значення
$U_{АЦПm}$ + $U_{ДОН}$	0x3FF	1023
$U_{АЦПm}$ + 0.999 $U_{ДОН}$	0x3FF	1023
$U_{АЦПm}$ + 0.998 $U_{ДОН}$	0x3FE	1022
…	…	…
$U_{АЦПm}$ + 0.001 $U_{ДОН}$	0x001	1
$U_{АЦПm}$	0x000	0

$U_{АЦПm}$ – вхідна напруга, яка дорівнює нулю, $U_{АЦПn}$ – поточне значення вхідної напруги.

Приклад. Нехай ADMUX = 0x00…0x07 (будь-який однополярний вхід), напруга на одному з входів 1000 мВ, напруга ДОН дорівнює 2,56 В.

Тоді:

$$\text{Код АЦП} = 1024 * 1000 / 2560 = 400 = 0x190.$$

Для каналів з диференціальним входом результат перетворення описується в [1; 10].

14.2.2 Моделювання модуля АЦП у пакеті PROTEUS 8.6

Схема моделі

Схему моделювання модуля АЦП у пакеті PROTEUS 8.6 зображено на рисунку 14.11.

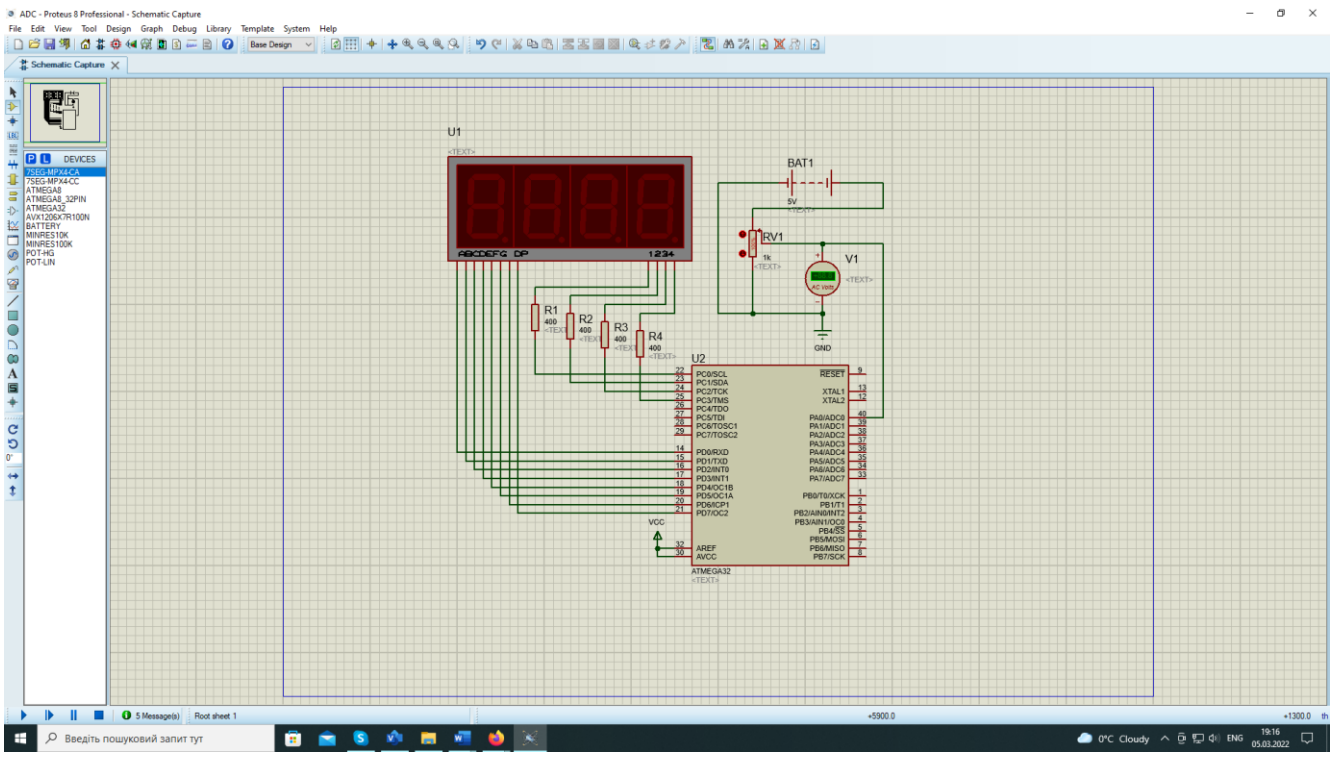


Рисунок 14.11 – Схема моделювання модуля АЦП у пакеті PROTEUS 8.6

Дана схема складається з наступних елементів:

– U2 – AVR-мікроконтролер ATMEGA32;
– R1…R4 – резистори опором 400Ом;
– RV1 – резистор змінного опором 10кОм;
– BAT1 – джерело напруги 5В;

– VCC – джерело живлення мікроконтролера;
– U1 – семисегментний чотирипозиційний індикатор;
– V1 – вольтметр для вимірювання вхідної напруги.

Вхідна аналогова напруга подається на лінію PA0 (ADC0) порту A з виходу дільника напруги +5В (RV1). За допомогою резистора RV1 можна змінювати величину вхідної напруги від 0В до +5В.

Після перетворення її значення у цифровий еквівалент, відповідна величина виводиться у порти мікроконтролера для відображення на індикаторі. Індикатор підключено до ліній PC0…PC3 порту C та ліній PD0…PD7 порту D.

АЦП програмується у режим безперервного перетворення з використанням каналів з однополярним входом. Вище для однополярного режиму представлено функцію перетворення АЦП (рисунок 14.10), а таблиця 14.10 відображає зв'язок між вхідним сигналом та вихідними кодами.

Для каналів з однополярним (несиметричним) входом результат перетворення визначається виразом (14.1), а коефіцієнт передачі – виразом (14.2), які наведено вище.

При U_{REF} = 5В, $К_{ПЕР}$ = 1023/5000 = 0,2046 [МЗР/мВ].

Наприклад, якщо Uвх = 5В = 5000мВ, то десятковий код на виході буде дорівнювати

$$ADC = 5000 \bullet 0{,}2046 = 1023.$$

Нижче на рисунку 14.12 наведено схему моделі після запуску процесу моделювання.

Якщо за допомогою дільника RV1 встановити на вольтметрі V1 напругу +5В, то на індикаторі U1 побачимо число 1023, що відповідає наведеному вище розрахунку.

На початковому етапі, вхідна напруга для АЦП подається на лінію PA0 порту A з виходу дільника напруги +5В (RV1). Після перетворення її значення у цифровий еквівалент, розрахована величина виводиться у порти мікроконтролера

для відображення на індикатори. Індикатори підключені до порту C (PC0…PC3) та порту D (PD0…PD7).

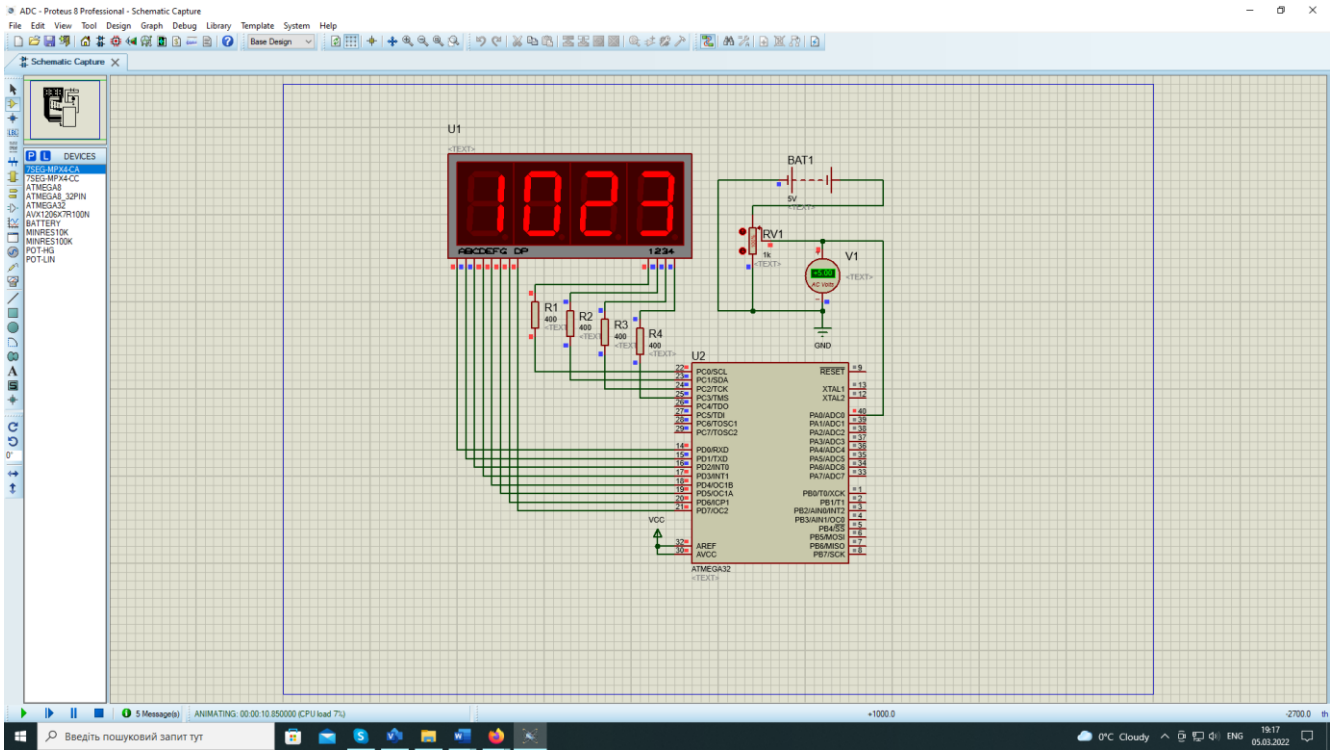


Рисунок 14.12 – Схема моделі АЦП після запуску процесу моделювання

За допомогою резистора RV1, можна змінювати величину вхідної напруги у режимі реального часу. Значення вхідної напруги може змінюватися від 0В до 5В.

Для налаштування параметрів АЦП у програмі виконуються такі дії: у регістрі ADCSRA встановлюються в логічну одиницю біти: ADEN – дозвіл АЦП, ADSC – запуск перетворення, ADATE – безперервний режим роботи, ADPS2 та ADPS1 – коефіцієнт ділення переддільника: 64, ADIE – дозвіл переривань від АЦП. Щоб обрати зовнішнє джерело опорної напруги біти REFS1 і REFS0 у регістрі ADMUX скидаються у логічний нуль.

Для відображення отриманого десяткового коду перетворення АЦП на семисегментних індикаторах використовується таймер/лічильник T2 мікроконтролера. При його переповненні відбувається переривання та результат аналого-цифрового перетворення відображається на чотирьохпозиційному семисегментному індикаторі.

Для цього у порт С подається число, в якому в логічну одиницю встановлено біт для вибору активного індикатора. Перший індикатор, якщо лічити зліва направо, обирається розрядом PC0 = 1 (інші біти порту С дорівнюють нулю). При виборі другого індикатора PC1 = 1, інші біти порту С дорівнюють нулю, і т. д. Після вибору індикатора, за допомогою математичної операції отримання остачі, визначається відповідна цифра, яка є індексом в масиві SEG (див. робочу програму) та відповідає своєму семисегментному аналогу. Оскільки окремі сегменти індикатора, який використано у моделі, активуються нульовим сигналом, то відповідне значення масива SEG у програмі перед виведенням в порт D інвертується.

Для дозволу переривань від таймера/лічильника Т2 використовується регістр TIMSK (Timer/Counter InterruptMaSK Register – регістр маски переривань від таймерів/лічильників) [1; 2]. Для дозволу переривання необхідно встановити в одиницю розряд TOIE2 цього регістра, а також встановити в одиницю прапорець глобального дозволу переривань – I регістра SREG.

В якості тактового сигналу fclk2 таймера/лічильника Т2 використовується масштабований системний тактовий сигнал: fclk2 = fclkI/0/N, де N – коефіцієнт ділення попереднього дільника. Програмування N здійснюються за допомогою розрядів CS22...CS20 регістра керування таймером – TCCR2. В моделі обрано N = 8. Для програмування цього значення згідно з [1] треба задати: CS22 = CS20 = 0, CS21 = 1.

14.2.3 Схема алгоритму роботи моделі

Схему алгоритму роботи моделі наведено на рисунках 14.13…14.16.

Після виконання блоку ініціалізації (блок 1) АЦП запускається в режим безперервного перетворення. Паралельно починає роботу таймер/лічильник Т2, який при кожному переповненні викликає підпрограму його обробки. Задачею цієї підпрограми є виведення на дисплей результату перетворення від АЦП, який формується у десятковому коді. До завершення першого перетворення АЦП на дисплей виводиться нульове значення.

Коли АЦП закінчує перше перетворення, встановлюється відповідний прапорець, виконання програми переривається та викликається підпрограма обробки цього переривання.

При черговому переповненні таймер отримує результат від АЦП та відповідна підпрограма виводить його на дисплей.

Далі АЦП буде безперервно виконувати наступні перетворення, результат яких за допомогою таймера також буде виводитись на дисплей.

Схему загального алгоритму роботи модуля наведено на рисунку 14.13.

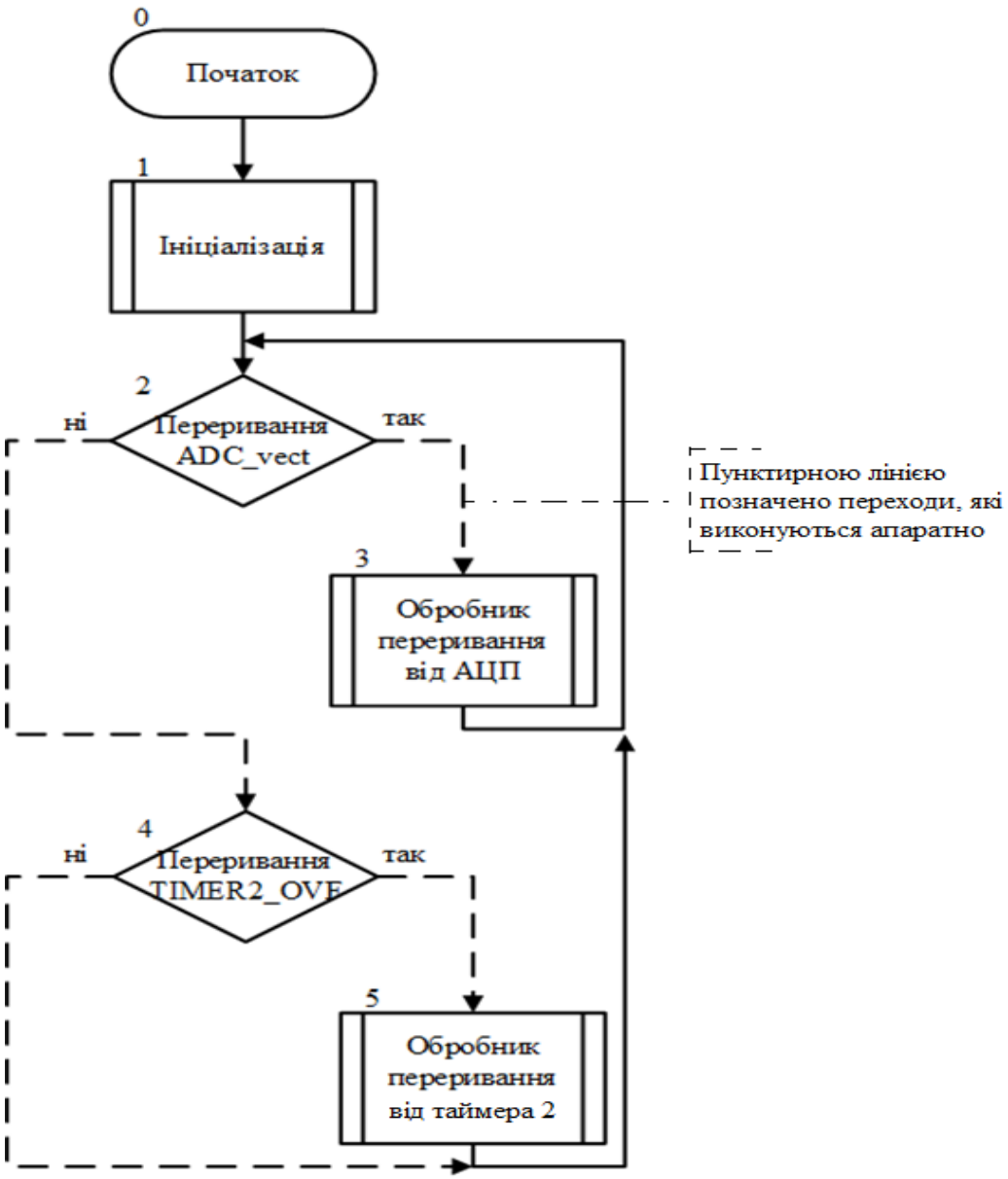


Рисунок 14.13 – Схема загального алгоритму роботи модуля

Схему алгоритму ініціалізації наведено на рисунку 14.14.

Рисунок 14.14 – Схема алгоритму ініціалізації

Схему алгоритму обробки переривання від таймера 2 наведено на рисунку 14.15.

Рисунок 14.15 – Схема алгоритму обробки переривання від таймера 2

Схему алгоритму обробки переривання завершення перетворення АЦП наведено на рисунку 14.16.

Рисунок 14.16 – Схема алгоритму обробки переривання завершення перетворення АЦП

14.2.4 Робоча програма

```
// Підключення заголовних файлів бібліотек
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

// Масив значень для відображення цифр на семисегментних індикаторах
//-------------0-----1-----2-----3-----4-----5-----6-----7-----8-----9----dp
char SEG[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F, 0x80};

// Ініціалізації глобальних змінних
volatile unsigned char current_indicator = 0;
volatile unsigned int display = 0;

// Блок 1 - Ініціалізація програми
```

```
int main (void)
{
     // Блок 1.1 – Налаштування портів семисегментних індикаторів на
     // виведення та їх обнулення
    DDRC = 0xFF;
    PORTC = 0x00;
    DDRD = 0xFF;
    PORTD = 0x00;
    // Блок 1.2 – Налаштування таймера 2
    TIMSK |= (1 << TOIE2); // Дозвіл переривання від таймера 2
    TCCR2 |= (1 << CS21); // Коефіцієнт ділення попереднього
          //дільника = 8

    // Блок 1.3 – Налаштування АЦП
    ADCSRA |= (1 << ADEN) // Дозвіл АЦП
    |(1 << ADSC) // Запуск перетворення
    |(1 << ADATE) // Безперервний режим роботи АЦП
    |(1 << ADPS2)|(1 << ADPS1) // Коефіцієнт ділення переддільника
                     // АЦП = 64
    |(1 << ADIE); // Дозвіл переривань від АЦП
    ADMUX &= (~(1 << REFS1))&(~(1 << REFS0)); // Зовнішнє ДОН

    // Блок 1.4 – Глобальний дозвіл переривань
    sei();

    // Блок 1.5 – Основний цикл
    while(1)
    {
        _delay_ms(50); // Блок 1.6   Затримка 50 мс
    }
}

// Блок 2 – Умова переривання від АЦП
ISR (ADC_vect)
{// Блок 3 – Обробник перериваннь від АЦП
    Display = ADC; // Блок 3.1 – Присвоюємо глобальній змінній
//поточне значення АЦП
}

// Блок 4 – Умова переривання від таймера Т2
ISR (TIMER2_OVF_vect)
{// Блок 5 – Обробник переривань від таймера Т2
    PORTD = 0xFF; // Блок 5.1 – Вимикаємо всі сегменти
    PORTC = (1 << current_indicator); // Блок 5.2 – Обираємо
поточний //індикатор

    //Блок 5.3
    switch (current_indicator)
    {
        case 0:
        PORTD = ~(SEG[display % 10000 / 1000]); // Вмикаємо цифру
//одиниць
```

```
            break;
            case 1:
            PORTD = ~((SEG[display % 1000 / 100])); // Вмикаємо цифру
//десятків
            break;
            case 2:
            PORTD = ~(SEG[display % 100 / 10]); // Вмикаємо цифру сотень
            break;
            case 3:
            PORTD = ~(SEG[display % 10 / 1]); // Вмикаємо цифру тисяч
            break;
        }
        if ((current_indicator++) > 2) // Блок 5.4 – Переходимо на
//наступний індикатор
        current_indicator = 0; // Блок 5.5 – Обнуляємо, якщо
                    //current_indicator за межами
}
```

Нижче наведено пояснення фрагменту програми виведення на чотири семисегментних індикатори результату перетворення АЦП – ADC, який надходить у десятковому коді згідно з формулою 14.1, яку наведено вище. Для виведення ADC використовується математична операція «остача», яка в мові програмування «С» обчислюється оператором «%».

Наприклад, при Uвх АЦП = 5В значення ADC згідно з роботою моделі в Proteus 8.6 дорівнює 1023. Це значення програма обробляє наступним чином:

1023%10000/1000 = 1023/1000 = 1,023. Цифра 1 виводиться на перший індикатор (зліва направо).

Далі відбуваються наступні дії:

1023%1000/100 = 23/100 = 0,23. Цифра 0 виводиться на другий індикатор;

1023%100/10 = 23/10 = 2,3. Цифра 2 виводиться на третій індикатор;

1023%10/1 = 3/1 = 3. Цифра 3 виводиться на четвертий індикатор.

Наприклад, ADC = 686, тоді:

686%10000/1000 = 686/1000 = 0,686. Цифра 0 виводиться на перший індикатор.

686%1000/100 = 686/100 = 6,86. Цифра 6 виводиться на другий індикатор;

686%100/10 = 86/10 = 8,6. Цифра 8 виводиться на третій індикатор;

686%10/1 = 6/1 = 6. Цифра 6 виводиться на четвертий індикатор.

14.3 Зміст звіту

Звіт по роботі повинен містити:

- схему моделі;
- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

14.4 Дослідження моделі цифрового вольтметра

14.4.1 Порядок виконання роботи

1) Створити модель пристрою в пакеті Proteus 8.6.
2) Розробити схему алгоритму роботи моделі та робочу програму.
3) Створити hex-файл та підключити його до мікроконтролера.
4) Запустити модель та виконати її дослідження згідно методичних вказівок.
5) Зробити відповідні висновки.

14.4.2 Стислі теоретичні відомості

Основним елементом цифрового вольтметра є аналого-цифровий перетворювач, в якості якого в роботі використовується відповідний модуль мікроконтролера ATmega32. Архітектуру цього модуля розглянуто у підрозділі 14.1.

14.4.3 Опис моделі

Робочу модель цифрового вольтметра показано на рисунку 14.17.

З лівого боку моделі зображено 7-сегментний чотирьохпозиційний цифровий дисплей – U1, який з’єднано з портами C та D мікроконтролера ATMega32 відповідними лініями зв’язку. У верхній частині рисунку знаходиться батарея BAT1 та резистор RV1, напругу з виходу якого можна змінювати. Ця вхідна аналогова напруга подається на лінію PA0 (ADC0) порту A

мікроконтролера та перетворюється модулем АЦП у цифровий еквівалент. Праворуч від RV1 знаходиться вольтметр для вимірювання вхідної напруги, яка знімається зі змінного резистора – V1.

Особливості керування цифровим дисплеєм описано у попередньому підрозділі. Різниця між виведенням результату перетворення АЦП, який описано вище, полягає в тому, що раніше на дисплей виводилася цифрова інформація у чотирьохрозрядному десятковому коді, а в моделі цифрового вольтметра відображається абсолютне значення вхідної напруги у вольтах з точністю до сотих долей вольта.

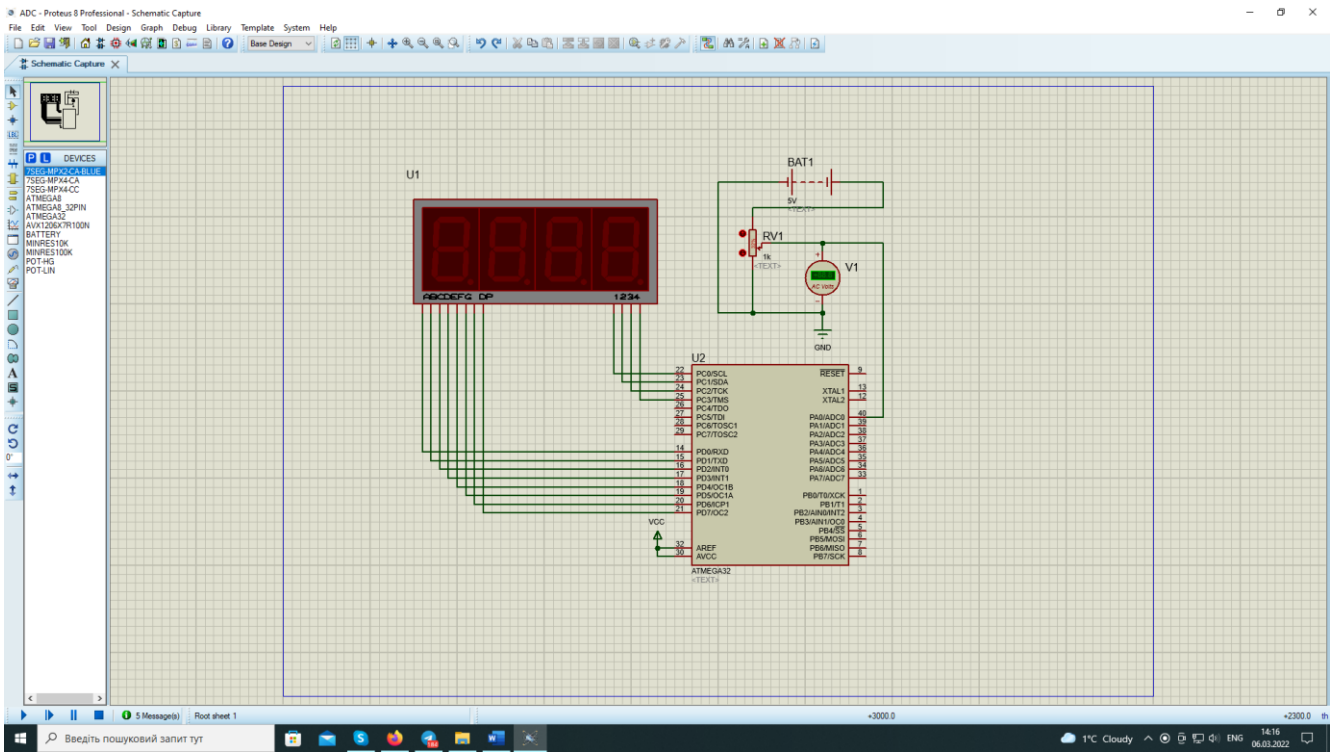


Рисунок 14.17 – Схема моделі цифрового вольтметра

На рисунку 14.18 наведено зовнішній вигляд 7-сегментного індикатора, який входить до складу чотирьохпозиційного цифрового дисплею. Для отримання на індикаторі чисел потрібно керувати сегментами індикатора A, B, C, D, E, F, G та точкою Dp згідно до таблиць 14.11, 14.12.

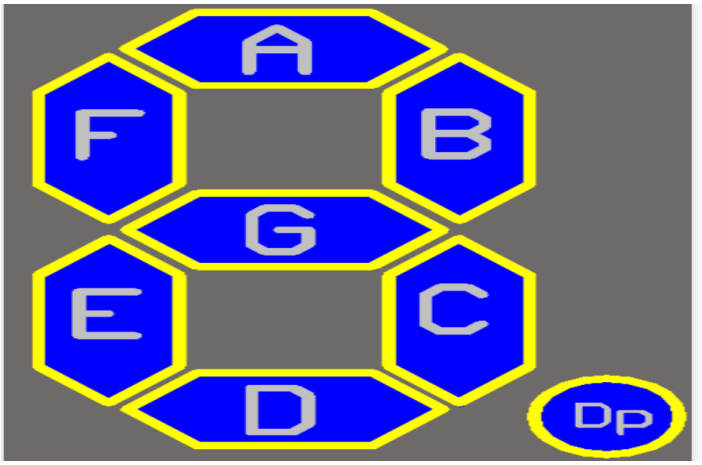


Рисунок 14.18 – 7-сегментний дисплей

Таблиця 14.11 – Зв’язок між цифрами на дисплеї та значенням керуючих сигналів без використання точки

Цифра на дисплеї	Dp	G	F	E	D	C	B	A	Значення керуючих сигналів
0	0	0	1	1	1	1	1	1	0x3F
1	0	0	0	0	0	1	1	0	0x06
2	0	1	0	1	1	0	1	1	0x5B
3	0	1	0	0	1	1	1	1	0x4F
4	0	1	1	0	0	1	1	0	0x66
5	0	1	1	0	1	1	0	1	0x6D
6	0	1	1	1	1	1	0	1	0x7D
7	0	0	0	0	0	1	1	1	0x07
8	0	1	1	1	1	1	1	1	0x7F
9	0	1	1	0	0	1	1	1	0x6F

Таблиця 14.12 – Зв’язок між цифрами з точкою на дисплеї та значенням керуючих сигналів з використанням точки

Цифра на дисплеї	Dp	G	F	E	D	C	B	A	Значення керуючих сигналів
0	1	0	1	1	1	1	1	1	0xBF
1	1	0	0	0	0	1	1	0	0x86
2	1	1	0	1	1	0	1	1	0xDB
3	1	1	0	0	1	1	1	1	0xCF
4	1	1	1	0	0	1	1	0	0xE6
5	1	1	1	0	1	1	0	1	0xED
6	1	1	1	1	1	1	0	1	0xFD
7	1	0	0	0	0	1	1	1	0x87
8	1	1	1	1	1	1	1	1	0xFF
9	1	1	1	0	0	1	1	1	0xEF

Нижче на рисунках 14.19, 14.20 наведено декілька прикладів роботи моделі.

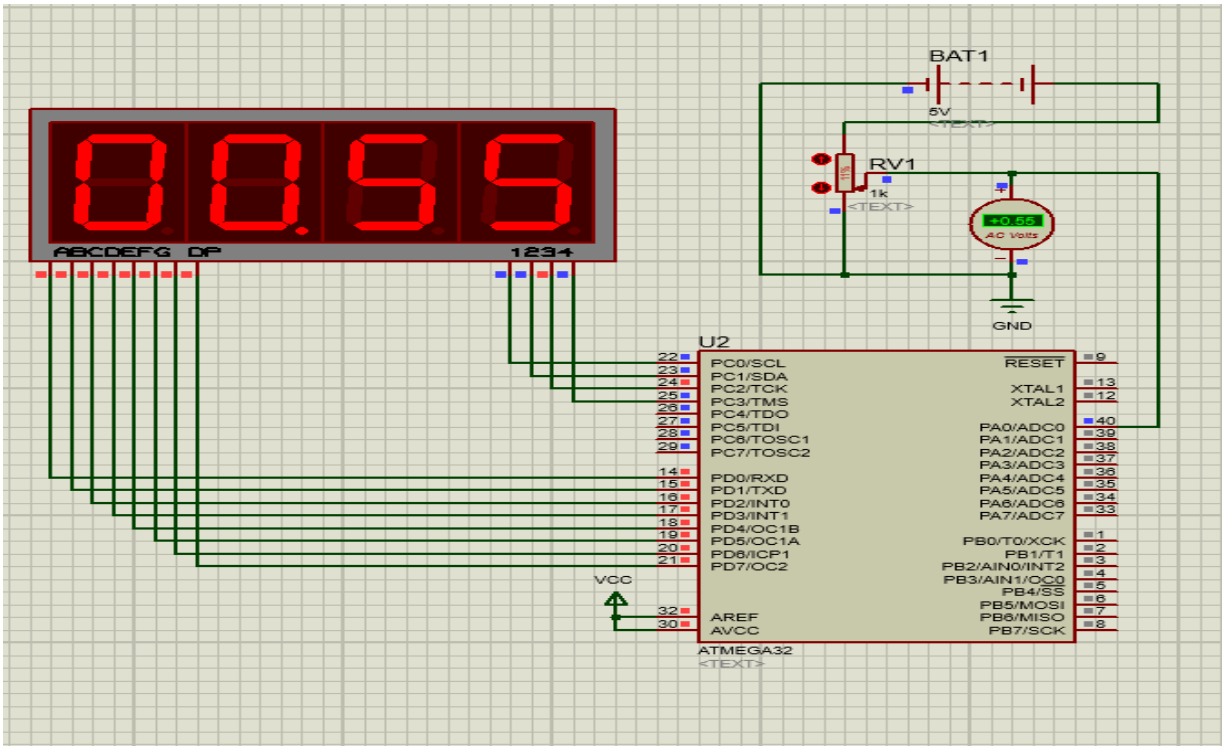


Рисунок 14.19 – Робота цифрового вольтметра при U_{BX} = 0,55В

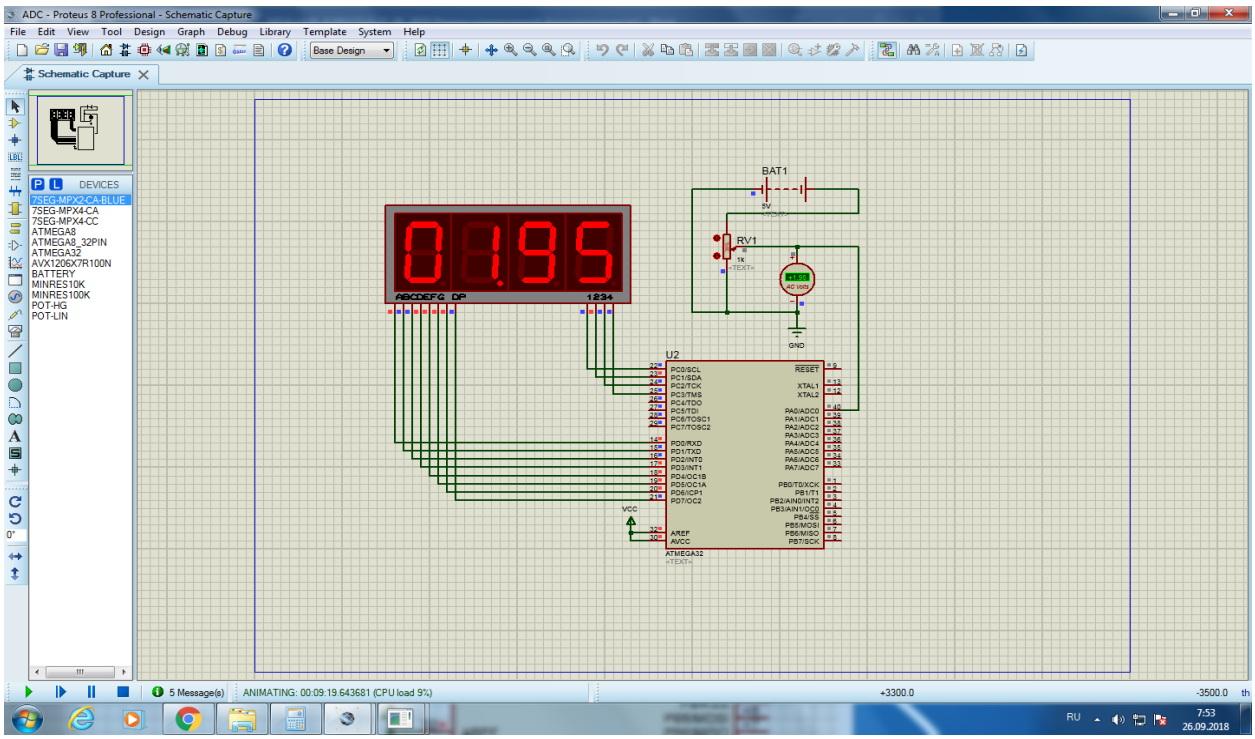


Рисунок 14.20 – Робота цифрового вольтметра при U_{BX} = 1,95В

14.4.4 Схема алгоритму роботи моделі

Схема алгоритму роботи моделі цифрового вольтметра подібна схемі алгоритму роботи моделі модуля АЦП, яку наведено вище на рисунках 13.14…13.17.

Ініціалізація модуля АЦП, таймера Т2 мікроконтролера та робота моделі у більшості виконується аналогічно описанному у підрозділі 13.2.2. Різниця полягає у виведенні результату моделювання на дисплей.

Нижче наведено пояснення перетворення значення ADC, яке отримується після завершення роботи модуля АЦП у десятковому коді, у значення вхідної напруги у вольтах для його виведення на дисплей.

Як відмічено у підрозділі 13.2.1 для каналів з однополярним (несиметричним) входом результат перетворення АЦП визначається виразом:

$$ADC = 1023 \bullet U_{IN}/U_{REF},$$

де U_{IN} – значення вхідної напруги, U_{REF} – величина опорної напруги, ADC – десятковий еквівалент двійкового коду на виході АЦП.

Коефіцієнт передачі АЦП

$$К_{ПЕР} = 1023/U_{REF}\ [МЗР/мВ].$$

При U_{REF} = 5В, $К_{ПЕР} = 1023/5000 = 0{,}2046$ [МЗР/мВ].

Значення вхідної напруги

$$U_{IN} = \frac{ADC}{K_{пер}} = \frac{ADC \cdot 5}{1023}.$$

Наприклад, якщо на вхід АЦП було подано напругу 0,55В, тоді згідно з результатом моделюванням АЦП (див. підрозділ 13.2.2) ADC = 113.

Перетворення значення ADC_value = ADC в змінну «display», виконується згідно з виразом:

$$display = (ADC\_value) \bullet (5/1023) \bullet 100.$$

Тоді при ADC = 113

$$display = 0113 \bullet (5/1023) \bullet 100 = 55.$$

Тобто, при U_{IN} = 0,55В значення «display» згідно з робочою програмою дорівнює 55. Для виведення цього значення використовується математична операція «остача», яка в мові програмування «С» обчислюється оператором «%».

Це значення програма обробляє наступним чином:

1) 55%10000/1000 = 55/1000 = 0,055. Цифра 0 виводиться на перший індикатор.
2) Далі відбуваються наступні дії:

   на другий індикатор завжди виводиться число з крапкою

   55%1000/100 = 55/100 = 0,55. Цифра 0 з десятковою крапкою виводиться на другий індикатор;
3) 55%100/10 = 55/10 = 5,5. Цифра 5 виводиться на третій індикатор;
4) 55%10/1 = 5/1 = 5. Цифра 5 виводиться на четвертий індикатор.

Наприклад, якщо на вхід АЦП було подано напругу 1,95В, тоді згідно з моделюванням АЦП (див. підрозділ 12.2.2) ADC = 399.

Значення «display = 0399 • (5/1023) • 100 = 195.

Це значення програма обробляє наступним чином:

195%10000/1000 = 195/1000 = 0,195. Цифра 0 виводиться на перший індикатор.

195%1000/100 = 195/100 = 1,95. Цифра 1 з десятковою крапкою виводиться на другий індикатор;

195%100/10 = 95/10 = 9,5. Цифра 9 виводиться на третій індикатор;

195%10/1 = 5/1 = 5. Цифра 5 виводиться на четвертий індикатор.

14.4.5 Робоча програма

```
// Підключення файлів бібліотек
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

// Масив значень для відображення цифр на семисегментних індикаторах
char SEG[] = {0x3F, 0x06, 0x5B, 0x4F, 0x66, 0x6D, 0x7D, 0x07, 0x7F, 0x6F};
```

```
char SEG_with_dot[] = {0xBF, 0x86, 0xDB, 0xCF, 0xE6, 0xED, 0xFD, 0x87, 0xFF,
0xEF};

// Ініціалізації глобальних змінних
volatile unsigned char current_indicator = 0;
volatile unsigned int display = 0;
volatile unsigned int ADC_value;

// Блок 1 – Ініціалізація програми
int main (void)
{
// Блок 1.1 – Налаштування та обнулення портів C та D керування
// семисегментними індикаторами
   DDRC = 0xFF;
   PORTC = 0x00;
   DDRD = 0xFF;
   PORTD = 0x00;

   // Блок 1.2 – Налаштування таймера T2
   TIMSK |= (1 << TOIE2); // Дозвіл переривання від таймера T2
   TCCR2 |= (1 << CS21);  // Переддільник на 8

   // Блок 1.3 – Налаштування АЦП
   ADCSRA |= (1 << ADEN) // Дозвіл АЦП
   |(1 << ADSC) // Запуск перетворення
   |(1 << ADATE) // Безперервний режим роботи АЦП
   |(1 << ADPS2)|(1 << ADPS1) // Переддільник на 64
   |(1 << ADIE); // Дозвіл переривань від АЦП

   ADMUX &= (~(1 << REFS1))&(~(1 << REFS0)); // Зовнішнє ДОН

   // Блок 1.4 – Глобальний дозвіл переривань
   sei();

   // Блок 1.5 – Основний цикл
   while(1)
   {
      _delay_ms(50); // Затримка 50 мс
```

```
   }
}
// Блок 2 – Умова переривання від АЦП
ISR (ADC_vect)
{// Блок 3 – Обробник переривань від АЦП
   ADC_value = ADC; // Блок 3.1 – Присвоювання глобальній змінній поточного
значення АЦП
}
// Блок 4 – Умова переривання від таймера T2
ISR (TIMER2_OVF_vect)
{   // Блок 5 – Обробник переривань від таймера T2
   PORTD = 0xFF; // Блок 5.1 – Вимикання всіх сегментів
   PORTC = (1 << current_indicator); // Блок 5.2 – Обирання поточного індикатора
                                          //починаючи зліва направо)
   display = (ADC_value)*(5/1023)*100; // Обрахування значення напруги, яка
                                            //виводиться на дисплей у вольтах
   //Блок 5.3
   switch (current_indicator)
   {
      case 0:
      PORTD = ~(SEG[display % 10000 / 1000]); // Вмикання цифри десятків
      break;
      case 1:
      PORTD = ~(SEG_with_dot[display % 1000 / 100]); // Вмикання цифри
                                            //одиниць з десятковою крапкою
      break;
      case 2:
      PORTD = ~(SEG[display % 100 / 10]); // Вмикання цифри десятих
      break;
      case 3:
      PORTD = ~(SEG[display % 10 / 1]); // Вмикання цифри сотих
      break;
   }
   if ((current_indicator++) > 2) // Блоки 5.4; 5.5 – Перехід на наступний
// індикатор, якщо current_indicator не більше двох
   current_indicator = 0; // Блок 5.6 – Обнулення current_indicator, якщо він
                                          //більше двох
}
```

14.5 Зміст звіту

Звіт по роботі повинен містити:

- схему моделі;
- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

Контрольні запитання та завдання

1) Назвіть розрядність модуля АЦП AVR-мікроконтролерів?.
2) Назвіть основні параметри АЦП.
3) Назвіть основні джерела опорної напруги для АЦП.
4) Назвіть режими роботи АЦП.
5) У якому регістрі зберігається результат перетворення?
6) Поясніть як саме відбувається перетворення вхідного сигналу у двійковий код?
7) Яку роль у функціональній схемі модуля АЦП виконує ПВЗ?
8) Який принцип перетворення використовується в АЦП? Поясніть особливості цього принципу.
9) Який регістр відповідає за налаштування мультиплексора АЦП?
10) Наведіть та поясніть формати регістрів стану і керування АЦП.
11) Який розряд регістра ADCSRA відповідає за дозвіл роботи АЦП?
12) Як обрати джерело опорної напруги?
13) Як вибрати режим роботи АЦП?
14) За якими перериваннями може відбуватися запуск АЦП?
15) Яку функцію виконує попередній дільник?
16) Як формується тактовий сигнал АЦП?
17) Який регістр використовується для налаштування вхідного мультиплексора?
18) Поясніть часові діаграми роботи АЦП при першому одиночному перетворенні в режимі одиночного перетворення.
19) Поясніть часові діаграми роботи АЦП в режимі наступного одиночного перетворення.
20) Поясніть часові діаграми роботи АЦП в режимі запуску за перериванням.
21) Поясніть часові діаграми роботи АЦП в режимі автоматичного перезапуску для однополярного перетворення.

22) Скільки тактів синхронізації АЦП витрачається на перетворення в режимах: першого одиночного перетворення та наступного одиночного перетворення?
23) Скільки однополярних входів має вхідний мультиплексор модуля АЦП?
24) Назвіть способи вирівнювання результату АЦП.
25) Яке вирівнювання слід використовувати якщо досить точності 8-розрядного значення?
26) Які особливості перемикання каналів при одиночному перетворенні?
27) Які особливості перемикання каналів при перетворенні у режимі автоматичного перезапуску?
28) Що може виступати у ролі джерела опорної напруги?
29) Яким буде результат перетворення для каналів з однополярним входом?
30) Яким буде результат перетворення для каналів з диференціальним входом?
31) Якою формулою визначається коефіцієнт передачі АЦП?
32) Поясніть функцію перетворення АЦП при зміні однополярного сигналу.
33) Поясніть зв'язок між вхідним сигналом та вихідними кодами для однополярного режиму.
34) Назвіть методи підвищення точності перетворення АЦП.
35) Чому дорівнює похибка зсуву та як її можна програмно врахувати?
36) Опишіть особливості програмування мовою С виведення на дисплей результату роботи модуля АЦП та цифрового вольтметра.
37) Як розраховується відносна похибка АЦП від квантування за рівнем?
38) Опишіть схему моделювання модуля АЦП в пакеті PROTEUS 8.6.
39) В який режим програмується модуль АЦП при його моделюванні?
40) Чому дорівнює коефіцієнт передачі модуля АЦП при його моделюванні?
41) В якому вигляді виводиться на індикацію результат моделювання модуля?
42) За допомогою якої математичної операції визначається відповідна цифра при її виведенні на індикатор? Відповідь пояснити.
43) Який таймер/лічильник мікроконтролера використовується при виведенні на семисегментні індикатори?
44) Який сигнал використовується в якості тактового для таймера 2? Відповідь пояснити.
45) Чим відрізняється виведення результату моделювання цифрового вольтметра від моделювання модуля АЦП?

15 ЛАБОРАТОРНА РОБОТА №12. МОДЕЛЮВАННЯ МІКРОКОНТРОЛЕРНОЇ МЕРЕЖІ RS-485-232

Тема: Моделювання мікроконтролерної мережі RS-485-232 з використанням мікроконтролера сім'ї AVR.

Мета: Користуючись пакетом Proteus 8.6 дослідити роботу мікроконтролерної мережі RS-485-232.

15.1 Порядок виконання роботи

1) Створити модель мікроконтролерної мережі в пакеті Proteus 8.6.
2) Розробити схему алгоритму роботи мережі та робочу програму.
3) Створити hex-файл та підключити його до мікроконтролера.
4) Запустити модель та виконати її дослідження згідно методичних вказівок.
5) Зробити відповідні висновки.

15.2 Опис інтерфейсу RS-485

15.2.1 Загальна характеристика

RS-485/ EIA-485 (RS485 – англ. Recommended Standard 485, EIA-485 – англ. Electronic Industries Alliance-485) – стандарт передачі даних по двопровідному напівдуплексному багатоточковому послідовному каналу зв'язку.

Стандарт RS-485 розроблений спільно двома асоціаціями: Асоціацією електронної промисловості (EIA – Electronics Industries Association) та Асоціацією промисловості засобів зв'язку (TIA – Telecommunications Industry Association). Раніше EIA маркірувала всі свої стандарти префіксом «RS» (англ. Recommended Standard – рекомендований стандарт). Багато інженерів продовжують використовувати це позначення, проте EIA/TIA офіційно замінив «RS» на «EIA/TIA» з метою полегшити ідентифікацію походження своїх стандартів. На сьогоднішній день, різні розширення стандарту RS-485 охоплюють широке розмаїття програм.

Цей стандарт став основою для створення цілого сімейства обчислювальних мереж, які широко використовуються в промисловій автоматизації.

В стандарті RS-485 для передачі і прийому даних часто використовується єдина вита пара проводів. Передача даних здійснюється за допомогою диференціальних сигналів. Різниця напруг між провідниками однієї полярності означає логічну одиницю, різниця іншої полярності – нуль.

Стандарт RS-485 це назва популярного інтерфейсу, що використовується в промислових АСК ТП для з'єднання контролерів та іншого обладнання. Головна відмінність RS-485 від також широко розповсюдженого RS-232 – можливість об'єднання декількох пристроїв в мережу.

Стандартом RS-485 описується використаний в інтерфейсі спосіб передачі диференціальних сигналів у мікроконтролерній мережі і визначає характеристики лінійних формувачів і приймачів. Таким чином, інтерфейс RS-485 передає і приймає лише послідовності логічних рівнів, не піклуючись про логічні подробиці передачі даних у вигляді поділу на байти, видачі старт-стопних сигналів, корекції помилок, керування напрямком і черговістю передачі даних. Все це повинно бути реалізовано окремо, але саме тому на RS-485 можна виконати безліч інтерфейсних протоколів – від найпростіших типу «точка – точка» до складних інтелектуальних інтерфейсів розподілених систем.

Для передачі сигналів формувач передавача генерує дві комплементарні напруги (логічно протилежних: високий рівень на виході A, низький рівень на виході B, або високий рівень на B, низький рівень на A). Всі інші передавачі в цей час, щоб уникнути конфлікту і спотворення сигналу перебувають у третьому (високоімпедансному) стані [12]. На рисунку 15.1 показано, як стандарт EIA-RS-485 визначає напруги Uoa, Uob, і Uo.

Коли рівень напруги U_{oa} – низький, тоді U_{ob} – високий, а коли U_{oa} – високий, тоді U_{ob} – низький.

Незважаючи на те, що на схемах включення RS-485 вказується двопровідна вита пара, RS-485 не є струмовою петлею з взаємно-протилежними струмами.

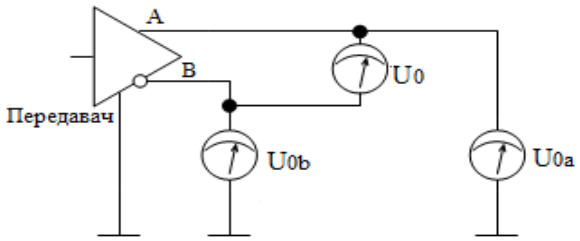
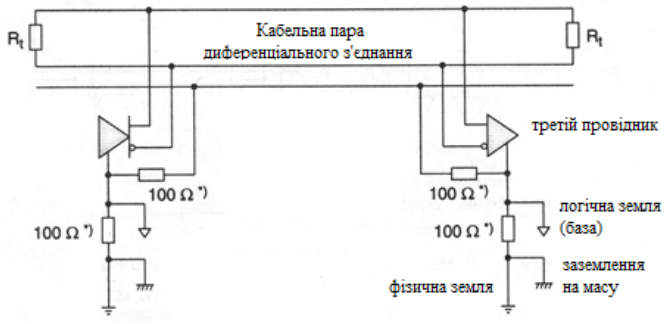


Рисунок 15.1 – Вимірювання напруг U_{0a}, U_{0b} і U_0

Існує інтерфейс – «current loop», в якому логічна 1 представляється імпульсом струму в петлі, а логічний 0 – відсутністю струму. Фактично це дві струмові петлі, два електричні контури. Формувачі і приймачі мережі RS-485 повинні використовувати спільну землю, тому термін "двопровідна лінія" непридатний до RS-485.

У загальному випадку, необхідно використовувати трьохпровідну лінію (рисунок 15.2). Третім проводом може бути високоякісне реальне заземлення з низьким власним опором. Іноді рекомендують логічну «землю» заземлювати на реальну землю (на шасі) через резистор в 100 Ом для обмеження струму.



Примітка: *) резистори, включені послідовно з лініями повернення сигналу, обмежують циркулюючі струми.

Рисунок 15.2 – Схема підключення інтерфейсу RS-485/ EIA-485 до трьохпровідної лінії

Правильна робота передавача і приймача вимагає наявності шляху повернення сигналу між підключеними до ланцюга землями обладнання в кожній точці підключення.

Приймачі мережі розроблені так, щоб реагувати на різницю напруг U_o = A-B, більшу ніж 200 мВ (напруги вимірюються щодо спільного потенціалу (землі)). При досить довгих лініях зв'язку та наявності струму через третій провід (при різних значеннях падіння напруги в локальних заземлюючих ланцюгах) можлива поява значного постійного потенціалу, який додається до напруги в сигнальних проводах A і B. Саме тому приймачі RS-485 працюють у діапазоні напруг: -7 … +12 В, а не 0 … +5 В (діапазон вихідної напруги передавача). Таким чином допускається наявність неузгодженості локальних потенціалів землі до 7 В.

Кількість вузлів

Стандартом EIA RS-485 визначається приймач з питомим навантаженням (Unit-Load) на передавач в один UL. При цьому передавач повинен формувати 32 UL, звідки випливає, що максимальна кількість вузлів у мережі не може перевищувати 32/1 = 32. Але вже існують приймачі з питомим навантаженням в 1/4 UL і навіть 1/8 UL, що робить можливим створення мережі з 256 вузлів. При використанні повторювачів (ретрансляторів) можна з'єднувати разом складні мережі, об'єднуючи фактично необмежену кількість вузлів, але одночасно зі зростанням діаметра топології мережі збільшуються і затримки, а швидкість передачі даних може стати неприйнятно низькою.

Швидкість та дальність

Мережа RS-485 забезпечує передачу даних зі швидкістю до 10 Мбіт/с. Максимальна дальність залежить від швидкості. Наприклад, при швидкості 10 Мбіт/с максимальна довжина лінії – 120 м, а при швидкості 100 кбіт/с – 1200 м.

Протоколи та роз’єми

Стандарт не нормує формат інформаційних кадрів і протокол обміну. Найбільш часто для передачі байтів даних використовуються протоколи, що і в

інтерфейсі RS-232: стартовий біт, біти даних, біт паритету (якщо потрібно), стоповий біт.

Протоколи обміну в більшості систем працюють за принципом "ведучий" – "ведений". Один пристрій на магістралі є ведучим (master) і ініціює обмін посилкою запитів веденим пристроям (slave), які розрізняються логічними адресами. Одним з популярних протоколів є протокол Modbus RTU.

Тип з'єднувачів і розпаювання також не обумовлюються стандартом. Зустрічаються з'єднувачі DB9, клемні з'єднувачі й т. ін.

Підключення інтерфейсів до мережі

Підключення інтерфейсів RS-485 до локальної мережі відбувається за наступною схемою (рисунок 15.3):

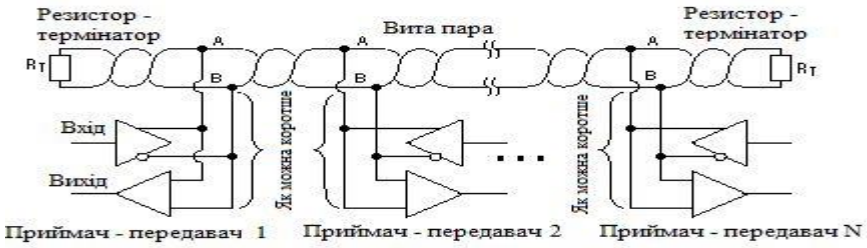


Рисунок 15.3 – Схема підключення інтерфейсів RS-485 до локальної мережі

При підключенні слід правильно приєднати сигнальні ланцюги, що зазвичай звуться A і B.

Загальні рекомендації щодо використання інтерфейсу RS-485

Існують наступні загальні рекомендації щодо використання RS-485:

1) Кращим середовищем передачі сигналу є кабель на основі крученої пари.

2) Кінці кабелю мають бути заглушені резисторами-термінаторами (зазвичай 120 Ом).

3) Мережа повинна бути прокладена за топологією шини.

4) Пристрої слід підключати до кабелю проводами мінімальної довжини.

5) Вита пара є оптимальним рішенням для прокладки мережі, оскільки має найменше паразитне випромінювання сигналу і добре захищена від наведень. В

умовах підвищених зовнішніх завад застосовують кабелі з екранованою крученою парою, при цьому екран кабелю з'єднують із захисною "землею" пристрою.

Узгодження «відкритого» кінця кабелю

Для узгодження "відкритого" кінця кабелю з рештою лінії використовують резистори-термінатори, які забезпечують усування відбиття сигналу.

Номінальний опір резисторів відповідає хвильовому опору кабелю, і для кабелів на основі витої пари зазвичай становить: 100 – 120 Ом. Наприклад, широко поширений кабель UTP-5, використовуваний для прокладки Ethernet, має імпеданс 100 Ом. Для іншого типу кабелю може знадобитися інший номінал.

Резистори можуть бути запаяні на контакти кабельних роз'ємів наприкінці лінії. Іноді резистори бувають вмонтовані в самому пристрої і для підключення резистора потрібно встановити перемичку. У цьому випадку при від'єднанні пристрою лінія розузгоджується, і для нормальної роботи решти системи потрібне підключення узгоджуючої заглушки.

Рівні сигналів в мережі

Інтерфейс RS-485 використовує балансну (диференційну) схему передачі сигналу. Це означає, що рівні напруг на сигнальних ланцюгах A і B змінюються в протифазі, як показано на наведеному нижче рисунку 15.4.

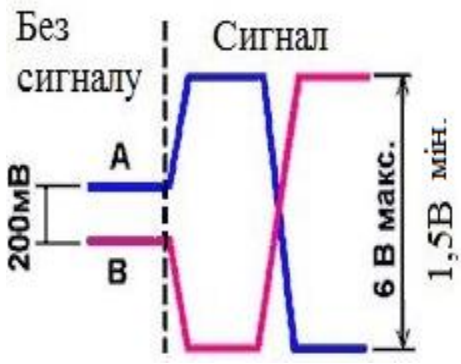


Рисунок 15.4 – Рівні сигналів в мережі

Передавач повинен забезпечувати мінімальний рівень сигналу 1,5 В при максимальному навантаженні – 32 стандартних входи і 2 резистора-термінатора, і не більше 6 В на холостому ході. Рівні напруг вимірюють диференціально, один сигнальний провід відносно іншого.

Пороговий діапазон прийнятого сигналу RS-485: ±200 мВ.

Зсув на сигнальних ланцюгах

За відсутності сигналу в мережі на сигнальних ланцюгах є невеликий зсув. Цей зсув призначений для захисту приймачів від помилкових спрацьовувань.

Рекомендується створювати зсув трохи більше 200 мВ (зона недостовірності вхідного сигналу відповідно до стандарту). При цьому ланцюг А "підтягують" до додатного полюса джерела, а ланцюг В – до "спільного" (рисунок 15.5).

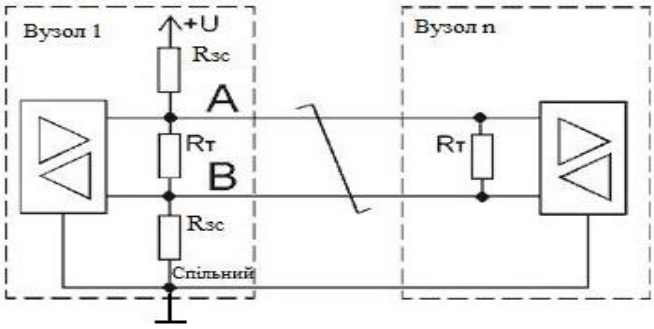


Рисунок 15.5 – Схема реалізації ланцюга зсуву на сигнальних ланцюгах

Номінали резисторів розраховують, виходячи з необхідного зсуву і напруги джерела живлення.

Величини опорів для резисторів захисного зсуву Rзс розраховуються за дільником (рисунок 15.6).

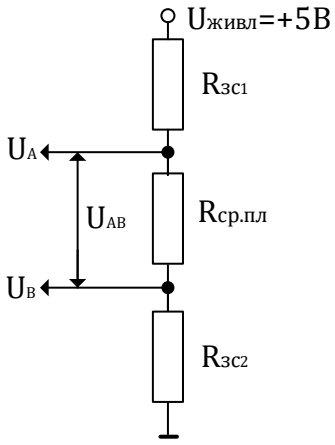


Рисунок 15.6 – Еквівалентна схема захисного зсуву

Необхідно забезпечити U_{AB} > 200мВ. Напруга живлення: +5В.

Опір середнього плеча Rср.пл дорівнює еквівалентному значенню паралельно включених двох резисторів-термінаторів Rт, які дорівнюють 120Ом, і еквівалентного опору, наприклад, п'яти приймачів RS-485, кожен з яких має Rвх = 12 кОм. Тоді

$$R_{ср.пл} = \frac{R_{вх}}{5} \left\| \frac{R_{т}}{2} = \frac{12000}{5} \right\| \frac{120}{2} = 2400 \left\| 60 = \frac{2400 \cdot 60}{2400 + 60} = 59 Ом. \right.$$

Приймемо Rзс1 = Rзс2 = 560 Ом, тоді напруга

$$U_A = \frac{U_{пит}(R_{ср.пл} + R_{зс2})}{R_{зс1} + R_{ср.пл} + R_{зс2}} = \frac{5 \cdot (59 + 560)}{560 + 59 + 560} = \frac{5 \cdot 619}{1179} = 2{,}63 B.$$

Напруга $U_в = \frac{U_{пит} \cdot R_{зс2}}{R_{зс1} + R_{ср.пл} + R_{зс2}} = \frac{5 \cdot 560}{560 + 59 + 560} = \frac{5 \cdot 560}{1179} = 2{,}38 B.$

Тоді різницю напруг $U_{AB} = U_A - U_в$=2,63В-2,38В=0,25В=250мВ, тобто. це відповідає умові U_{AB}>200мВ, отже інтерфейс RS-485 подає на інтерфейс RS-232 логічну одиницю.

Якщо на лінії висить багато приймачів, то номінал Rзс повинен бути менше. У довгих лініях передачі необхідно також враховувати опір крученої пари, який може "з'їдати" частину зміщуючої різниці потенціалів для віддалених від місця підтяжки пристроїв. Для довгої лінії краще ставити два комплекти підтягуючих резисторів в обидва віддалені кінці поруч із термінаторами.

При наявності зміщення потенціал ланцюга А на холостому ході додатний відносно ланцюга В. Це може служити орієнтиром при підключенні нового пристрою до кабелю за немаркованими проводами та може говорити приймачу інтерфейсу RS-232 про наявність одиничного логічного рівня, після якого в мережі при початку передачі приймач отримує нульовий стартовий біт (див нижче опис інтерфейсу RS-232).

15.2.2 Реалізація інтерфейсу RS-485

Нижче розглянуто практичну реалізацію інтерфейсу RS-485 на основі недорогих, економічних мікросхем фірми MAXIM: MAX481, MAX483, MAX485, MAX487…MAX491 і MAX1487.

Ці мікросхеми є малопотужним приймачами-передавачами (трансиверами) для взаємодії за протоколами RS-485 і RS-422 (таблиця 15.1).

Таблиця 15.1 – Параметри RS-485 мікросхем MAXIM

Найменування	Дуплекс/ напівдуплекс	Швикість передачі (Мбіт/с)	Обмеження наростання вихідної напруги	Режим мікро–спожи–вання	Дозвіл приймача/передавача	Струм спокою (мкА)	Число пристроїв на шині	Число виводів
MAX481	Напів–дуплекс	2.5	Ні	Так	Так	300	32	8
MAX483	Напів–дуплекс	0.25	Так	Так	Так	120	32	8
MAX485	Напів–дуплекс	2.5	Ні	Ні	Так	300	32	8
MAX487	Напів–дуплекс	0.25	Так	Так	Так	120	128	8
MAX488	Дуплекс	0.25	Так	Ні	Ні	120	32	8
MAX489	Дуплекс	0.25	Так	Ні	Так	120	32	14
MAX490	Дуплекс	2.5	Ні	Ні	Ні	300	32	8
MAX491	Дуплекс	2.5	Ні	Ні	Так	300	32	14
MAX1487	Напів–дуплекс	2.5	Ні	Ні	Так	230	128	8

Кожна мікросхема включає в себе один вихідний формувач і один приймач. Конструктивно вона оформлюється в корпусах типу DIP (англ. Dual-in-line package), SO, μMAX. Мікросхеми: MAX483, MAX487, MAX488 та MAX489 містять вихідні формувачі з обмеженням швидкості наростання вихідної напруги. Це зменшує електромагнітне випромінювання і відбиття сигналу, викликані неправильним термінуванням крученої пари, тим самим дозволяючи передачу даних без помилок на швидкостях до 250 Кбіт/с. Вихідні формувачі мікросхем MAX481, MAX485, MAX490, MAX491 і MAX1487 не обмежують швидкість наростання вихідної напруги і дозволяють передавати дані на швидкості до 2.5 Мбіт/с.

Розглянуті приймачі-передавачі споживають від 120 мкА до 300 мкА при відсутності навантаження або наявності повного навантаження, але з відключеними вихідними формувачами.

Додатково, MAX481, MAX483 і MAX487 мають режим зниженого споживання енергії, в якому вони споживають лише 0,1 мкА.

Всі мікросхеми живляться від однополярної напруги: +5 В. Приймачі розглянутих мікросхем працюють в діапазоні: – 7 ... +12 В.

Мікросхеми MAX487 і MAX1487 мають вчетверо більше навантаження на лінію, що дозволяє підключити до 128 приймачів на спільну шину.

Мікросхеми MAX488…MAX491 забезпечують повнодуплексну передачу даних, а мікросхеми MAX481, MAX483, MAX485, MAX487 та MAX1487 розроблено для напівдуплексного режиму (рисунок 15.7).

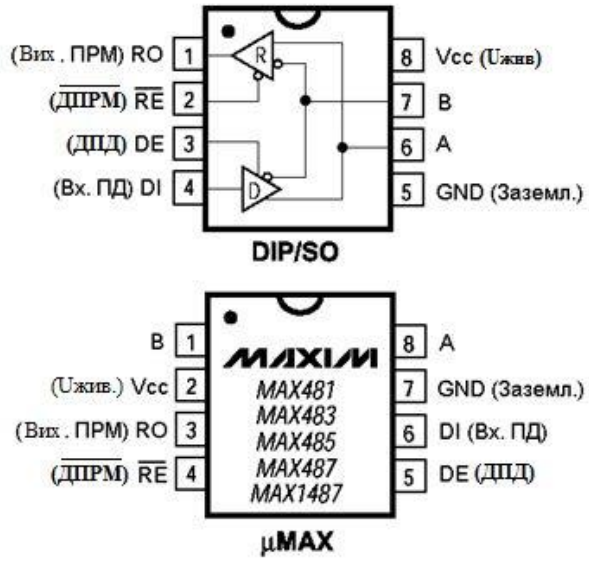


Рисунок 15.7 – Призначення виводів напівдуплексних мікросхем RS-485 фірми MAXIM

Розглянуті мікросхеми можуть працювати не тільки в мережі (рисунок 15.3), а й у простому з'єднанні типу «точка-точка» (рисунок 15.8).

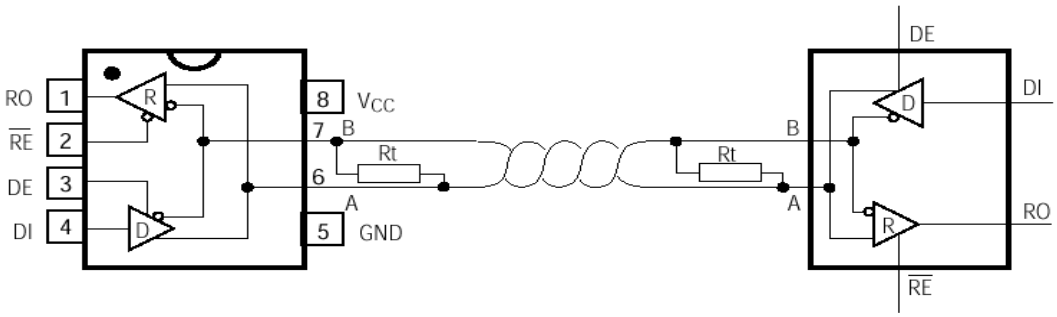


Рисунок 15.8 – Приклад простого з’єднання мікроссхем типу точка-точка (напівдуплекс)

Нижче наведено приклад типового використання мікросхем RS-485 в напівдуплексній (рисунок 15.9) та в дуплексній мережах (рисунок 15.10).

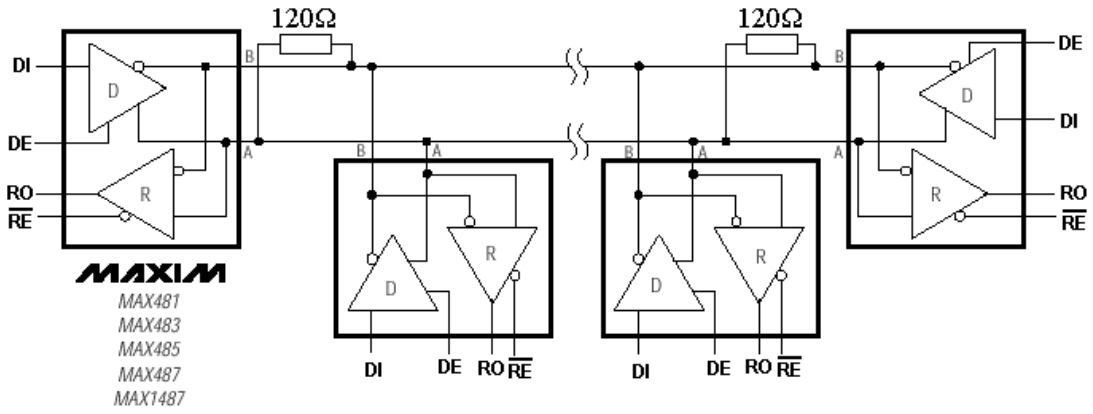
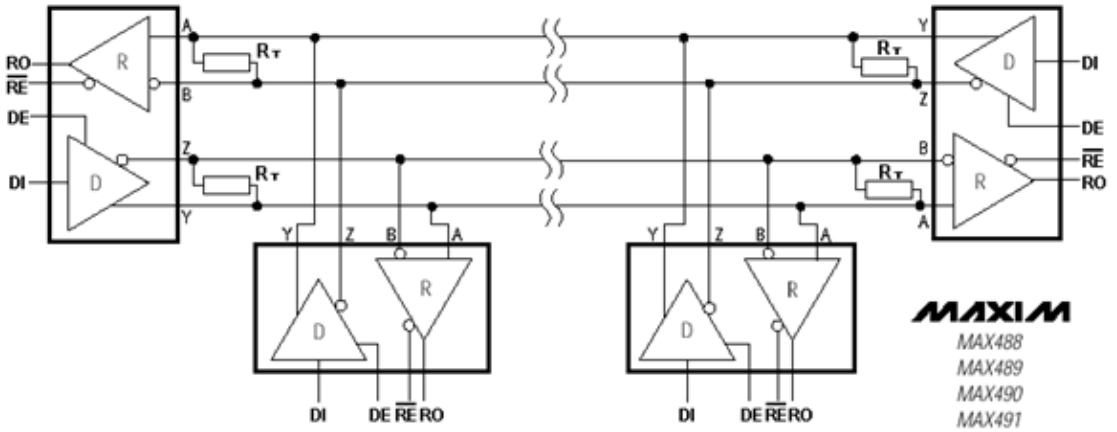


Рисунок 15.9 – Типове використання мікросхем RS-485 в напівдуплексній мережі



Примітка: $\overline{RE}$ і DE тільки для MAX489/MAX491. R_T – термінатор: 120 Ω.

Рисунок 15.10 – Типове використання мікросхем RS-485 в дуплексній мережі (мікросхеми MAX488…MAX491)

Нижче наведено приклад використання мікросхеми MAX485 спільно з мікроконтролером AT89C51 (рисунок 15.11).

На рисунку 15.12 представлено функціональну схему мікроконтролерної мережі RS-485 в мікропроцесорній системі з розподіленим керуванням.

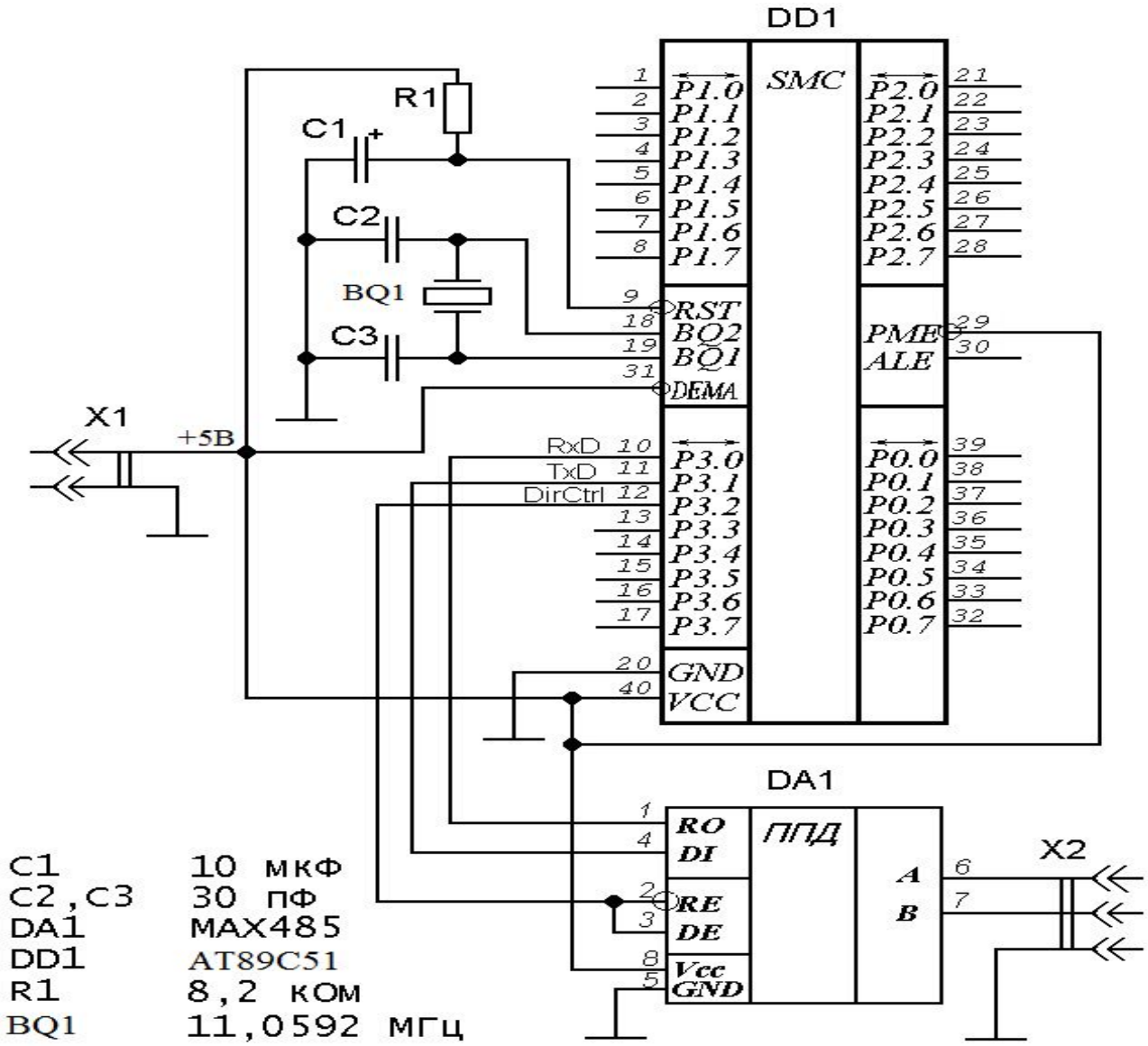


Рисунок 15.11 – Принципова схема підключення мікроконтролера AT89C51 до мікросхеми RS-485

15.3 Опис інтерфейсу RS-232

15.3.1 Загальна характеристика

Обмін інформацією між мікропроцесором (МП) або мікроконтролером (МК) і, наприклад, модемом може здійснюватися через інтерфейс RS-232.

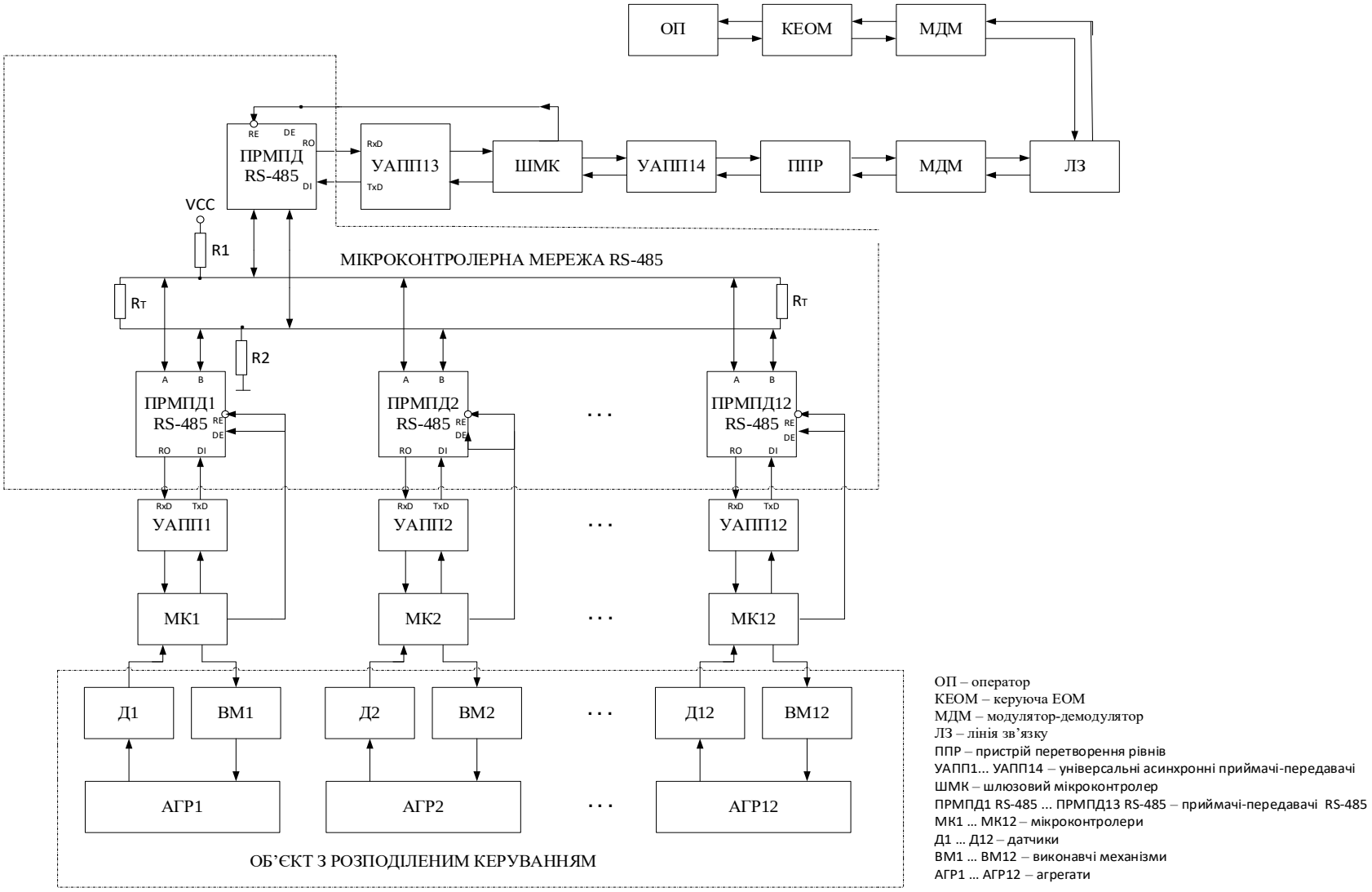
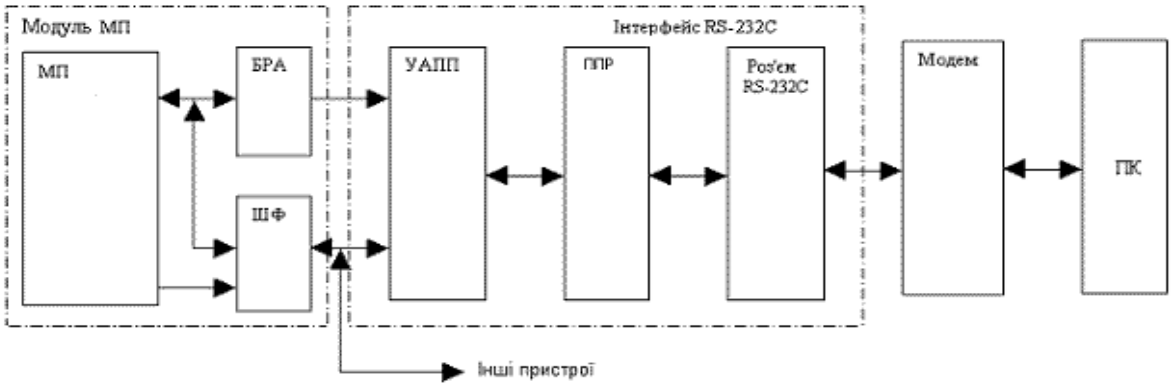


Рисунок 15.12 – Функціональна схема промислової мережі RS-485 для мікропроцесорної системи з розподіленим керуванням

Нижче наведено структурну схему сполучення МП/МК–ра з модемом через інтерфейс RS-232 (рисунок 15.13).



МП - мікропроцесор; БРА – буферний регістр адреси
ШФ – шинний формувач; УАПП – універсальний асинхронний послідовний програмовуваний приймач – передавач;
ППР – пристрій перетворення рівнів; ПК - персональний комп'ютер

Рисунок 15.13 – Структурна схема сполучення МП/МК-ра з модемом через інтерфейс RS-232

Схема містить:

– УАПП – універсальний асинхронний послідовний програмовуваний приймач/передавач;
– ППР – пристрій перетворення рівнів;
– роз'єм RS-232.

Крім інтерфейсу RS–232 схема сполучення містить:

– БРА – буферний регістр адреси;
– ШФ – шинний формувач.

15.3.2 Універсальний асинхронний послідовний програмовуваний приймач/передавач (УАПП)

Універсальний асинхронний послідовний програмовуваний приймач/передавач (УАПП) перетворює дані з паралельного формату в послідовний при передачі (виведенні) з МП/МК-ра і з послідовного формату в паралельний при прийомі (введенні) у МП/МК.

В якості УАПП може використовуватися мікросхема TL16C450/550, або модуль УАПП (UART) у складі мікроконтролера [7].

Більшість мікроконтролерів сімейства AVR мають у своєму складі модуль універсального асинхронного приймача/передавача (УАПП), або універсального синхронно/асинхронного приймача/передавача (УСАПП). У деяких моделях міститься по декілька таких модулів [1; 10].

В іноземній літературі модуль УАПП позначається, як UART (Universal Asynchronous Receiver and Transmitter) – універсальний асинхронний приймач-передавач, а модуль УСАПП – як USART (Universal Synchronous and Asynchronous Receiver and Transmitter) – універсальний синхронний/асинхронний приймач-передавач.

Формат даних інтерфейсу, які передаються у канал зв'язку в послідовному форматі представлено на рисунку 15.14.

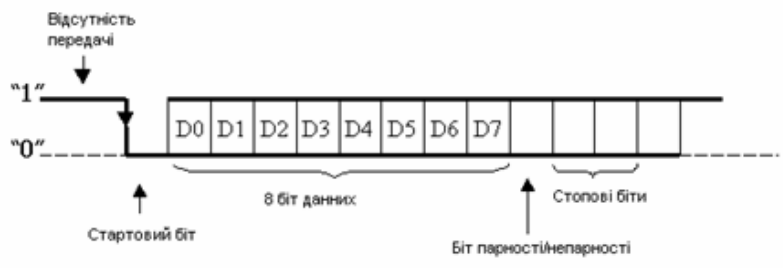


Рисунок 15.14 – Формат даних інтерфейсу RS-232

Безпосередньо дані (5, 6, 7 чи 8 біт) супроводжуються стартовим (нульовим) бітом. Потім передаються біти даних, біт парності/непарності (якщо такий контроль програмно передбачений), і стоповий одиничний сигнал, що може включати 1; 1,5 чи 2 стоп-біта. Одержавши стартовий біт, приймач вибирає з лінії біти даних через визначені інтервали часу. Дуже важливо, щоб тактові частоти приймача і передавача були стабільними та однаковими (допустима розбіжність – не більше 10 %).

Швидкість передачі за RS-232 може вибиратися з ряду: 110, 150, 300, 600, 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200 біт/с (бод).

Більш детально архітектуру універсального асинхронного послідовного програмовуваного приймача/передавача розглянуто у [1; 7].

Пристрій перетворення рівнів (ППР)

Всі сигнали R-232 передаються/приймаються спеціально обраними рівнями, що забезпечують високу завадостійкість зв'язку (рисунок 15.15). Слід зазначити, що дані передаються/приймаються в інверсному вигляді: логічній одиниці відповідає низький рівень, а логічному нулю – високий рівень.

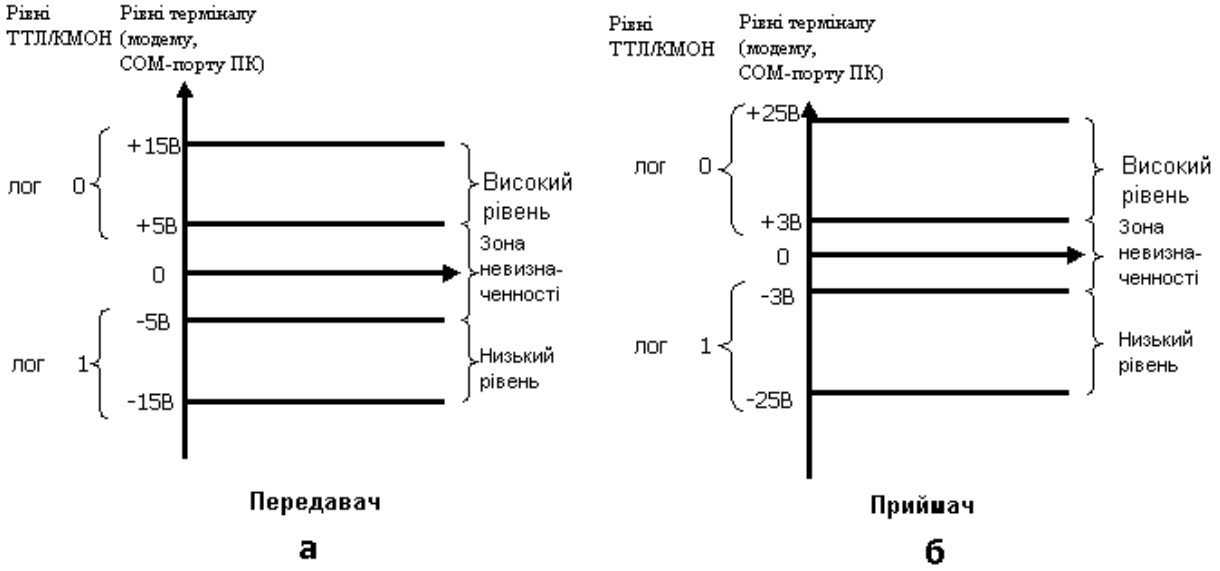


Рисунок 15.15 – Рівні сигналів RS-232 на передаючій та приймальній кінцях лінії зв’язку

Як видно з рисунку 15.15 під час передачі логічного нуля на виході інтерфейсу формується високий рівень напруги в діапазоні: +5В…+15В, під час передачі логічної одиниці – низький рівень напруги в діапазоні: –5В…–15В.

Під час прийому на вхід інтерфейсу надходить високий рівень напруги в діапазоні: +3В…+25В, що несе інформацію про логічний 0, чи низький рівень напруги в діапазоні: -3В…-25В, що відображає логічну одиницю.

Таким чином, для узгодження ТТЛШ/КМОН рівнів сигналів, що діють у мікропроцесорній системі, з рівнями сигналів послідовного інтерфейсу, які передаються у лінію зв'язку або приймаються з лінії зв'язку, використовують пристрої перетворення рівнів (ППР).

Різні варіанти схемної реалізації ППР розглянуто у [20], одним із яких є застосування мікросхеми фірми MAXIM: MAX232A. Дана мікросхема (рисунок 15.16) вимагає одне джерело живлення +5В та ряд додаткових елементів – конденсаторів: С1, С2, … , С5, що не є надмірною ціною за переваги її застосування.

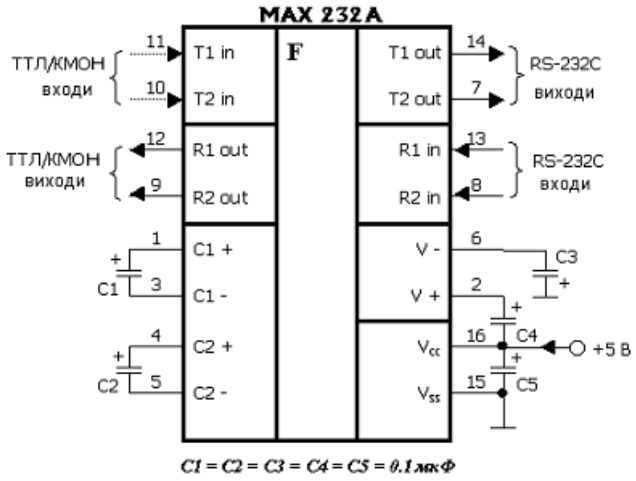


Рисунок 15.16 – Позначення і особливості підключення мікросхеми MAX232A

Роз’єм RS-232C

Для зв'язку інтерфейсу RS-232 із зовнішнім терміналом – модемом або ПК може використовуватися 9-контактний роз’єм (рисунок 15.17).

Роз’єм RS-232

Рисунок 15.17 – 9-контактний роз'єм RS-232C

Нижче наведено призначення його основних контактів:

- SG – сигнальне заземлення, нульовий провід;
- TxD – дані, що передаються мікроконтролером у послідовному коді (від'ємна логіка);
- RxD – дані, що приймаються мікроконтролером у послідовному коді (від'ємна логіка);
- DCD – виявлення несучої даних (детектування сигналу, що приймається мікроконтролером від модему);
- DTR – запит передавача терміналу (модему або ПК);
- DSR – готовність передавача терміналу (модему або ПК);
- RTS – запит приймача терміналу (модему або ПК);
- CTS – готовність приймача терміналу (модему або ПК);
- RI – індикатор виклику (свідчить про прийом модемом сигналу виклику від телефонної мережі).

15.4 Опис моделі мережі RS-485-232

15.4.1 Опис моделі мережі RS-485-232-1

В роботі розглянуті два приклади моделей мережі: RS-485-232-1 та RS-485-232-2.

Нижче розглядається модель RS-485-232-1, схему якої наведено на рисунку 15.18.

На схемі розміщено 3 мікроконтролера (лівий працює в режимі «Master», два інші – в режимі «Slave»). В якості мікроконтролерів використовуються мікросхеми ATmega32.

До ліній портів D мікроконтролерів: RxD, TxD та INT0 підключені приймачі-передавачі MAX485. Останні зв'язані з проводами A та B мережі. Ці провоа підключені до сигнальної «землі» двома резисторами-термінаторами, значенням 560Ом.

До ведучого мікроконтролера через його порт B підключено 8-ми клавішний перемикач для задання байта адреси для обраного веденого мікроконтролера.

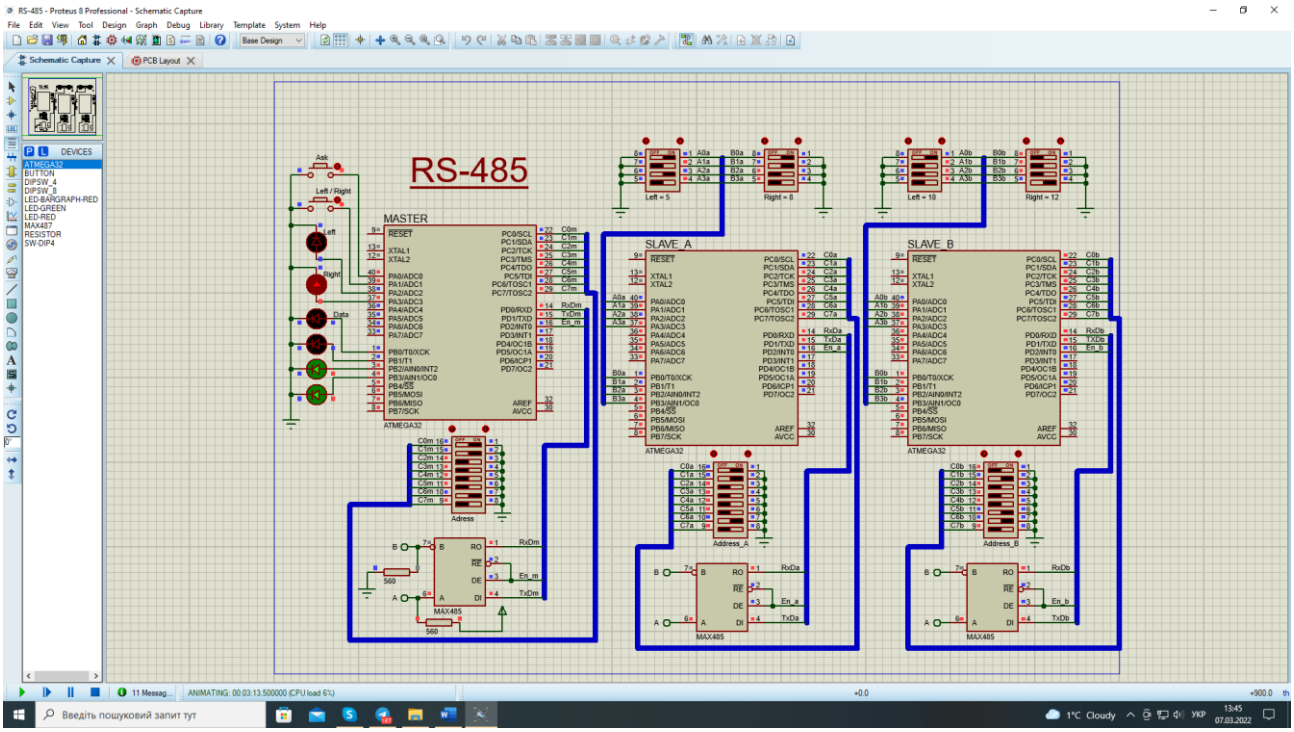


Рисунок 15.18 – Схема робочої моделі мережі RS-485-232-1

До двох ведених (підлеглих) мікроконтролерів через лінії портів C підключені вісьмипозиційні перемикачі, які задають адреси цих МК в мережі.

Кожен ведений мікроконтролер має два 4-позиційні перемикачі, що підключені до молодших тетрад портів A (Left) та B (Right) для задання інформації при її передачі мережею до ведучого мікроконтролера.

До ведучого МК через молодшу тетраду порту B підключено блок світлодіодів «Data» для відображення інформації, отриманої від ведених мікроконтролерів, та два світлодіоди: «Left» та «Right» для відображення лівого або правого каналу обраного веденого мікроконтролера, які підключено до ліній PA2 та PA3 портуA.

До ліній PA0 та PA1 порту A підключено дві кнопки керування: «Ask» та «Left / Right».

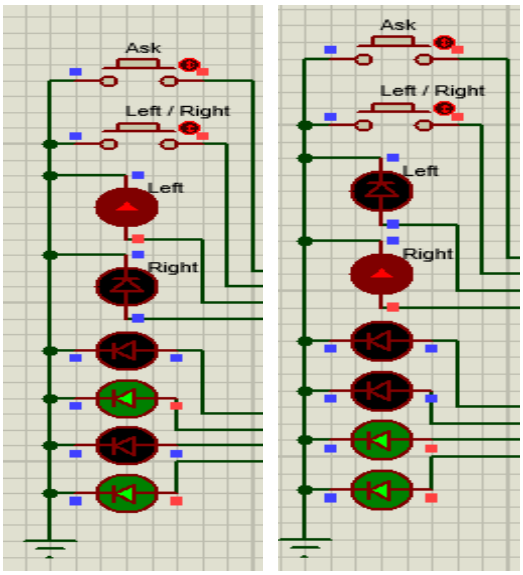
До порту C ведучого МК підключено 8-ми позиційний перемикач для задання адреси одного з ведених МК, з якого ми хочемо отримати інформацію.

Для запуску моделі (рисунок 13.18) натискаємо кнопку «Run the simulation  » в правому нижньому куті Proteus.

Далі вводимо адресу лівого веденого мікроконтролера на 8-ми позиційному перемикачі. Ця адреса буде вводитись у ведучий МК через його лінії введення: $C0_m$, $C1_m$,…,$C7_m$. Після першого натискання кнопки «Left/Right» 

: ON – > OFF – > ON – > OFF буде обрано канал Left. Після натискання та відпускання кнопки «Ask» 

інформація на блоці світлодіодів «Data» відображає стан лівого 4-х позиційного перемикача лівого веденого МК (рисунок 15.19, а). Для отримання інформації від правого 4-х позиційного перемикача веденого треба повторно натиснути кнопку «Left/Right» та «Ask» (рисунок 15.19, б).



а б

Рисунок 15.19 – Інформація на блоці світлодіодів «Data»: а – отримання інформації від лівого 4-х позиційного перемикача; б – отримання інформації від правого 4-х позиційного перемикача

Таким чином, в моделі RS 485-232-1 обирається один з двох ведених МК за їх адресами в мережі. Після чого від обраного веденого до ведучого МК передаються 4-ри біти від одного з двох 4-позиційних перемикачів.

15.4.2 Схема алгоритму роботи моделі мережі RS-485-232-1

Схема алгоритму роботи ведучого МК представлено на рисунку 13.20.

Схема алгоритму роботи підлеглого МК представлено на рисунку 15.21.

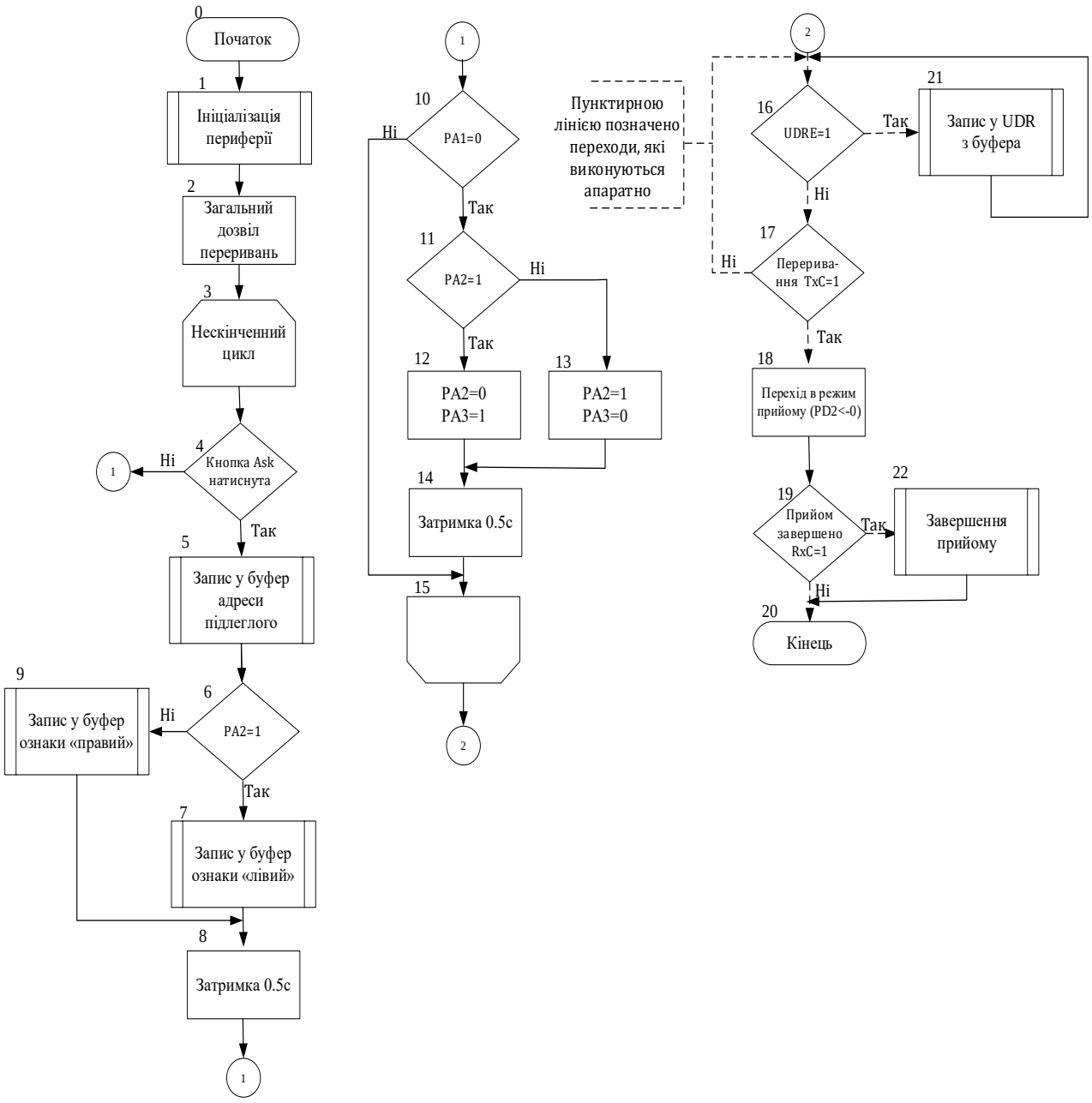
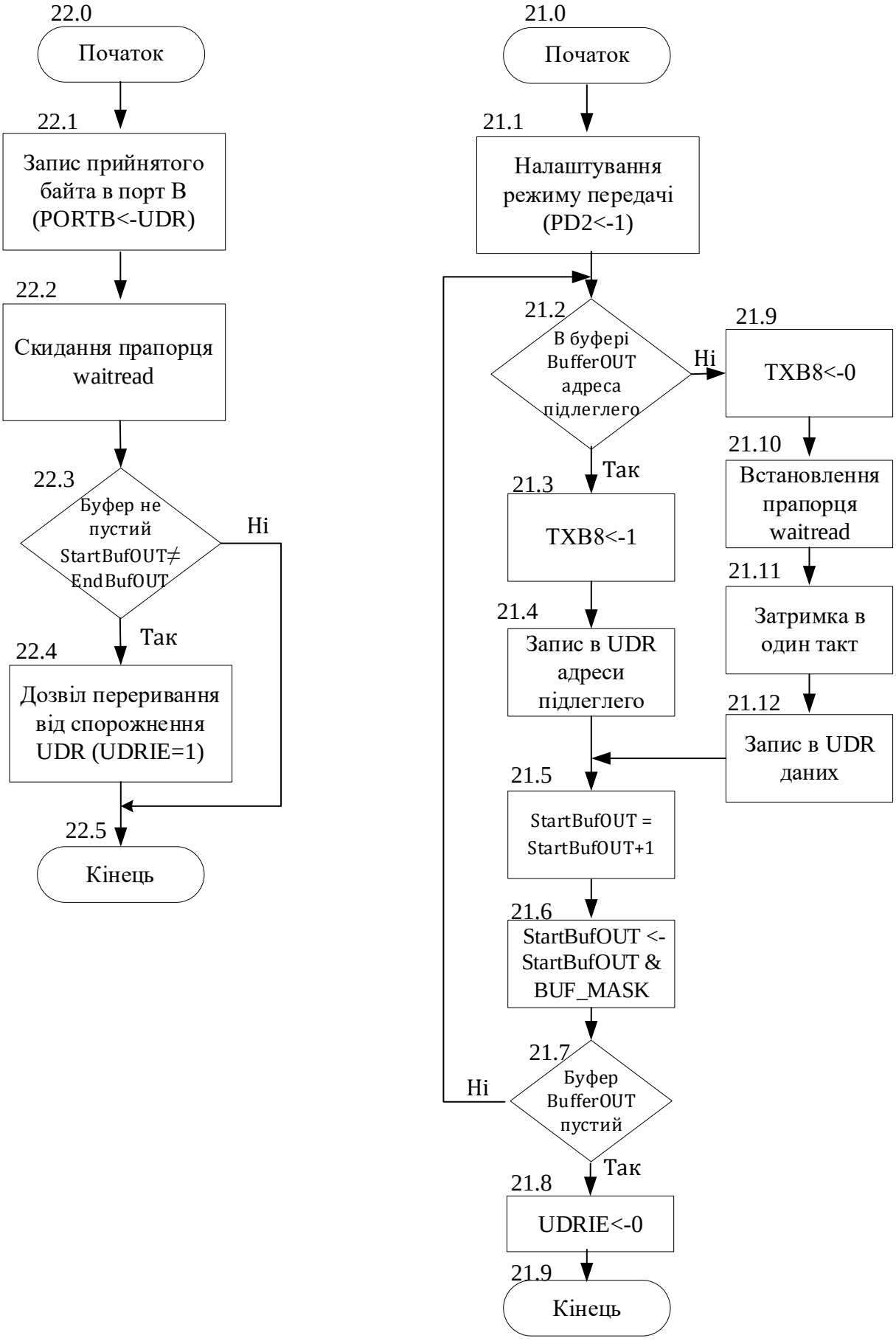
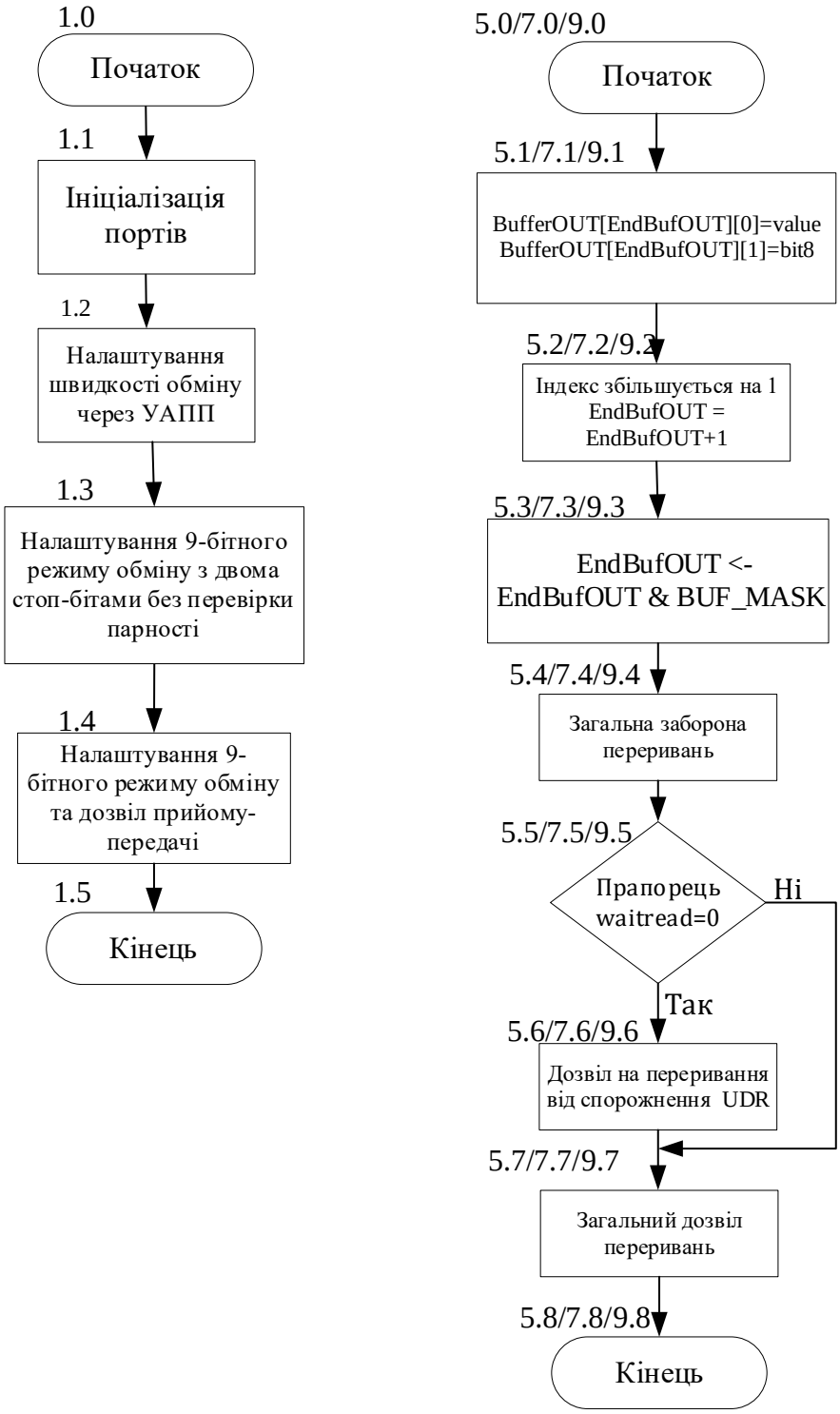


Рисунок 15.20 – Схема алгоритму роботи ведучого МК



Продовження рисунку 15.20 (стор. 443)



Закінчення рисунку 15.20 (стор. 443, 444)

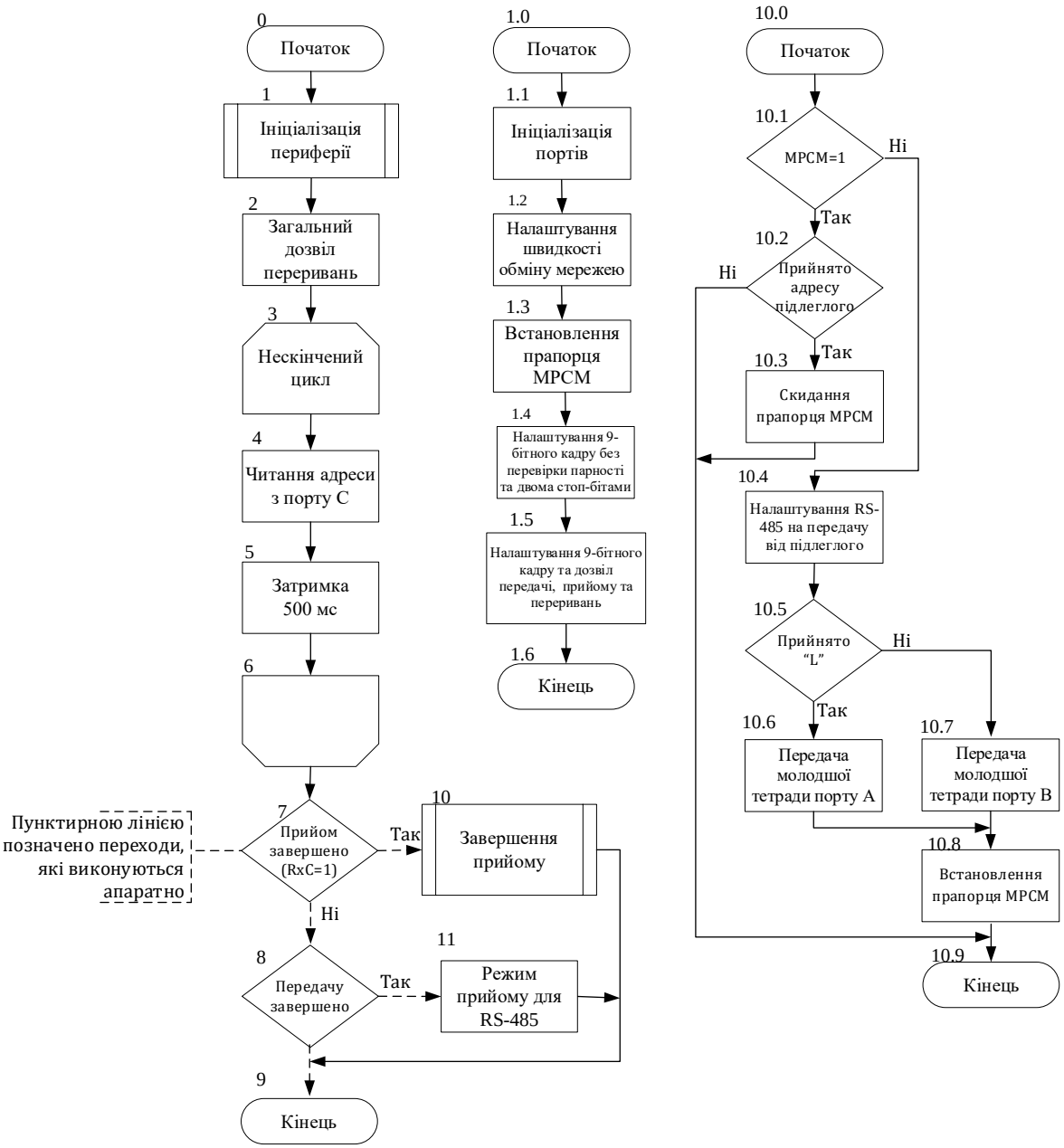


Рисунок 15.21 – Схема алгоритму роботи підлеглого МК

15.4.3 Робоча програма моделі мережі RS-485-232-1

Програмний код МК «master»

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define F_CPU 8000000L
#define BAUD 9600
#define UBRRcalc (F_CPU/(BAUD*16L)-1)
#define BUF_SIZE 16
#define BUF_MASK (BUF_SIZE-1)
unsigned char BufferOUT[BUF_SIZE][2], StartBufOUT = 0, EndBufOUT = 0;
volatile unsigned char waitread = 0; // прапорець (очикування читання)
// --------------- запис у буфер передачі ------------------------------------------------------
void WriteBufOUT(unsigned char value, unsigned char bit8)
{       // 5.0/7.0/9.0
// 5.1/7.1/9.1
        BufferOUT[EndBufOUT][0] = value;   // запис адреси або «лівий/правий» у буфер
        BufferOUT[EndBufOUT++][1] = bit8; // запис ознаки адреси у буфер , та
                                                // 5.2/7.2/9.2
                                                 // збільшення індексу EndBufOUT  на 1
        EndBufOUT &= BUF_MASK;          // 5.3/7.3/9.3 обнулення EndBufOUT при досягненні
                                                 // максимального розміру буфера
                                                // EndBufOUT  <- EndBufOUT&BUF_MASK
        cli(); // 5.4/7.4/9.4  загальна заборона переривань
        if( waitread == 0 )                     // 5.5/7.5/9.5 перевірка прапорця waitread
        UCSRB |= 1<<UDRIE;          // 5.6/7.6/9.6 дозвіл переривання від спорожнення UDR
        sei();                                  // 5.7/7.7/9.7   загальний дозвіл переривання
}       // 5.8/7.8/9.8
// --------------- переривання при прийнятому байті даних ------------------------------
ISR(USART_RXC_vect)     //19
{       //22.0
        PORTB = UDR;        //22.1 запис прийнятого байта з UDR в PORTB
        waitread = 0;           //22.2 скидання прапорця waitread (очікування читання) в 0
        // перевіряємо на спорожнення буфера передачі
        if( StartBufOUT != EndBufOUT )    //22.3 якщо буфер не пустий, то
        UCSRB |= 1<<UDRIE;          // 22.4 дозвіл переривання від спорожнення UDR
}               //22.5
// --------------- переривання при завершенні передачі ----------------------------------
ISR(USART_TXC_vect )                //17
{
PORTD &= ~(1<<PD2);                 //18  перехід в режим прийому
} //
// --------------- переривання при спорожненні буферу UDR ------------------------------
ISR(USART_UDRE_vect )               //16
{                                       //21.0
        PORTD |= 1<<PD2;            //21.1 режим передачі
        if( BufferOUT[StartBufOUT][1] == 1 )      //21.2 якщо ознака адреси, то
                UCSRB |= 1<<TXB8;                   //21.3 bit8=1
        else {
```

```
                UCSRB &= ~(1<<TXB8);    //21.9 якщо «лівий» чи «правий», bit8=0
                waitread = 1;                   //21.10   прапорець waitread встановлюємо в 1 ;
                 // і очікуємо прийому з буферу в UDR
              }
        asm("nop");             // 21.11   затримка в 1 такт
//21.4  і 21.12 читання з FIFO-буфера в UDR ( перший раз -адреси; другий раз -даних)
        UDR = BufferOUT[StartBufOUT++][0];    // 21.5 StartBufOUT= StartBufOUT +1
          StartBufOUT &= BUF_MASK; // 21.6 обнулення StartBufOUT при  досягненні
                                   // максимального розміру буфера
                                   // StartBufOUT  <- StartBufOUT&BUF_MASK
        // перевіряємо на спорожнення буфера передачі
        if( StartBufOUT == EndBufOUT || waitread == 1 )//21.7 //якщо буфер не пустий,
                                                       //виконується блок 21.2, інакше
        UCSRB &= ~(1<<UDRIE);    // 21.8  UDRIE <-0  заборона на переривання від
                                 //спорожнення буферу в UDR
}               //21.9
int main(void)
{
        Init();    // 1  ініціалізація периферії
        sei();     // 2  загальний дозвіл на переривання
        while (1)
        {       //3 початок нескінченного циклу
                if( bit_is_clear(/*PINC*/ PINA, 0) )    // 4  якщо Ask=0
                {
                        WriteBufOUT(PINC, 1);      // 5 запис в буфер адреси підлеглого
                        if( bit_is_set(PINA, 2) )       //6 якщо PA.2=1
                              WriteBufOUT('L', 0);     // 7  запис в буфер ознаки «лівий»
                        else WriteBufOUT('R', 0);      //  9  запис в буфер ознаки «правий»
                        _delay_ms(500);                // 8  затримка 0.5 сек.
                }
                if( bit_is_clear(/*PINC*/PINA, 1) )  //10   якщо PA.1=0
                {
                        if( bit_is_set(PINA, 2) )                       //11 якщо PA.2=1
                       {PORTA &= ~(1<<2); PORTA |= 1<<3;}   //12 PA.2=0; PA.3=1
                          else {PORTA &= ~(1<<3); PORTA |= 1<<2;}    //13 PA.2=1; PA.3=0
                        _delay_ms(500);                         // 14  затримка 0.5 сек.
                }
        }               //15  кінець нескінченого циклу
}   //20  кінець програми
void Init() // ініціалізація периферії
{               //1.0
        //1.1  ініціалізація портів
        DDRA |= 0b11111100; PORTA |= 0x03; //PA2…PA7 – на вихід, PA0 – PA1– на вхід,
        //PORTA=0x03 – Rпідт.
        DDRB=0xFF; PORTB=0x00; // порт B – на вихід, початкове значення = 0
        DDRC=0x00; PORTC=0xFF; // порт C – на вхід,  початкове значення = 0xFF – Rпідт.
        DDRD=0b11111110; PORTD=0b00000001; // pin0 – Rпідт.
        //1.2  налаштування швидкості обміну через УАПП
        UBRRL = (unsigned char)(UBRRcalc);
        UBRRH = (unsigned char)(UBRRcalc>>8);
```

```
// 1.3 налаштування 9-бітного режиму обміну з двома стоп-бітами без перевірки парності
        UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0)|(1<<USBS);
// 1.4 налаштування 9-бітного режиму обміну, дозвіл прийому-передачі та переривань
        UCSRB = (1<<UCSZ2)|(1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<TXCIE);
}               //1.5
```

Програмний код МК «slave»

```
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define F_CPU 8000000L
#define BAUD 9600
#define UBRRcalc (F_CPU/(BAUD*16L)-1)
unsigned char Address;
// --------------- переривання при прийнятому байті даних ------------------------------
ISR(USART_RXC_vect)             //7
{       //10.0
if( bit_is_set(UCSRA, MPCM) ) //10.1 якщо розряд MPCM у регістрі
                              //керування UCSRA встановлено в «1»
        {
                if( UDR == Address )          //10.2 якщо прийнято адресу підлеглого

                UCSRA &= ~(1<<MPCM); //10.3      скидання прапорця MPCM

        }
        else
        {
                PORTD |= 1<<PD2;    //10.4   налаштування RS-485 на передачу від підлеглого
                if( UDR == 'L' )    //10.5       прийнято Left від ведучого
             UDR = PINA;  //10.6 передача молодшої тетради порту A (надсилання значення
                          //Left)
        else
                 UDR = PINB;   //10.7 передача молодшої тетради порту B (надсил. знач. Right)
                UCSRA |= (1<<MPCM); // 10.8          встановлення прапорця MPCM
        }
}               //10.9
// --------------- переривання при завершенні передачі ----------------------------------
ISR(USART_TXC_vect )     //8
{
PORTD &= ~(1<<PD2); //11          режим прийому для RS-485
}
int main(void)
{       //0
        Init(); //1         ініціалізація периферії
        sei(); //2          загальний дозвіл переривань
        while (1)
        {               //3 початок нескінченного циклу
                Address = PINC;       //4 читання адреси з порту C
                _delay_ms(500);       //5 затримка
        }       //6
}       //9
void Init() // ініціалізація периферії
```

```
{       //1.0
        //1.1
        DDRA=0x00; PORTA=0xFF; // порт A – на вхід, підключення R підтягування
        DDRB=0x00; PORTB=0xFF; // порт B – на вхід, підключення R підтягування
        DDRC=0x00; PORTC=0xFF; // порт C – на вхід, підключення R підтягування
        DDRD=0b11111110; PORTD=0b00000001; // pin PD0 – на вхід, підключення
                                           // R підтягування
        //1.2 налаштування швидкості обміну мережею
        UBRRL = (unsigned char)(UBRRcalc);
        UBRRH = (unsigned char)(UBRRcalc>>8);
        // 1.3 встановлення прапорця MPCM
        UCSRA = (1<<MPCM);
        //1.4 налаштування 9-бітного кадру без перевірки парності та двома стоп-бітами
        UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0)|(1<<USBS);
        //1.5    дозвіл прийому, передачі, переривань; 9-бітний кадр
        UCSRB = (1<<UCSZ2)|(1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<TXCIE);
            } //1.6
```

13.4.4 Опис моделі мережі RS-485-232-2

Нижче розглядається модель RS-485-232-2, схему якої наведено на рисунку 15. 22.

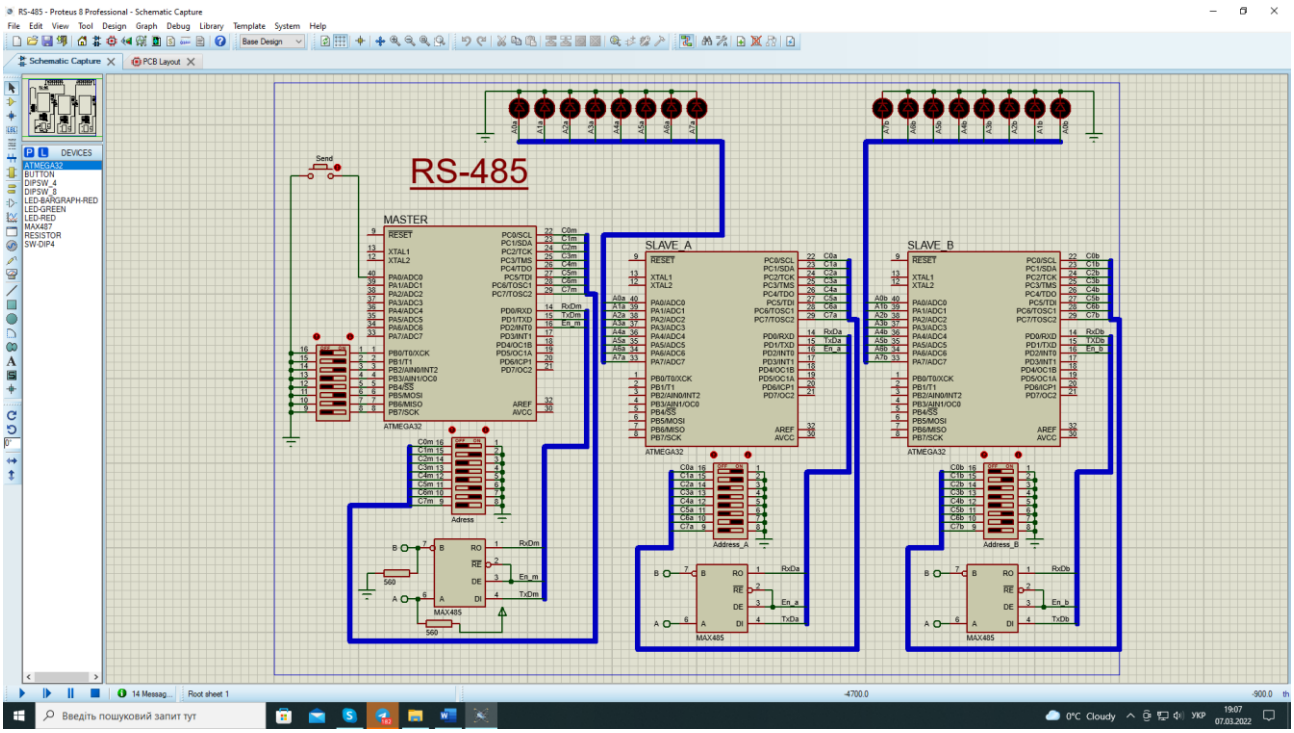


Рисунок 15.22 – Схема робочої моделі мережі RS-485-232-2

На схемі розміщено 3 мікроконтролери (лівий працює в режимі «Master», інші два – в режимі «Slave»). В якості мікроконтролерів використовуються мікросхеми ATMEGA32.

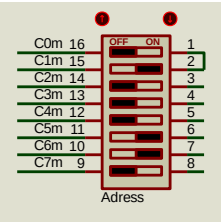
До ліній портів D мікроконтролерів: RXD, TXD та INT0 підключені приймачі-передавачі MAX485. Останні зв’язані з проводами A та B мережі. Ці проводи підключено до сигнальної «землі» двома резисторами-термінаторами, значенням 560Ом.

До двох ведених (підлеглих) мікроконтролерів через лінії портів C підключено 8-ми позиційні перемикачі, які задають адреси цих МК в мережі.

До ведучого мікроконтролера до порту B підключено 8-ми клавішний перемикач для задання байта даних для обраного веденого мікроконтролера, а до порту C підключено 8-ми клавішний перемикач для задання байта адреси обраного веденого мікроконтролера.

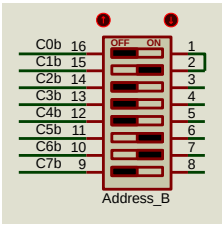
До лінії PA0 порту A підключено кнопку керування: «Send».

Після відкриття проекту до запуску програми на 8-ми клавішному перемикачі (знаходиться знизу від мікроконтролера «МАСТЕР»вводимо адресу одного з ведених, наприклад, правого:

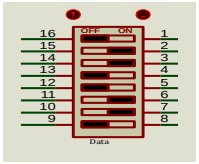


.

Такуж адресу на 8-ми клавішному перемикачі (знаходиться знизу від мікроконтролера «SLAVE_B») вводимо адресу цього правого веденого:



На 8-ми клавішному перемикачі (знаходиться зліва від мікроконтролера «МАСТЕР») задаємо байт даних для передачі обраному веденому:



Після цього для запуску програми натискаємо кнопку «Run the imulation

», яка знаходиться в лівому нижньому куті Proteus (рисунок 15.23).

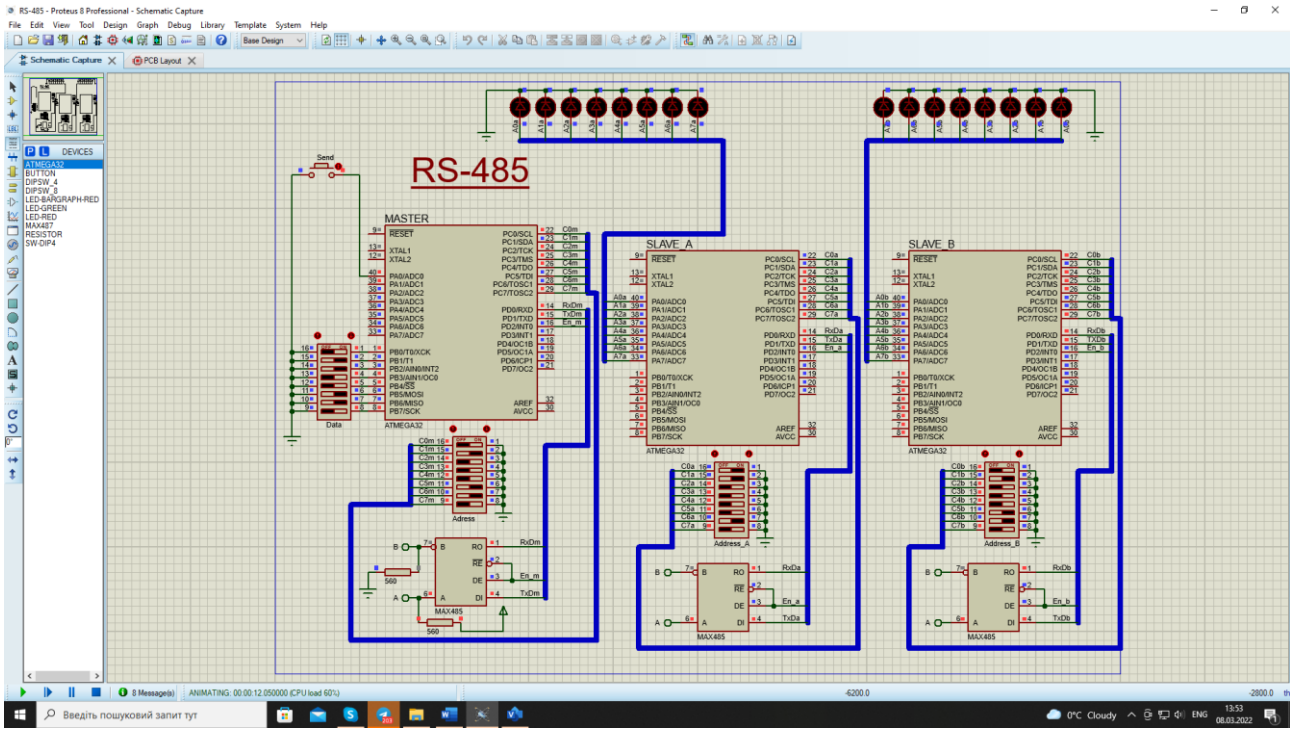
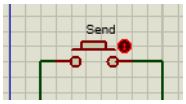
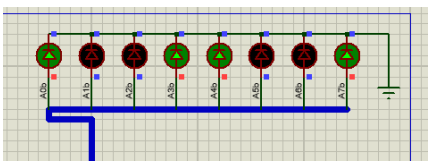


Рисунок 15.23 – Схема робочої моделі мережі RS-485-232-2 після запуску моделювання

Потім натискаємо кнопку «Send» 

для відправки повідомлення від ведучого до обраного веденого.

Після цього значення заданого байта відобразиться на панелі світодіодів мікроконтролера з відповідною адресою:



.

Можна змінити байт даних для відправки, використовуючи 8-ми клавішний перемикач, що під'єднаний до ведучого мікроконтролера, та адресувати лівий ведений мікроконтролер (рисунок 15.24).

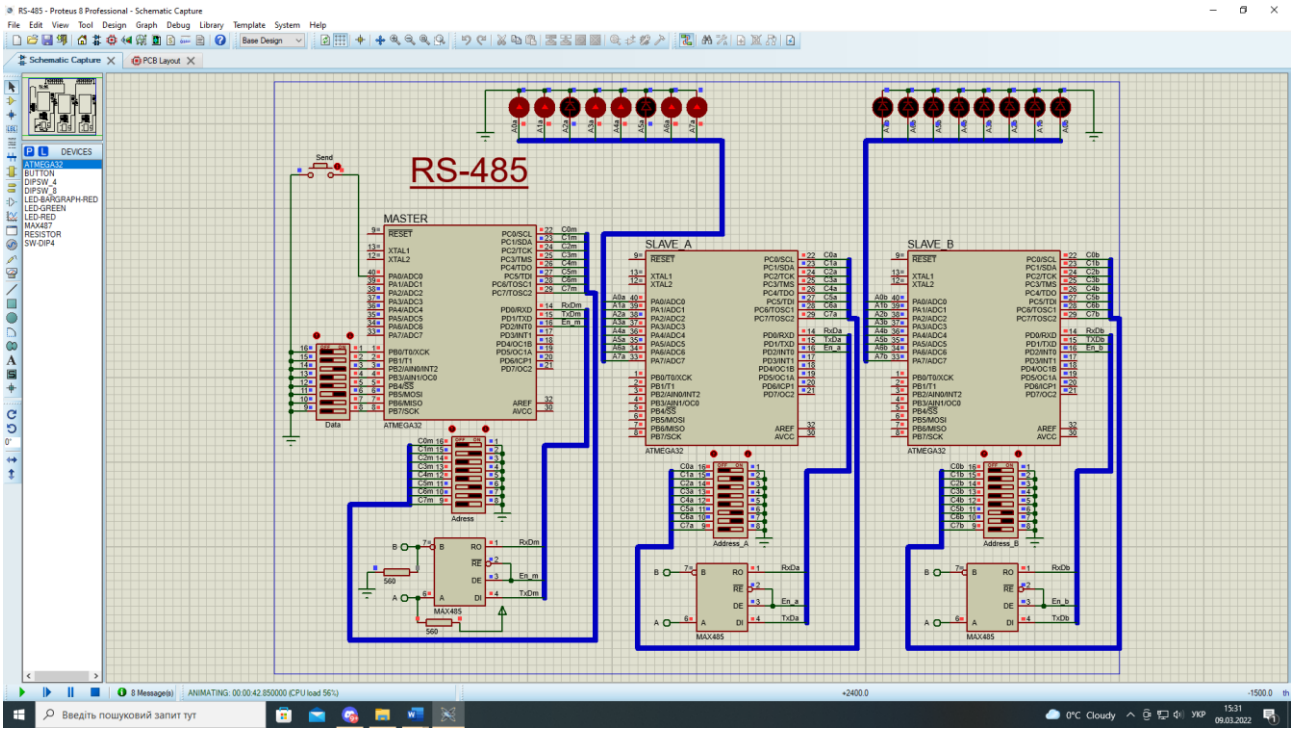


Рисунок 15.24 – Схема робочої моделі мережі RS-485-232-2 після запуску моделювання при зміни байта та адресації лівого веденого МК

Таким чином, моделювання RS-485-232-2 підтвердило можливість передачі байта даних від ведучого МК мережі до одного з двох ведених з використанням мультипроцесорного режиму роботи AVR-мікроконтролерів (див. 9.2.6).

15.4.5 Схема алгоритму роботи моделі мережі RS-485-232-2

Схема алгоритму роботи ведучого МК представлено на рисунку 15.26.

Схема алгоритму роботи веденого МК представлено на рисунку 15.27.

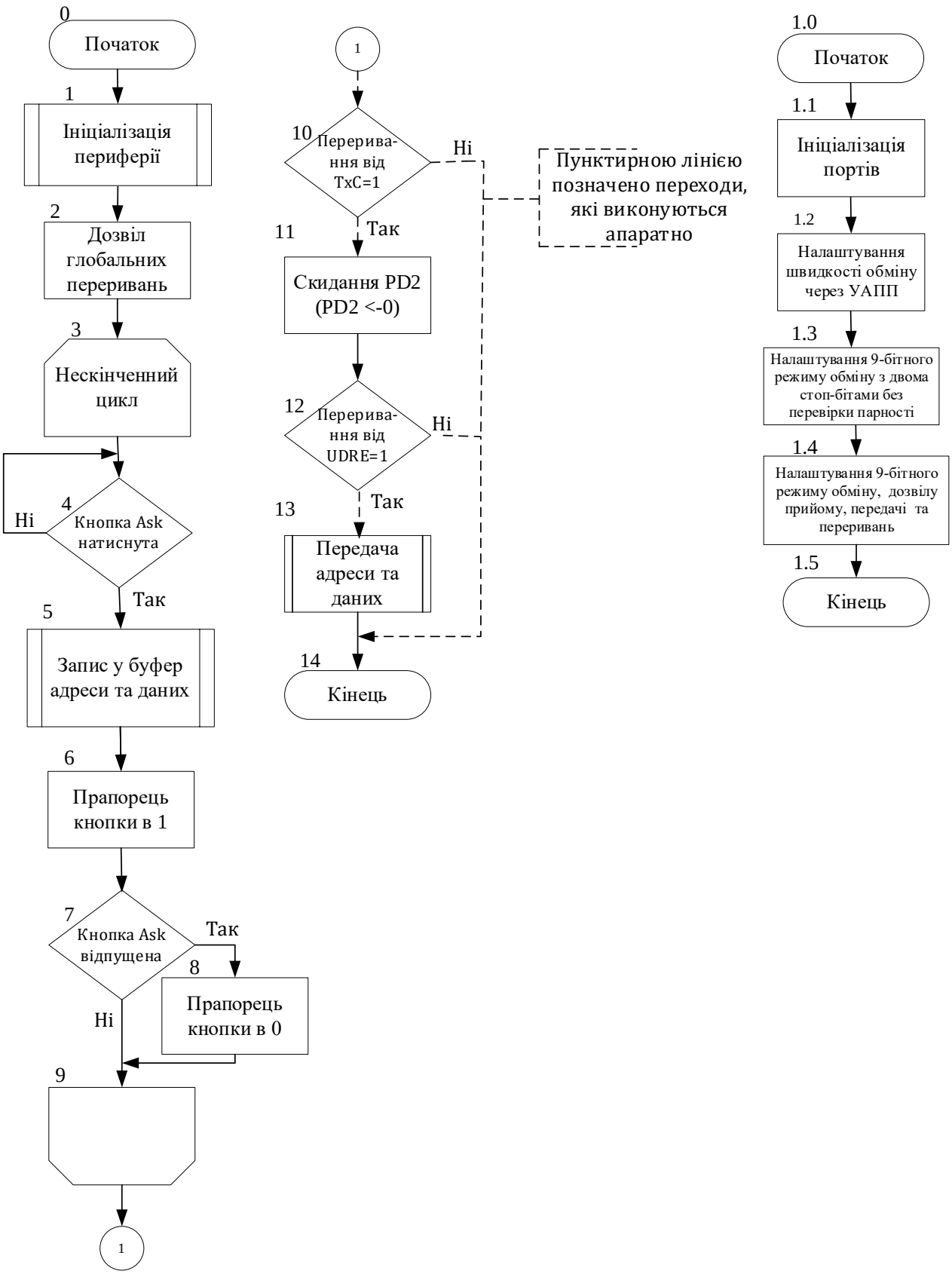
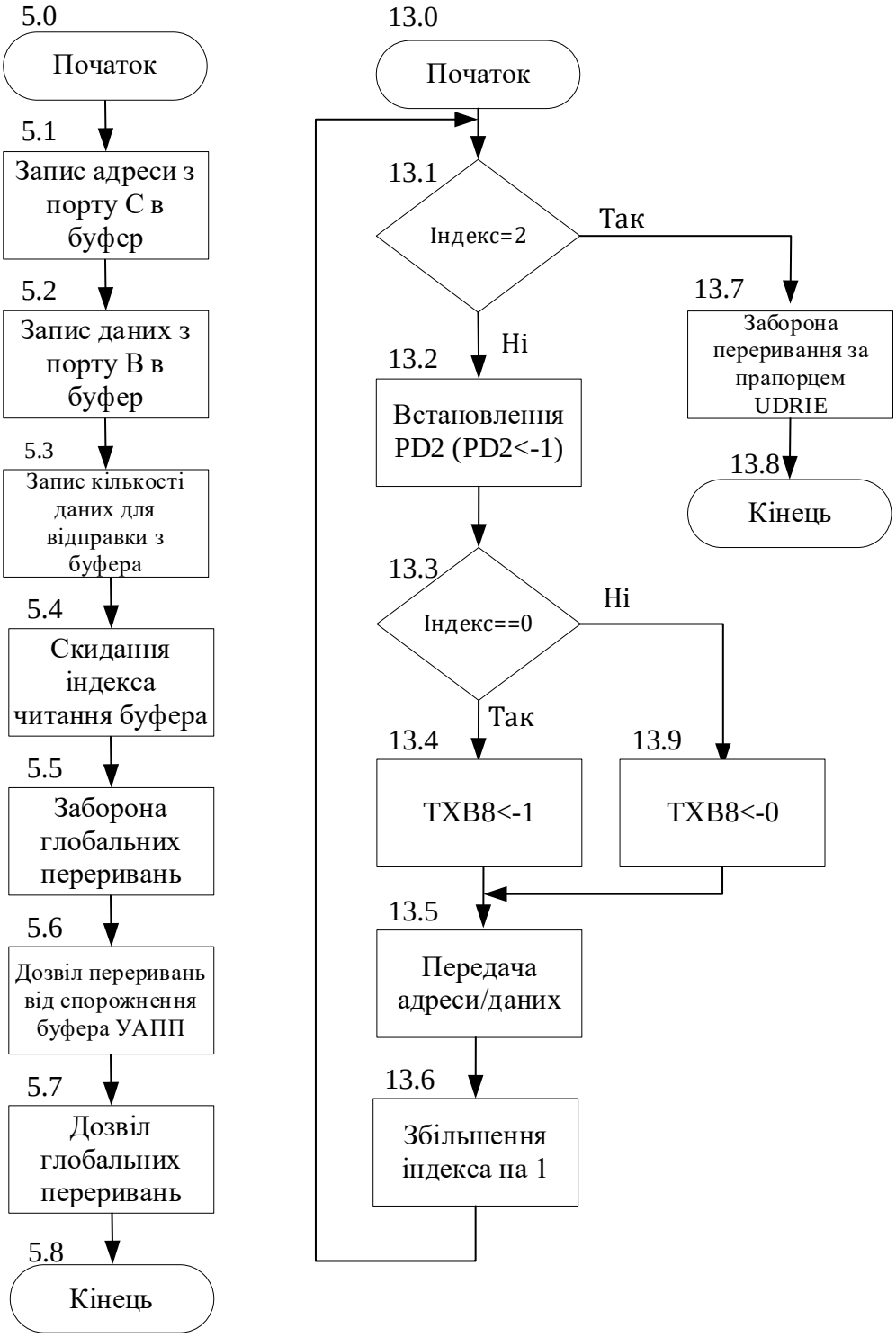


Рисунок 15.26 – Схема алгоритму роботи ведучого МК



Закінчення рисунку 15.26 (стор.454)

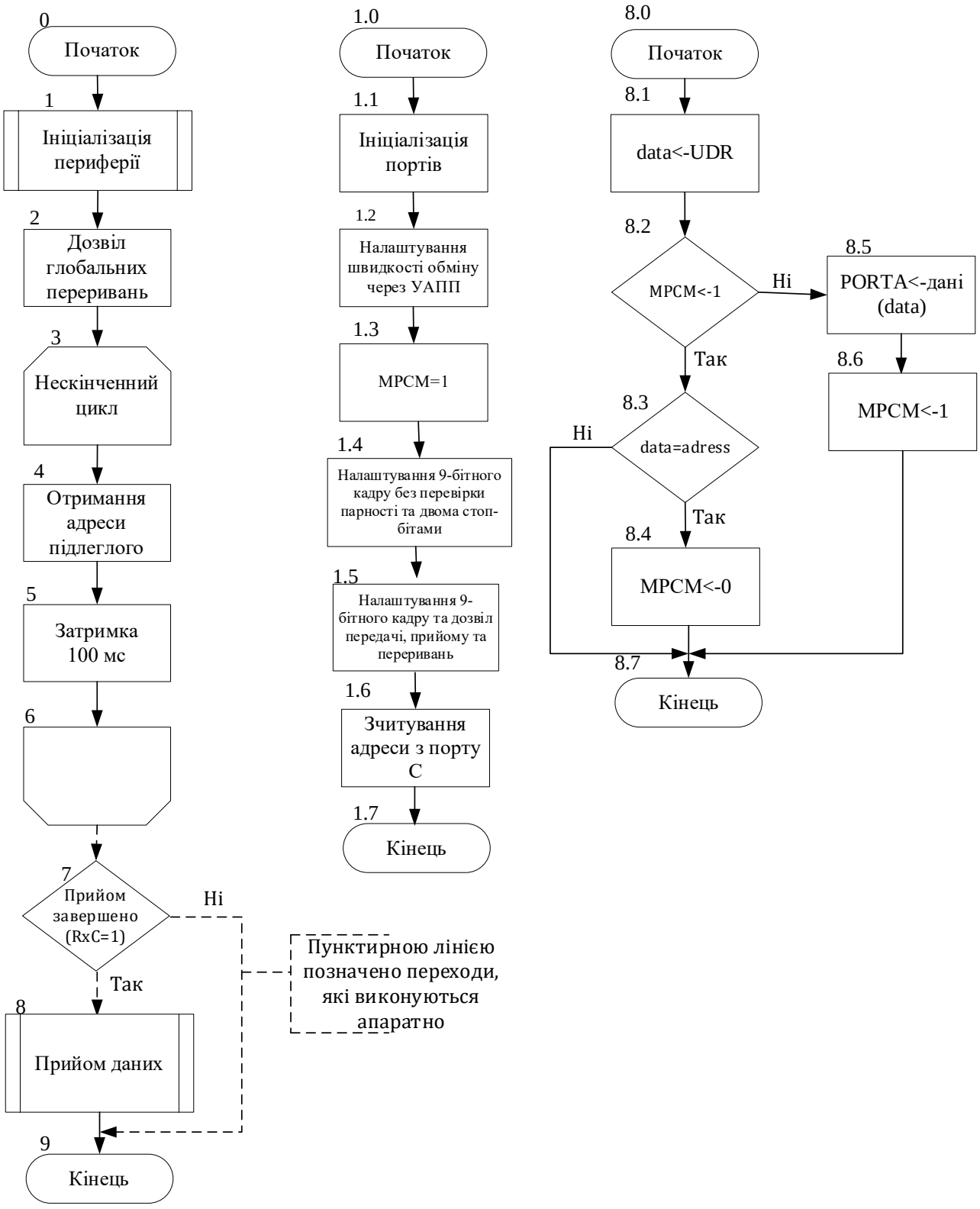


Рисунок 15.27 – Схема алгоритму роботи веденого МК

15.4.6 Робоча програма моделі мережі RS-485-232-2

Програмний код МК «master»

```
#define F_CPU 8000000L
#include <avr/io.h>
#include <stdbool.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#define BAUD 9600
#define UBRR_CALC (F_CPU/(BAUD*16L)-1)
#define BUF_SIZE 16
volatile uint8_t bufferOUT[BUF_SIZE];
volatile uint8_t dataCount = 0;
volatile uint8_t readIdx = 0;
bool buttonFlag = 0;
void WriteBufOUT(uint8_t address, uint8_t value);
void Init();
int main(void)
{	// 0
	Init(); // 1
	sei(); // 2
	while (1)
	{	// 3
		if( bit_is_clear(PINA, 0) )	// 4
		{
			if(!buttonFlag){
				WriteBufOUT(PINC, PINB); // 5
				buttonFlag = 1;		// 6
			}
		}
		if( bit_is_set(PINA, 0) )	//7
		{
			buttonFlag = 0;		//8
		}
	}	//9
}	//14
void Init() // 1
{	//1.0
	// 1.1
	DDRA |= 0b11111100; PORTA |= 0x03; //PA2…PA7 – на вихід, PA0 – PA1– на вхід,
	//PORTA=0x03 – Rпідт.
	DDRB=0x00; PORTB=0xFF; // порт B – на вхід, початкове значення = 0xFF – Rпідт.
	DDRC=0x00; PORTC=0xFF; // порт C – на вхід, початкове значення = 0xFF – Rпідт.
	DDRD=0b11111110; PORTD=0b00000001; //PD1… PD7 – на вихід, PD0 – на вхід,
	//PD0=1 – Rпідт.
	// 1.2 налаштування швидкості обміну через УАПП
	UBRRL = (unsigned char)(UBRR_CALC);
	UBRRH = (unsigned char)(UBRR_CALC>>8);
```

```
        // 1.3 налаштування 9-бітного режиму обміну з двома стоп-бітами без перевірки
        //парності
        UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0)|(1<<USBS);
        // 1.4 налаштування 9-бітного режиму обміну, дозвіл прийому-передачі та переривань
        UCSRB = (1<<UCSZ2)|(1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<TXCIE);
}       // 1.5

void WriteBufOUT(uint8_t address, uint8_t value)
{       // 5.0
        bufferOUT[0] = address; // 5.1
        bufferOUT[1] = value; // 5.2
        dataCount = 2; // 5.3
        readIdx = 0; // 5.4
        cli();    // 5.5
        UCSRB |= 1<<UDRIE; // 5.6
        sei();    // 5.7
}       // 5.8


ISR(USART_TXC_vect)      // 10
{
 PORTD &= ~(1<<PD2); // 11
}

ISR(USART_UDRE_vect ) // 12
{       //13.0
        if(readIdx == dataCount)   //13.1 якщо індекс=2
        {
                UCSRB &= ~(1<<UDRIE); // 13.7 заборона переривання за прапорцем UDRIE
                return; // на кінець
        }

        PORTD |= 1<<PD2; // 13.2
        if(readIdx == 0){       // 13.3 address byte
                UCSRB |= 1<<TXB8; // 13.4
                } else {
                UCSRB &= ~(1<<TXB8); // 13.9
        }
        UDR = bufferOUT[readIdx]; // 13.5
        readIdx++;      // 13.6
}       // 13.8
```

Програмний код МК «slave»

```
#define F_CPU 8000000L
#define F_CPU 8000000L
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <avr/sfr_defs.h>
#define BAUD 9600
#define UBRR_CALC (F_CPU/(BAUD*16L)-1)
uint8_t address;
```

```
void Init();
int main(void)
{
	Init(); // 1
	sei(); // 2
	while (1)
	{	// 3
		address = PINC;	// 4
		_delay_ms(100);	// 5
	}	// 6
}	// 9
void Init() // 1
{	// 1.0
	DDRA=0xFF; PORTA=0x00; // 1.1 порт А – на вихід, PORTA=0x00;
	DDRB=0x00; PORTB=0xFF; // порт В – на вхід, початкове значення = 0xFF – Rпідт.
	DDRC=0x00; PORTC=0xFF; // порт С – на вхід, початкове значення = 0xFF – Rпідт.
	DDRD=0b11111110; PORTD=0b00000001; //PD1… PD7 – на вихід, PD0 – на вхід,
	//PD0=1 – Rпідт.

	// 1.2 Налаштування швидкості обміну через УАПП
	UBRRL = (unsigned char)(UBRR_CALC);
	UBRRH = (unsigned char)(UBRR_CALC>>8);
	// 1.3
	UCSRA = (1<<MPCM);
	// 1.4 Налаштування 9-бітного кадру без перевірки парності та двома стоп-бітами
	UCSRC = (1<<URSEL)|(1<<UCSZ1)|(1<<UCSZ0)|(1<<USBS);
	// 1.5 Налаштування 9-бітного кадру та дозвіл передачі, прийому та переривань
	UCSRB = (1<<UCSZ2)|(1<<RXEN)|(1<<TXEN)|(1<<RXCIE)|(1<<TXCIE);
	// 1.6 зчитування адреси з порту С
	address = PINC;
}	// 1.7

ISR(USART_RXC_vect)	// 7
{	// 8.0
	uint8_t data = UDR;	// 8.1
	if( bit_is_set(UCSRA, MPCM) )	// 8.2
	{
		if( data == address ) // 8.3
		UCSRA &= ~(1<<MPCM); // 8.4
	}
	else
	{
		PORTA = data; // 8.5
		UCSRA |= (1<<MPCM);	// 8.6
	}
}	// 8.7
```

15.5 Зміст звіту

Звіт по роботі повинен містити:

- схему моделі;
- схему алгоритму роботи моделі;
- робочу програму;
- формули за потребою.

Контрольні запитання та завдання

1) Дайте загальну характеристику інтерфейсу RS-485/EIA-485.
2) Наведіть та опишіть схему підключення інтерфейсу RS-485/EIA-485 до трьохпровідної лінії.
3) Яку кількість вузлів можна підключити в мережу RS-485/EIA-485?
4) Якою може бути швидкість обміну та відстань в мережі RS-485/EIA-485?
5) Наведіть та поясніть схему підключення інтерфейсів RS-485 до локальної мережі.
6) Назвіть загальні рекомендації щодо використання інтерфейсу RS-485.
7) Наведіть та поясніть схему реалізації ланцюга зсуву на сигнальних ланцюгах інтерфейсу RS-485.
8) Назвіть параметри інтерфейсу RS-485 для мікросхем фірми MAXIM.
9) Опишіть призначення виводів напівдуплексних мікросхем RS-485 фірми MAXIM.
10) Наведіть та опишіть схему типу точка-точка (напівдуплекс) з’єднання двох мікросхем RS-485.
11) Наведіть та опишіть схеми типового використання мікросхем RS-485 в напівдуплексній та дуплексній мережах.
12) Наведіть та опишіть схему підключення мікроконтролера AT89C51 до інтерфейсу RS-485.

13) Дайте загальну характеристику інтерфейсу RS-232.
14) Наведіть та опишіть структурну схему сполучення МП/МК-ра і ПК за допомогою інтерфейсу RS-232C і модему.
15) Наведіть та опишіть формат даних інтерфейсу RS-232.
16) Опишіть призначення та наведіть приклад реалізації пристрою перетворення рівнів (ППР).
17) Опишіть рівні сигналів RS-232 на передаючій та приймальній кінцях лінії зв’язку.
18) Опишіть позначення і особливості підключення мікросхеми MAX232.
19) Наведіть та опишіть 9-контактний роз'єм RS-232.
20) Наведіть та опишіть схему робочої моделі мережі RS-485-232-1.
21) Наведіть та опишіть схему робочої моделі мережі RS-485-232-2.
22) Наведіть та опишіть схеми алгоритмів роботи та робочі програми моделей: RS-485-232-1 та RS-485-232-2.
23) Виконайте порівняння моделей RS-485-232-1 та RS-485-232-2.

СПИСОК ЛІТЕРАТУРИ

Базова

1. Мікропроцесорні та мікроконтролерні системи: Ч.2 «Проектування мікропроцесорних систем» [Електронний ресурс] : підручник для студ. освітньої програми «Інтегровані інформаційні системи» за спеціальністю 126 «Інформаційні системи та технології» / А.О. Новацький ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 20,3 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2020. – 460 с.
2. Проектування мікропроцесорних систем: Проектування мікропроцесорних систем на базі AVR-мікроконтролерів: Периферійні модулі AVR-мікроконтролерів: Навчальний посібник для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / Укл.: А.О. Новацький– К: НТУУ „КПІ”, 2012. – 470 с.
3. Навчальний посібник з дисципліни «Проектування мікропроцесорних систем», розділ «Програмування мікроконтролерів родини AVR» для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / Укл.: А.О. Новацький, Є.В. Глушко – К: НТУУ „КПІ”, 2013. – 109 с.
4. Мікропроцесорні та мікроконтролерні системи : підручник. У 2 ч. Ч. 1. Мікропроцесорні системи [Електронний ресурс] / А. О. Новацький. – Електронні текстові дані (1 файл: 43,8 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, Вид-во «Політехніка», 2019. – 367 с. : ил.
5. Мікропроцесорні та мікроконтролерні системи : лаб. практикум [Електронний ресурс] : навч. посіб. для студ. освітньої програми «Інтегровані інформаційні системи» спец. 126 «Інформаційні системи та технології» / Уклад. А. О. Новацький. – Електронні текстові дані (1 файл: 18,97 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018. – 415 с.: ил.
6. Проектування мережі 1-WIRE: Навчальний посібник для студентів спеціальностей 7.05020101, 8.05020101 «Комп’ютеризовані системи управління та автоматика» кафедри автоматики та управління в технічних системах / Автор: А.О. Новацький– К: НТУУ „КПІ”, 2016 – 141с.
7. Конспект лекцій з дисципліни «Проектування та програмування мікропроцесорних систем та мереж », розділ «Проектування промислових мереж на базі стандартів передачі даних RS-485, RS-232 та 1-Wire » для студентів напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / Укл.: А.О. Новацький – К: НТУУ „КПІ”, 2012 – 178 с.

8. Проектування мікропроцесорних систем: Проектування мікропроцесорних систем на базі мікроконтролерів сімейства MCS-51: Периферійні модулі мікроконтролерів сімейства MCS-51 :навч. посіб. для студ. напряму підготовки 6.050201 «Системна інженерія» кафедри Автоматики та управління у технічних системах / А. О. Новацький. – Київ : НТУУ «КПІ», 2016. – 399 с.
9. Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL – М.: Додэка, 2005 – 560 с.
10. Евстифеев А. В. Микроконтроллеры AVR семейства Mega. Руководство пользователя. – М.:Издательский дом «Додэка-XXI», 2007 – 592 с.
11. Проектування CAN-мережі: Навчальний посібник для студентів спеціальності 8.050201.01 «Комп'ютеризовані системи управління та автоматика» кафедри Автоматики та управління у технічних системах / Автор: А.О. Новацький К: НТУУ „КПІ", 2011 – 169 с.
12. Комп'ютерна електроніка [Електронний ресурс] : підручник для студ. спеціальності 126 «Інформаційні системи та технології», спеціалізації «Інтегровані інформаційні системи» /А.О. Новацький ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 80.9 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2018 – 468 с.

Допоміжна

13. Програмування мікроконтролерів систем автоматики: конспект лекцій для студентів базового напряму 050201 "Системна інженерія" / Укл.: А.Г. Павельчак, В.В. Самотий, Ю.В. Яцук – Львів: Львівська політехніка. – 2012. – 143 с.
14. Голубцов М.С., Кириченкова А.В. Микроконтроллеры AVR: от простого к сложному. – М.: Солон-Пресс, 2005.
15. Трамперт Вольфган. AVR-RISC микроконтроллеры. – Перевод с немецкого. – Киев.: МК – Пресс, 2006.
16. Баранов В.Н. Применения микроконтроллеров AVR: схемы, алгоритмы, программы. – М.: Додэка, 2004.
17. Ревич Ю. В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера. – 2-е изд., испр. – СПб.:БХВ-Петербург, 2011.
18. Александров Е. К., Грушвицкий Р. И. Микропроцессорные системы: Учебное пособие для вузов. – СПб.: Политехника, 2002.
19. Официальное описание микроконтроллеров XMEGA http://www.gaw.ru/html.cgi/txt/ic/Atmel/micros/avr_xmega/start.htm