МІКРОПРОЦЕСОРНІ ТА МІКРОКОНТРОЛЕРНІ СИСТЕМИ

Частина 2. Проектування мікропроцесорних 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
2021

Мікропроцесорні та мікроконтролерні системи: Частина 2. Проектування мікропроцесорних систем: Лабораторний практикум [Електронний ресурс] : навч. посіб. для студ. освітньої програми «Інтегровані інформаційні системи» спеціальності 126 «Інформаційні системи та технології» / А.О. Новацький ; КПІ ім. Ігоря Сікорського. – Електронні текстові дані (1 файл: 22,38 Мбайт). – Київ : КПІ ім. Ігоря Сікорського, 2021. – 268 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 6 від 25.02.2021 р.) за поданням Вченої ради факультету Інформатики та обчислювальної техніки (протокол № 6 від 20. 01. 2021 р.)

Електронне мережне навчальне видання

Новацький Анатолій Олександрович, к.т.н., доцент

МІКРОПРОЦЕСОРНІ ТА МІКРОКОНТРОЛЕРНІ СИСТЕМИ

Частина 2. Проектування мікропроцесорних систем

Лабораторний практикум

Відповідальний редактор: Полторак В. П., к.т.н., доцент, КПІ ім. Ігоря Сікорського, факультет інформатики та обчислювальної техніки, кафедра автоматики та управління в технічних системах

Рецензент: Ткач Михайло Мартинович, к.т.н., доцент, КПІ ім. Ігоря Сікорського, факультет інформатики та обчислювальної техніки, кафедра технічної кібернетики

Навчальний посібник охоплює теоретичний матеріал та практичні завдання, які необхідні для виконання лабораторного практикуму з дисципліни «Мікропроцесорні та мікроконтролерні системи, кредитний модуль «Проектування мікропроцесорних систем»». Практикум виконується у комп'ютерному класі кафедри з використанням симулятора «AVR-studio» та моделюючого пакету PROTEUS 8.6. В посібнику наводяться рекомендації по використанню цих програмних продуктів та виконанню робіт з наступної тематики: дослідження команд AVR-мікроконтролерів, моделювання периферійних модулів: універсального асинхронного приймача-передавача; SPI; I^2C; АЦП; пристрою керування двигуном постійного струму та цифрового вольтметру. 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.

ЗМІСТ

ВСТУП

При виконанні лабораторних робіт використовуються AVR-мікроконтролери, які є одним з цікавих напрямів, що розвиваються корпораціями Atmel та Microchip. Об’єми продажів AVR у світі подвоюються щорічно, неухильно росте число сторонніх фірм, що випускають програмні й апаратні засоби підтримки розробок для цих мікроконтролерів. Можна вважати, що AVR поступово стає ще одним індустріальним стандартом серед 8-розрядних мікроконтролерів загального призначення. В даний час у виробництві в Atmel Corp. знаходяться три сімейства AVR – "Tiny", "Mega" "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 покажчика 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). Усі 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 таємності 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. Усі 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 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-мікроконтролері вбудована система 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 зарезервовані 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 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 та I2C (TWI), 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 установки 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 цифрово-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 вводу/виводу 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 РОБОТИ

Згідно робочої програми дисципліни «Мікропрцесорні та мікроконтролерні системи», кредитний модуль «Проектування мікропроцесорних 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 навики 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 вивконання лабораторних робіт студенти повинні

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 на мовах 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 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 … 3; 7; 8]. 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. У частини моделей мікроконтролерів сімейства Mega пам’ять програм логічно поділено на 2 рівні частини: область прикладної програми і область завантажувача [1; 3; 7].

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

вивантаженням прикладних програм. Використання цієї області та реалізація програми-завантажувача розглянуто у [1; 2; 7]. 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; 7]. При виникненні переривання після збереження у стеку поточного значення лічильника команд відбувається виконання команди, розташованої за адресою відповідного вектора. За цими адресами розташовуються 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) і енергонезалежна EEPROM-пам'ять.

Регістрова пам'ять включає 32 регістри загального призначення (РЗП), об'єднаних у файл, і 64 службові 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 змінних, 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'ятю 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 складає для різних моделей 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.

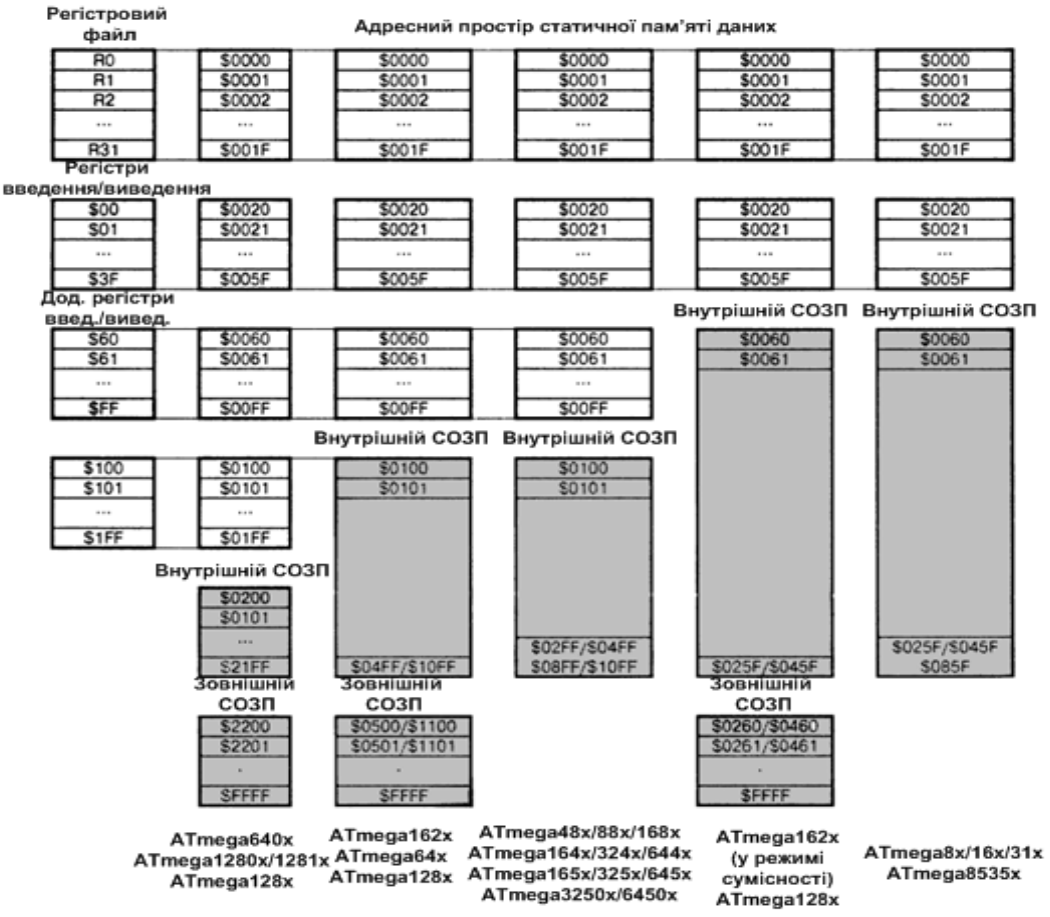


Рисунок 1.2 – Організація статичної пам’яті даних

Регістри загального призначення

32 регістри загального призначення об’єднано в регістровий файл швидкого доступу, структуру якого показано на рисунку 1.3.

В AVR-мікроконтролерах усі РЗП безпосередньо доступні арифметико-логічному пристрою (АЛП), на відміну від 8-бітових 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.

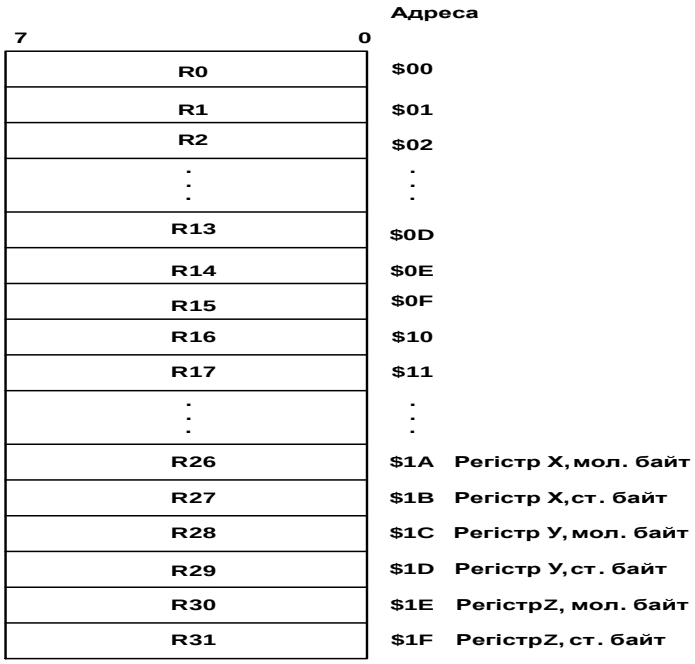


Рисунок 1.3 – Структура регістрів загального призначення

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.

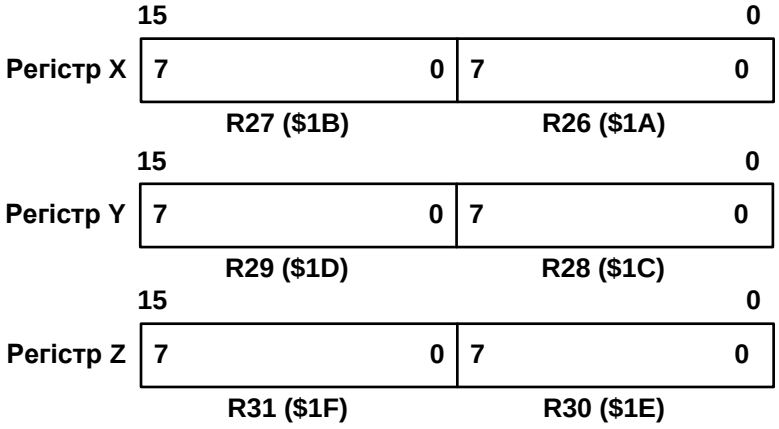


Рисунок 1.4 – Регістри-покажчики X,Y та Z

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, і його глибина визначається тільки розміром вільної області 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, 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; 7].

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; 7].

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.

	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

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. 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 підручника [1].

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 регістрів керування EEPROM-пам’яті наведено у [1; 7].

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; 7]. 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; вийти 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 регістрів 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 енергозалежною.

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, у яких є тільки один такий регістр, наприклад, робочий регістр A/W – акумулятор.

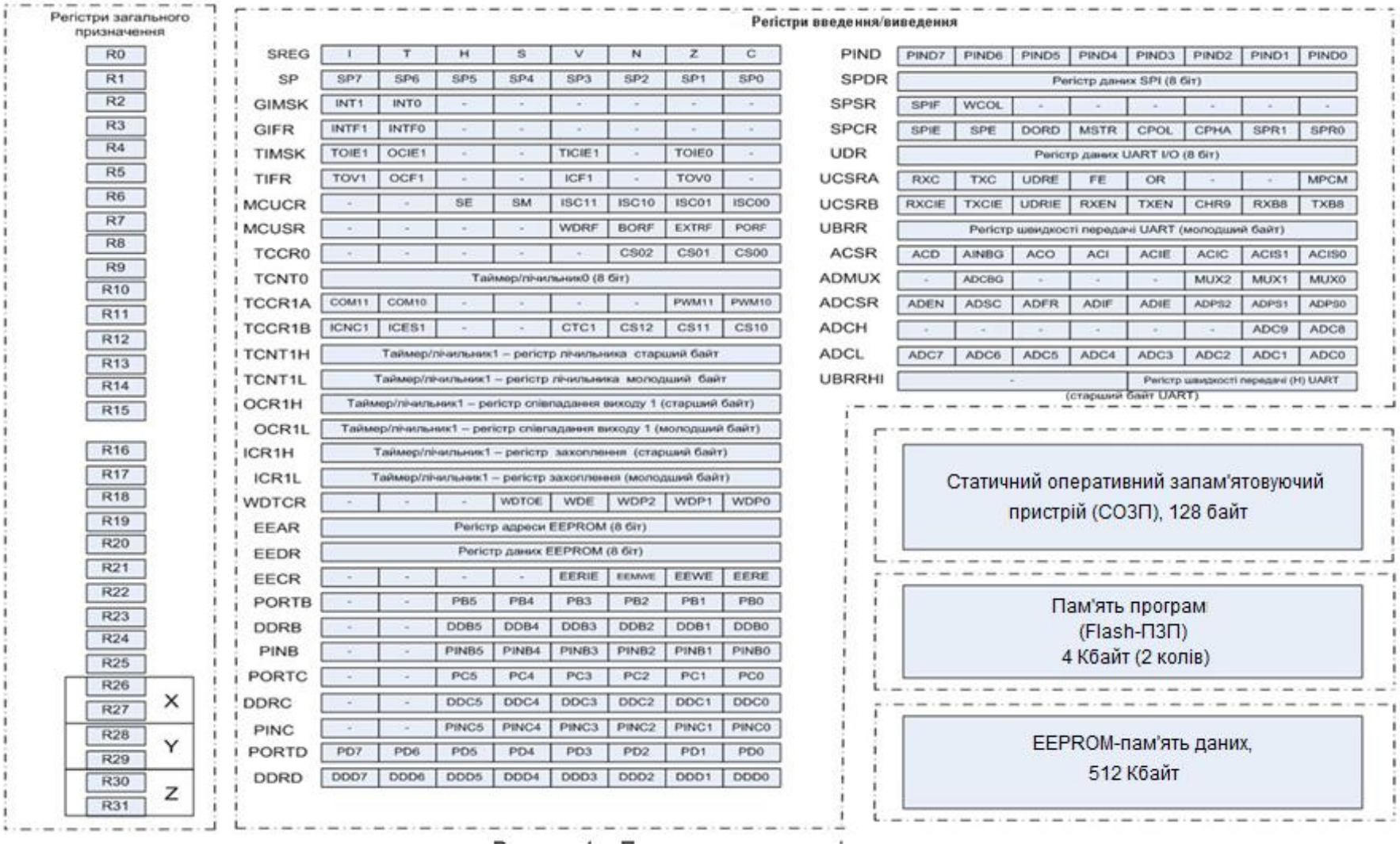


Рисунок 1.7 – Програмна модель мікроконтролера

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

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 1.8 – Структура файлу регістрів загального призначення

На рисунку 1.9 наведено регістри-покажчики X,Y і Z, які використовуються при непрямій адресації операндів.

Рисунок 1.9 – Регістри-покажчики X,Y і Z

1.2.3 Регістри введення/виведення

Регістри введення/виведення (РВВ) розташовуються в так званому просторі введення/виведення розміром 64 байти. 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; 7]. 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 … 3; 7]. 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	$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; 7]. Регістр розташовано за адресою $34 ($54).

1.2.4 Функціонування конвеєра

Для підвищення швидкодії AVR-мікроконтролерів при виконанні програми використовується дворівневий конвеєр (рисунок 1.10).

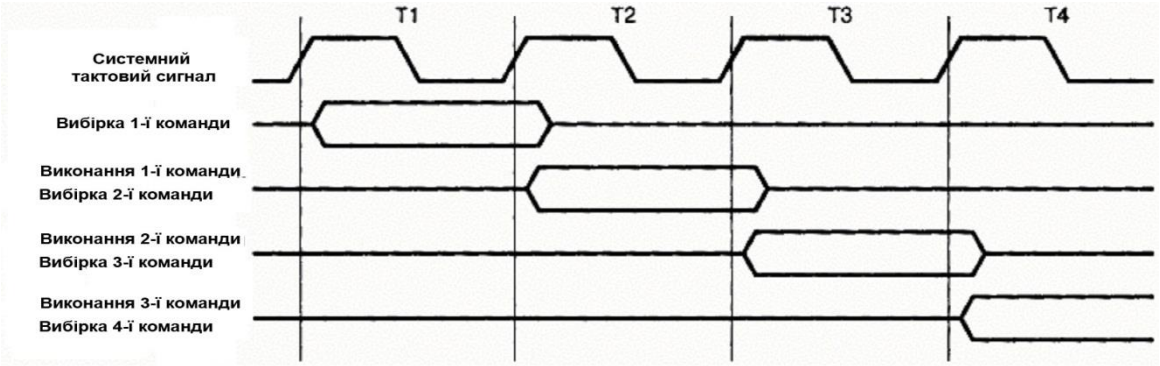


Рисунок 1.10 – Послідовність виконання команд у конвеєрі

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 1.11).

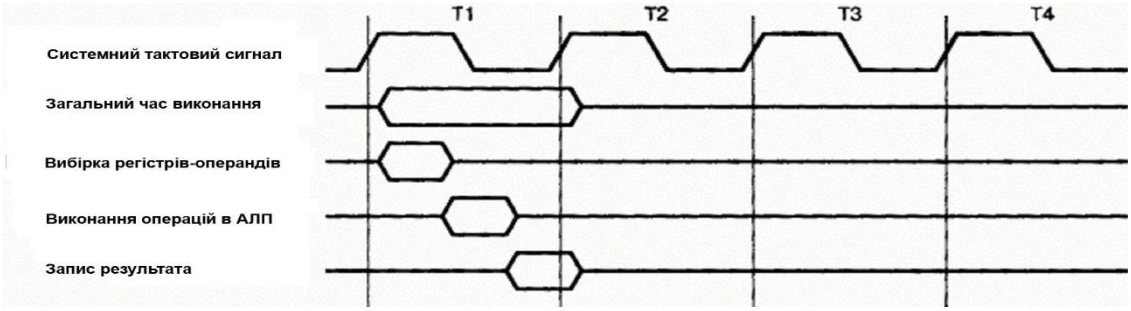


Рисунок 1.11 – Функціонування АЛП

Звернення до внутрішнього СОЗП виконується за два такти [1; 2; 7].

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 контролер 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. У відповідності до її структури 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 1.12).

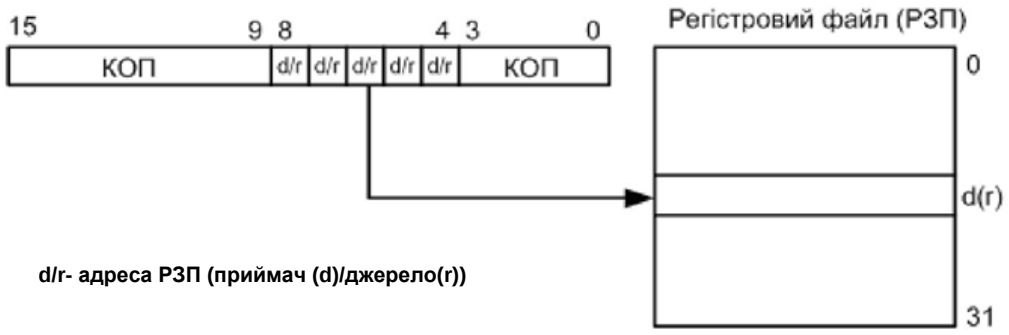


Рисунок 1.12 – Пряма адресація одного регістра загального призначення

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 1.13).

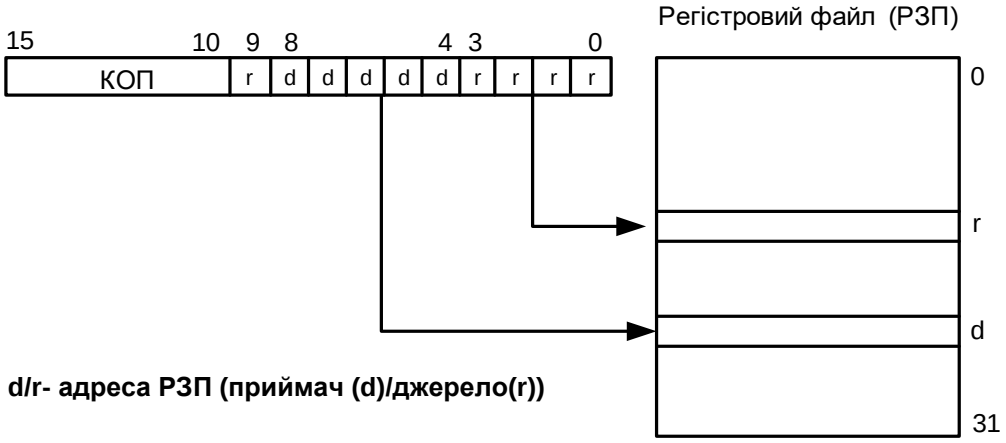


Рисунок 1.13 – Пряма адресація двох регістрів загального призначення

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 1.14).

Рисунок 1.14 – 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 1.15).

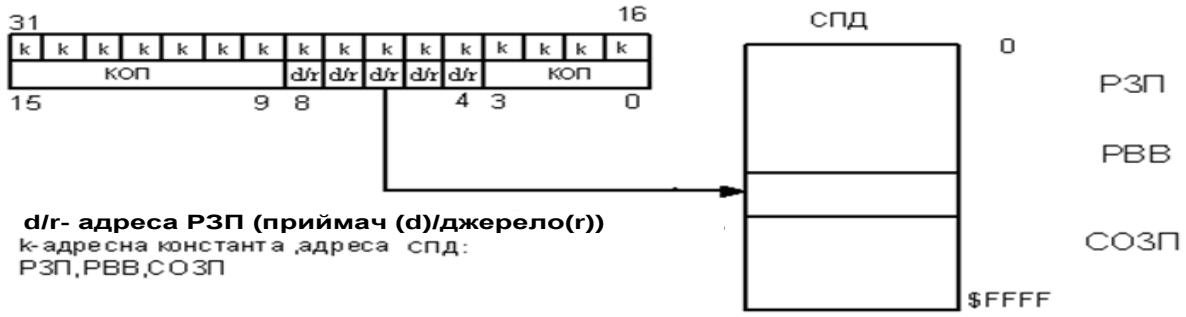


Рисунок 1.15 – Пряма адресація статичної пам’яті даних

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 1.16). Жодних дій із вмістом індексного регістра при цьому не виконується.

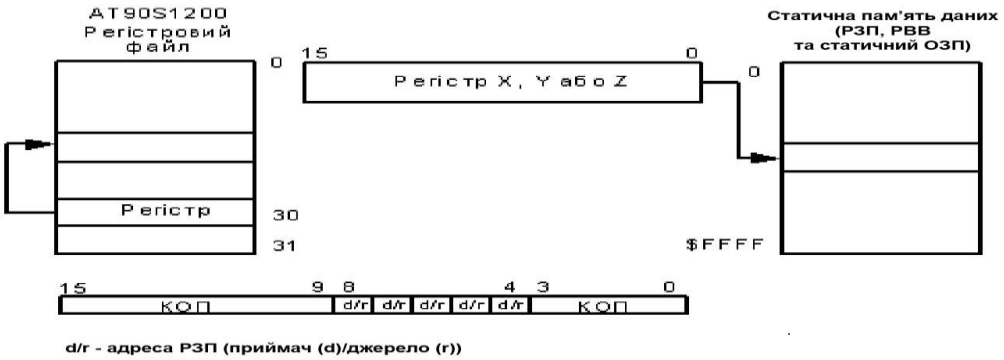


Рисунок 1.16 – Проста непряма адресація

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.

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 1.17).

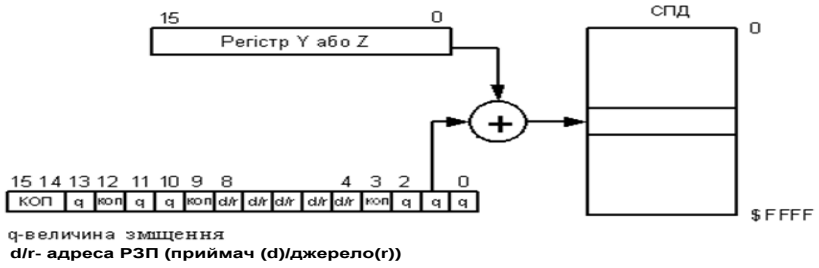


Рисунок 1.17 – Відносна непряма адресація

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 1.18).

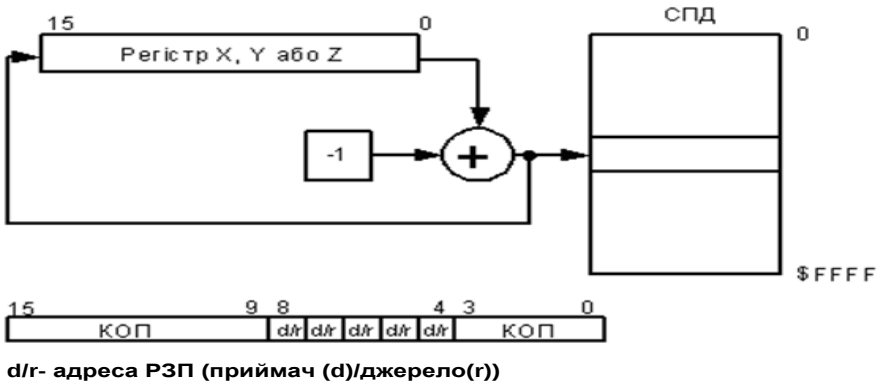


Рисунок 1.18 – Непряма адресація з переддекрементом

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 1.19).

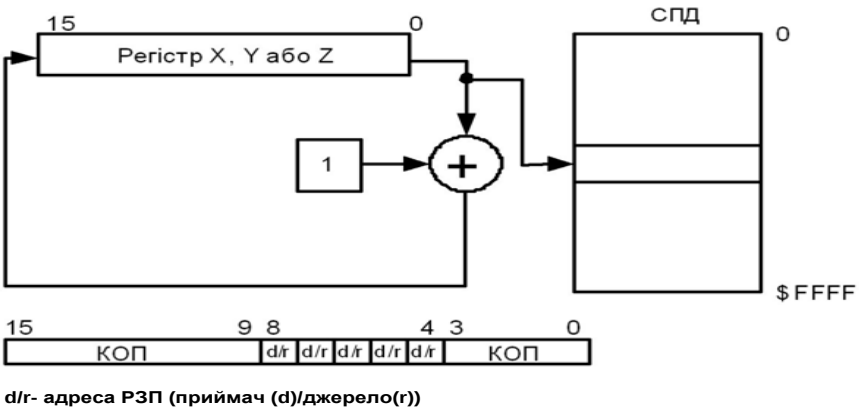


Рисунок 1.19 – Непряма адресація з постінкрементом

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 1.20).

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.

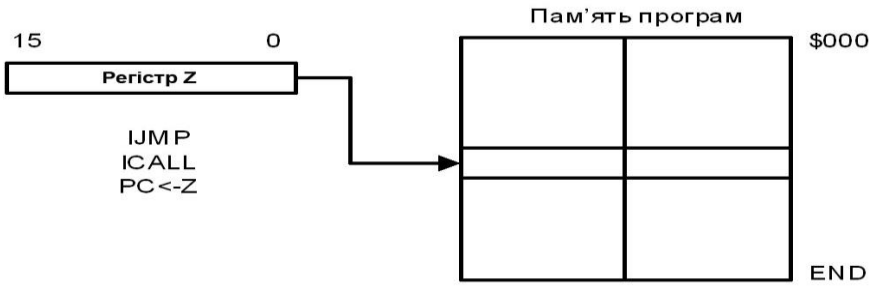


Рисунок 1.20 – Непряма адресація пам’яті програм

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 1.21).

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).

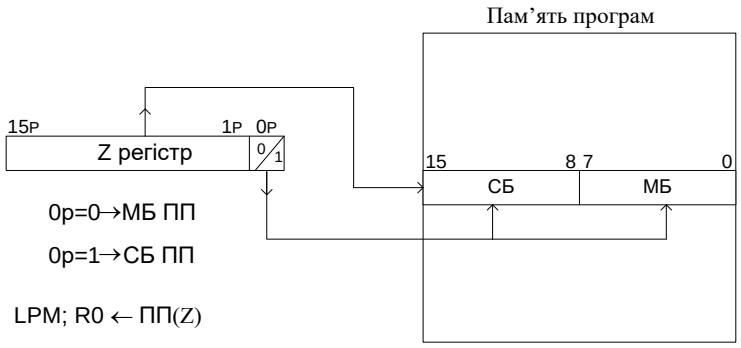


Рисунок 1.21 – Непряма адресація констант в пам’яті програм

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 1.22).

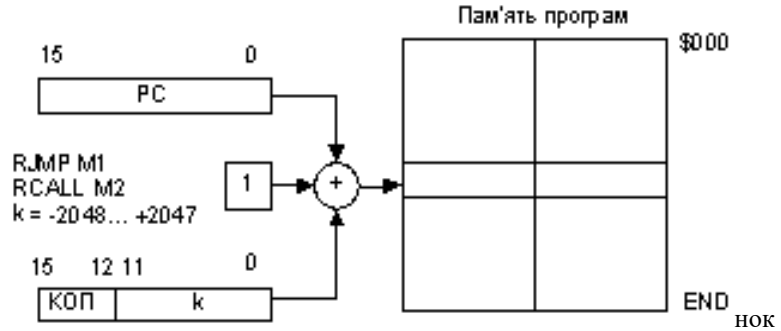


Рисунок 1.22 – Відносна адресація пам’яті програм

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 Загальна характеристика команд AVR-мікроконтролерів

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, 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. 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 (опеації). В залежності від способу адресації операндів, дані можуть знаходитися у регістрах та пам’яті.

1.5 Формати команд

Більшість мікроконтролерів сімейства AVR мають 14 форматів (типів) базового набору команд, наведених на рисунку 1.23. 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 двоопернадні команди, в яких один операнд: K6 = 6 біт (K = 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. Наприклад, команди ADIW Rdl, K6; SBIW Rdl, K6, де Rdl = R24/R26/R28/R30.

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

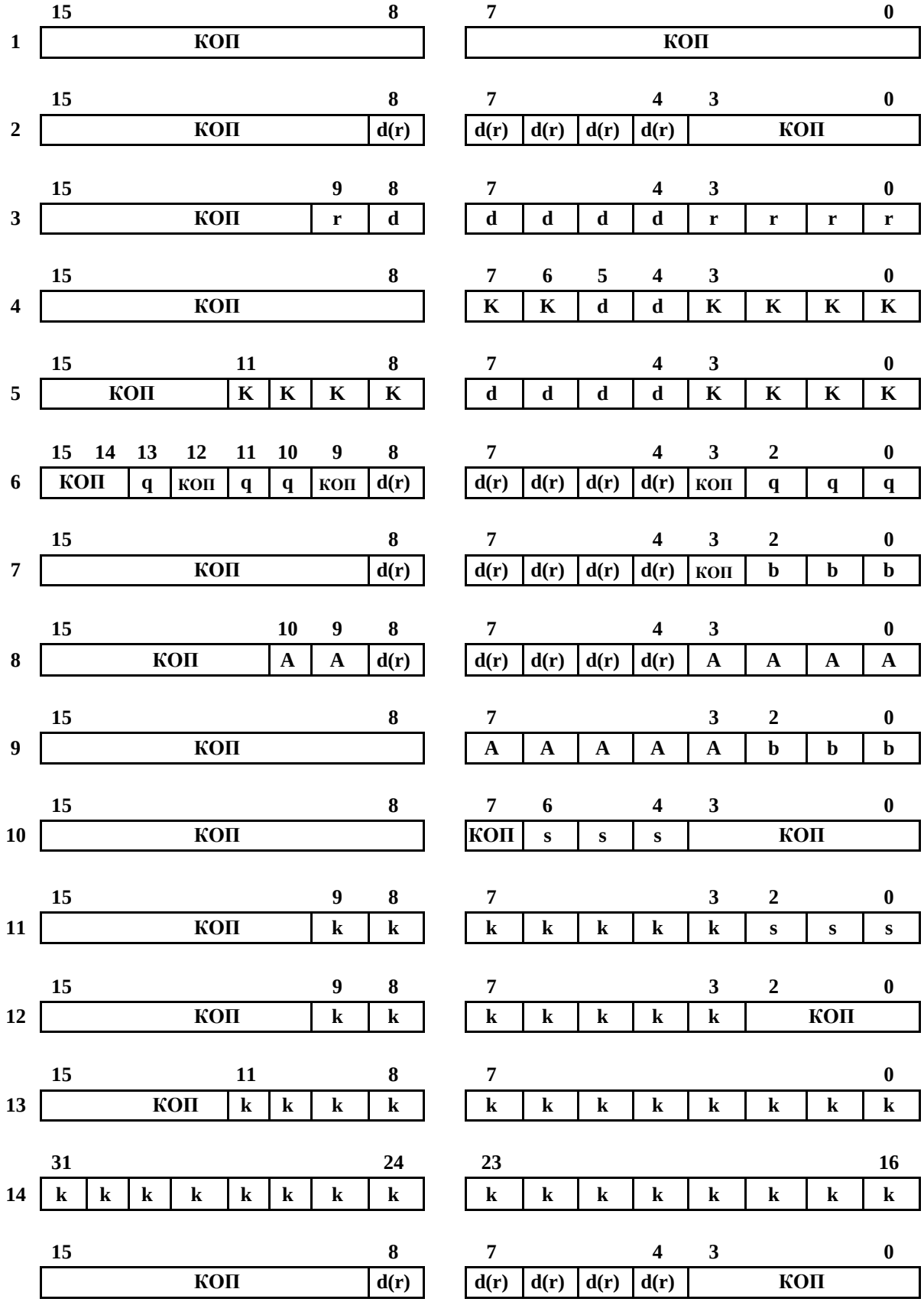


Рисунок 1.23 – Формати базового набору команд

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 у регістрі 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 у 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 в СПД (РЗП, РВВ та СОЗП). Оскільки k = 16 біт, то кількість комірок СПД = 2^{16} = 65536.

1.6 Формати даних

При роботі з мікроконтролером необхідно знати не тільки формати (типи) команд (рисунок 3.18), які керують роботою мікроконтролера, але й формати (типи) даних (операндів), що беруть участь у виконанні тієї або іншої команди (операції) (рисунок 1.24).

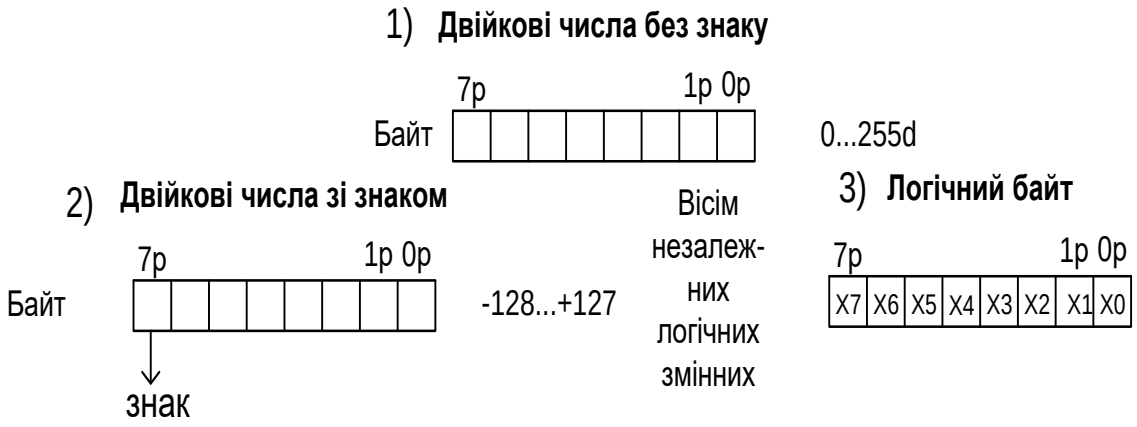


Рисунок 1.24 – Типи (формати) даних мікроконтролера

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

Відповідно до рисунку 1.23 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.

1.8 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.

1.9 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}.

1.10 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; 7].

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 – 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 (додатковий код)	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 С	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 С	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 пост/інкрементом	(Z) ← Rr, Z ← Z + 1	-	2	2
24	ST	-Z, Rr	Непряме збереження з перед/декрементом	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 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; 7]. 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; 2; 7].

У 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 змінюєтьсь 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).

У 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.

У 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

У 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 з нього деякого зміщення. Оскільки під значення зміщення 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.

Усі 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.

Таблиця 1.4 – Зведена таблиця команд умовного переходу

Перевірка	**Логіч. умова**	**Команда**	**Зворотна перевірка**	**Логіч. 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 в зворотному порядку, тобто замість CP Rd, Rr –> CP Rr, Rd.

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).

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. У 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; 7].

1.11 Нові команди AVR-мікроконтролерів

Архітектура AVR-мікроконтролерів постійно оновлюється. 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; 7; 19].

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 ≤ d ≤ 23; 16 ≤ r ≤ 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: $0 \le d \le 31$, 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; 7; 19].

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 ($0 \le d \le 31$) – 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).

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: $0 \leq d \leq 31$.

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: $0 \leq d \leq 31$.

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).

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).

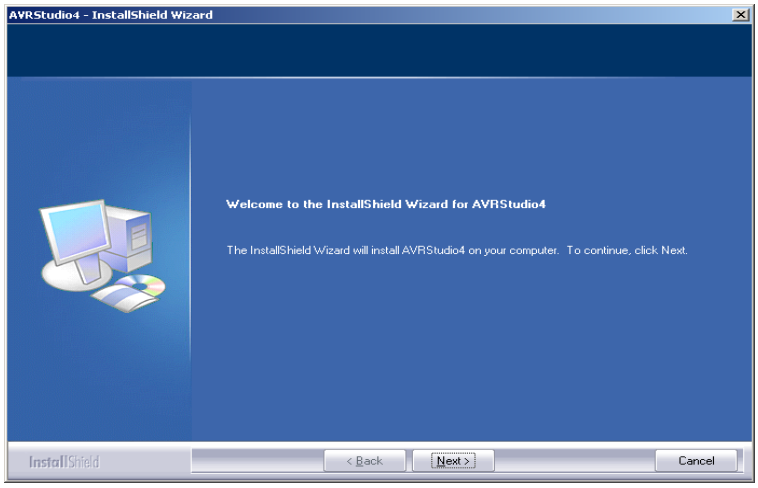


Рисунок 2.1 – Вікно привітання

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

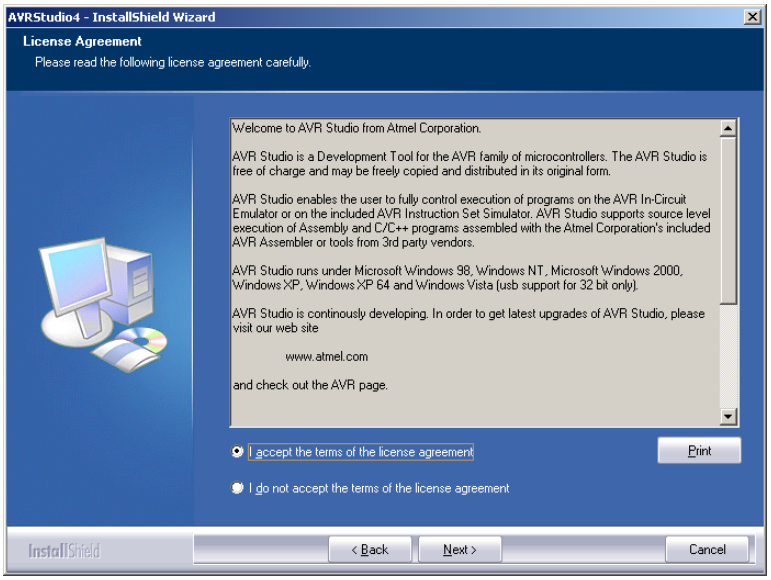


Рисунок 2.2 – Умови ліцензійної угоди

Після цього оберіть місце встановлення (рисунок 2.3).

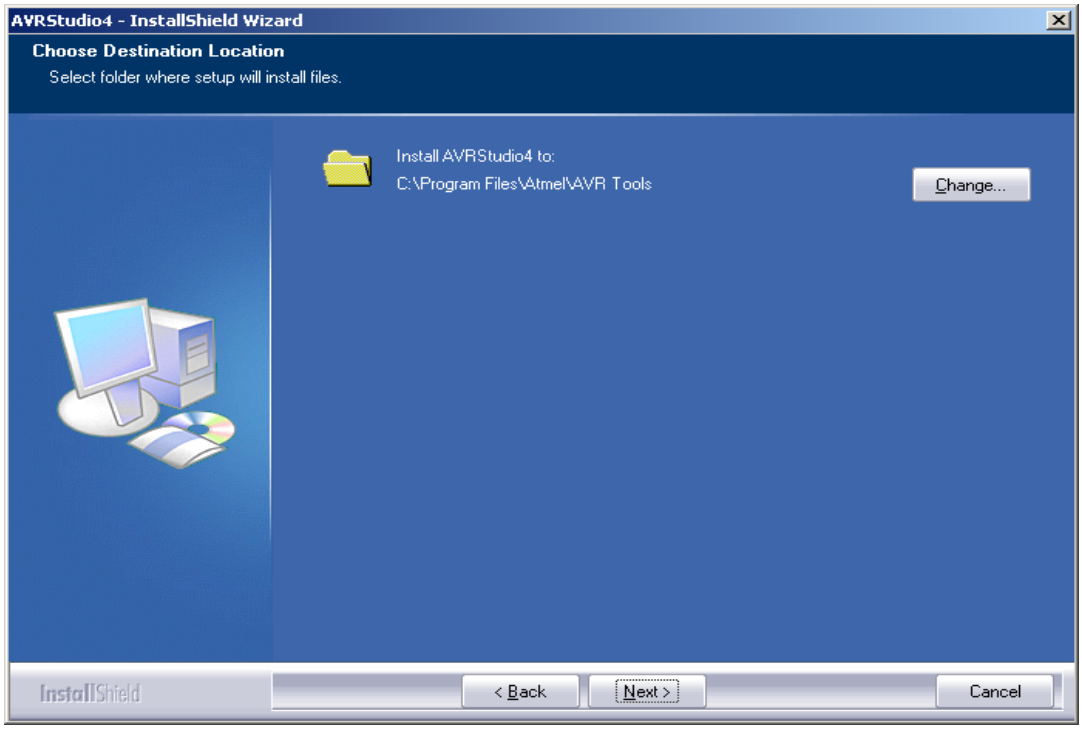


Рисунок 2.3 – Вибір місця інсталяції

Далі оберіть додаткові компоненти для встановлення (рисунок 2.4).

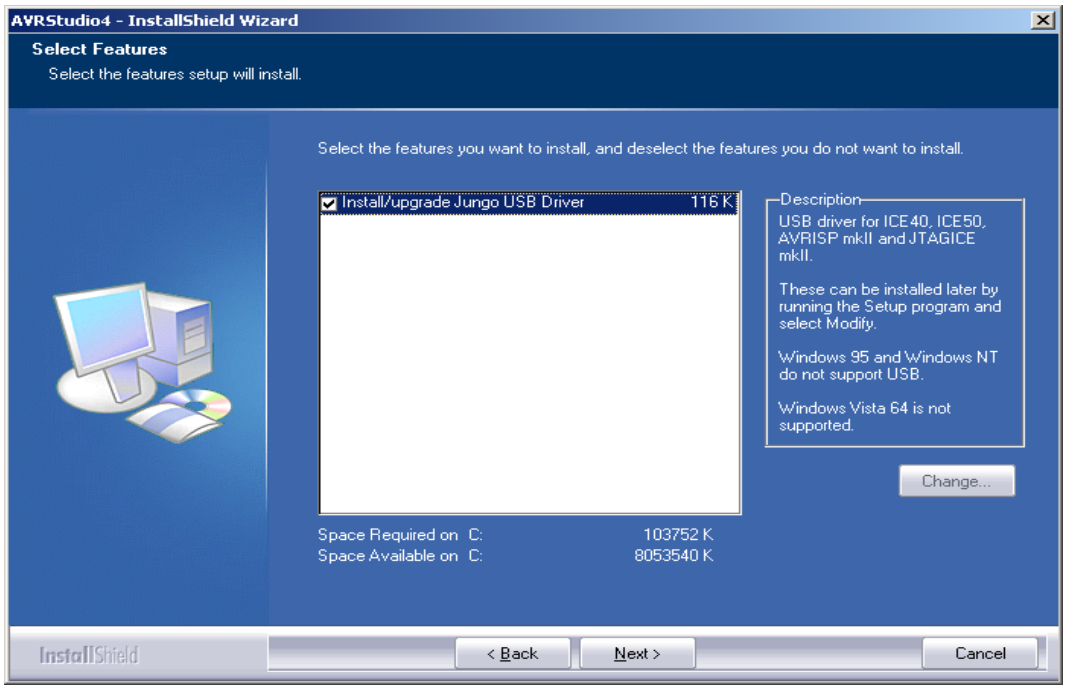


Рисунок 2.4 – Вибір додаткових компонентів

Натиснувши кнопку Install розпочніть установку (рисунок 2.5).

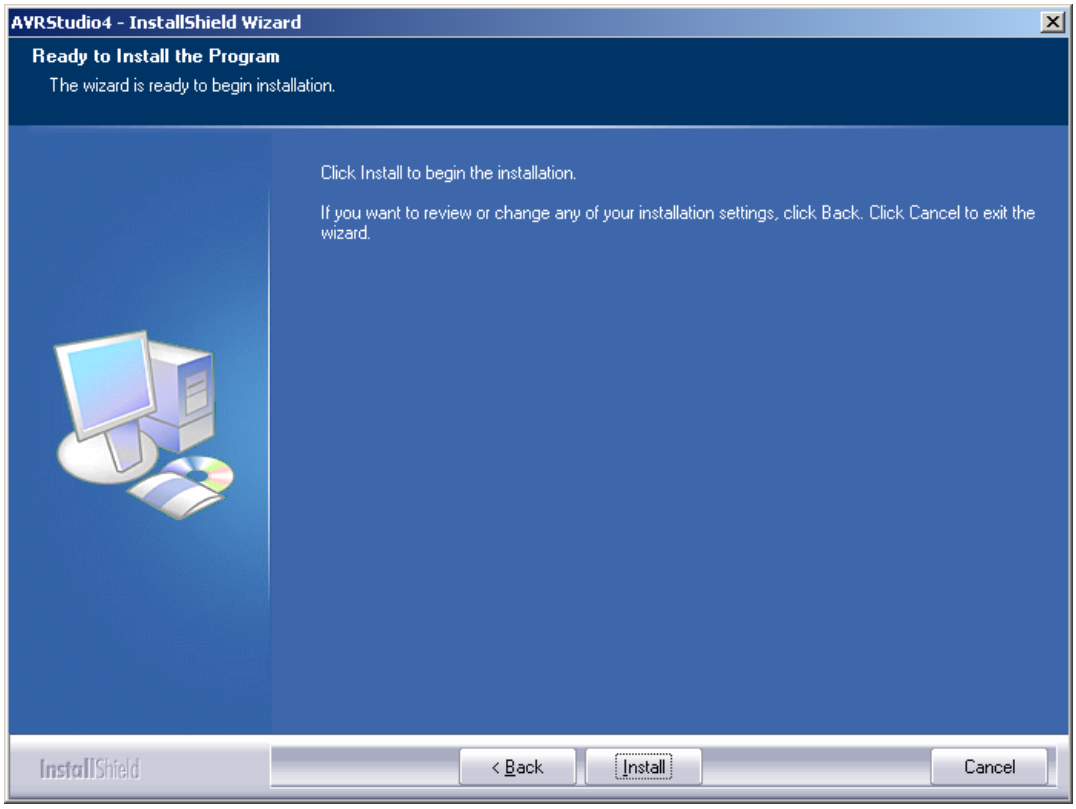


Рисунок 2.5 – Початок інсталяції

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

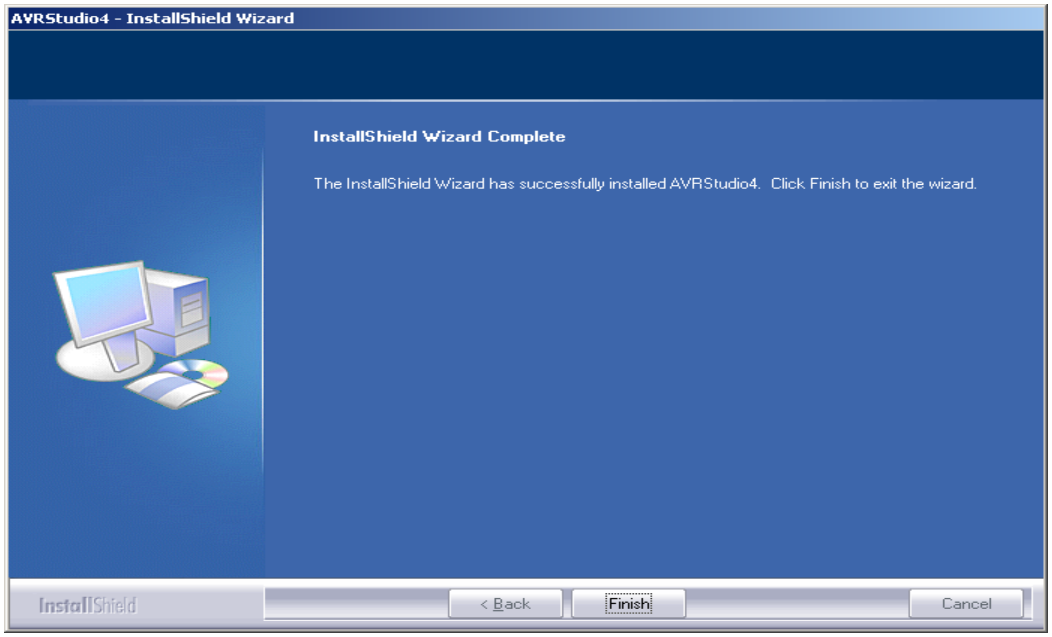


Рисунок 2.6 – Успішне закінчення інсталяції

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.

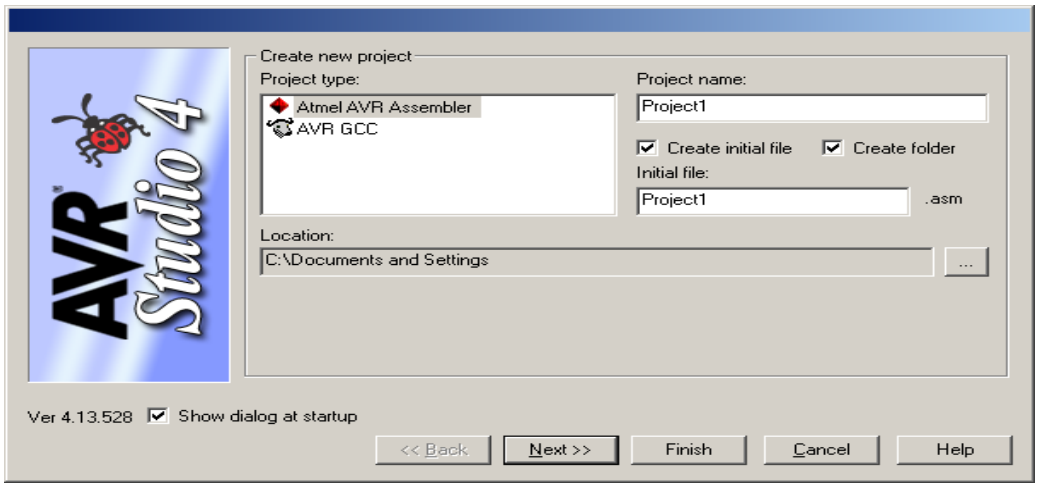


Рисунок 2.7 – Створення проекту

Натисніть Next.

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

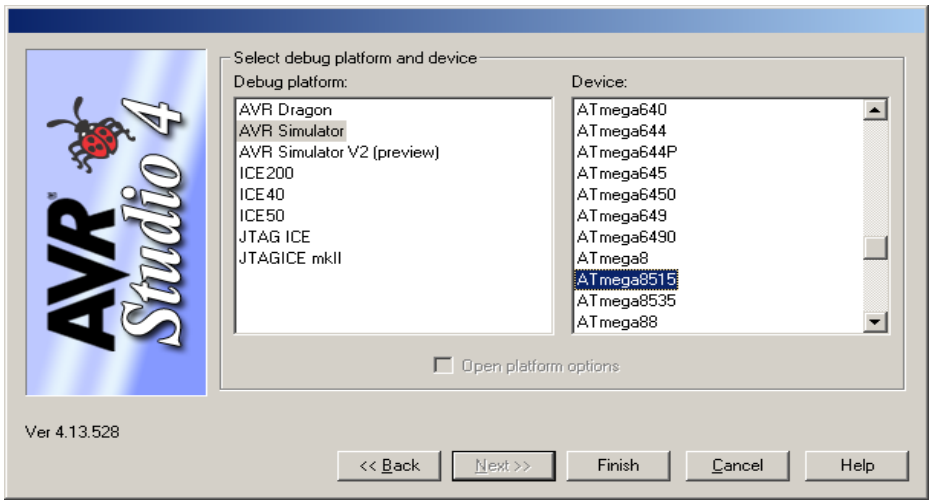


Рисунок 2.8 – Вибір платформи та пристрою

Натисніть Finish.

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

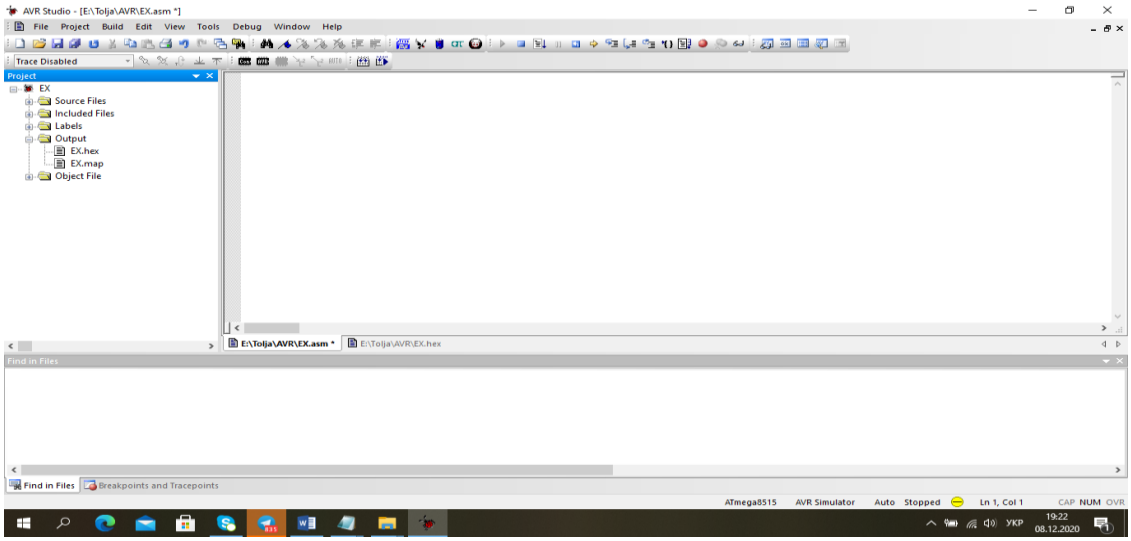


Рисунок 2.9 – Вікно для запису нового проекту

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

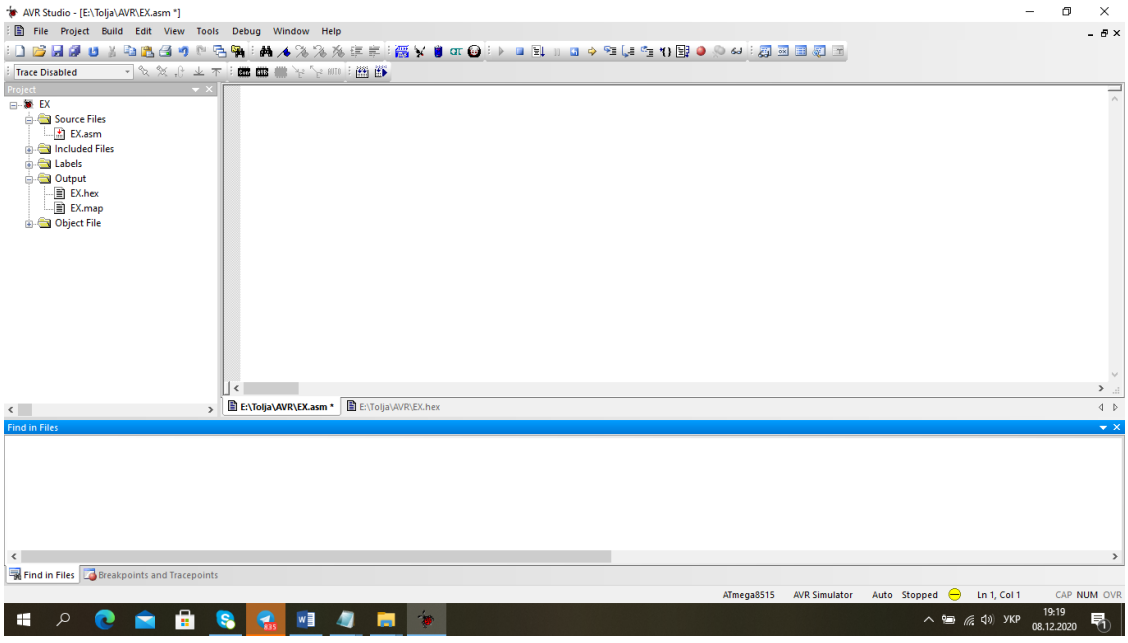


Рисунок 2.10 – Вікно пусте для запису файла *.asm

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

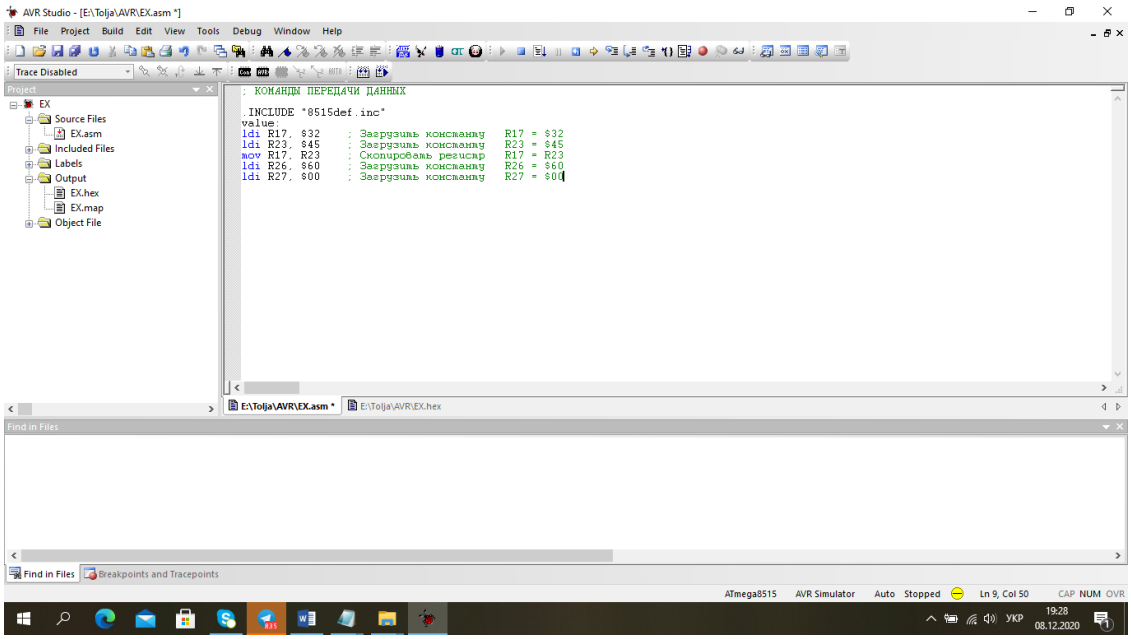


Рисунок 2.11 – Вікно с текстом файла *.asm

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.

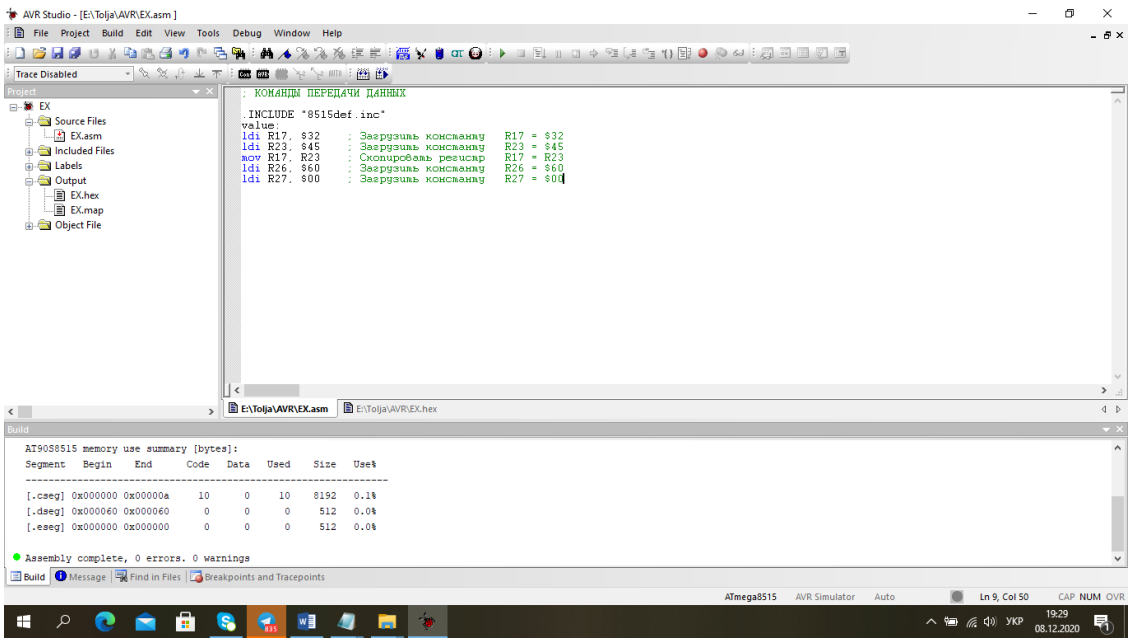


Рисунок 2.13 – Вікно повідомлення про завершення асемблерування

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).

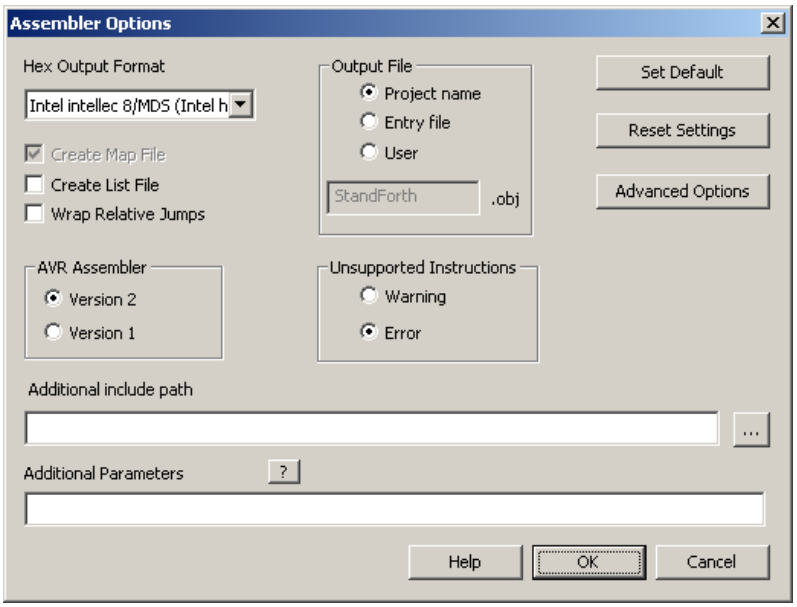


Рисунок 2.14 – Вікно параметрів асемблерування

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

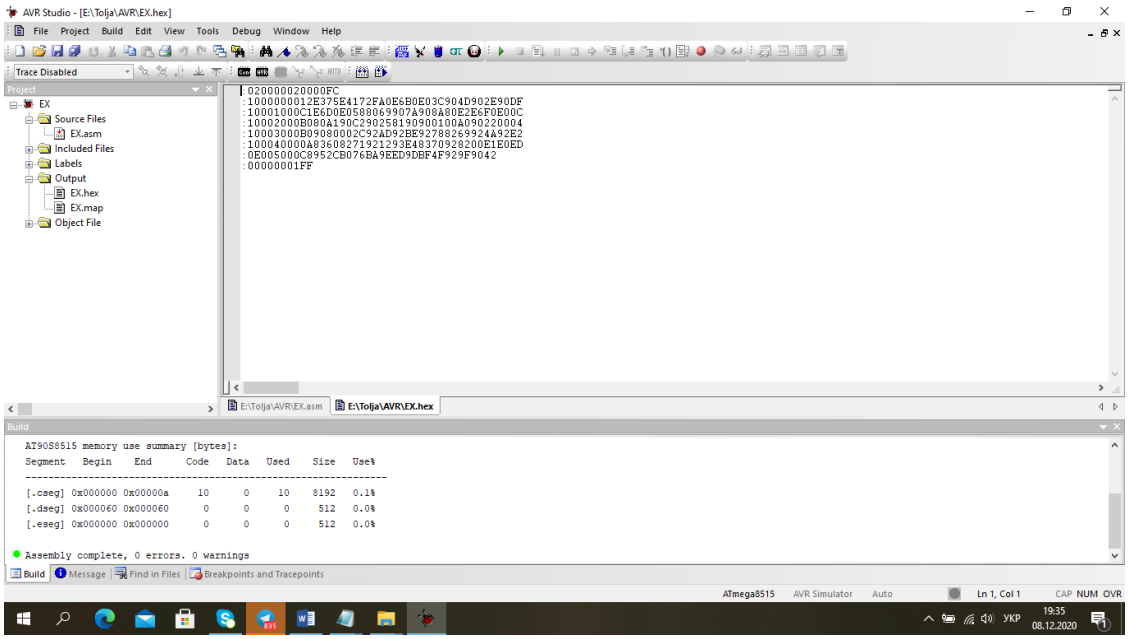


Рисунок 2.15 – Вікно з створеним файлом *.hex

2.5 Налагоджування 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.

2.5.1 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).

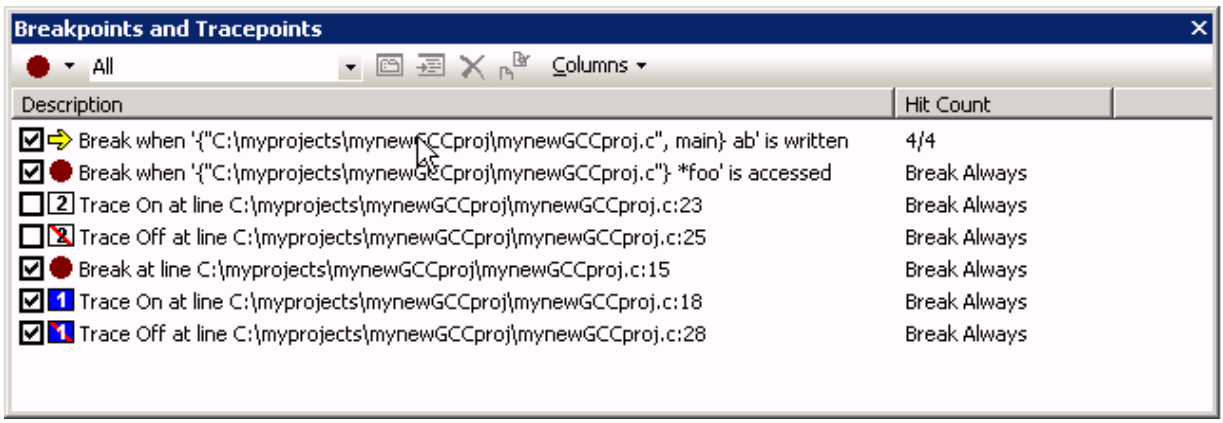


Рисунок 2.18 – Вікно перегляду контрольних точок

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.

2.5.2 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 та регістри від R0 до R31 (рисунок 2.19).

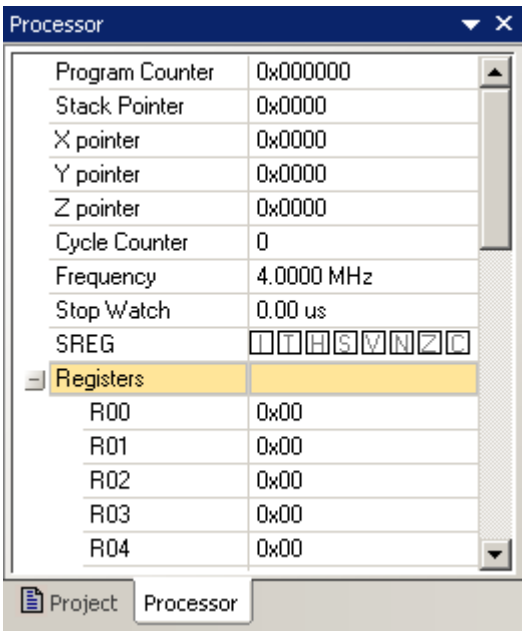


Рисунок 2.19 – Вікно перегляду стану процесора та регістрів

2.5.3 Перегляд стану пам’яті

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

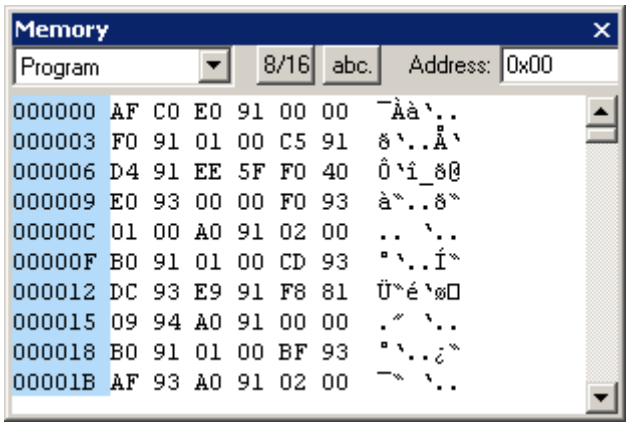


Рисунок 2.20 – Вікно перегляду стану пам’яті

Вікно для редагування комірки пам’яті викликається подвійним кліком.

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

3 МОДЕЛЮВАННЯ ОКРЕМИХ МОДУЛІВ МІКРОКОНТРОЛЕРІВ СІМ'Ї AVR У ПАКЕТІ PROTEUS 8.6

Вступ

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, просто-напросто спалювали насилу придбані дорогі радіодеталі. І скоріше за все більшість, обпікшись на перших невдачах, закидали заняття радіоелектронікою назавжди.

У 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 виготовлення 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.

3.1 Опис налагоджувача PROTEUS 8.6

3.1.1 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).

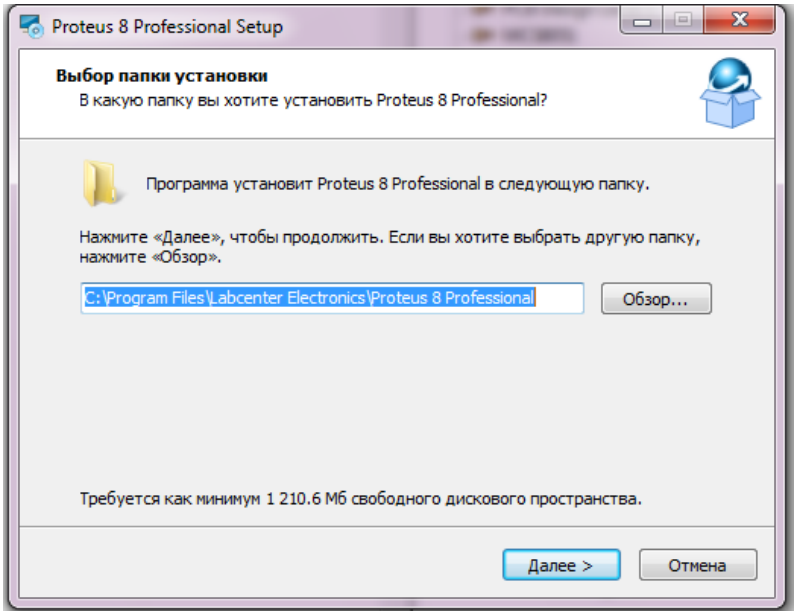


Рисунок 3.1 – Вікно привітання та вибір папки для програми

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).

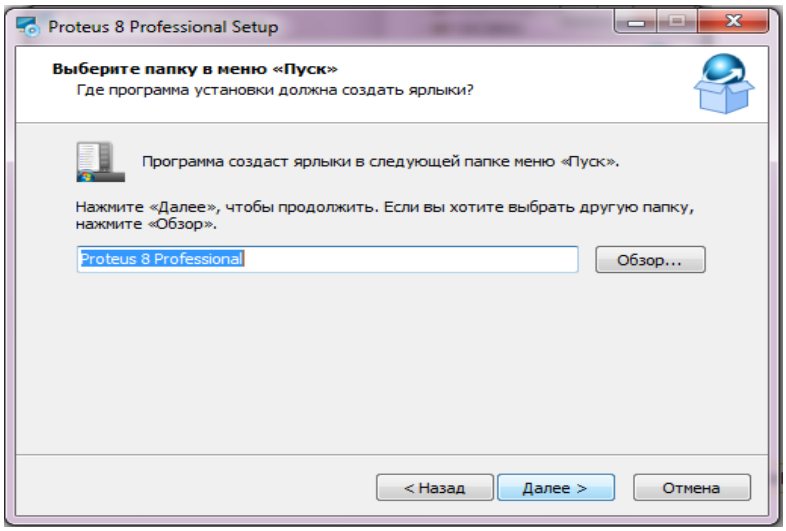


Рисунок 3.2 – Вибір локації у меню «Пуск»

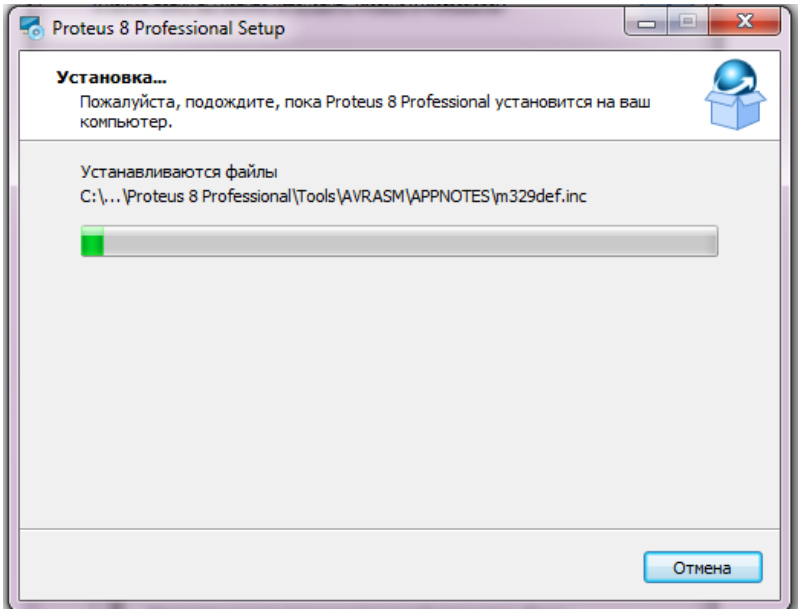


Рисунок 3.3 – Екран процесу встановлення

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.

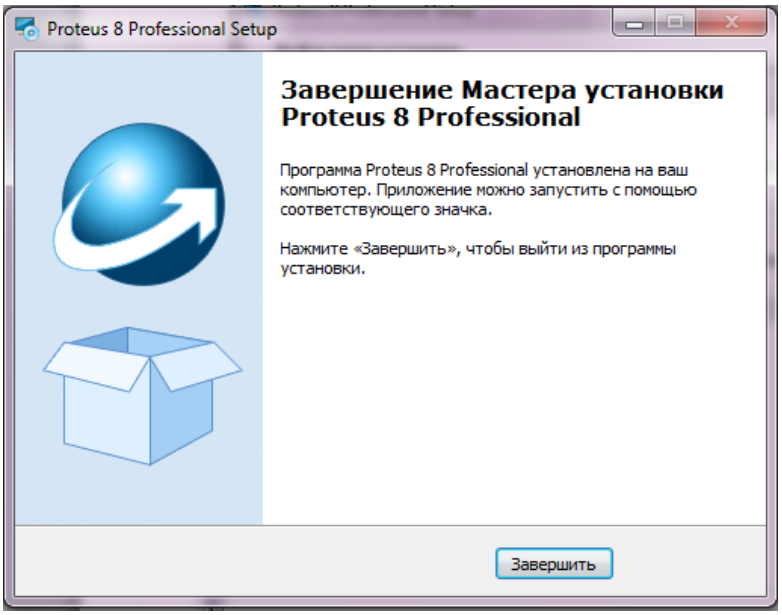


Рисунок 3.4 – Кінець інсталяції програми

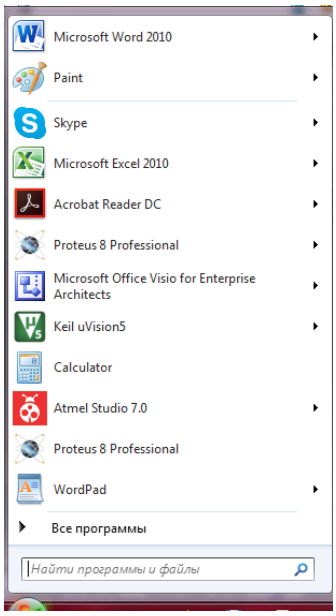


Рисунок 3.5 – Запуск програми за кнопкою “ПУСК”

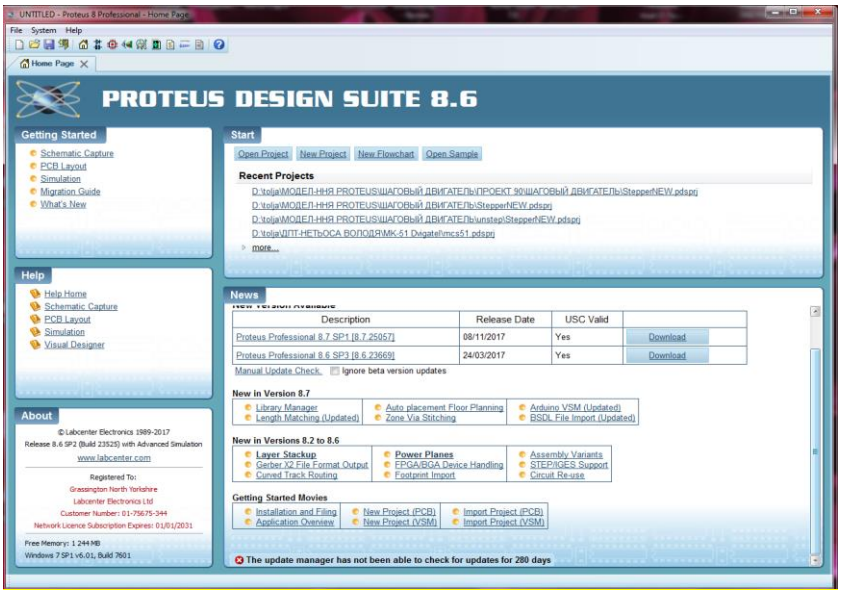


Рисунок 3.6 – Головний екран програми

3.1.2 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.

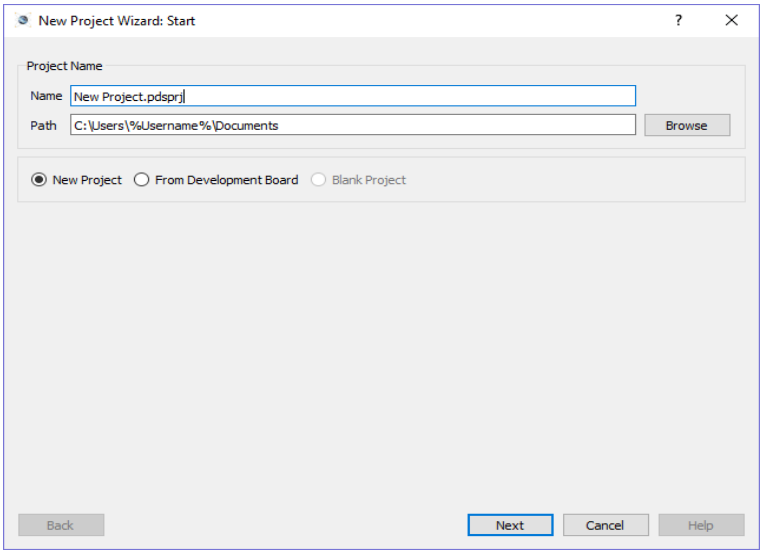


Рисунок 3.7 – Створення нового проекту у Proteus

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

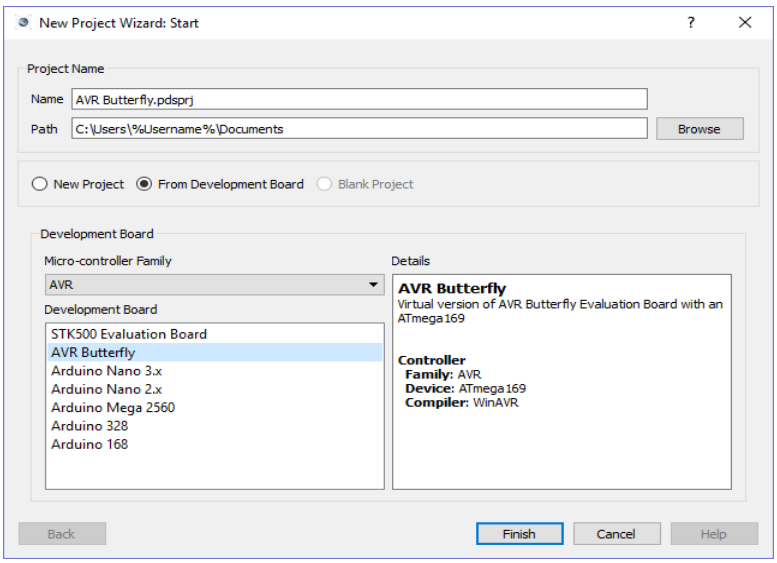


Рисунок 3.8 – Заздалегідь налаштовані проекти

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

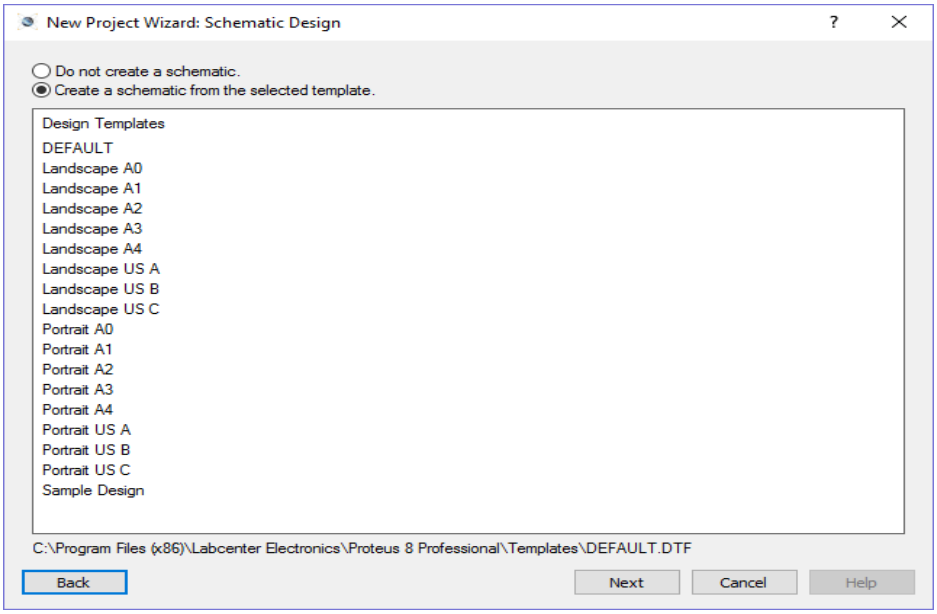


Рисунок 3.9 – Налаштування розмірів полотна схеми

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

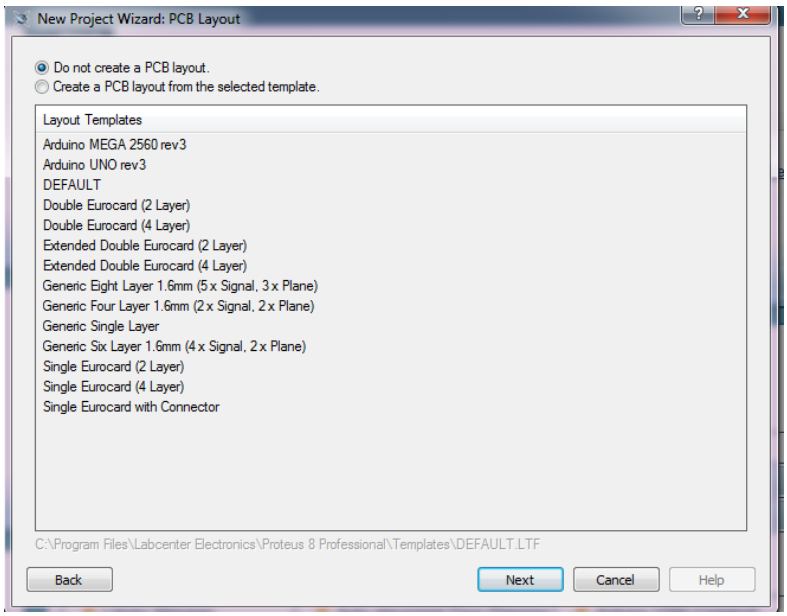


Рисунок 3.10 – Вибір схеми

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

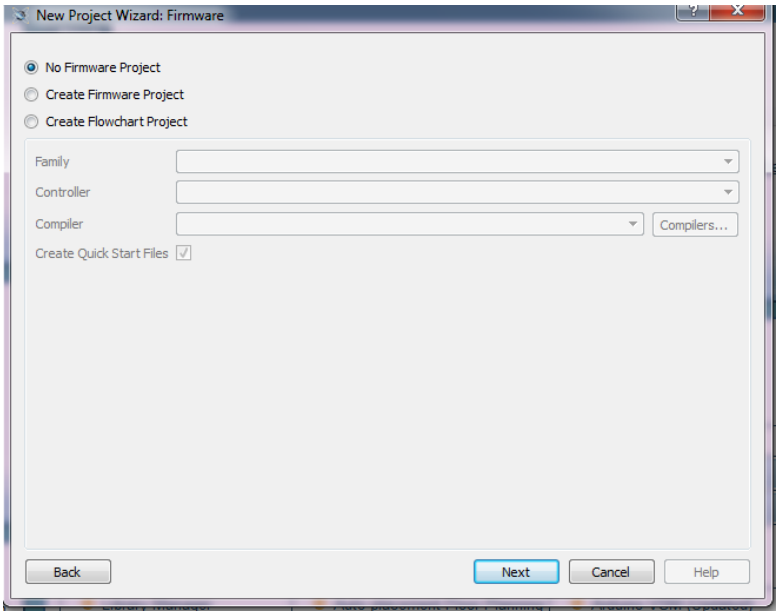


Рисунок 3.11 – Вибір налаштувань проекту прошивки

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

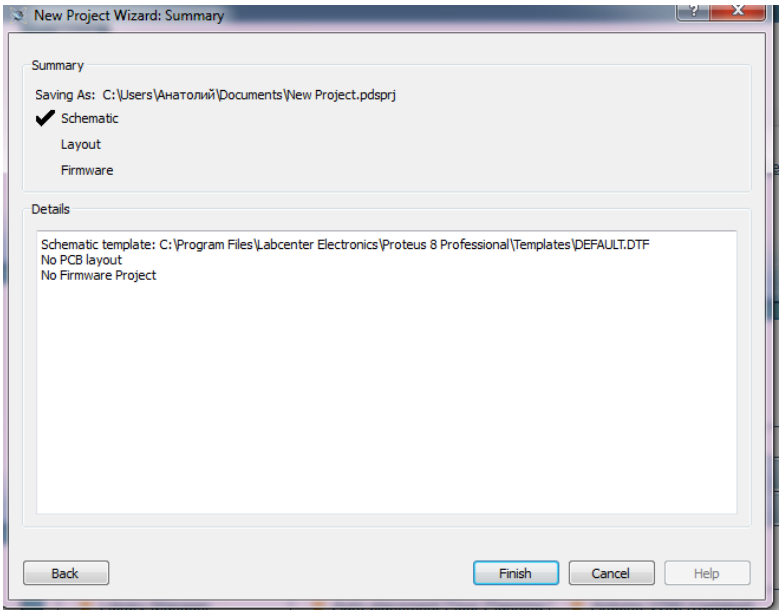


Рисунок 3.12 – Вікно з підсумками процесу налаштування

Врешті відкриється головний екран середовища розробки ISIS (рисунок 3.13)

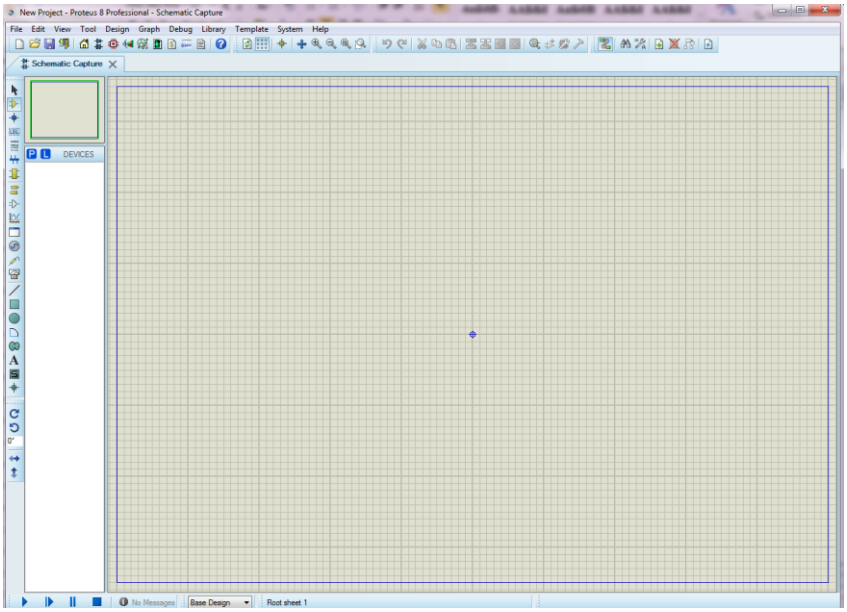


Рисунок 3.13 – Вигляд середовища ISIS після налаштувань

3.1.3 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.

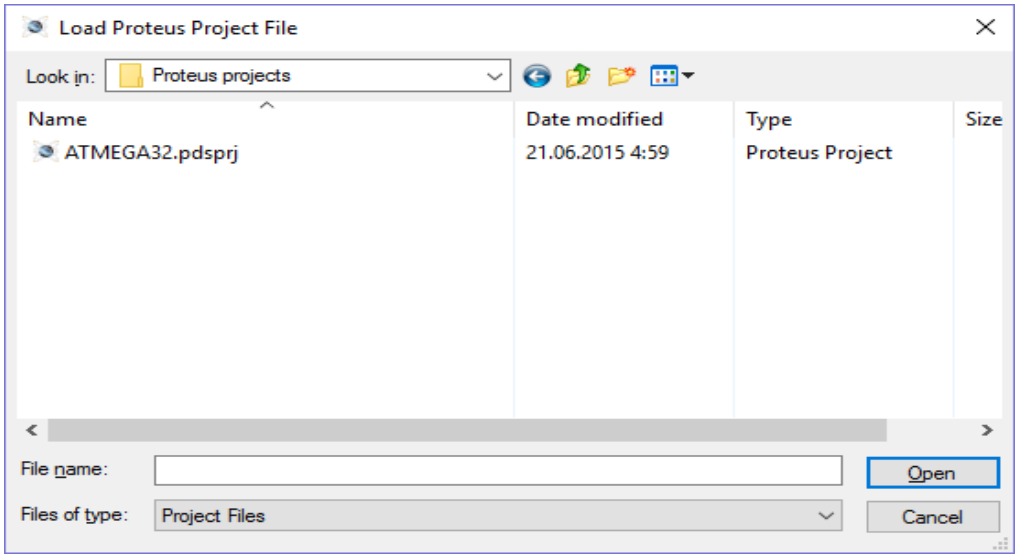


Рисунок 3.14 – Вікно вибору файлу

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

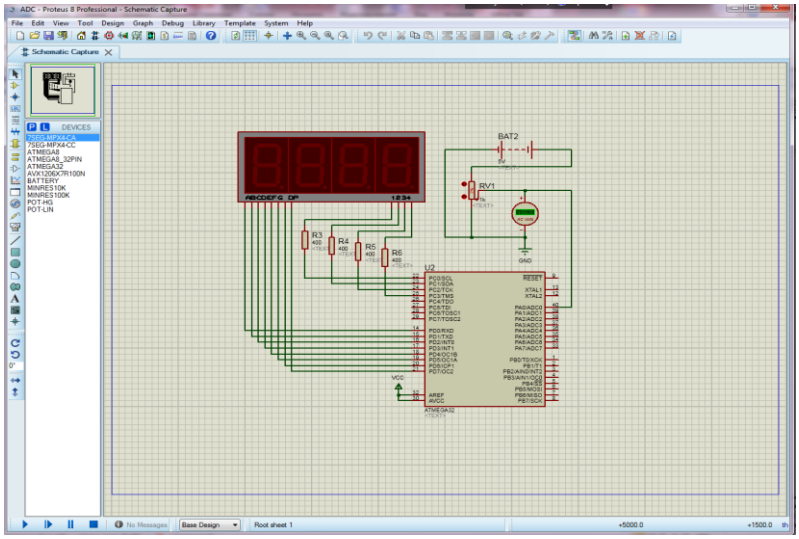


Рисунок 3.15 – Приклад відкритого проекту

3.1.4 Знайомство з інтерфейсом середовища ISIS

Це питання розглянемо на прикладі завантаженої моделі принципової схеми цифрового вольтметру, яку ми далі побудуємо самостійно (рисунок 3.16).

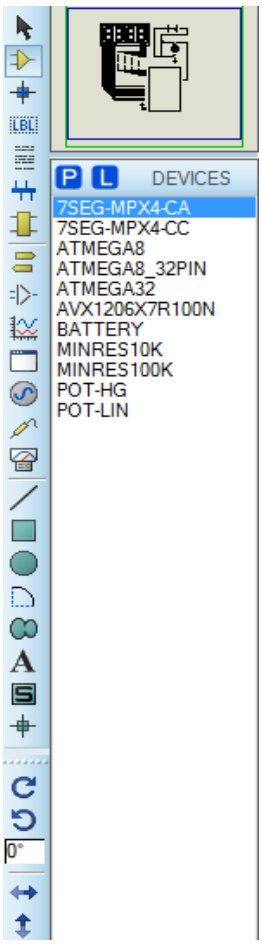


Рисунок 3.16 – Вид інтерфейсу бокового меню

На рисунку 3.16 бачимо список компонентів, що були використані для побудови даної схеми. Стрілки повороту та стрілки напрямів у меню посередині (рисунок 3.16) 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.

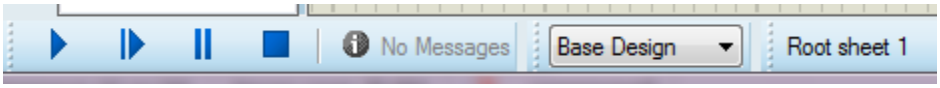


Рисунок 3.17 – Меню запуску моделювання

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.

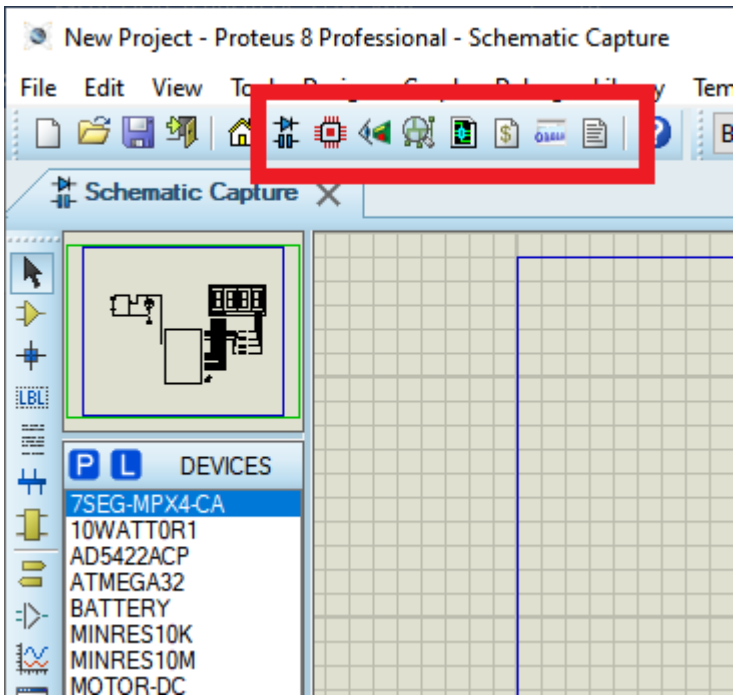


Рисунок 3.18 – Меню вибору вигляду схеми

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

3.1.5 Додавання моделей

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

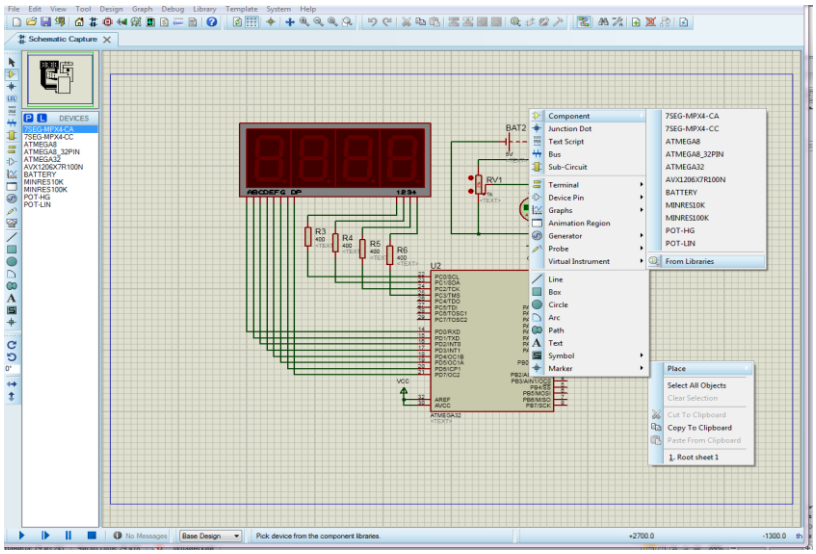


Рисунок 3.19 – Додавання моделі

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).

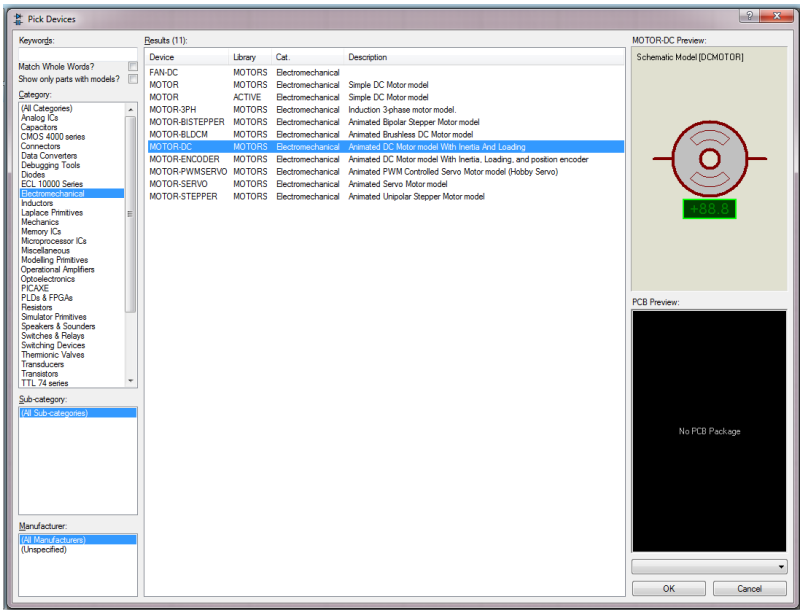


Рисунок 3.20 – Вікно вибору потрібного елемента

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.

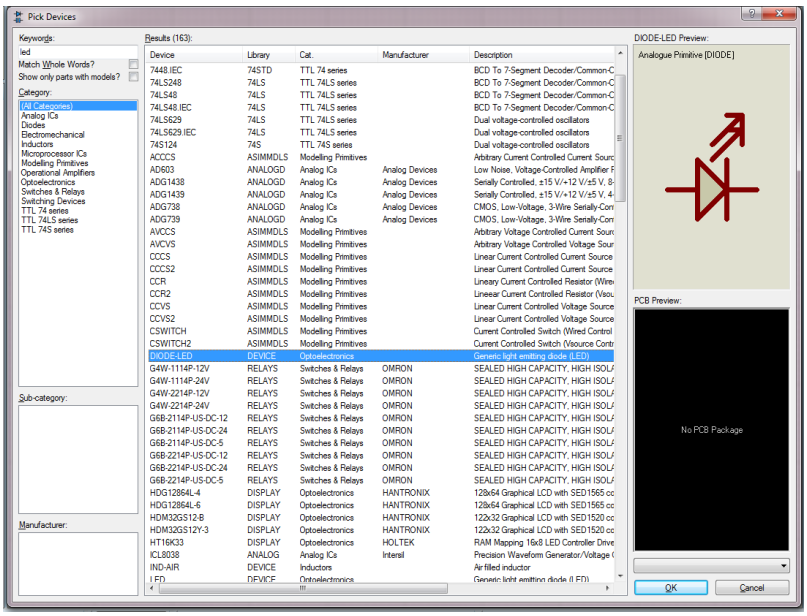


Рисунок 3.21 – Вибір потрібного елемента зі списку

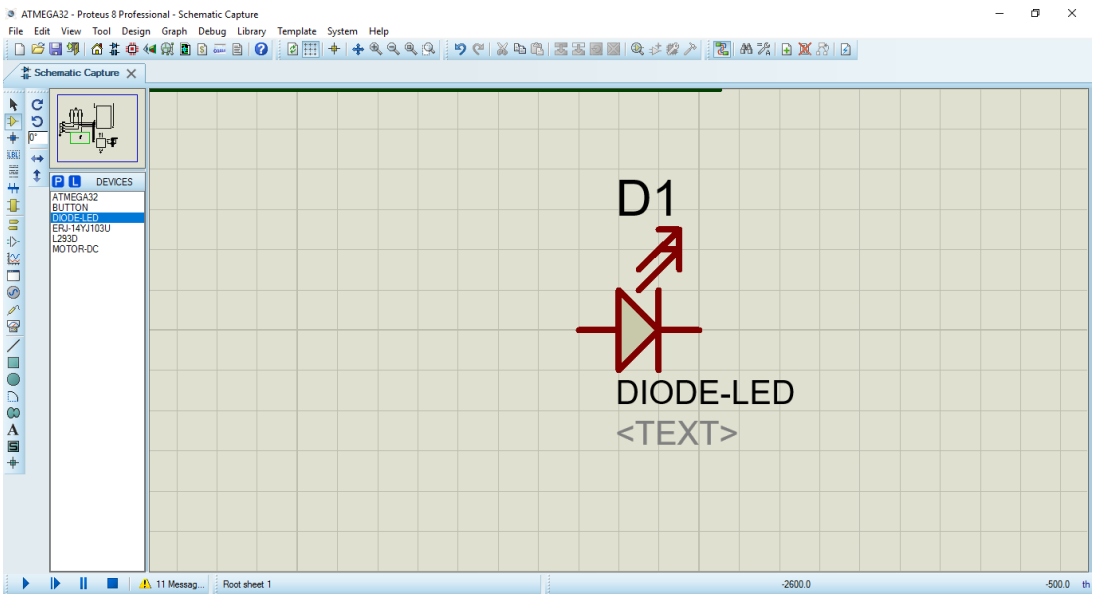


Рисунок 3.22 – Розміщення елемента на полотні

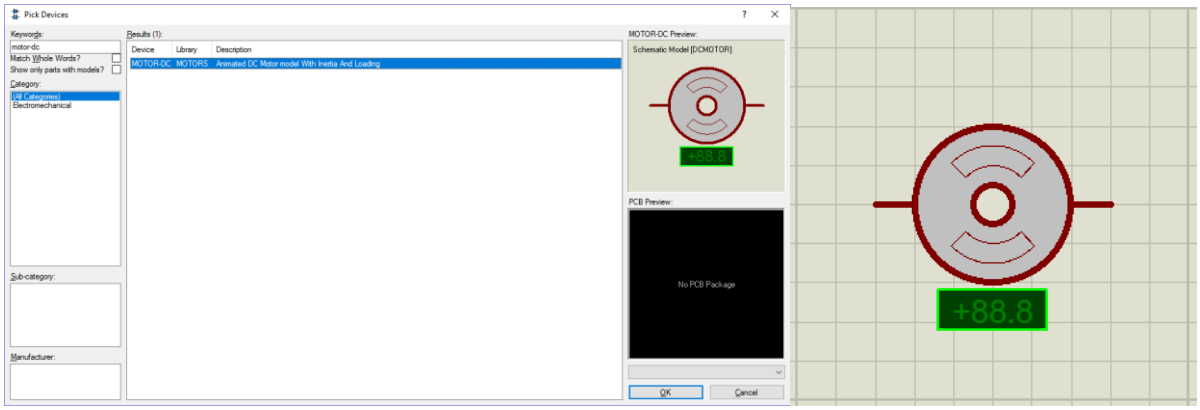


Рисунок 3.23 – Розміщення двигуна постійного струму

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.

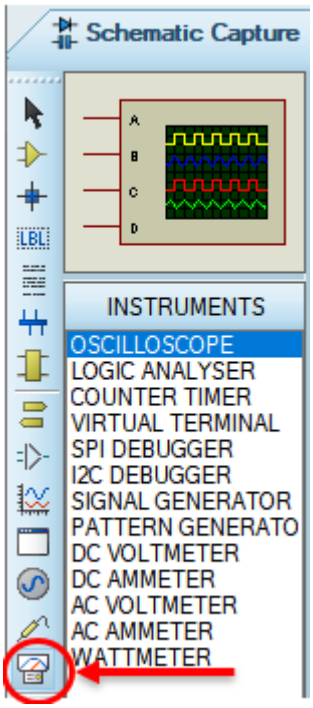


Рисунок 3.24 – Меню вибору інструментів

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).

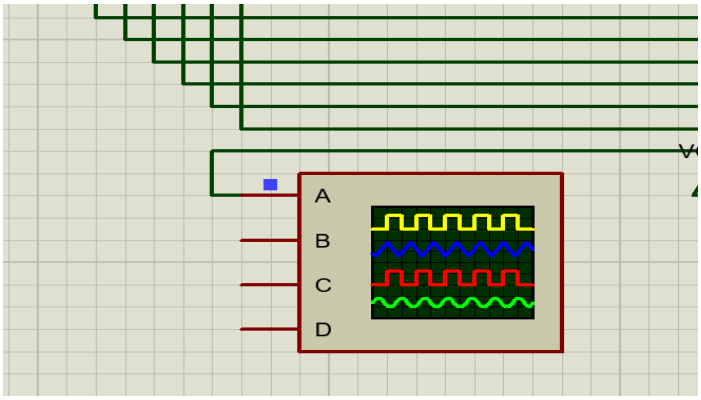


Рисунок 3.25 – Осцилограф на схемі

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.

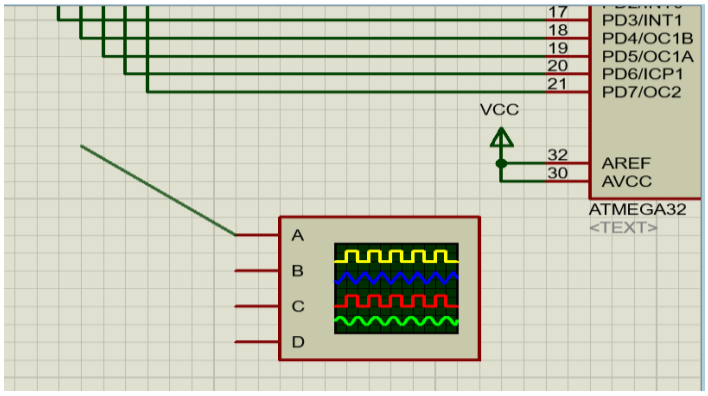


Рисунок 3.26 – Приклад приєднання осцилографа

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).

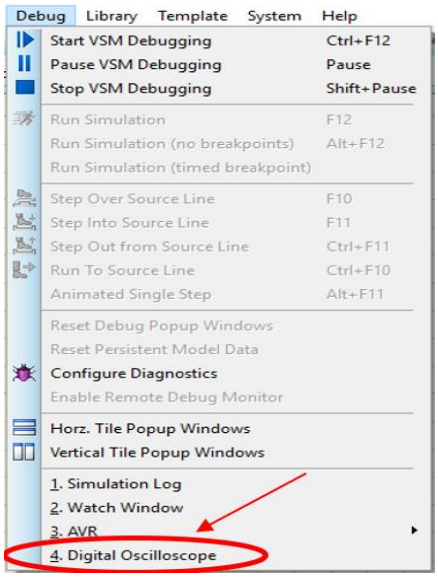


Рисунок 3.28 – Список компонентів у системі

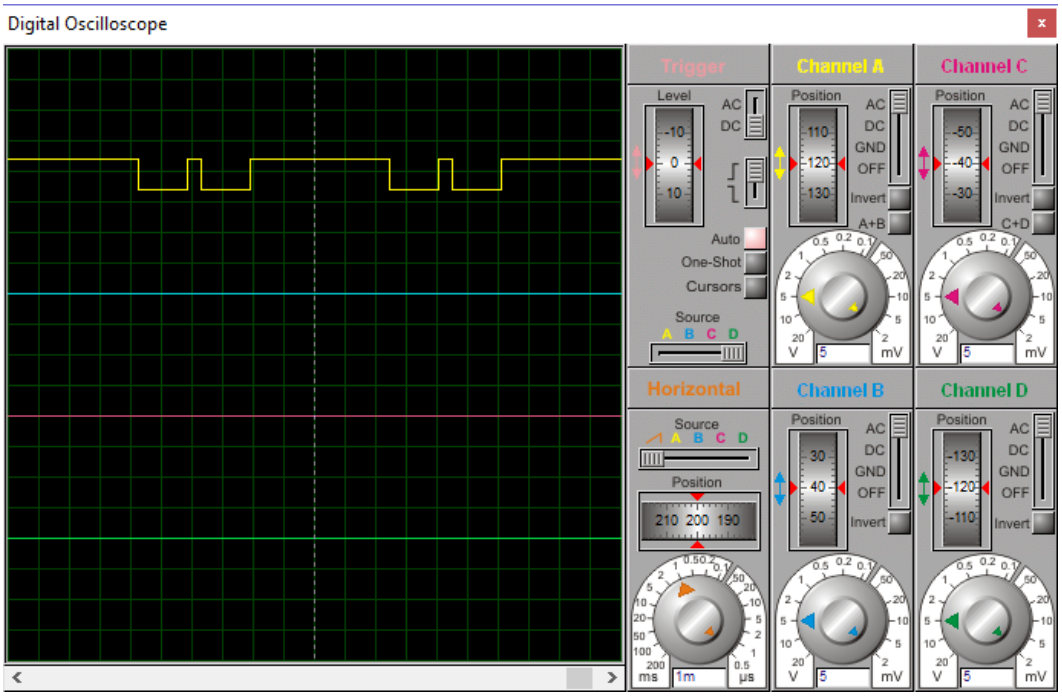


Рисунок 3.29 – Вікно осцилографа

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).

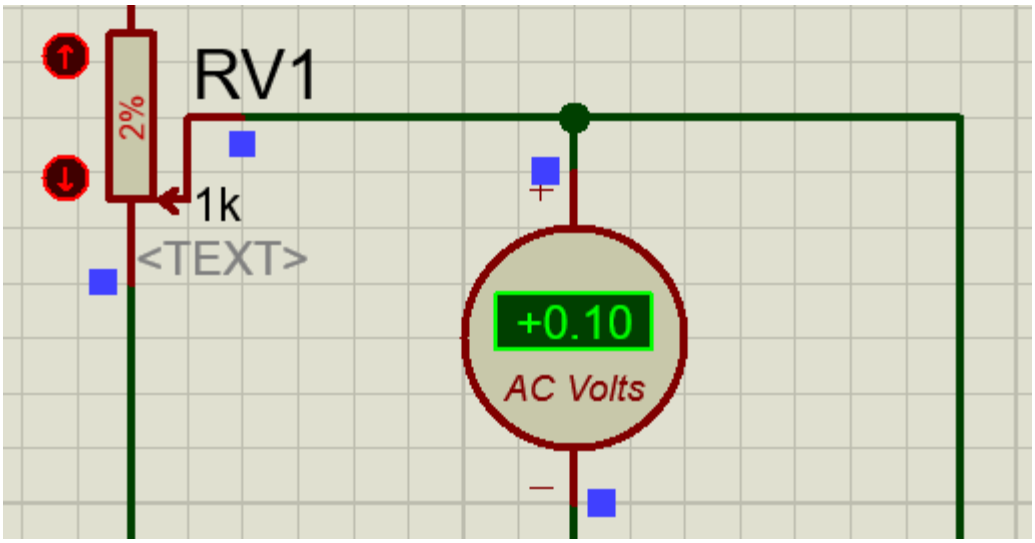


Рисунок 3.30 – Вольтметр, який розміщено на схемі

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).

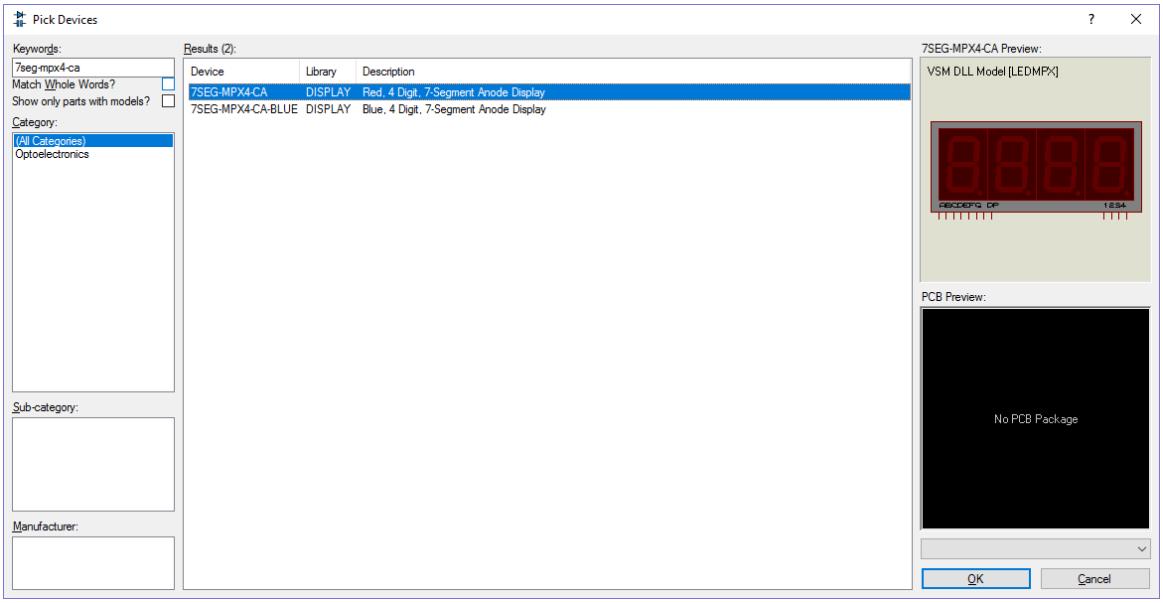


Рисунок 3.31 – Вікно додавання семисегментного індикатора

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

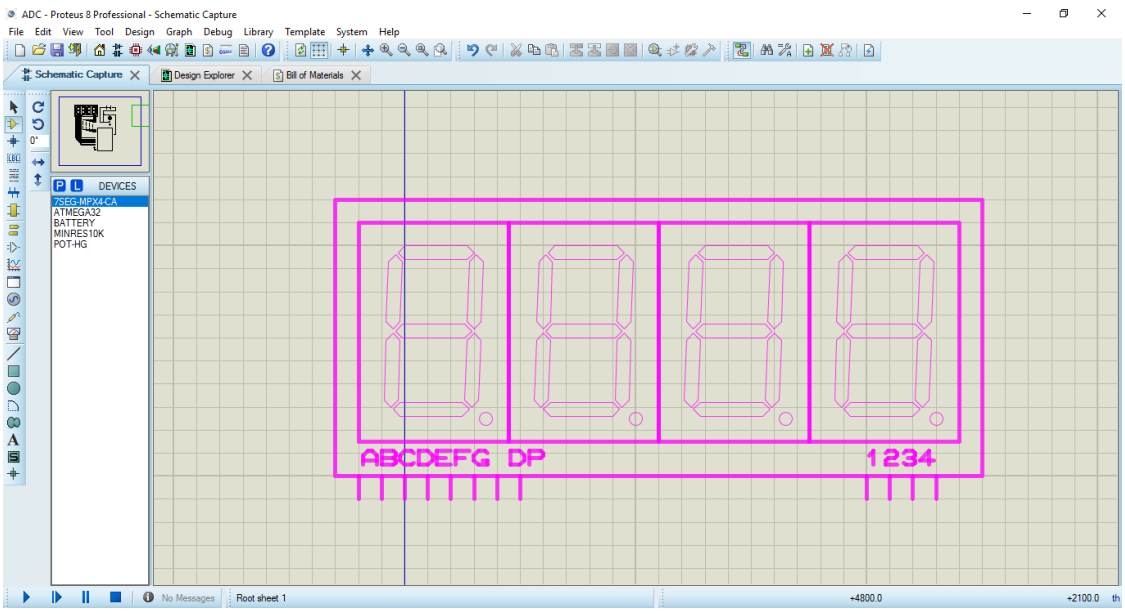


Рисунок 3.32 – Розміщення індикатора на схемі

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).

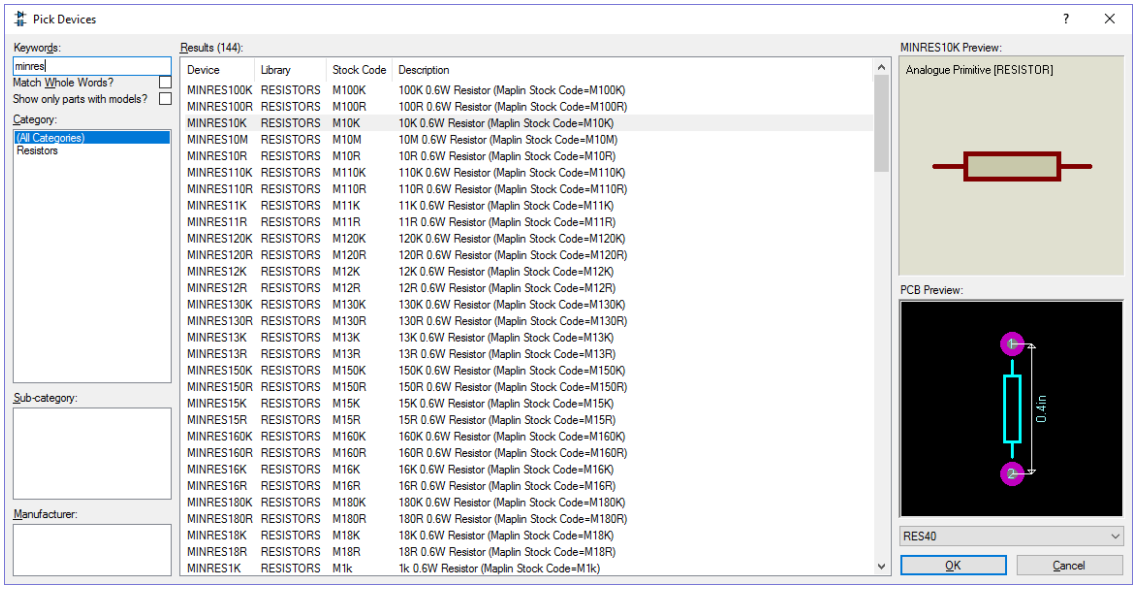


Рисунок 3.33 – Вікно вибору потрібного резистора

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.

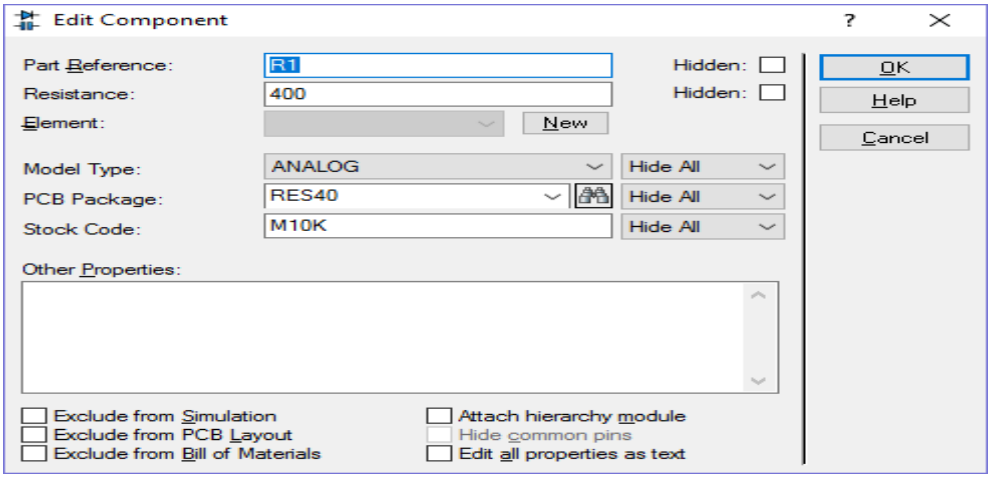


Рисунок 3.34 – Вікно налаштувань резистора

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).

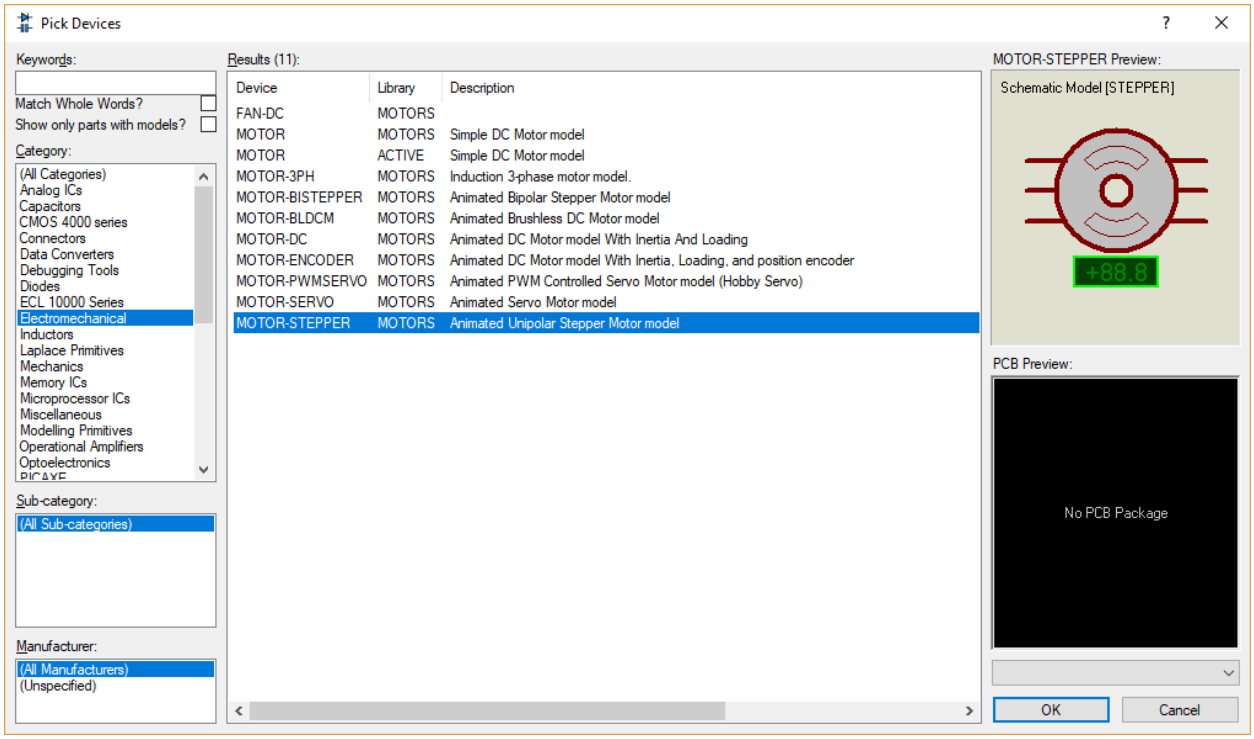


Рисунок 3.35 – Уніполярний кроковий двигун (STEPPER)

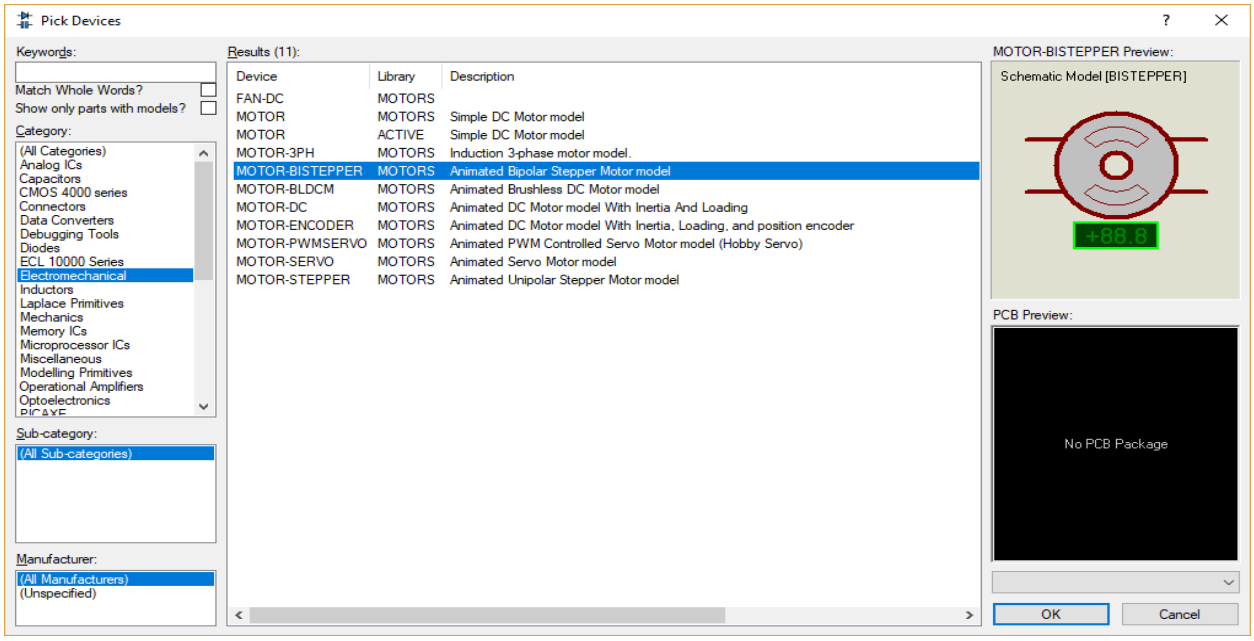


Рисунок 3.36 – Біполярний кроковий двигун (BISTEPPER)

3.1.6 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).

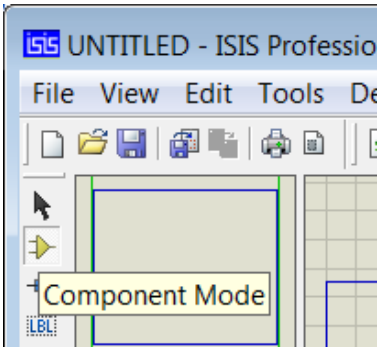


Рисунок 3.37 – Перехід у режим Component Mode

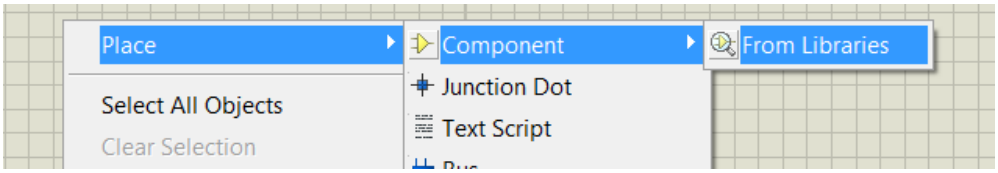


Рисунок 3.38 – Меню додавання нового компоненту

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.

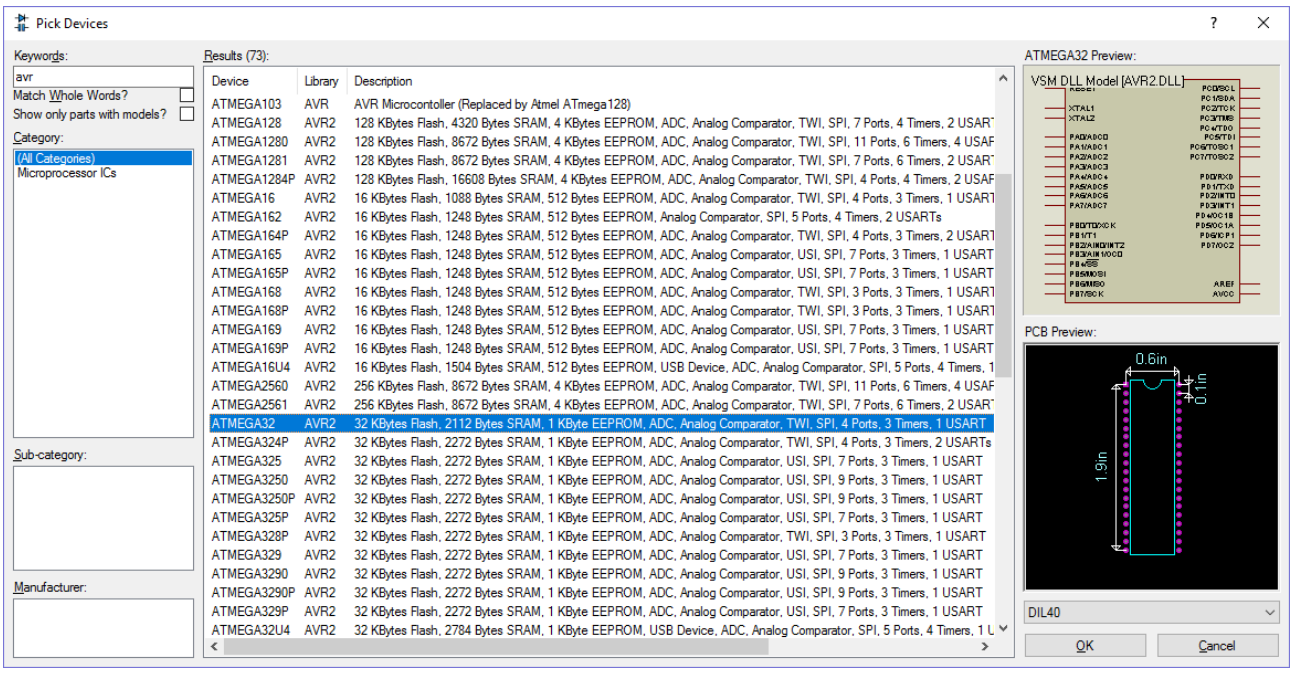


Рисунок 3.39 – Вибір мікроконтролера

3.1.7 Приклад побудови схеми цифрового вольтметра

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

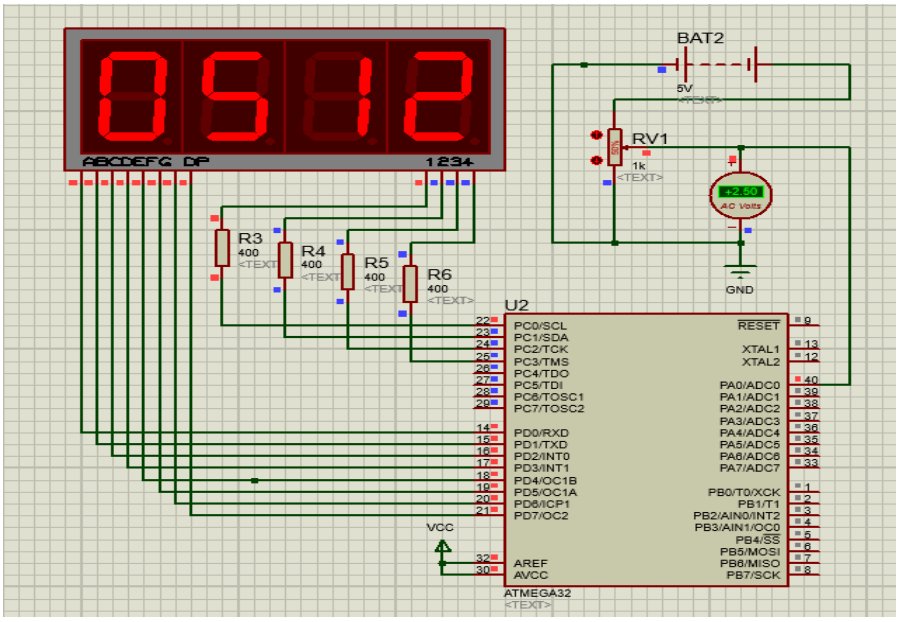


Рисунок 3.40 – Приклад схеми цифрового вольтметра

Роботу цієї схеми буде розглянуто у підрозділі 4.3.3.2. 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).

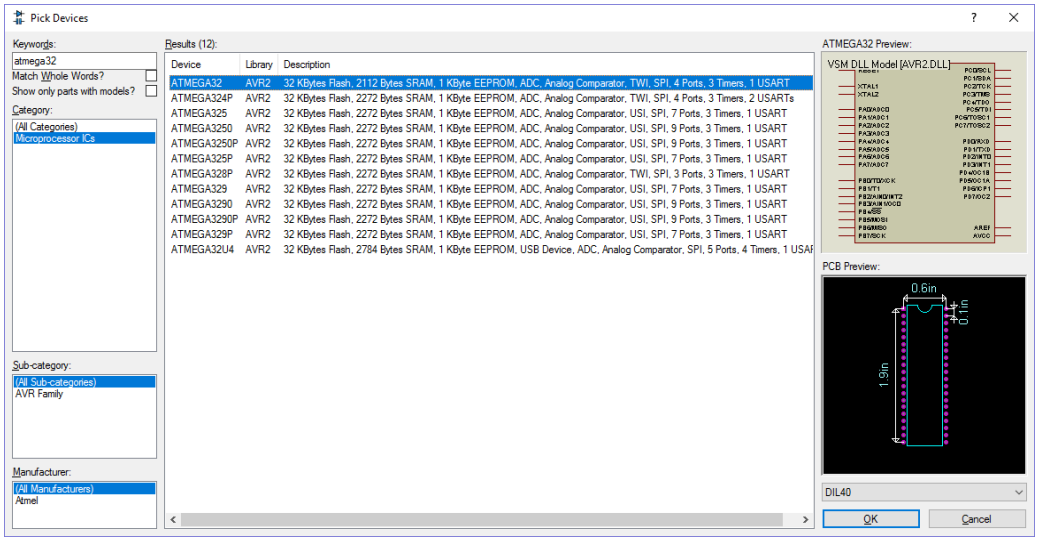


Рисунок 3.41 – Пошук МК у бібліотеці компонентів

Зображення обраного МК на схемі моделі наведене на рисунку 3.42.

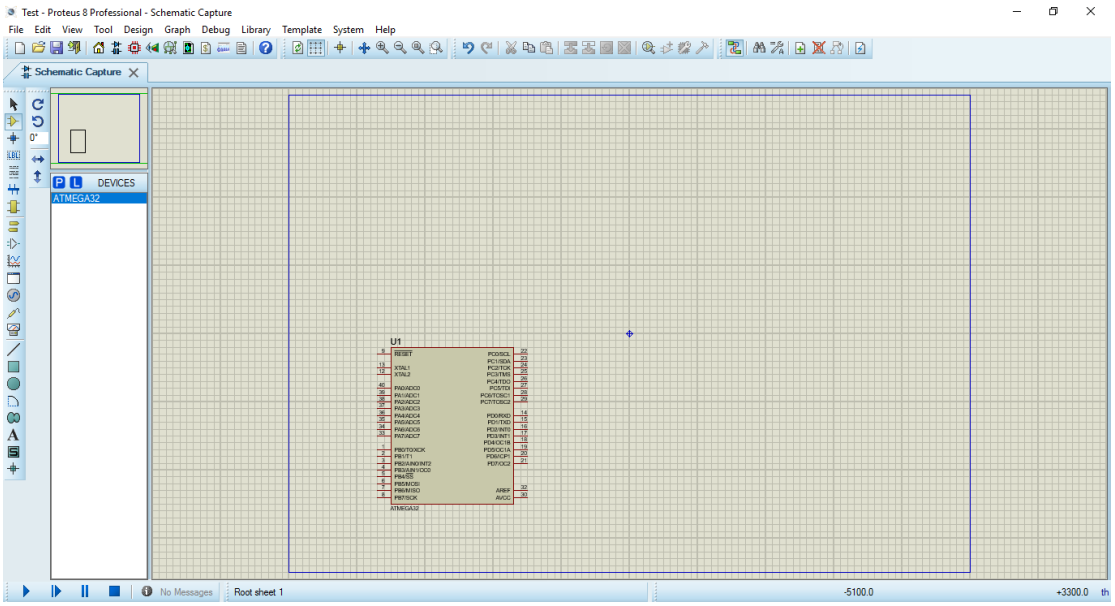


Рисунок 3.42 – Зображення обраного МК на схемі моделі

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).

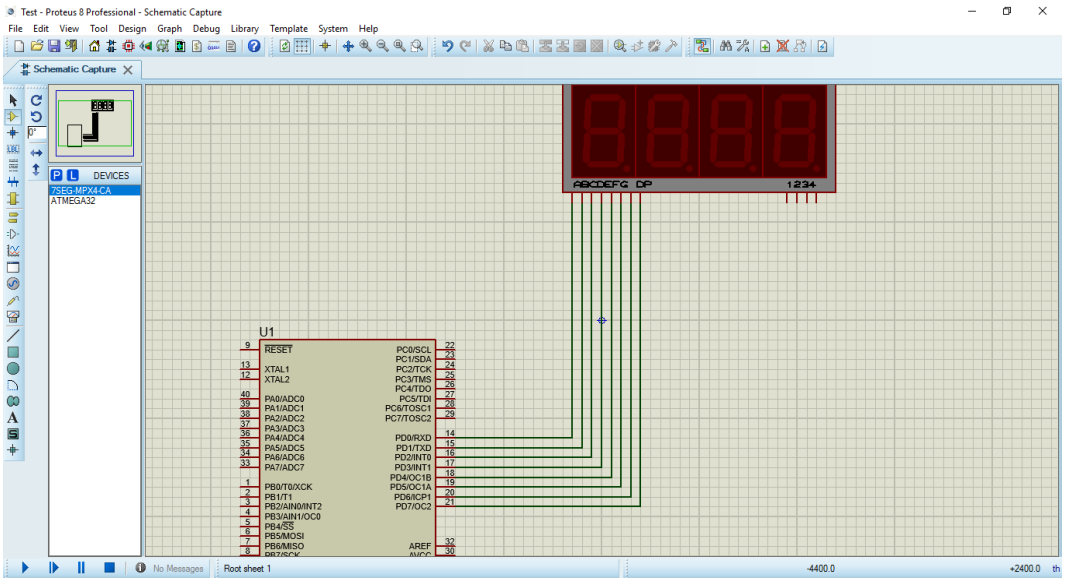


Рисунок 3.43 – З’єднання МК з індикатором

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.

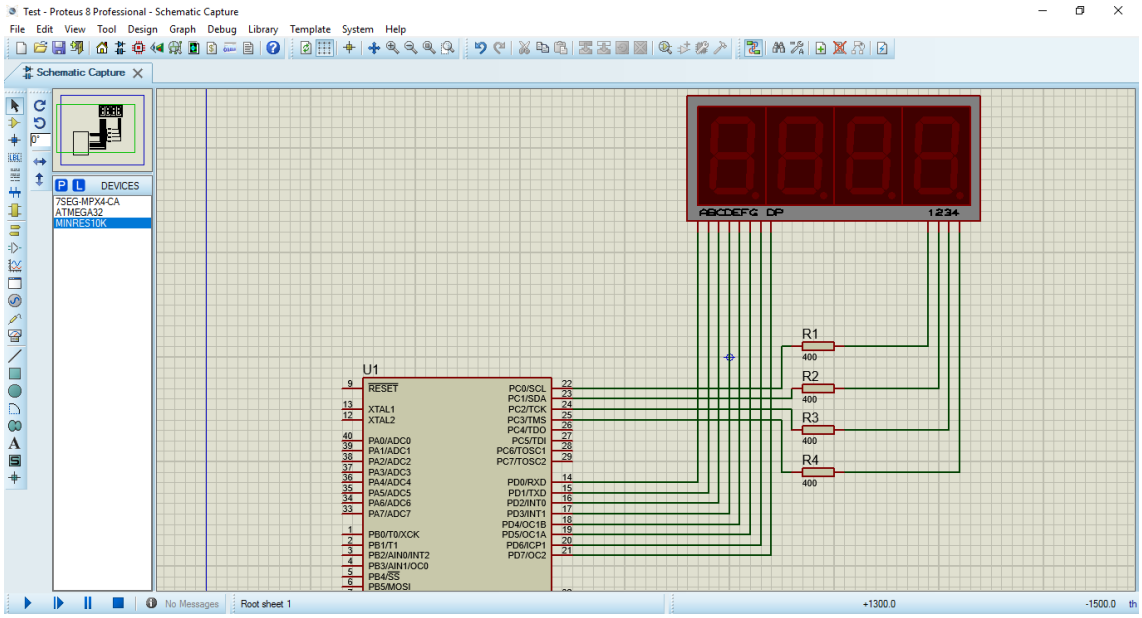


Рисунок 3.44 – Під’єднання резисторів

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

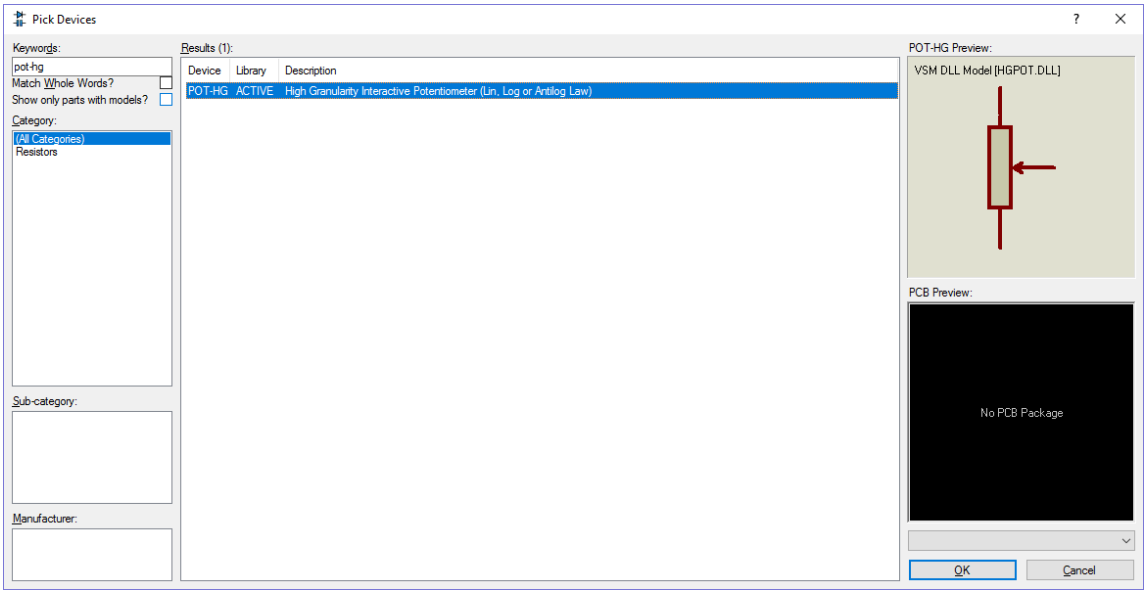


Рисунок 3.45 – Додавання на схему потенціометра

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.

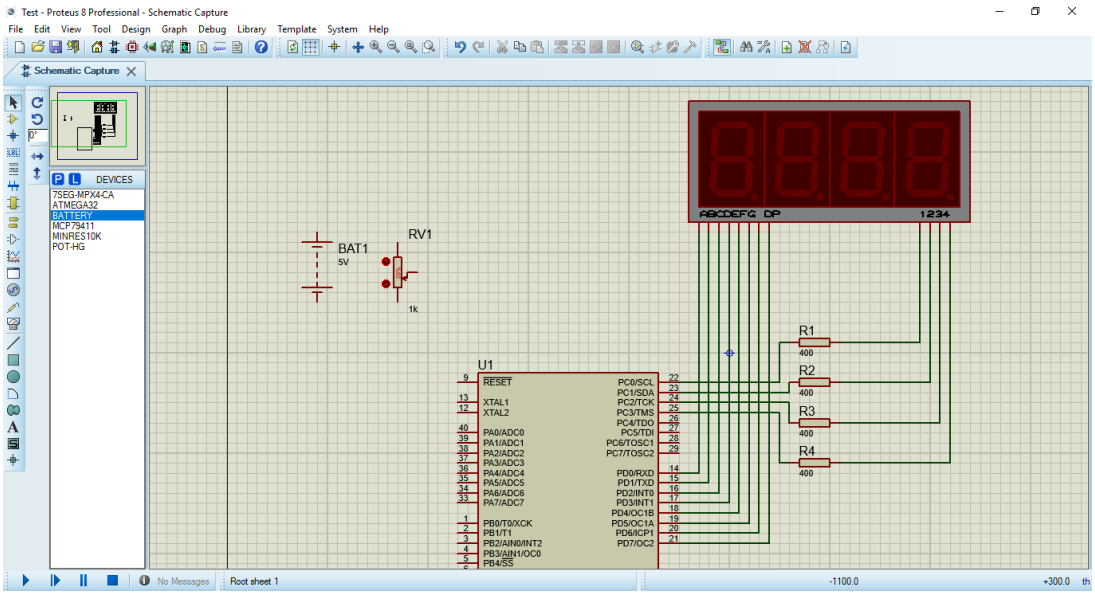


Рисунок 3.46 – Розташування на схемі батарей та потенціометра

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

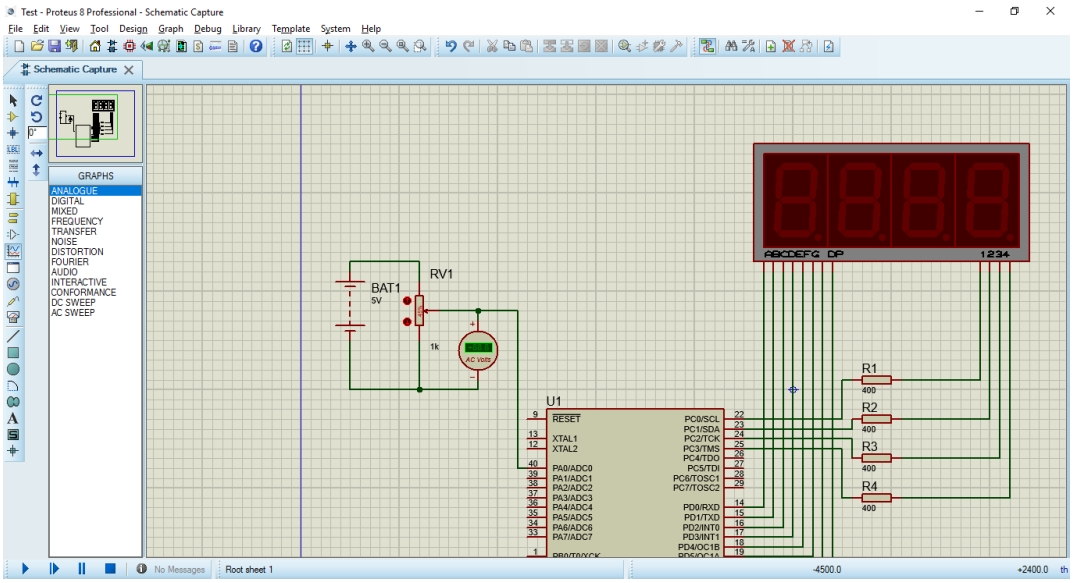


Рисунок 3.47 – Схема з’єднання компонентів

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).

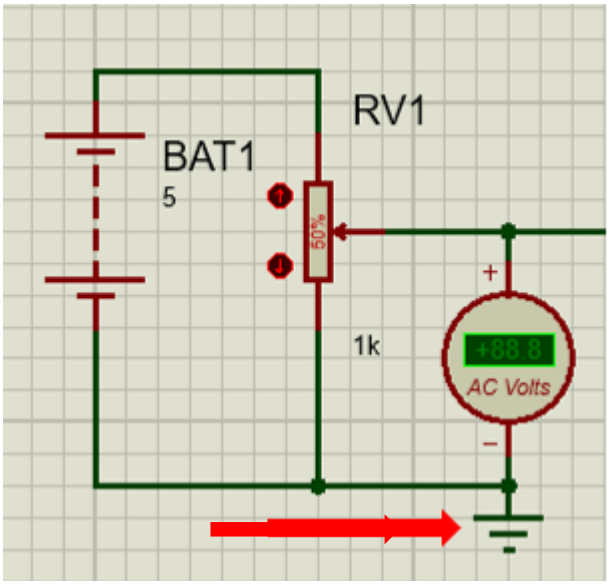


Рисунок 3.48 – Значок заземлення на схемі

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

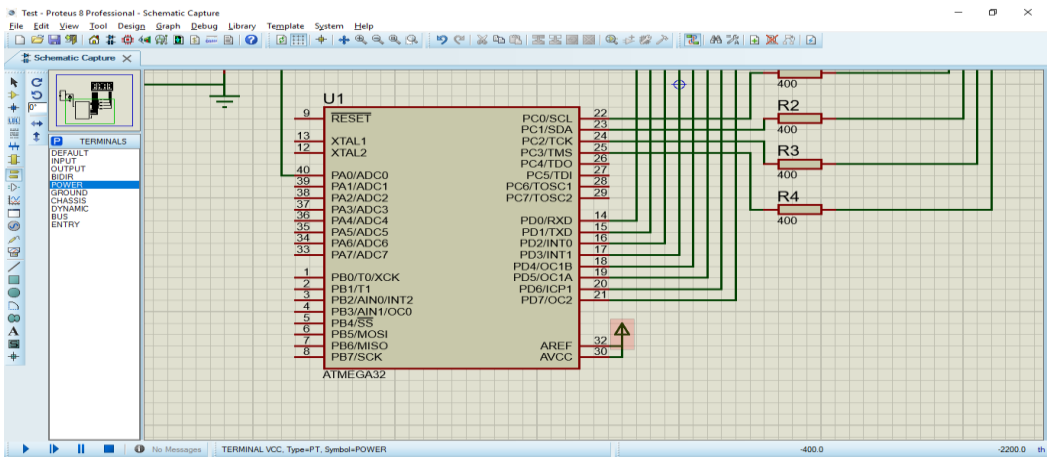


Рисунок 3.49 – Місце під’єднання живлення

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

3.1.8 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.

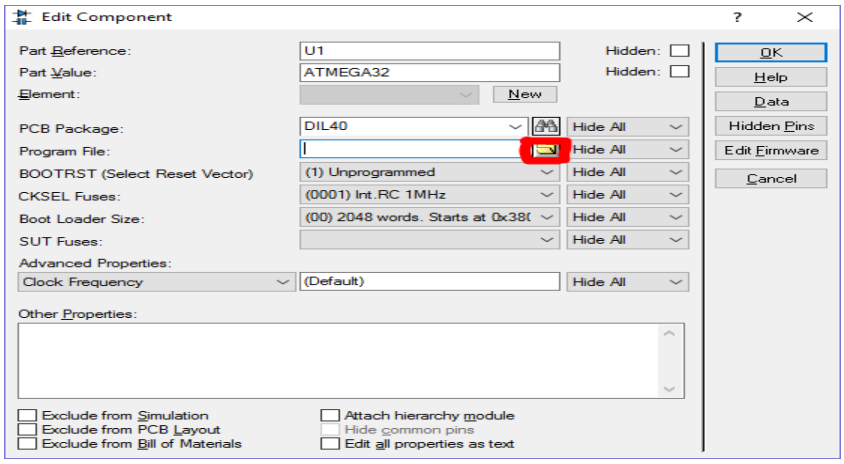


Рисунок 3.50 – Вікно налаштувань мікроконтролера

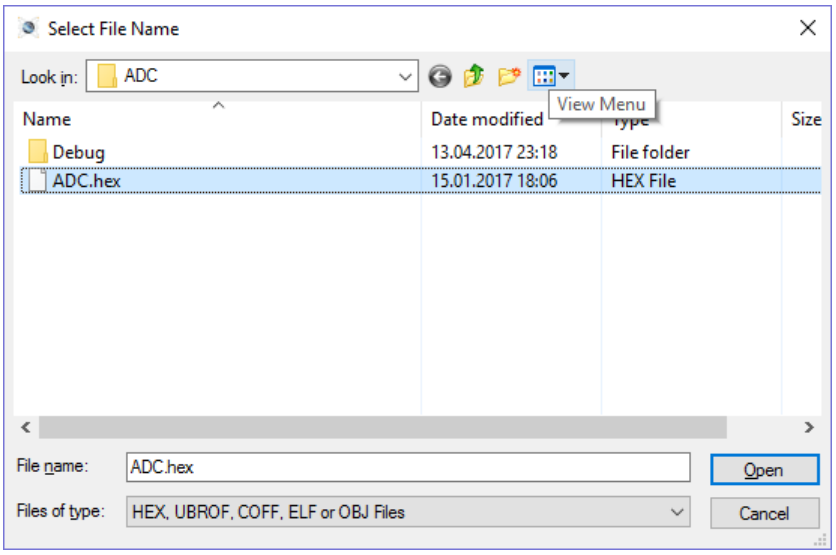


Рисунок 3.51 – Вибір файлу програми

3.2 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.

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 3.1.1.3…3.1.1.8.

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.

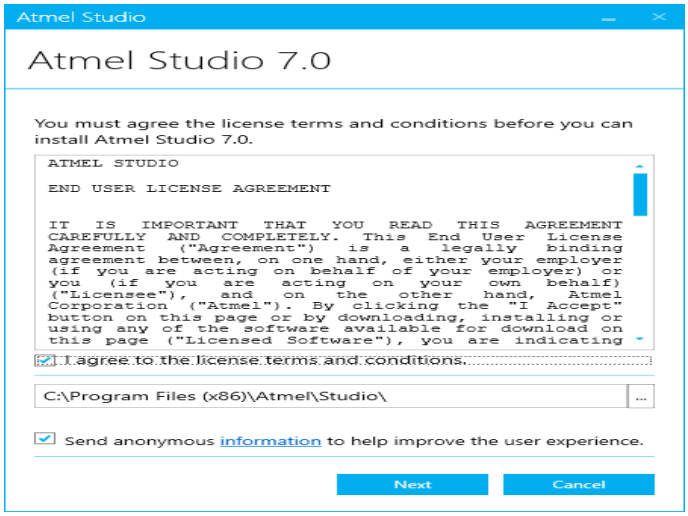


Рисунок 3.53 – Вікно ліцензійного погодження з правилами

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

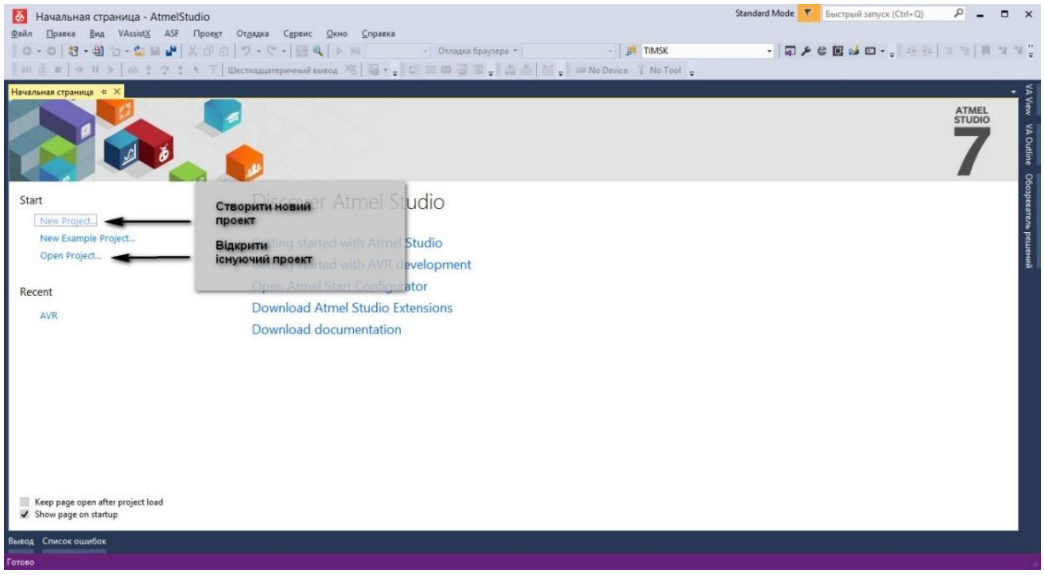


Рисунок 3.54 – Початкове вікно Atmel studio

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

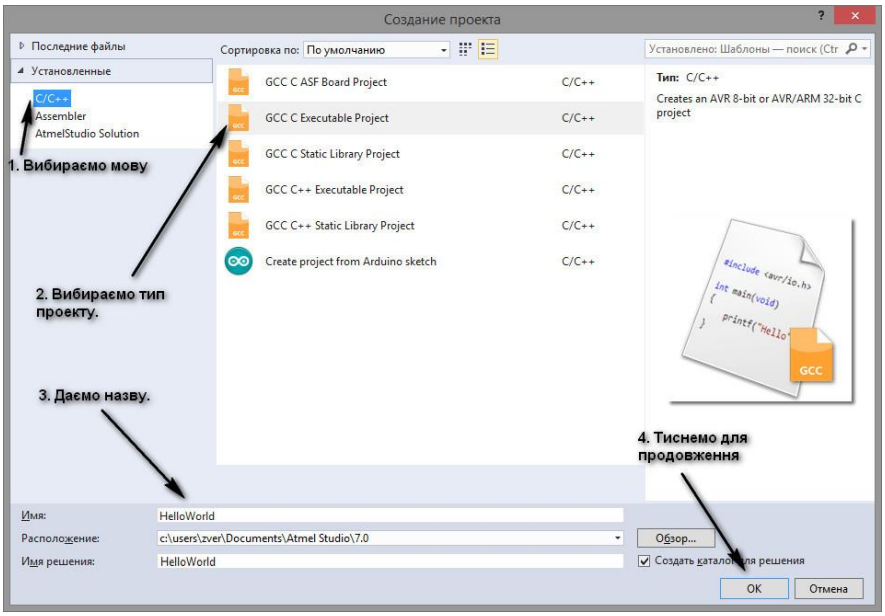


Рисунок 3.55 – Меню вибору мови, типу проекту та назви

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).

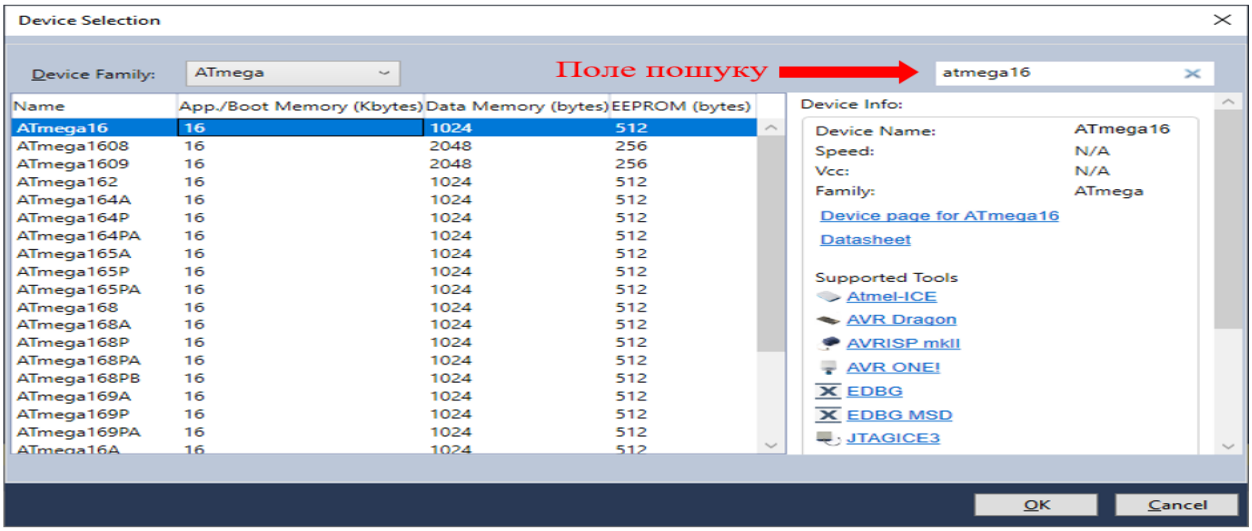


Рисунок 3.56 – Меню вибору мікроконтролера

Як приклад, вибираємо ATmega16 та будемо використовувати даний мікроконтролер як цільовий пристрій, для якого буде створюватись виконувана програма. Після цього необхідно натиснути «ОК». Atmel Studio створить проект та покаже вікно середовища розробки (рисунок 3.57).

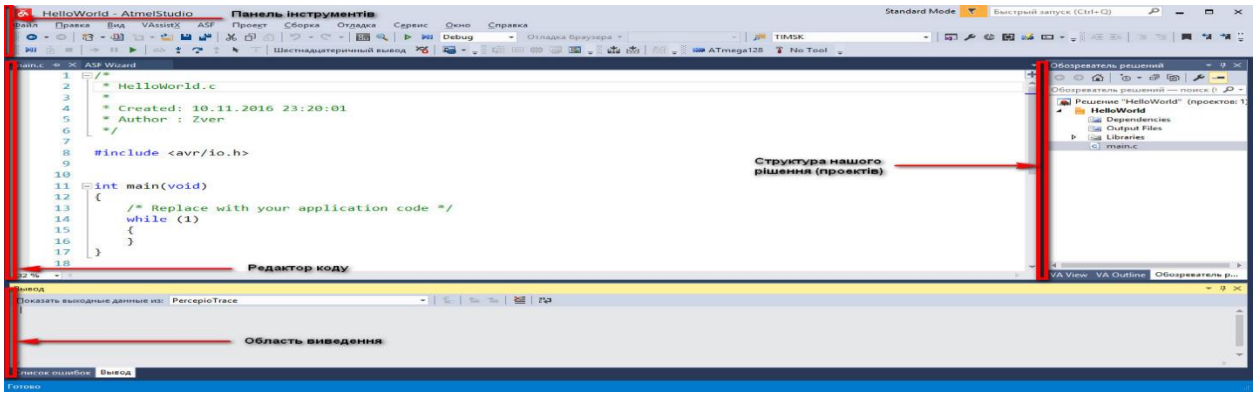


Рисунок 3.57 – Головне вікно середовища розробки

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.

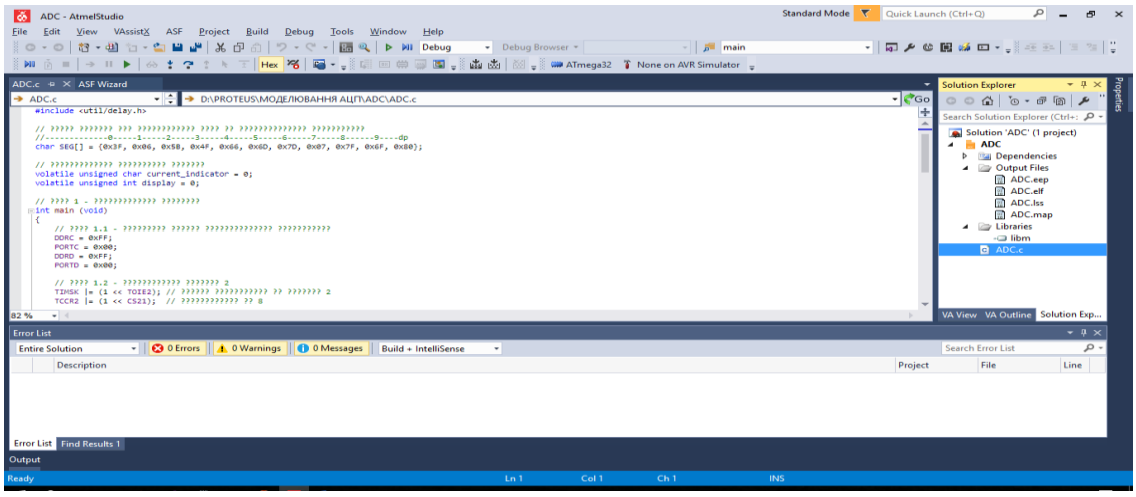


Рисунок 3.58 – Загальний вигляд середовища розробки

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.

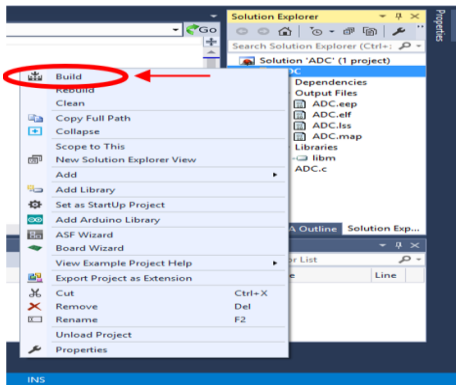


Рисунок 3.59 – Меню налаштувань проекту

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

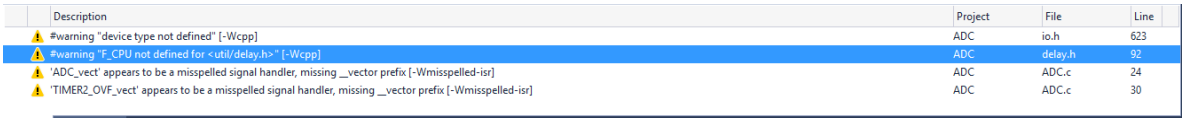


Рисунок 3.60 – Попередження від компілятора

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

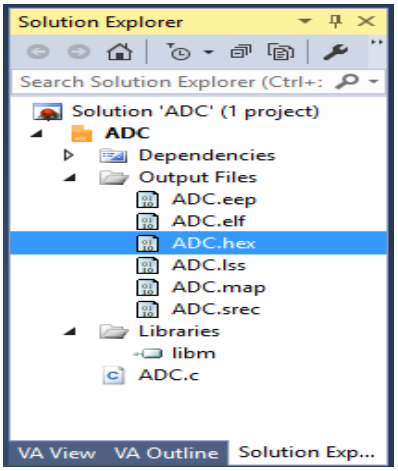


Рисунок 3.61 – Новий файл

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

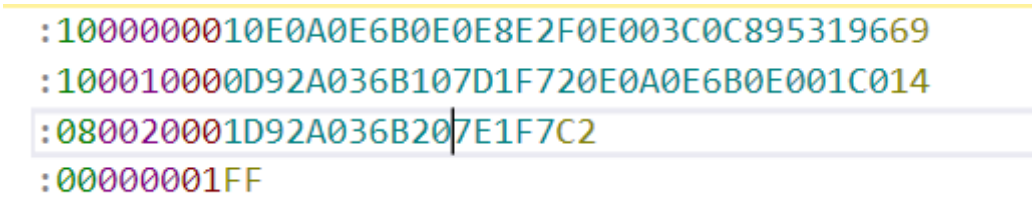


Рисунок 3.62 – Дані з файлу

4 ВИКОНАННЯ ОКРЕМИХ ЛАБОРАТОРНИХ РОБІТ

4.1 Перелік тем лабораторних робіт

Розділ 1. Програмування AVR-мікроконтролерів

Лабораторна робота 1. Дослідження команд пересилання, арифметичних, логічних, роботи з окремими бітами та зсуву – 2 години.

Лабораторна робота 2. Дослідження команд передачі керування, виклику та повернення із підпрограм – 2 години.

Лабораторна робота 3. Дослідження нових команд МК-рів Mega та Xmega – 2 години.

Розділ 2. Периферійні модулі AVR-мікроконтролерів

Лабораторна робота 4. Дослідження моделі пристрою керування двигуном постійного струму – 2 години.

Лабораторна робота 5. Дослідження моделі АЦП – 2 години.

Лабораторна робота 6. Дослідження моделі цифрового вольтметра – 2 години.

Лабораторна робота 7. Дослідження моделі послідовного асинхронного інтерфейсу – 2 години.

Лабораторна робота 8. Дослідження моделі послідовного синхронного інтерфейсу SPI – 2 години.

Лабораторна робота 9. Дослідження моделі послідовного синхронного інтерфейсу I^2C – 2 години.

4.2 Виконання лабораторних робіт з використанням симулятора AVR Studio 4

4.2.1 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 дослідити виконання команд пересилання даних, арифметичних, логічних, роботи з окремими бітами та зсуву у покроковому режимі.

Порядок виконання лабораторної роботи

1) Вивчити теоретичні відомості з теми «Команди пересилання даних, арифметичні, логічні, роботи з окремими бітами та зсуву». 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 програму з наведеними нижче командами пересилання даних у симулятор AVR-Studio 4.
3) Вивчити програмно-налагоджувальні засоби (ПНЗ) у AVR-Studio 4.
4) За допомогою ПНЗ проаналізувати виконання програми з наведеними нижче командами пересилання. Переконатися в правильному виконанні програми. При негативному результаті здійснити зміну програми та повторити перевірку.
5) Розробити алгоритм для виконання індивідуального завдання.
6) Розробити програму для виконання індивідуального завдання.
7) Ввести програму індивідуального завдання у симулятор AVR-Studio 4.
8) За допомогою ПНЗ проаналізувати виконання індивідуальної програми. Переконатися в правильному виконанні індивідуального завдання, при негативному результаті здійснити зміну алгоритму або програми, повторити перевірку.
9) Роздрукувати лістинг правильно працюючої програми.
10) Відповісти на контрольні питання виклада

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 стекові команди PUSH і POP (відсутні в AT90S1200), що дозволяють зберігати в 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).

Тип команди визначає як вона буде представлена у пам’яті програм процесора. Тип кожної команди зазначено в 6-му стовпці таблиці 1.3 і представлено на рисунку 4.1.

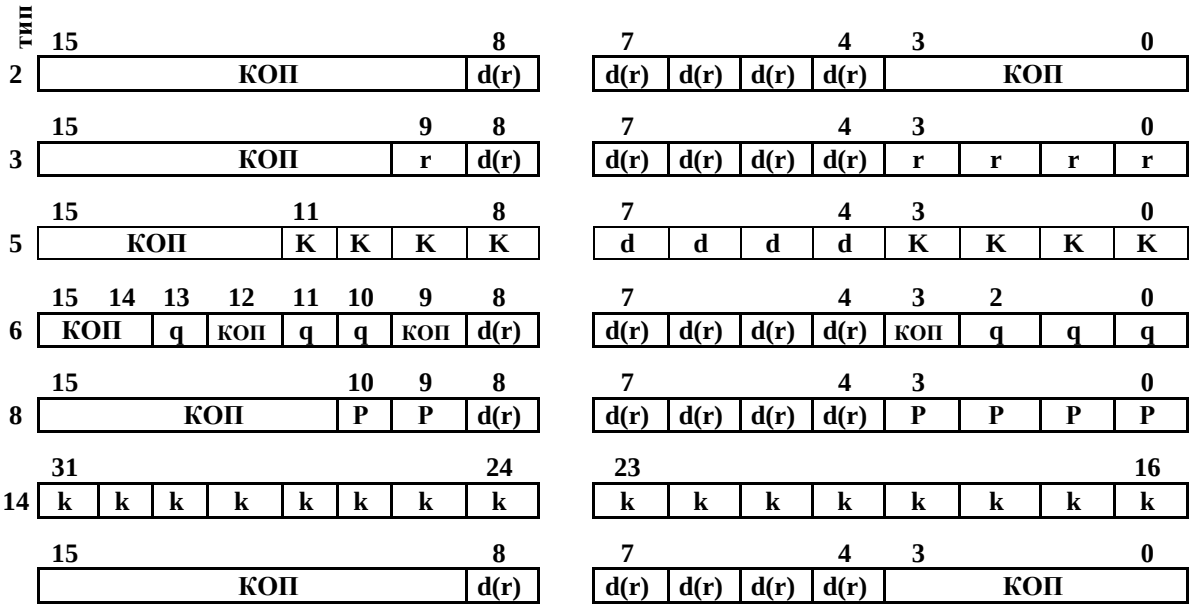


Рисунок 4.1 – Типи команд, що використовуються у командах пересилання даних

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

Опис: команда копіює вміст одного регістра в інший. 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), лише замість коду операції (КОП) компілятор підставляє конкретні цифри. КОП надає пристрою керування мікропроцесора інформацію які дії потрібно виконати при виконанні команди. $КОП_{команди}$ = 001011. Його можна визначити за допомогою симулятора AVR STUDIO4. Якщо створити проект, який складається лише з однієї команди MOV R1, R31, скомпілювати її, то область пам’яті програм матиме вигляд, який зображено на рисунку 4.2.

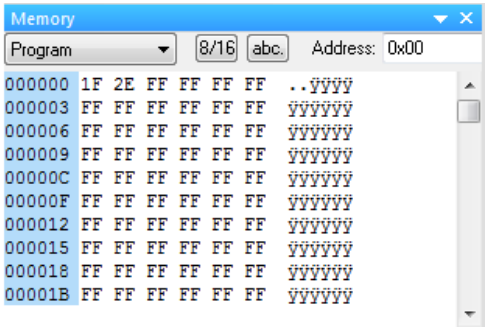


Рисунок 4.2 – Вигляд пам’яті програм після компіляції команди MOV R1, R31

Нижче наведено аналіз вмісту пам’яті програм.

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		

Слід зазначити що в пам’яті програм спочатку розміщається другий байт команди, потім – перший: 2E 1F, у той час як 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 дозволяють перевірити правильність розрахунків. Розглядалась команда MOV R1, R31. Розраховані значення r = 31, d = 1, робимо висновок, що розрахунки проведено вірно.

Довжина команди: 1 слово (2 байти).

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

Нижче наведено програму з командами 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 завантаження                      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
```

Варіанти індивідуальних завдань

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.

Контрольні запитання

1) Які області пам’яті виділяють в програмістській моделі мікроконтролерів сімейства AVR?
2) Які операції пересилання даних дозволяє виконувати система команд мікроконтролерів AVR?
3) Який спосіб адресації використовується у цих командах?
4) Опишіть операнди, що входять до складу команд.
5) Звідки береться цифра 12 у формулі розрахунку часу виконання команди у секундах?

Команди арифметичні, логічні, зсуву та роботи з окремими бітами

Стислі теоретичні відомості

Команди даної групи наведено у таблиці 1.3. Формати команд наведено на рисунку 1.23, а їх опис – у 1.5.

Нижче наведено приклади арифметичних, логічних команд та команд зсуву.

Приклади 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        ; Встановити  парні біти в регістрі     R17 = R17 v $55
cbr R23, $AA; Скинути непарні біти  в регістрі      R23 = R23 * ($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     ; Скинути регістр                               R23 = $00
ser R17     ; Встановити регістр                            R17 = $FF
swap R24    ; Обмін тетрадами регістра                      R24
ldi R25, $05; Завантажити константу                         R25 = $05
ldi R26, $C0        ; Завантажити константу                 R26 = $C0
lsl R25     ; Логічний зсув ліворуч R25
lsl R26      ; Арифметичний зсув вліво R26
ldi R27, 12  ; Завантажити константу                        R27 = 12
ldi R28, $F4; Завантажити константу                         R28 = $F4
lsr R27     ; Логічний зсув праворуч  R27
asr R28             ; Арифметичний зсув праворуч  R28
ldi R29, $A5        ; Завантажити константу                 R29 = $A5
rol R28     ; Циклічний зсув вліво R28
ror R28     ; Циклічний зсув вправо  R28
```

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 двухбайтного 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.
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, 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 K15 відповідно.

Закінчення таблиці 4.2

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 приклади команд роботи з бітами.

Приклади програми з використанням команд роботи з бітами

```
sbi $ 1C, 0; Встановити біт читання в EECR
cbi $ 12, 7; Скинути біт 7 у порту D
lsl R1      ; Логічний зсув вліво R1 (n +1) <= R1 (n), R1 (0) <= 0
lsr R1      ; Логічний зсув вправо  R1 (n) <= R1 (n +1), R1 (7) <= 0
rol R1      ; Циклічний зсув вліво через C R1 (0) <= C, R1 (n +1) <= R1 (n), C <= R1 (7)
ror R1      ; Циклічний зсув вправо через C R1 (7) <= C, R1 (n) <= R1 (n +1), C <= R1 (0)
asr R1      ; Арифметичний зсув вправо  R1 (n) <= R1 (n +1), n = 0 .. +6
swap R2   ; Перестановка тетрад R1 (3 .. 0) <= R1 (7 .. 4), R1 (7 .. 4) <= R1 (3 .. 0)
bset 7      ; Встановлення прапорця SREG (s) <= 1 SREG (s)
bclr 7      ; Скидання прапорця SREG (s) <= 0 SREG (s)
bld R1, 7   ; Завантажити біт з T в регістр R1 (b) <= T
bst R1, 7    ; Зберегти біт з регістра в T, T <= R1 (b)
set         ; Встановити прапорець перенесення C <= 1
clc         ; Очистити прапорець перенесення C <= 0
sen         ; Встановити прапорець від'ємного числа N <= 1
cln         ; Скинути прапорець від'ємного числа N <= 0
sez         ; Встановити прапорець нуля Z <= 1
clz             ; Скинути прапорець нуля Z <= 0
sei             ; Встановити прапорець переривань I <= 1
cli             ; Скинути прапорець переривань I <= 0
ses             ; Встановити прапорець числа зі знаком S <= 1
cls             ; Скинути прапорець числа зі знаком S <= 0
sev             ; Встановити прапорець переповнення V <= 1
clv             ; Скинути прапорець переповнення V <= 0
set             ; Встановити прапорець T, T <= 1
clt             ; Скинути прапорець T, T <= 0
seh             ; Встановити прапорець допоміжного перенесення      H <= 1
clh             ; Скинути прапорець допоміжного перенесення H <= 0
nop             ; Немає операції
sleep           ; Спати (зменшити енергоспоживання)
wdr             ; Скидання вартового таймера
```

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 (TCCR1B). 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 PORTB b) Реалізувати комбінаційну двійкову функцію, задану аналітично: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.
4	a) Скинути третій біт у порту PORTC. b) Знайти алгебраїчну суму бітів bit1 + bit2 + bit3 + bit4. Результат помістити у регістрі R7, у 3 молодших біта. 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 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) Перші не зарезервовані 3 біта і останній біт з прямоадресуємої області 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 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 и R23.
9	a) Виконати арифметичний зсув регістра R17 на 5 вліво. Реалізувати логічну функцію: $PORTB(2) = \overline{R12(7) + bit1}$ c) Просумувати по модулю 2 біти регістра R5. Результат помістити в прапорець додаткового перенесення.

Закінчення таблиці 4.3

№	Текст індивідуального завдання
10	a) Проінвертувати перші 5 не зарезервованих біт з області прямоадресуємих біт і логічно підсумувати їх з 3-м бітом порту PORTC. b) Реалізувати логічну функцію: bit1=(bit1∧bit2∨(bit1∨bit2))∧bit1∨bit2∨bit3 c) В R12 помістити перші вісім не зарезервованих прямоадресуємих біт.

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

1) 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**

2) 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**

3) 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**

4) 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**

5) 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**

6) 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

1) Які арифметичні та логічні операції дозволяє виконувати система команд мікроконтролерів AVR?
2) Який спосіб адресації використовується у цих командах?
3) Як впливають арифметичні та логічні команди на прапорці?
4) Опишіть операнди, що входять до складу цих команд.
5) В якому поданні можуть бути використані числа при виконанні арифметичних та логічних операціях для мікроконтролерів AVR?
6) Які операції роботи з бітами дозволяє виконувати система команд мікроконтролерів AVR?
7) Які спосіби адресації використовуються у цих командах?
8) Опишіть операнди, що входять до складу цих команд.
9) Де знаходиться області прямоадресуємих біт?

4.2.2 Лабораторна робота № 2. Дослідження команд передачі керування, виклику та повернення із підпрограм

Тема: команди передачі керування

Мета: користуючись налагоджувачем дослідити виконання команд передачі керування у покроковому режимі.

Порядок виконання лабораторної роботи:

1) Вивчити теоретичні відомості з теми «Команди передачі керування». 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 програму з наведеними нижче командами передачі керування у симулятор AVR-Studio 4.
3) За допомогою програмно-налагоджувальних засобів (ПНЗ) у AVR-Studio 4 проаналізувати виконання програми з наведеними нижче

командами передачі керування. 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 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 викладача.

Стислі теоретичні відомості

Команди даної групи наведено у таблиці 1.3. Формати команд наведено на рисунку 1.23, а їх опис – у підрозділі 2.5.

Нижче наведено приклади команд передачі керування.

Приклад програми з використанням команд передачі керування

```
      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
      no                   ; 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 п / пр
        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   ; Сбросіть7-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 4.10 – Завдання до використання команд передачі керування

№	Текст індивідуального завдання
1	a) Вибрати з п’яти двобайтових чисел найбільше. b) Починаючи з адреси 71h, дана послідовність з 8-ми байт. 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 - 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 R21 необхідно помістити в 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.
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 71H по 91H на наявність послідовності з 5 нульових або одиничних біт. Якщо така послідовність є, то у R21 записати адресу початку послідовності.
5	b) Написати підпрограму, яка визначала б, до якого проміжку належить вміст R30: 0 - 99; 100 - 199, 200 - 255. Для видачі результату використовуються три молодших біта 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 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.

Закінчення таблиці 4.10

7	a) Вміст регістра SREG необхідно помістити в: – R21, якщо [R19] >1Fh; – R22, якщо [R19] < 1Fh. b) 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 проміжку належить вміст регістра R21: 0 - 99; 100 - 199, 200 - 255. Для видачі результату використовуються три молодших біта регістра R30. b) Починаючи з адреси 71h, дана послідовність з 10-ти байт. Виділити з послідовності байти з певними властивостями і помістити їх за адресою, заданому регістром 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 в 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

1) Які операції передачі керування дозволяє виконувати система команд мікроконтролерів AVR?
2) Який спосіб 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 і RET? Яка помилка виникне, якщо замість RETI застосувати RET, чому?

4.2.3 Лабораторна робота №3. Дослідження нових команд МК Mega та Xmega

Тема: Нові команди AVR-мікроконтролерів

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

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 1.54.
2) Ввести програму з використанням нових команд у симулятор AVR-Studio 4.
3) За допомогою програмно-налагоджувальних засобів у AVR-Studio 4 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 викладача.

Стислі теоретичні відомості

Команди даної групи наведено у таблиці 1.5, а їх опис – у підрозділі 1.11.

Нижче наведено програму з використанням нових команд.

Програма з використанням нових команд:

```
ok3:
nop                    ; Порожня операція
```

```
nop                 ; Порожня операція
; Множення
ldi r19, $ 2f       ; Завантажити константу r19 = $ 2f
ldi r20, $ 78       ; Завантажити константу 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       ; Завантажити константу r17 = $ 2f
ldi r18, $ 78       ; Завантажити константу r18 = $ 78
clr r0; обнулити r0
clr r1; обнулити r1
MULS r17, r18       ; Множення знакових чисел
ldi r17, $ 2f       ; Завантажити константу r17 = $ 2f
ldi r18, $ 78       ; Завантажити константу r18 = $ 78
clr r0;
clr r1;
MULSU r17, r18      ; Множення знакового числа на беззнакове
; Інструкція переходу
ldi r30, $ 1a       ; Завантажити константу r30 = $ 8a
ldi r31, $ 00       ; Завантажити константу r31 = $ 00
EIJMP               ; Безумовний непрямий перехід в пам’яті програм ємністю 4
                    ; Мслова на Z
nop                 ; Порожня операція
nop                 ; Порожня операція
nop                 ; Порожня операція
ok2:
; Множення
ldi r17, $ 2f       ; Завантажити константу r30 = $ 0d
ldi r18, $ 78       ; Завантажити константу r16 = $ 00MUL R1, R2; Множення бе
clr r0              ; обнулити r0
clr r1              ; обнулити r1
FMUL r17, r18       ; Множення дробових беззнакових чисел
ldi r17, $ 2f       ; Завантажити константу r17 = $ 2f
ldi r18, $ 78       ; Завантажити константу r18 = $ 78
clr r0              ; обнулити r0
clr r1              ; обнулити r1
FMULSU R17, R18     ; Множення дробового знакового числа на беззнаковое
; Виклик підпрограми
ldi r25, $ de       ; Завантажити константу r25 = $ de
out $ 3D, r25       ; Записати r25 в порт $ 3D
call ok1            ; Відносний безумовний виклик
; Множення
ldi r17, $ 2f       ; Завантажити константу r17 = $ 2f
ldi r18, $ 78       ; Завантажити константу r18 = $ 78
clr r0              ; обнулити r0
clr r1              ; обнулити r1
FMULS r17, r18      ; Множення дробових знакових чисел
; Виклик підпрограми
ldi r30, $ 4b       ; Завантажити константу r30 = $ 8a
```

```
ldi r31, $ 00          ; Завантажити константу r31 = $ 00
eicall                 ; Відносний безумовний виклик підпрограми
; Передача даних
ldi r19, $ 00          ; Завантажити константу r17 = $ 2f
ldi r18, $ 1b          ; Завантажити константу r18 = $ 78
clr r31                ; обнулити r31
clr r30                ; обнулити r30
MOVW r30, r18          ; Копіювання слова
LPM r1, Z              ; Завантаження регістра R1 з пам’яті програм з адресою в
                       ; індексному регістрі Z
LPM r2, Z +            ; Завантаження регістра R1 з пам’яті програм з адресою в
                       ; індексному регістрі Z з постинкрементом вмісту Z
SPM                    ; Самопрограмування мікроконтролера, у якого є відповідний блок
clr r0                 ; обнулити r0
ELPM                   ; Завантажити регістра r0 з розширеною пам’яті програм з
                       ; адресою в регістрах RAMPZ і Z
ELPM r3, Z             ; Завантажити регістра r3 з розширеної пам’яті програм з
                       ; адресою в регістрах RAMPZ і Z
ELPM r4, Z +           ; Завантажити регістра r4 з розширеною 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
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
nop                    ; Порожня операція
nop                    ; Порожня операція
nop                    ; Порожня операція
add r21, r19           ; Додати r21 до  r19 без перенесення
adc r22, r20           ; Додати r22 до  r20 з перенесенням
ret                    ; Повернення з підпрограми
ok4:
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           ; Віднняти r20 від r22 без перенесення
adc r23, r21           ; Віднняти r21 від r23 з перенесенням
ret                    ; Повернення з підпрограми
```

Контрольні запитання

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

4.3 Виконання лабораторних робіт з використанням Proteus 8.6

4.3.1 Лабораторна робота №4. Моделювання 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.

4.3.1.1 Порядок виконання роботи

– створити модель пристрою в пакеті Proteus 8.6

– розробити схему алгоритму роботи моделі та робочу програму

– створити hex-файл та підключити його до мікроконтролера

– запустити модель та виконати її дослідження згідно методичних вказівок

– зробити відповідні висновки.

4.3.1.2 Стислі теоретичні відомості

4.3.1.2.1 Опис моделі

Робочу модель пристрою керування двигуном постійного струму показано на рисунку 4.1.

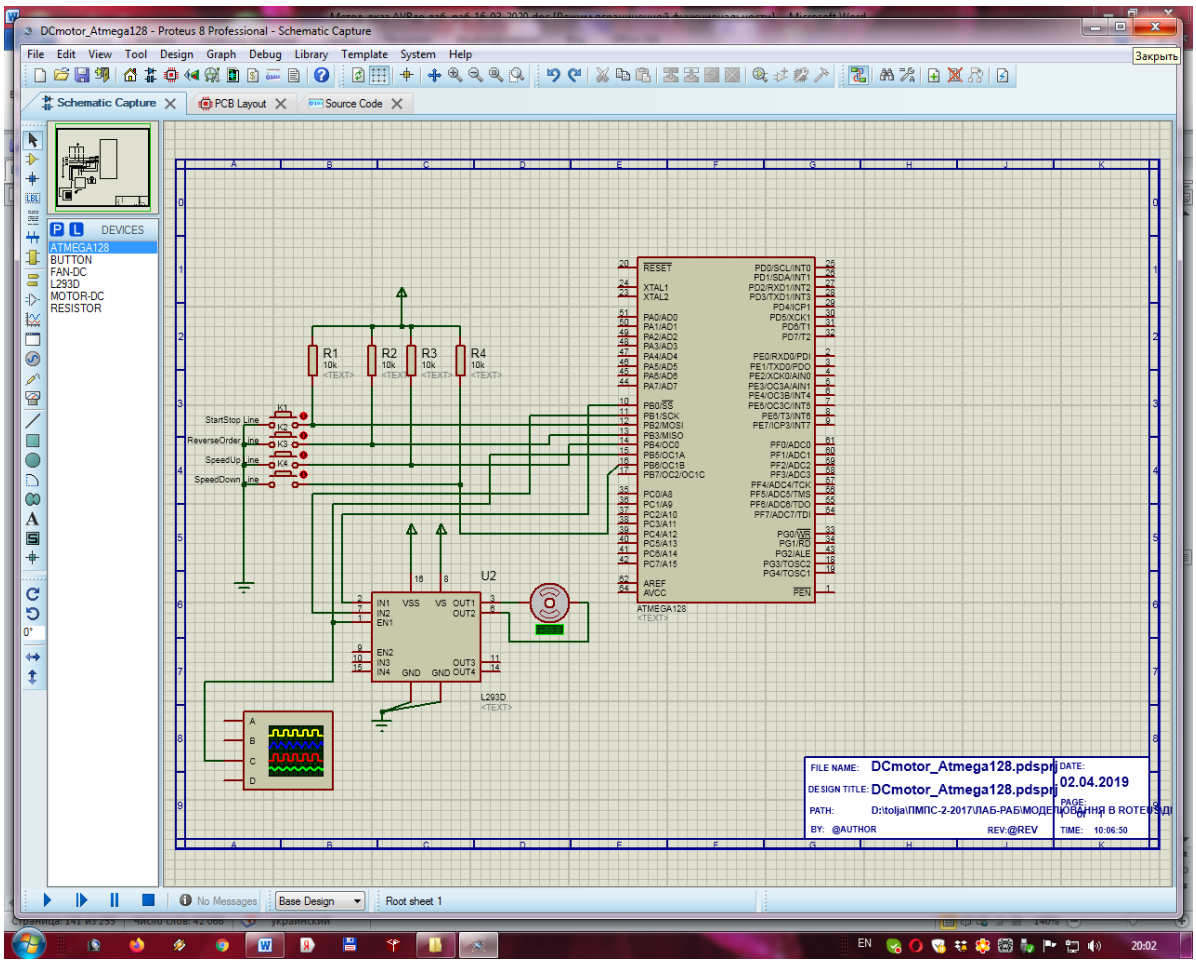


Рисунок 4.1 – Схема моделі пристрою керування двигуном постійного струму

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.

Зовнішні резистори ми використовуємо через те, що не використовуємо внутрішні підтягуючі резистори на вхідних лініях мікроконтролера. На рисунку 4.2 наведено позначення виводів мікроконтролера.

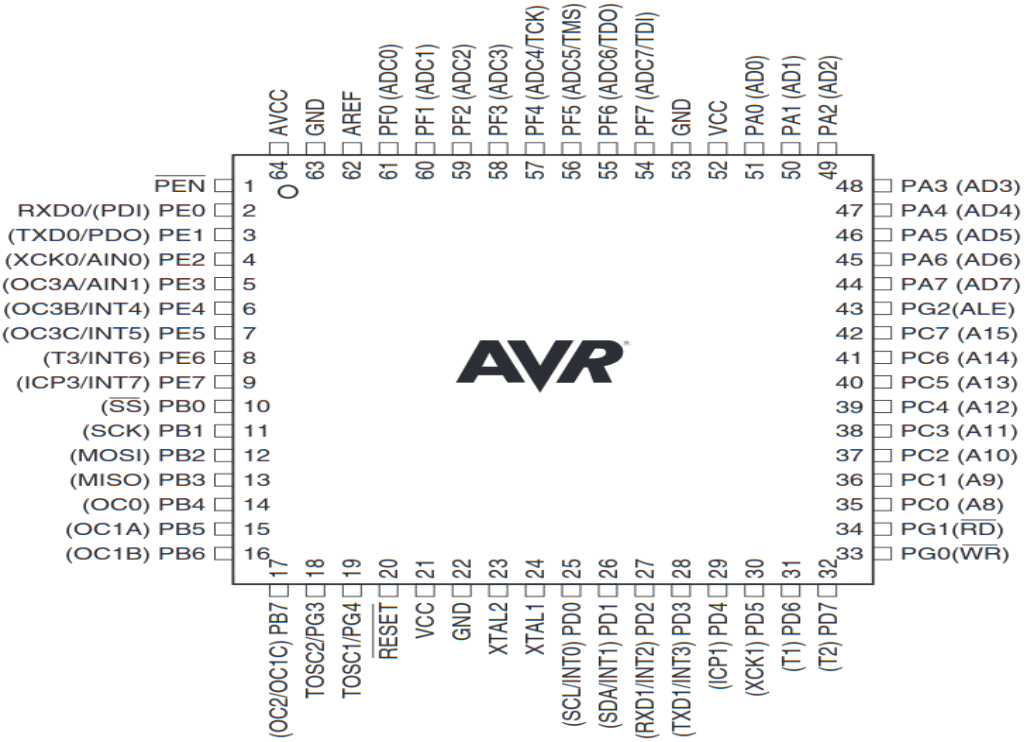


Рисунок 4.2 – Виводи мікроконтролера ATmega128

На рисунку 4.3 наведено збільшений вигляд кнопок керування.

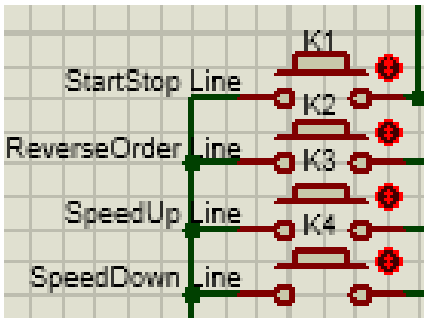


Рисунок 4.3 – Порядок розташування елементів керування

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 низьки 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 постійного струму DC Motor. 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 постійного струму DC Motor. Одне натискання – одна зміна напрямку. Проте потрібно мати на увазі, що двигун DC Motor інерційний, і він не зупиняється миттєво і одразу починає 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 4.4). Знову ж, двигун інерційний, тому швидкість обертання збільшується не стрибкоподібно, а поступово, і потребує певного проміжку часу, щоб вийти на новий програмно встановлений рівень.

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.

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.

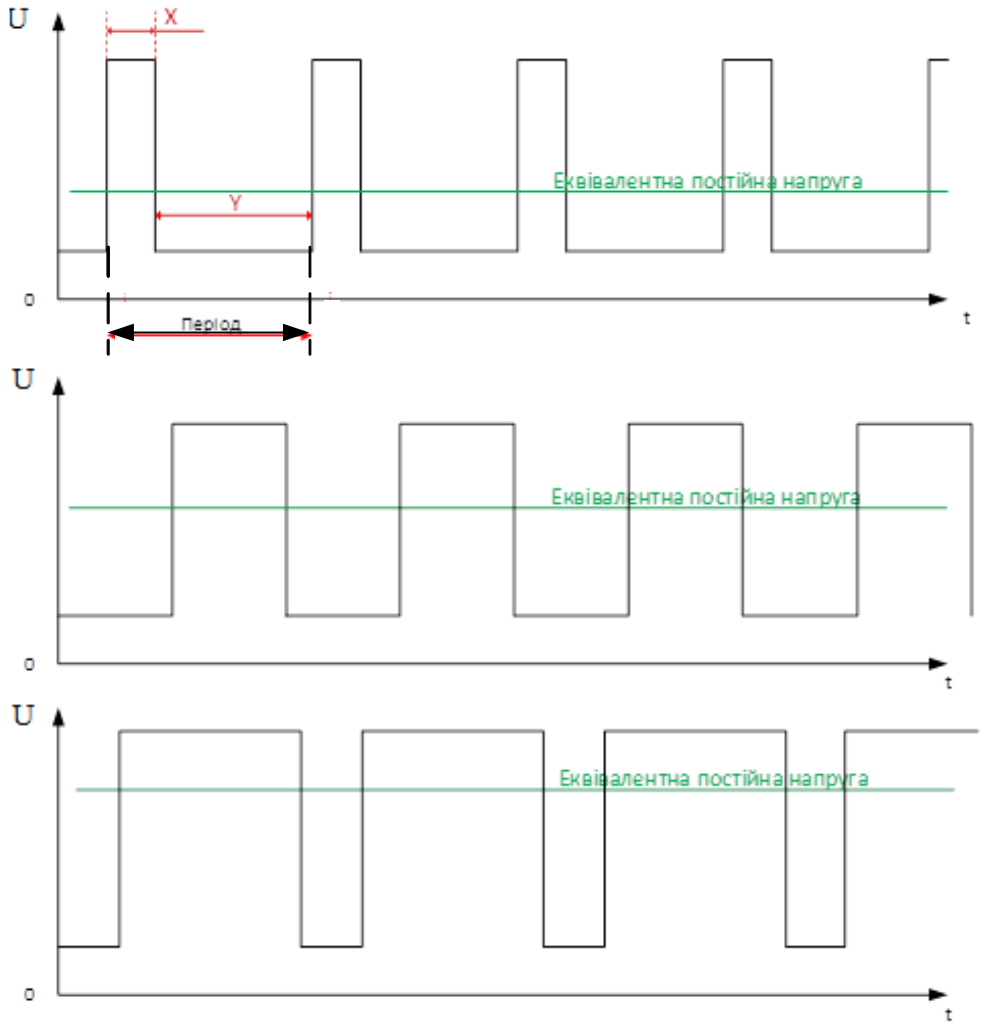


Рисунок 4.4 – Еквівалентна постійна напруга при ШІМ-сигналі

На рисунку 4.5 наведено підключення до мікроконтролера мікросхеми L293D.

Виводи 10 та 11 мікроконтролера, тобто PB0 та PB1 запрограмовані як виходи і підключені, відповідно, до входів IN1 та IN2 мікросхеми L293D (U2).

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

15-й вивід мікроконтролера – PB5 запрограмовано як вихід і підключено до входу EN1 мікросхеми L293D. З цього виходу мікроконтролера знімається ШІМ-сигнал. У свою чергу вхід EN1 відповідає за роботу першої мостової схеми у складі мікросхеми (4.3.1.5). Коли на ньому високий рівень, на перший міст подається живлення, яке підведено до виводу VS мікросхеми (рисунок 4.15). Коли на вході EN1 низький рівень, на перший міст напруга живлення не подається. Завдяки сигналу на цьому вході можна змінювати швидкість обертання двигуна.

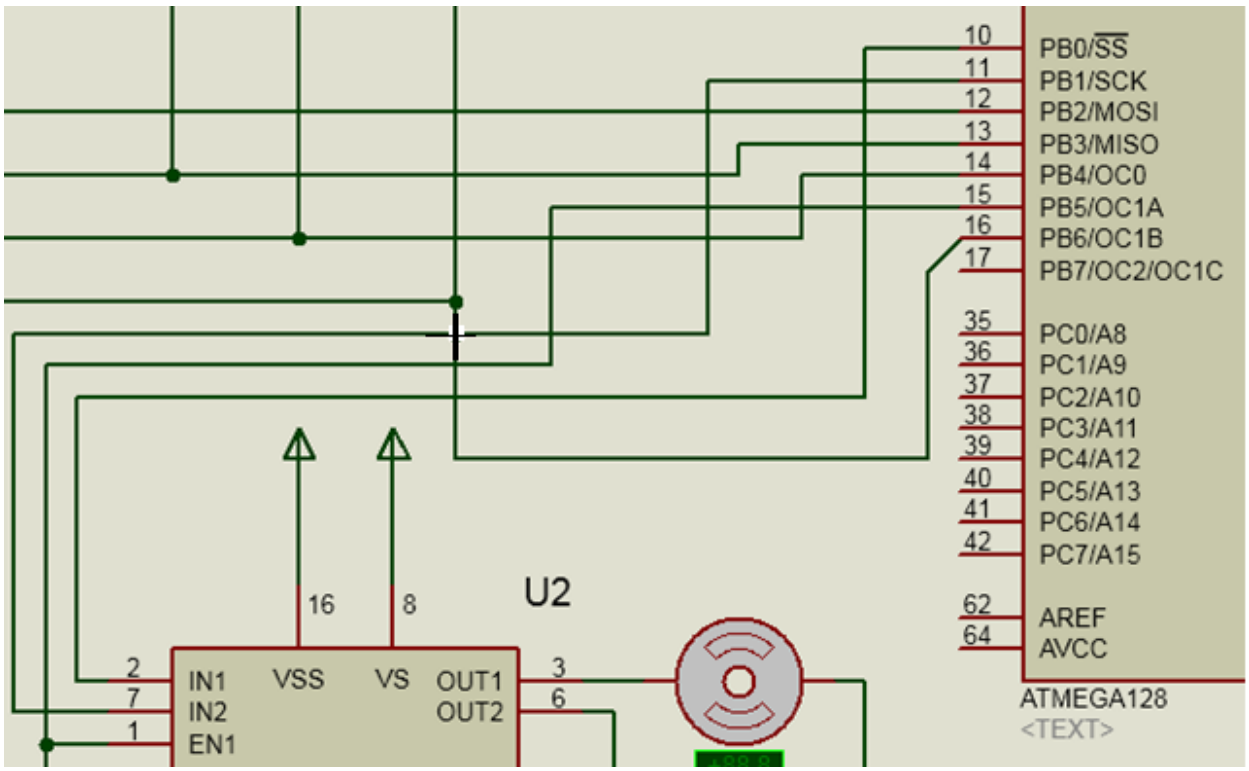


Рисунок 4.5 – Підключення мікросхеми L293D та мікроконтролера ATmega128

Два входи мікросхеми L293D – GND заземлено. На вхід VSS (рисунок 4.15) подається напруга живлення самої мікросхеми, а на вхід VS – напруга живлення двигуна.

Підключенням двигуна до мікросхеми L293D зображено на рисунку 4.6.

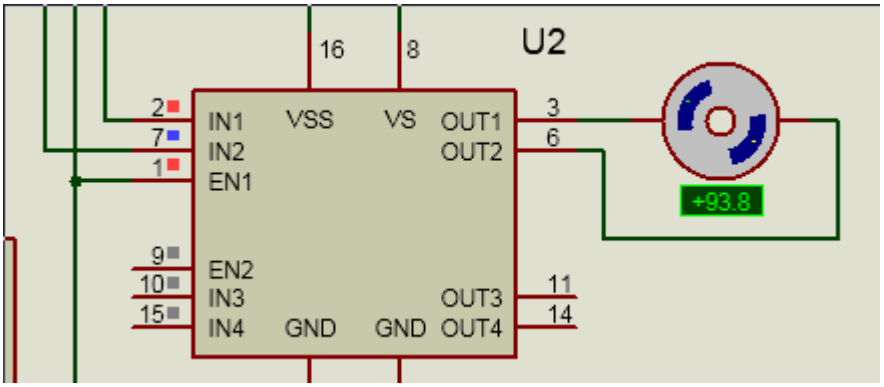


Рисунок 4.6 – Підключення двигуна постійного струму до мікросхеми L293D

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.

На рисунках 4.7…4.10 наведено деякі фрагменти, які демонструють роботу моделі.

На рисунку 4.11 наведено стан усієї системи після запуску сценарію.

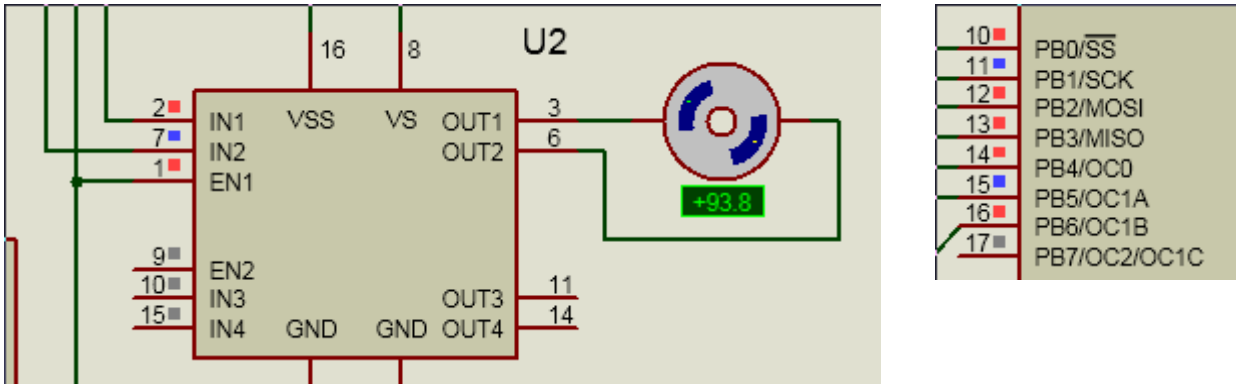


Рисунок 4.7 – Стан моделі після натискання кнопки «старт/стоп»

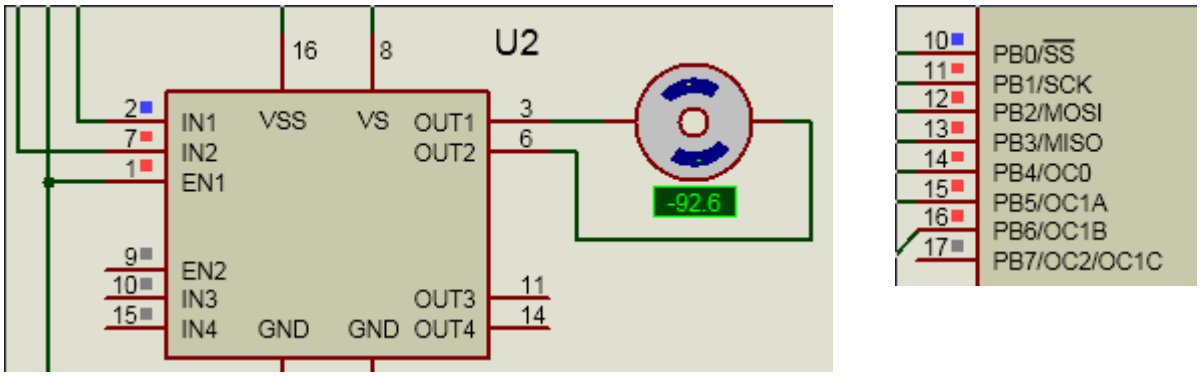


Рисунок 4.8 – Стан моделі після активації режиму реверсу

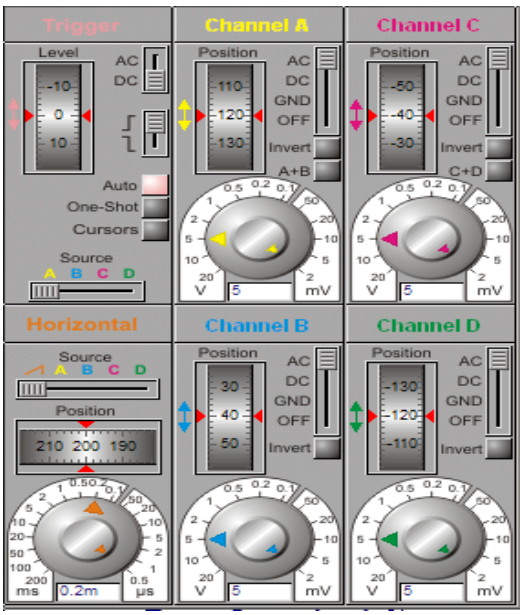


Рисунок 4.9 – Налаштування осцилографа

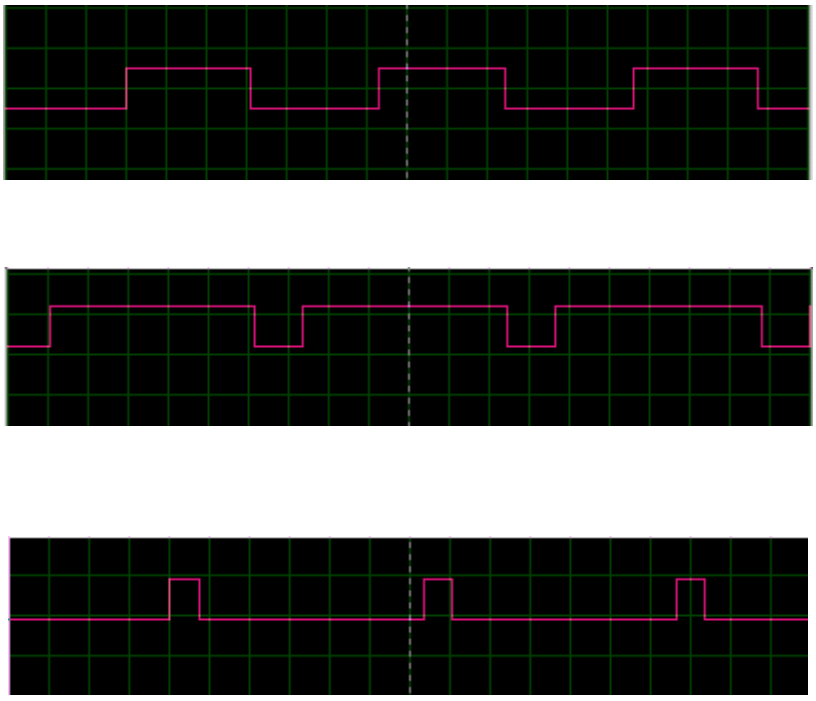


Рисунок 4.10 – ШІМ-сигнал, що подається на двигун (шпаруватість q = 2, q < 2, q > 2)

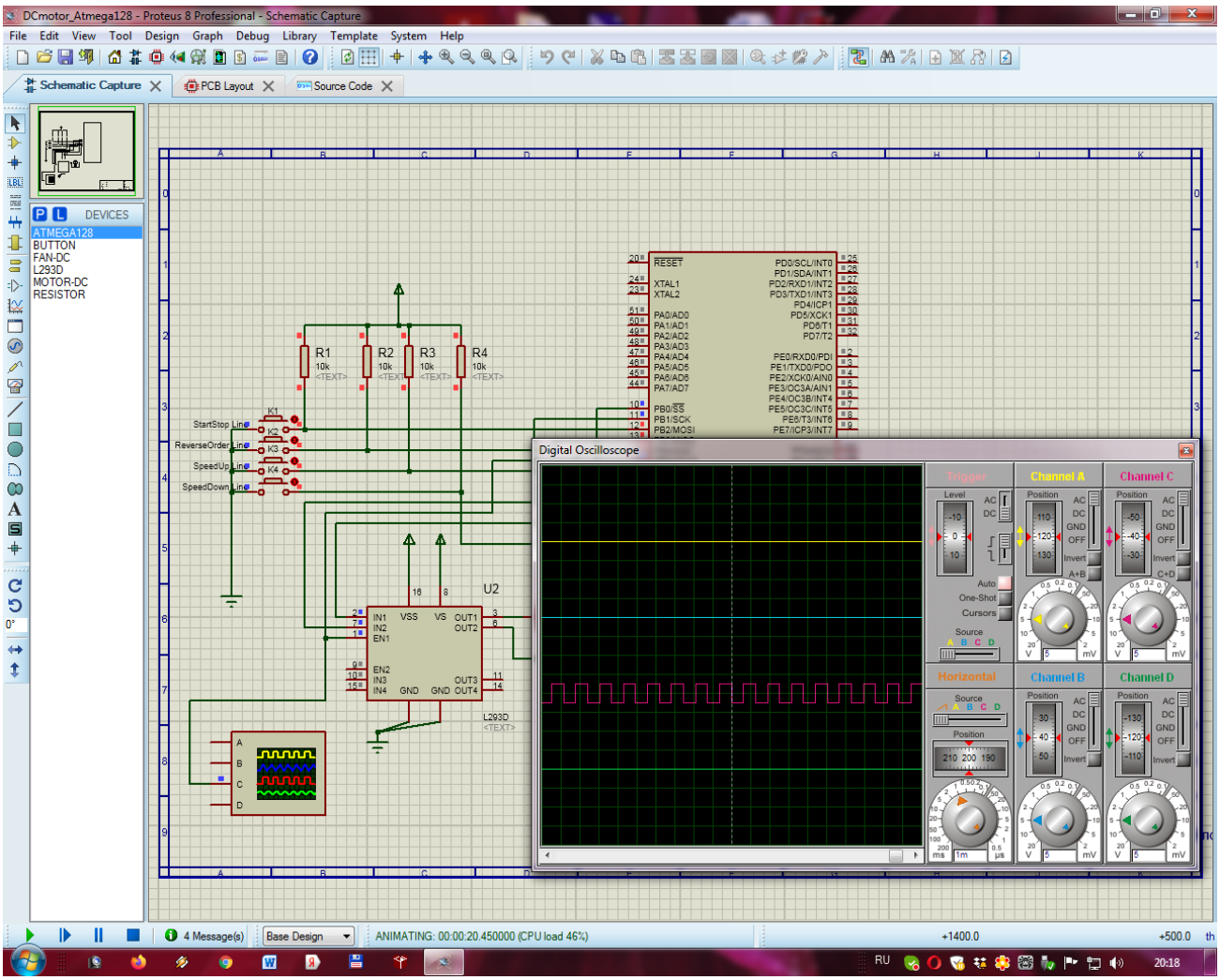


Рисунок 4.11 – Стан усієї системи після запуску сценарію

4.3.1.2.2 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]. 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) Високопродуктивний.
2) Споживає мало енергії.

3) Має розвинену RISC-архітектуру:

а) 131 основна команда, більшість з яких виконується за один машинний такт;

б) 32 робочих регістри загального призначення;

в) Повністю статичний режим роботи.

4) Має енергонезалежну пам’ять програм та даних.

5) Має 128 Кбайт внутрішньої Flash-пам’яті програм з кількістю циклів перепрограмування до 10 000.

6) Має 4096 байт EEPROM-пам’яті даних із кількістю циклів перепрограмування до 100 000.

7) Має 64 виводи типу вхід – вихід.

8) Має 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.

12) Вбудований аналоговий компаратор.

32) Спеціальні функції:

а) функція Reset для включення живлення і функція вимикання для зниження 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-цифрового перетворювача. 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;
– AREF – вхід підключення джерела опорної напруги 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.

4.3.1.2.3 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. Чим менше шпаруватість, а відповідно більше величина відношення «тривалість імпульсу/період», тим більша напруга надходить на двигун, який буде обертатись із більшою швидкістю.

Рекомендована частота сигналу з широтно-імпульсною модуляцією повинна перевищувати 20 кГц, що дозволяє виключити звукові ефекти у двигуні при зміні магнітного поля, яке в ньому утворюється [1].

4.3.1.2.4 Двигун

В моделі використовується електродвигун постійного струму. Двигуном можна керувати за допомогою широтно-імпульсної модуляції. Реалізувати широтну-імпульсну модуляцію можна з допомогою вбудованого в мікроконтроллер таймера. Вибір двигуна є важливим, оскільки від цього вибору буде залежати стабільність роботи системи. Реверсивний режим двигуна постійного струму здійснюється зміною полярності.

Двигунами постійного струму можна керувати за допомогою реле або транзисторів. Одиночне перемикання реле вмикає і вимикає двигун, а парне відповідає за напрямок обертання.

Регулювання швидкості обертання двигуна постійного струму 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.

4.3.1.2.5 Драйвер ШІМ

Для керування двигуном постійного струму нам потрібно підключати його до мікроконтролера за допомогою відповідної мостової схеми, яку зображено на рисунку 4.12.

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

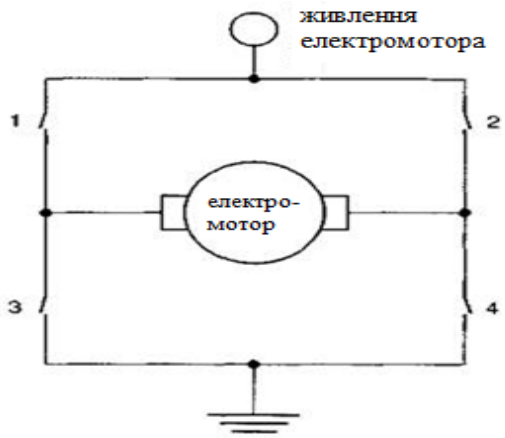


Рисунок 4.12 – Мостова схема підключення двигуна постійного струму до мікроконтролера

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 4.13.

Рисунок 4.13 – Зовнішній вигляд мікросхеми L293D

L293D включає одразу два драйвери для керування двома електродвигунами відносно невеликої потужності. Це відбувається через чотири незалежних канали, які об’єднано у дві пари. Мікросхема також має дві пари входів для сигналів керування і дві пари виходів для підключення електродвигунів. Крім того, у L293D є два входи для вмикання кожного з драйверів.

Ці входи використовуються для керування швидкістю обертання електродвигунів за допомогою широтно-імпульсної модуляції сигналів. Саме через це ця мікросхема нам добре підходить.

Також, L293D забезпечує розділене живлення для мікросхеми і для керованих нею двигунів, що дозволяє підключати електродвигуни з більшою

напругою живлення, чим у мікросхеми. Розділення живлення мікросхеми і електродвигунів нам також буде необхідним 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 розглянемо принцип роботи лише одного з них. Спрощений вид драйвера наведено на рисунку 4.14.

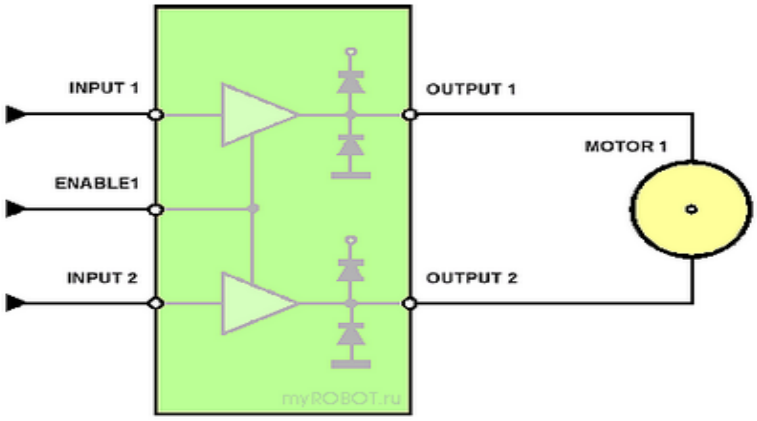


Рисунок 4.14 – Спрощений вид драйвера мікросхеми L293D

До виходів OUTPUT1 та OUTPUT2 підключається двигун постійного струму MOTOR1. На вхід ENABLE1, який відповідає за ввімкнення драйвера, подають керуючий сигнал, наприклад, під’єднують 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 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 подати сигнали одного рівня одразу на два керуючих входи INPUT1 та INPUT2, 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 наявності сигналів на входах INPUT1 та INPUT2 двигун також обертатися не буде. На вхід ENABLE1 подається імпульсний ШІМ-сигнал, який

формується відповідним модулем мікроконтролера. Змінюючи шпаруватість цього сигналу ми змінюємо постійну складову імпульсного сигналу, що в свою чергу змінює швидкість обертання двигуна.

На рисунку 4.14 показано нумерацію та позначення виводів мікросхеми L293D.

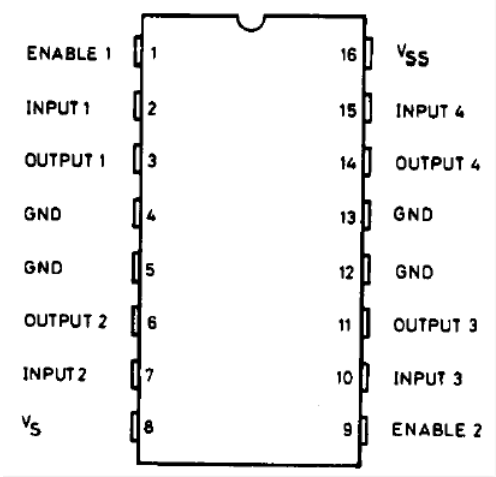


Рисунок 4.15 – Нумерація та позначення виводів мікросхеми L293D

Призначення виводів:

- 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 двигуном, який підключено до виходів OUTPUT1 та OUTPUT2;
- входи INPUT3 та INPUT4 керують другим двигуном, який підключено до виходів OUTPUT3 та OUTPUT4;
- 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. 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.

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;
– 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 і та спрощену внутрішню структуру мікросхеми зображено на рисунку 4.16.

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. Входи IN1 і IN2 (контакти 5 і 7) керують напрямком обертання двигуна для першого каналу, а IN3 і IN4 – другого. Емітери транзисторів з’єднано для підключення зовнішніх контролюючих датчиків.

Коли на вході EN A/B низький рівень, входи заблоковано і двигун не обертається. Якщо на цей вхід подати високий рівень, входи відкриваються. Входи IN1 і IN2 керують режимами роботи двигуна наступним чином:

– 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.

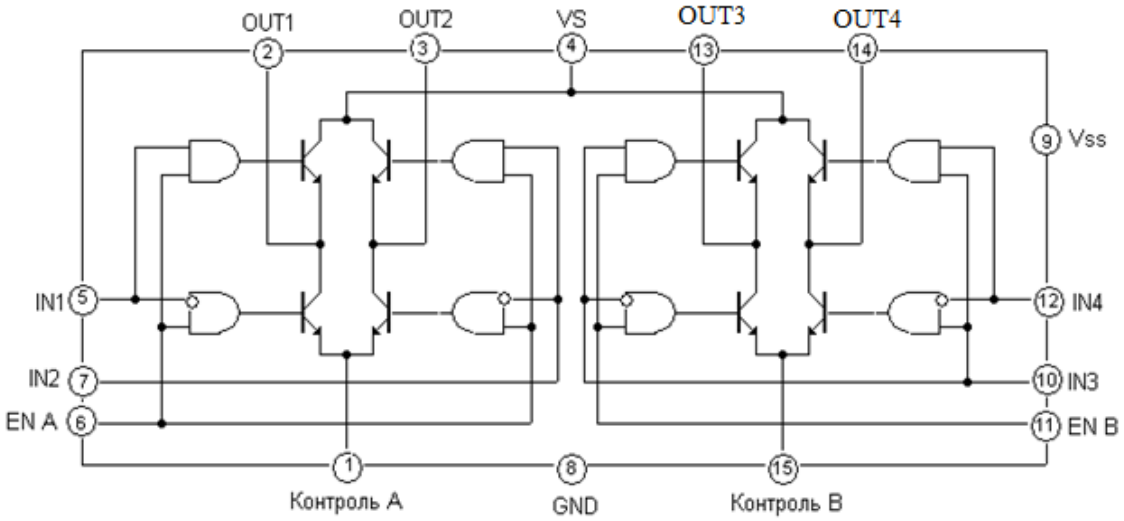


Рисунок 4.16 – Розташування видів і спрощена внутрішня структура мікросхеми L298N

Якщо, наприклад, вихідний струм, що надходить безпосередньо на обмотку двигуна постійного струму досягає величини до 5 А, то на вихід схеми ми ставимо додаткові пари транзисторів, шунтуючи їх також захисними діодами. Таким чином остаточна схема буде вигляд, показаний на рисунку 4.17.

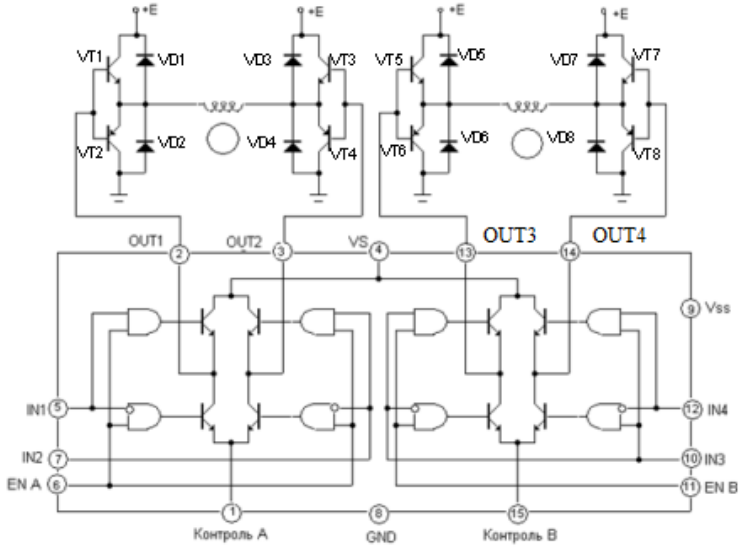


Рисунок 4.17 – Остаточна схема драйвера ШІМ

4.3.1.2.6 Захисні діоди

Як видно з рисунків 4.14, 4.17 схема драйверів містить 8 захисних діодів VD1…VD8. Ці діоди призначені 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 (ЕРС) індукції (ЕІ). Ця ЕРС протидіє зміні магнітного потоку (правило Ленца). Якщо, наприклад, струм у котушці різко збільшується (включення ключа), то наростаючий магнітний потік індукує в котушці ЕРС: ЕІ, під дією якої виникає струм, протилежний первісному струму і прагне загальмувати його. При цьому на ключі виникає від'ємний імпульс (рисунок 4.18, а) достатньо великої амплітуди, який може вивести ключ з ладу.

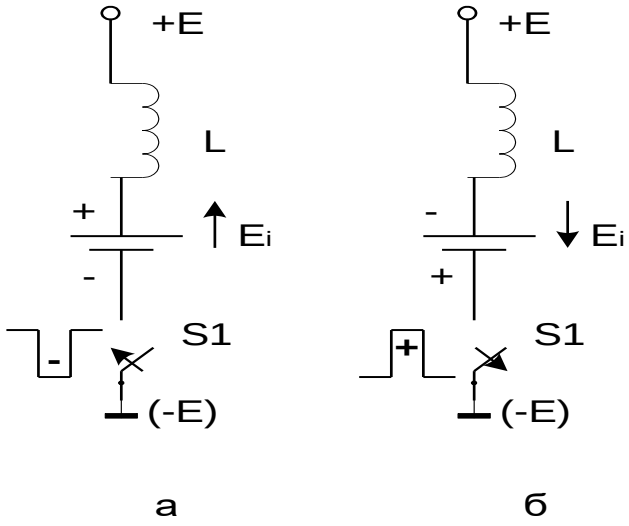


Рисунок 4.18 – Виникнення паразитних імпульсів при комутації ключів:
а – від'ємний імпульс при включенні ключа;
б – додатний імпульс при відключенні ключа

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

струм. При цьому на ключі виникає додатний імпульс (рисунок 4.18, б) достатньо великої амплітуди, який може вивести ключ з ладу.

4.3.1.2.7 Приклад програмування ШІМ-модуля мовою Асемблер

Вхідні дані

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

- тип МК-ра: ATmega 128;
- номер таймера: T/C 1;
- на виході PB5 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 4.19 наведено формування таймером ШІМ-сигналу в режимі Phase and Frequency Correct PWM.

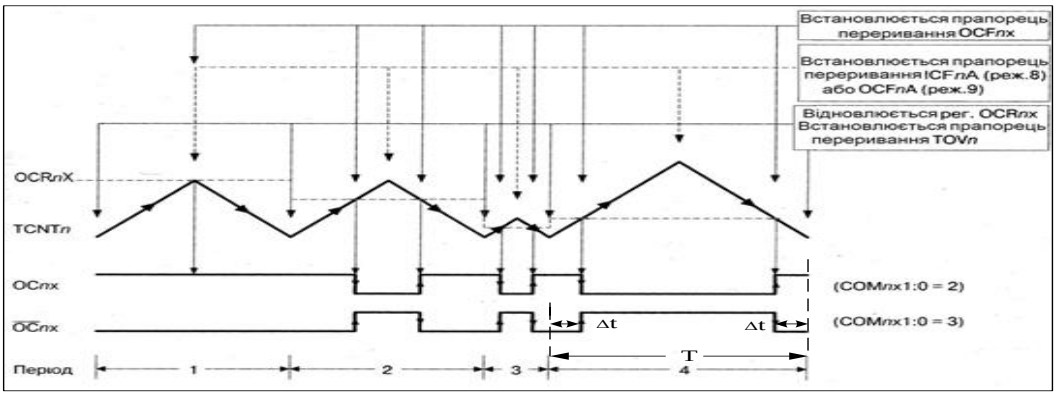


Рисунок 4.19 – Формування таймером ШІМ-сигналу в режимі Phase and Frequency Correct PWM

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

$$f_{PB5} = \frac{f_{CLK\ I/O}}{2N(TOP+1)}, \qquad (4.1)$$

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

Згідно таблиці 4.11 обираємо режим роботи таймера номер 8 [1].

Таблиця 4.11 – Режими роботи 16-розрядних таймерів/лічильників

Номер режиму	WGMn3	WGMn2	WGMn1	WGMn0	Режим роботи таймера/лічильника Tn	Модуль лічби (TOP)	Оновлення регістрів TOVn	Момент установки 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 при збігу)	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

Закінчення таблиці 4.11

Номер режиму	WGMn3	WGMn2	WGMn1	WGMn0	Режим роботи таймера/лічильника Tn	Модуль лічби (TOP)	Оновлення регістрів TOVn	Момент установки прапорця TOVn
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 при збігу)	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.

Тоді значення модуля лічби TOP визначається вмістом регістра ICR1. Значення f_{PB5} задано в завданні та дорівнює 25кГц. Згідно формули 4.1 при N = 8 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:

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

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

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

Згідно з рисунком 4.819

$$t_{імпPB5} = T_{PB5} - 2\,\Delta t. \qquad (4.2)$$

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

$$\Delta t = T_{CLKI/O} * N * OCR1A, \quad (4.3)$$

де OCR1A – регістр порівняння таймера 1.

Тоді:

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

Шпаруватість:

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

$$= \frac{1}{1 - 2 * \frac{1}{f_{CLKI/O}} * N * OCR1A * f_{PB5}}. \quad (4.4)$$

Розрахуємо значення шпаруватості Q при OCR1A = 20, яке повинно бути рівним 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 4.20), адреса якого згідно із таблицею 4.12 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)

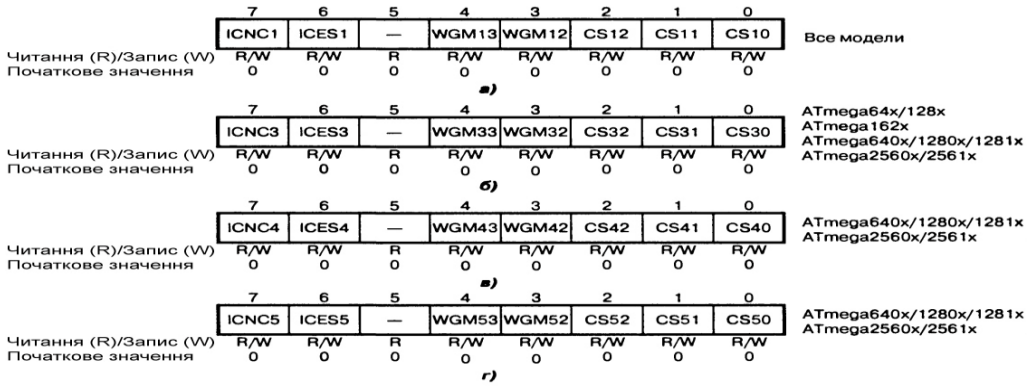


Рисунок 4.20 – Формат регістрів TCCR1B (а), TCCR3B (б), TCCR4B (в), TCCR5B (г)

Таблиця 4.12 – Регістри 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)						•		•	•	•	•
OCR1C	($79:$78)					•						
	($8D:$8C)											•

Закінчення табл. 4.12

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 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 4.21 формат регістра 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 В=$C0	

Для нашої задачі для формування інвертованого ШІМ-сигналу згідно таблиці 4.13 необхідно встановити біти COM1A1 = COM1A0 = 1.

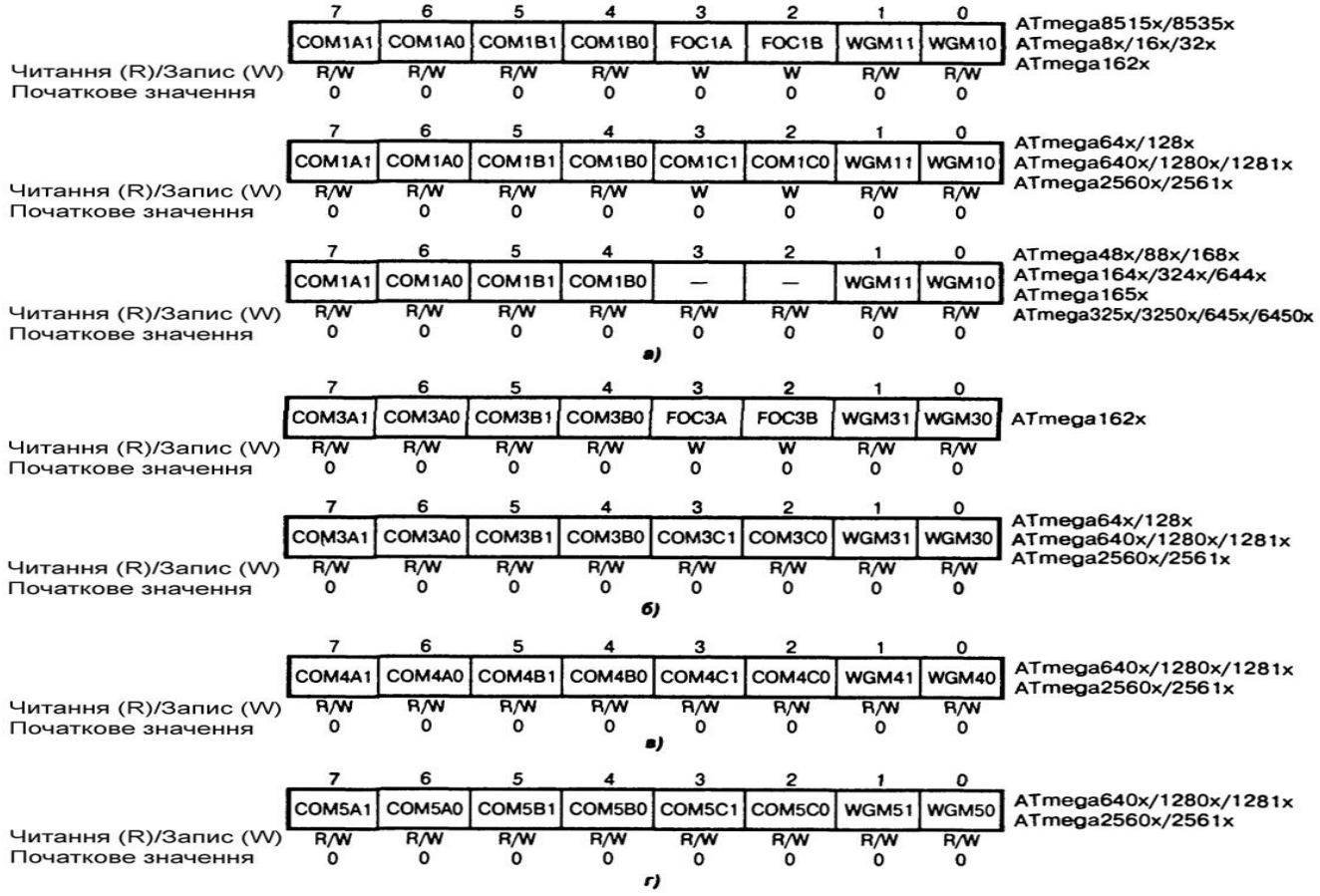


Рисунок 4.21 – Формат регістрів TCCR1A (а), TCCR3A (б), TCCR4A (в), TCCR5A (г)

Таблиця 4.13 – 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.

Для програмування режиму роботи 8 згідно таблиці 4.11 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 4.12 адреса регістра TCCR1A = $004F.

Тоді програма має вигляд:

```
LDI R17, $C0; R17← КС2 = $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 4.14 адреса регістра: $0037.

Тоді програма 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.

Таблиця 4.14 – Регістри 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 табл. 4.14

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
F	PORTF	$03 ($23)	–	–	–	–	($62)	–	–	$11 ($31)			
	DDRF	$02 ($22)	–	–	–	–	($61)	–	–	$10 ($30)			
	PINF	$01 ($21)	–	–	–	–	$00 ($20)	–	–	$0F ($2F)			
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 ICR1

Згідно таблиці 4.12 16-розрядний регістр ICR1 має адресу:

$0047 (СБ) : $0046 (МБ). В ICR1 треба завантажити: TOP = 40 = 00101000B = = $0028 (див. вище).

Тоді програма 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.
```

Завантаження регістра OCR1A

Згідно таблиці 4.12, 16-розрядний регістр OCR1A має адресу: $004B (СБ): $004A (МБ). В OCR1A треба завантажити: 20 = 00010100B = $0014 (див. вище).

Тоді програма 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/С 1

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 4.11).

Для програмування Кділ = 8 (табл. 4.15) та запуску таймера також використовують регістр TCCR1B (рисунок 4.20).

7 р.	6р.	5р.	4р.	3р.	2р.	1р.	0р.
ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
0	0	0	1	0	0	1	0B = $12

(режим 8) (Кділ = 8)

Таблиця 4.15 – 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 із табл. 4.12 (див. вище) дорівнює $004E.

Тоді програма має вигляд:

LDI R17, $12; R17← $12;

LDI R29, $00; R29← $00;

LDI R28, $4E; R28← $4E;

ST Y, R17; TCCR1B ← R17 = $12.

4.3.1.2.8 Схема алгоритму роботи моделі та керуюча програма

Схема алгоритму роботи моделі

Для моделювання пристрою керування двигуном постійного струму у пакеті PROTEUS 8.6 (рисунок 4.11) розроблено схему алгоритму роботи моделі (рисунок 4.22) та робочу програму мовою С, які наведено нижче.

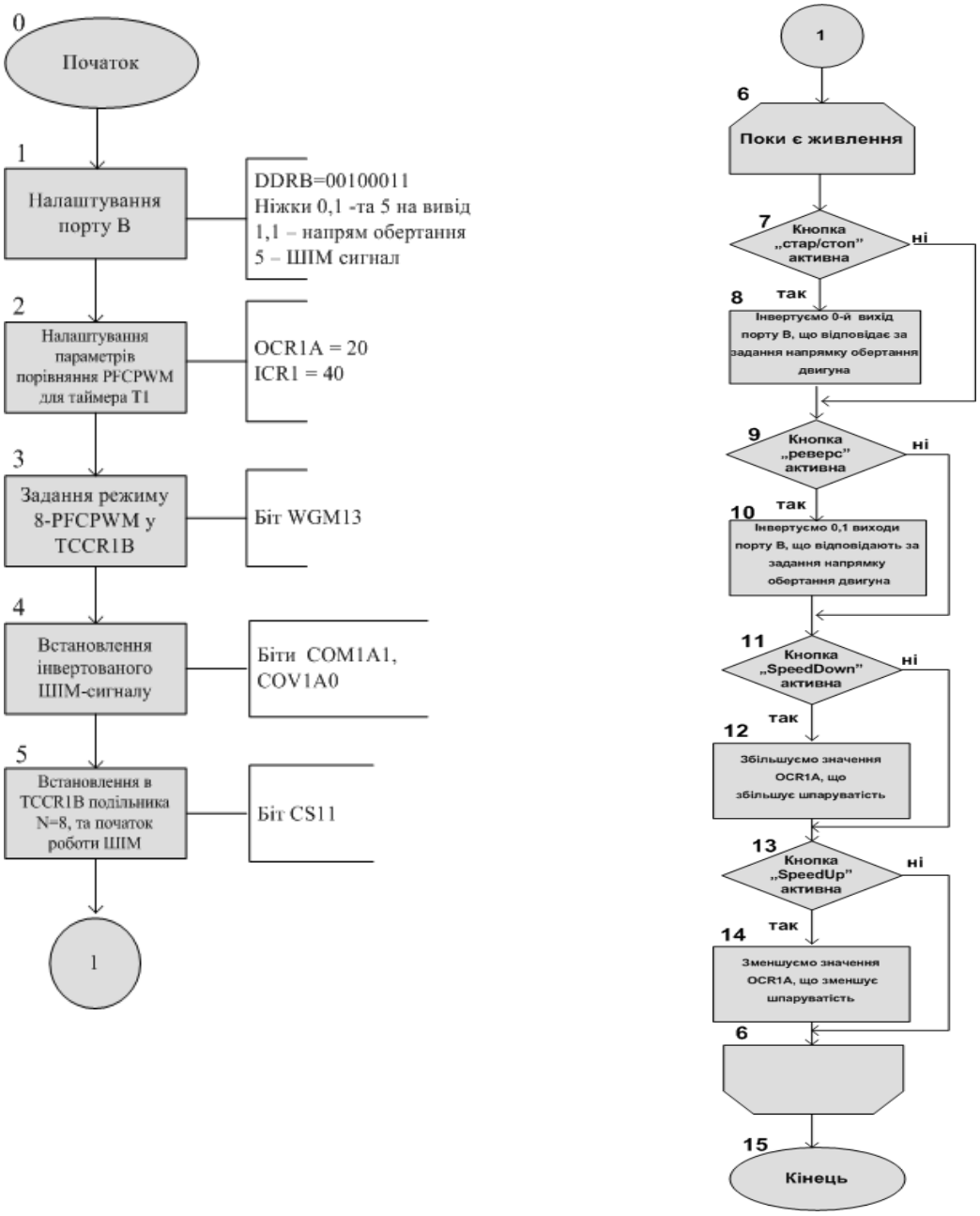


Рисунок 4.22 – Схема алгоритму роботи моделі

Керуюча програма 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;
// Init Phase and frequency correct PWM
TCCR1B = _BV(WGM13);  // 3
TCCR1A = 0;
TCCR1A |= _BV(COM1A0);// 4
TCCR1A |= _BV(COM1A1);
TCCR1B |= _BV(CS11);  // 5
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;
}
} // 6: End of loop
} // 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 1, то n-й вивід порту являє собою вихід, якщо ж цей біт скинутий у 0, то цей вивід функціонує як вхід. Загальний вигляд регістра DDRx показано на рисунку 4.23.


DDRx Register

Bit No.	7	6	5	4	3	2	1	0
Name	DDx7	DDx6	DDx5	DDx4	DDx3	DDx2	DDx1	DDx0
Initial Value	0	0	0	0	0	0	0	0

Upper Nibble Lower Nibble

Рисунок 4.23 – Вигляд регістра DDRx

В моделі виводи 0, 1 програмуються на виведення для передачі сигналів на входи драйвера двигуна, за допомогою яких відбувається керувати напрямом обертання двигуном постійного струму. Вивід 5 також програмується на вихід, оскільки з нього буде зніматися ШІМ-сигнал, який буде керувати швидкістю обертання двигуном постійного струму.

Тоді в регістр DDRB записується значення: 0b00100011.

Через можливе близьке знаходження двигуна до мікроконтролера і наведення завад вбудовані у мікроконтролер підтягуючі резистори використовувати не будемо, а використаємо в моделі 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 1, на виході встановлюється напруга високого рівня, коли ж він скинутий в 0, 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 1 підтягуючий резистор підключається між виводом мікроконтролера та лінією живлення. Загальний вигляд регістра PORTx показано на рисунку 4.24. В моделі регістр PORTB програмується нулями: PORTB = 0b00000000.

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 4.24 – 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 4.25.

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 4.25 – 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 схему

драйвера двигуна одразу після ініціалізації таймера. Проте на виводі порту, через який подається сигнал керування запуском/зупинкою двигуна постійного струму присутній високий рівень напруги. Після натискання кнопки на нульовий 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. Це дає змогу почергово змінювати рівні напруг на виході PB0, який використовується для керування 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 буде робитися 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. Лінію з низьким потенціалом будемо робити лінією з високим, а лінію з високим – лінією з низьким. Вище було відмічено, що після запуску двигуна з лінії PB0 знімається

високий рівень напруги, а з лінії PB1 – низький рівень напруги. Після натискання кнопки K2 (ReverseOrder Line) за допомогою операції побітового XOR ми будемо інвертувати перші два розряди регістра PORTB.

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 4.12

В даному випадку для нас важливі комбінації, коли є різниця потенціалів. 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 ми відкриваємо ключі за діагоналлю – наш 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 4.4).

Швидкість обертання двигуна залежить від напруги, яка на нього подається. На рисунку 4.4 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.

Тепер перейдемо до кнопок керування. Для збільшення швидкості використовується кнопка K3 (SpeedUp Line). Кнопку підключено до виводу PB4 мікроконтролера, який запрограмовано як вхід. Коли кнопку у початковому стані не натиснуто 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 встановлено в одиницю, і 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 наш двигун постійного струму буде обертається з максимальною швидкістю, яку ми можемо досягти при моделюванні.

Нижче наведено код на C, 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. Перевіряється рівень напруги на виводі PB6, який запрограмовано на вхід. В програмі зчитується рівень напруги на вході Px6 регістра PINB. 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;
   }
```

4.3.1.3 Зміст звіту

Звіт по роботі повинен містити:

– схему моделі;

– схему алгоритму роботи моделі;

– робочу програму;

– формули за потребою.

Контрольні запитання

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. У чому полягає функція вартового таймера?
15. Чим відрізняється розширений вартовий таймер від звичайного?

4.3.2 Лабораторна робота №5. Моделювання модуля АЦП

Тема: Моделювання модуля АЦП мікроконтролерів сім'ї AVR

Мета: Користуючись пакетом Proteus 8.6 дослідити роботу модуляАЦП

4.3.2.1 Порядок виконання роботи

– створити модель пристрою в пакеті Proteus 8.6

– розробити схему алгоритму роботи моделі та робочу програму

– створити hex-файл та підключити його до мікроконтролера

– запустити модель та виконати її дослідження згідно методичних вказівок

– зробити відповідні висновки.

4.3.2.2 Стислі теоретичні відомості

4.3.2.2.1 Особливості модуля АЦП у складі мікроконтролера ATmega32

До складу більшості моделей AVR-мікроконтролерів входить модуль 10-розрядного аналого-цифрового перетворювача (АЦП) послідовного наближення [1; 10]. Нижче 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 8/16-канальний аналоговий мультиплексор, що визначає кількість аналогових каналів перетворення. Деякі мікроконтролери, наприклад ATmega8x, мають тільки 6 каналів перетворення, два з яких – ADC4 та ADC5 є 8-розрядними.

У більшості моделей входи АЦП можуть також об`єднуватися попарно для формування каналів з диференціальним входом.

Для деяких каналів є можливість 10- та 200-кратного попереднього підсилення вхідного сигналу. При коефіцієнтах підсилення 1x та 10x діюча роздільна здатність АЦП складає 8 розрядів, а при коефіцієнті підсилення 200x – 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 пристрій вибірки-зберігання (ПВЗ,) що під час перетворення підтримує напругу на вході АЦП на постійному рівні.

4.3.2.2.2 Опис функціональної схеми модуля АЦП

На рисунку 4.26 наведено функціональну схему модуля АЦП. В деяких моделях мікроконтролерів елементи і пов'язані з ними сигнали, які виділено на цьому рисунку сірим кольором, відсутні, а не інвертуючий вхід компаратора з пристроєм вибірки-зберігання підключено безпосередньо до виходу мультиплексора (показано пунктирною лінією).

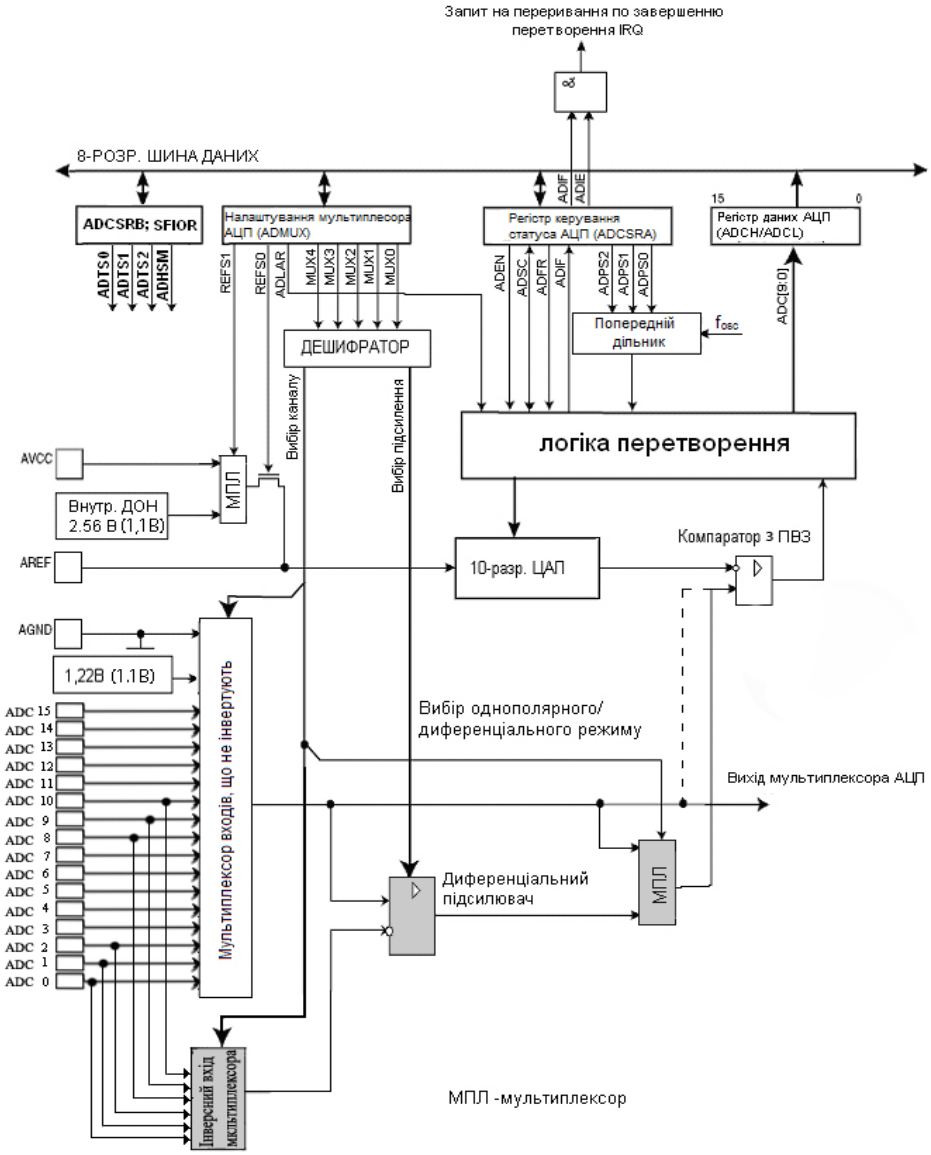


Рисунок 4.26 – Функціональна схема аналого–цифрового перетворювача

Вхідний аналоговий сигнал перед перетворюванням зберігається у ПВЗ. АЦП працює за принципом послідовного наближення [10]. 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…ADC15. При диференціальному режимі використовуються шість входів мультиплексора (ADC0, ADC1, ADC2, ADC8, ADC9, ADC10). Різниця сигналів між цими входами підсилюється за допомогою диференціального підсилювача і поступає на вхід компаратора.

4.3.2.2.3 Програмне керування модулем АЦП

Як приклад для деяких моделей сімейства AVR у таблиці 4.16 наведено регістри, що використовуються для керування модулем АЦП. Формат регістрів ADCSRA (ADCSR) і ADMUX наведено на рисунках 4.17 і 4.18 а короткий опис функцій їх розрядів наведено у таблицях 4.17 і 4.18 відповідно.

Таблиця 4.16 – 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)												♦	

Таблиця 4.17 – 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 6)
* В моделі ATmega8x ** В моделях ATmega8x, ATmega128x		

Таблиця 4.18 – Розряди регістра ADMUX

Розряд	Назва	Опис	Модель
7..6	REFS1:REFS0	Вибір джерела опорної напруги (див. таблицю 5)	Всі моделі
5	ADLAR	Ліве вирівнювання результату (див. таблицю 11)	Всі моделі
4	–	Зарезервовано	ATmega8x
	MUX4	Вибір вхідних каналів (див. таблицю 9)	Всі моделі крім ATmega8x
3..0	MUX3..MUX0	Вибір вхідних каналів і частоти перетворення (див. таблиці 8, 9)	Всі моделі

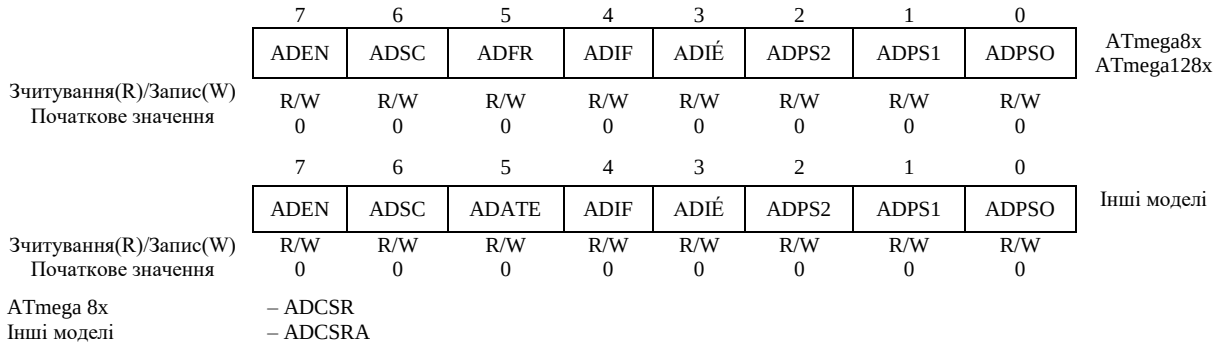


Рисунок 4.27 – Формат регістра ADCSRA (ADCSR)

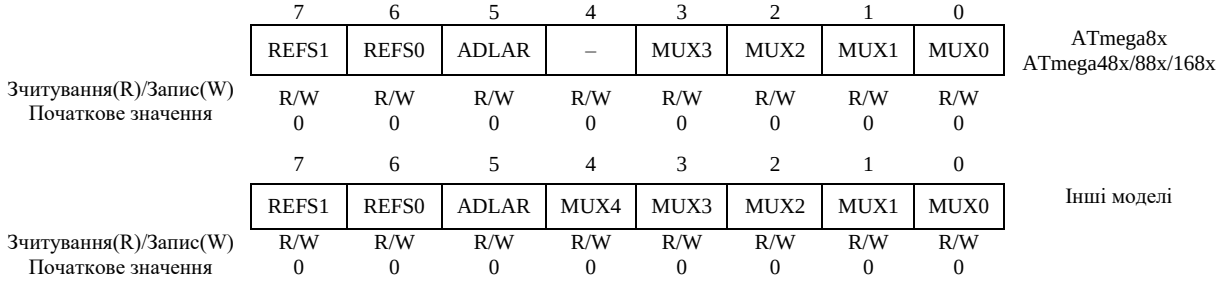


Рисунок 4.28 – Формат регістра ADMUX

Формат регістрів ADCSRB і SFIOR наведено рисунках 4.29 і 4.30 відповідно (не використовувані розряди регістра SFIOR позначені як «–»).

Для дозволу робити АЦП необхідно записати логічну 1 у розряд ADEN регістра ADCSRA (ADCSR), а для заборони – відповідно логічний 0. Якщо АЦП буде вимкнено під час циклу перетворення, то перетворення не буде завершено (в регістрі даних АЦП залишиться результат попереднього перетворення).

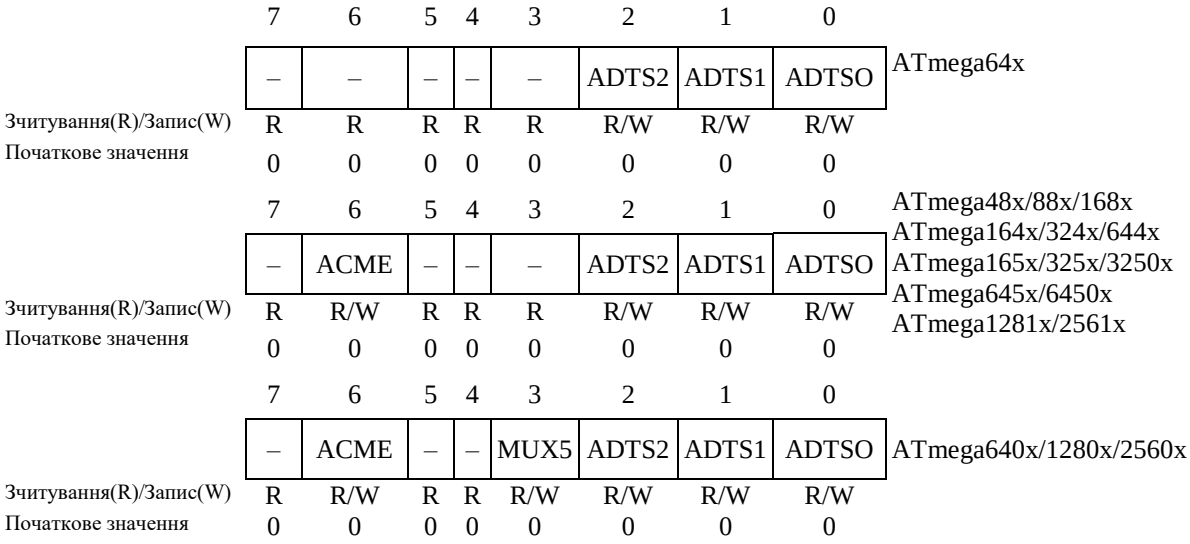


Рисунок 4.29 – Формат регістра ADCSRB

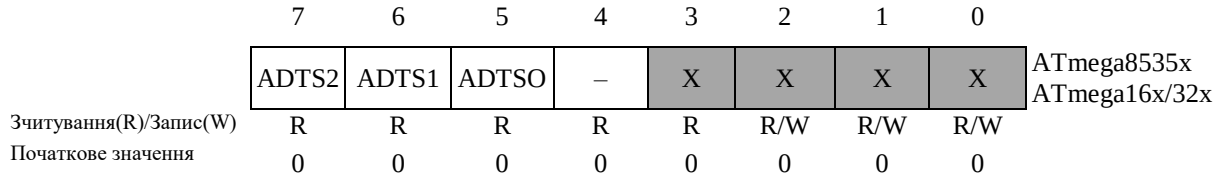


Рисунок 4.30 – Формат регістра SFIOR

У більшості моделей модуль АЦП може використовуватись як мультиплексор аналогових сигналів для аналогового компаратора. Для цього

необхідно встановити розряд ACME (Analog Comparator Multiplexer Enable) в одиницю [1; 2]. Інверсний вхід аналогового компаратора може бути з'єднано, або з його входом AIN1, або з будь-яким із входів АЦП в залежності від значення розряду ADEN (одиниця – AIN1, нуль – один із входів АЦП).

В моделях ATmega8x та ATmega128x режим роботи АЦП визначається станом розряду ADFR (таблиця 4.17). Якщо його встановлено в одиницю, АЦП працює в режимі безперервного перетворення. У цьому режимі запуск кожного наступного перетворення здійснюється автоматично після закінчення поточного перетворення.

Якщо розряд ADFR скинуто в нуль, АЦП працює в режимі одиночного перетворення та запуск кожного перетворення здійснюється командою користувача.

В більшості моделей, окрім ATmega8x та ATmega128x, 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 або ADCSRB.

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 4.19.

Таблиця 4.19 – 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 T0
1	0	0	Переривання за переповненням таймера/лічильника T0
1	0	1	Переривання за подією "Збіг В" таймера/лічильника T1
1	1	0	Переривання за переповненням таймера/лічильника T1
1	1	1	Переривання за подією "Захоплення" таймера/лічильника T1

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 4.20).

Як зазначено в таблиці 4.20, до виводу AREF мікроконтролера може бути підключено внутрішнє ДОН. Тому при його використанні для підвищення завадозахищеності до виводу AREF треба підключити зовнішній фільтруючий конденсатор.

Таблиця 4.20 – 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. Мінімальне значення відповідає рівню GND, а максимальне рівню AREF мінус 1 молодшого розряду.

Канал однополярного або диференціального аналогового введення та каскад диференціального підсилення обираються шляхом програмування розрядів MUXn

(n = 0,1…4) у регістрі ADMUX. У якості однополярного аналогового входу АЦП в залежності від моделі мікроконтролера може бути обраний один із входів ADC0...ADC7. У режимі диференціального введення передбачена можливість вибору входів, що інвертують і не інвертують диференціального підсилювача.

Якщо обрано диференціальний режим аналогового введення, то диференціальний підсилювач буде помножувати різницю напруг між обраною парою входів на заданий коефіцієнт підсилення. Підсилене в такий спосіб значення надходить на аналоговий вхід АЦП. Якщо обирається однополярний режим аналогового введення, то каскад підсилення пропускається.

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 4.18).

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 лог. 1 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 лог. 1 у розряд ADFR (ADATE) регістра ADCSR (ADCSRA). Перше перетворення ініціюється шляхом запису лог. 1 у розряд ADSC регістра ADCSR (ADCSRA). У даному режимі АЦП виконує послідовні перетворення, незалежно від того скидається прапорець переривання АЦП ADIF чи ні.

4.3.2.2.4 Формування тактової частоти АЦП

Для формування тактового сигналу для АЦП використовується попередній дільник, схему якого наведено на рисунку 4.31.

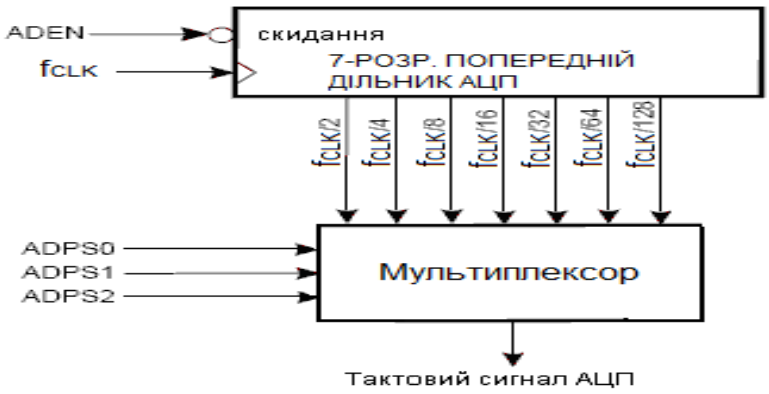


Рисунок 4.31 – Схема попереднього дільника АЦП

Попередній дільник формує похідні частоти відносно частоти синхронізації мікроконтролера. Коефіцієнт ділення встановлюється за допомогою розрядів ADPSn у регістрі ADCSRA (таблиця 4.21).

Попередній дільник починає лічбу з моменту включення АЦП встановленням розряду ADEN у регістрі ADCSRA. Попередній дільник 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 4.21 – 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

4.3.2.2.5 Часові діаграми роботи АЦП

На рисунках 4.32…4.35 наведено часові діаграми роботи АЦП у різних режимах.

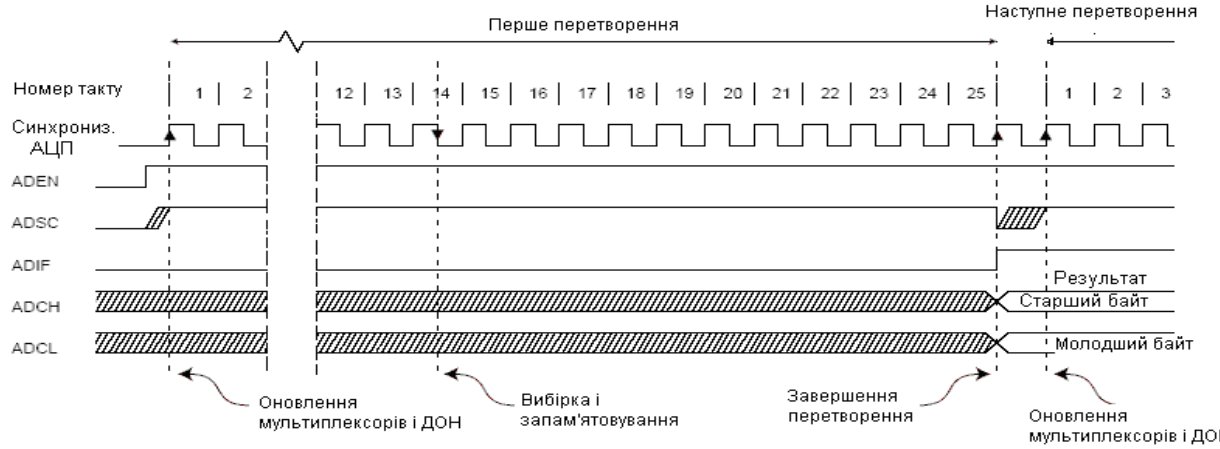


Рисунок 4.32 – Часові діаграми роботи АЦП при першому перетворенні в режимі одиночного перетворення

Рисунок 4.33 – Часові діаграми роботи АЦП у режимі одиночного перетворення

Рисунок 4.34 – Часові діаграми роботи АЦП у режимі запуску за перериванням

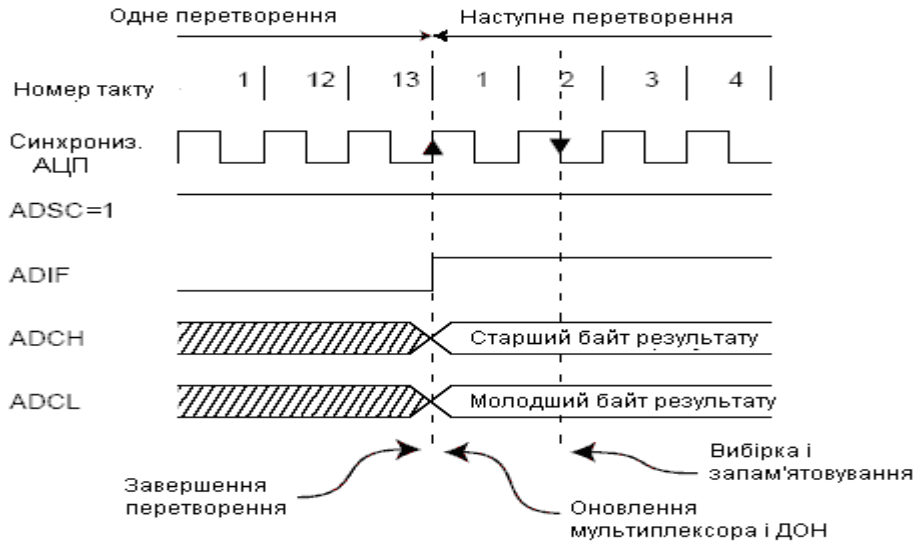


Рисунок 4.35 – Часові діаграми роботи АЦП у режимі автоматичного перезапуску для однополярного перетворення

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 4.22.

Таблиця 4.22 – 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

4.3.2.2.6 Керування вхідним мультиплексором

Виводи мікроконтролера, які підключені до входу АЦП, визначаються станом розрядів MUX4...MUX0 регістра ADMUX (рисунок 4.28), та регістра ADCSRB (рисунок 4.29). Для каналів з диференціальним входом зазначені розряди

визначають також коефіцієнт попереднього підсилення вхідного сигналу. Для мікроконтролера ATmega32 керування вхідним мультиплексором відображає таблиця 4.23.

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.

Таблиця 4.23 – 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				

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.

4.3.2.2.7 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 4.24 наведено приклади вирівнювання результату вліво та вправо.

Таблиця 4.24 – 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.

4.3.2.2.8 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 (1,1 В) або зовнішнє ДОН, що підключено до виводу AREF. AVCC підключається до АЦП через пасивний ключ. Внутрішня опорна напруга 2,56 В (1,1 В) генерується внутрішнім еталонним джерелом VBG, що буферизовано внутрішнім підсилювачем. У кожному разі зовнішній вивід AREF зв’язаний

безпосередньо з АЦП і, тому, можна знизити вплив шумів на опорне джерело за рахунок підключення конденсатора між виводом AREF і спільним виводом. Напруга $U_{ДОН}$ також може бути виміряна на виводі AREF вольтметром з високим вхідним опором. Зверніть увагу, що $U_{ДОН}$ є високоомним джерелом і, тому, зовні до нього може бути підключене тільки ємнісне навантаження.

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 (1,1 В) як опорне джерело. Результат першого перетворення після перемикання опорного джерела може характеризуватися низькою точністю, тому користувачеві рекомендується його ігнорувати.

4.3.2.2.9 Результат перетворення АЦП

Для каналів з однополярним (несиметричним) входом результат перетворення визначається виразом:

$$ADC = \frac{1023 U_{IN}}{U_{REF}}, \qquad (4.5)$$

де U_{IN} – значення вхідної напруги, U_{REF} – величина опорної напруги, ADC – десятковий еквівалент двійкового коду на виході АЦП.

Коефіцієнт передачі АЦП визначається формулою:

$$К_{ПЕР} = 1023/U_{REF} \text{ [МЗР/мВ]}. \qquad (4.6)$$

На рисунку 4.36 представлено функцію перетворення АЦП в однополярному режимі.

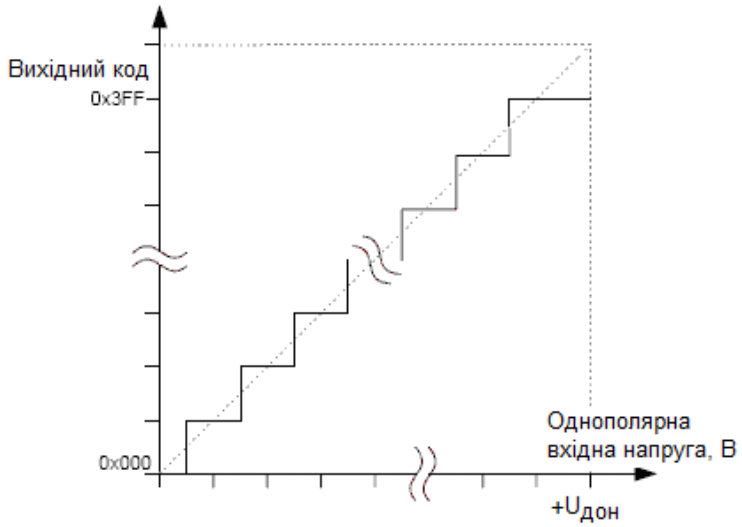


Рисунок 4.36 – Функція перетворення АЦП при зміні однополярного сигналу

Код 0x000 відповідає рівню аналогової землі, а 0x3FF – рівню напруги ДОН мінус 1 крок квантування за напругою.

Зв’язок між вхідним сигналом й вихідними кодами для однополярного режиму відображає таблиця 4.25.

Таблиця 4.25 – 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 рівна 2,56 В, тоді:

КодАЦП = 1024 * 1000 / 2560 = 400 = 0x190.

Для каналів з диференціальним входом результат перетворення описується в [1; 2].

4.3.2.2.10 Моделювання модуля АЦП у пакеті PROTEUS 8.6

Схема моделі

Схему моделювання модуля АЦП у пакеті PROTEUS 8.6 зображено на рисунку 4.37.

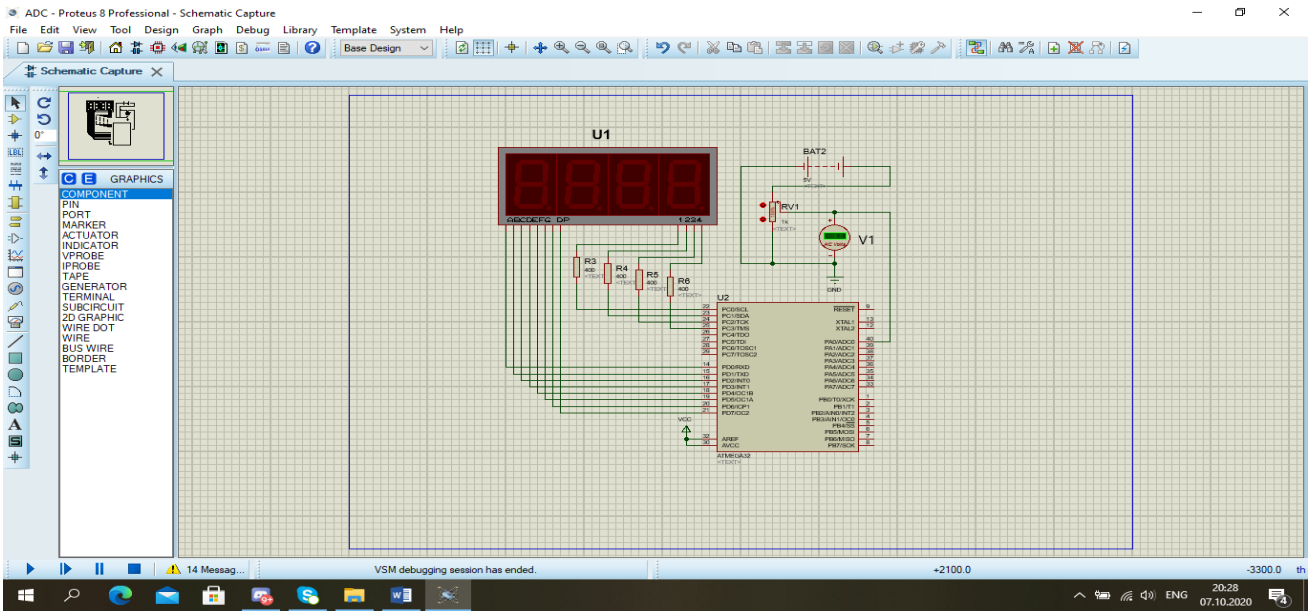


Рисунок 4.37 – Схема моделювання модуля АЦП у пакеті PROTEUS 8.6

Дана схема складається з наступних елементів:

– U2 – AVR мікропроцесор ATMEGA32;
– R1…R4 – резистори опором 400Ом;
– RV1 – резистор змінного опору на 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 4.36), а таблиця 4.25 відображає зв'язок між вхідним сигналом та вихідними кодами.

Для каналів з однополярним (несиметричним) входом результат перетворення визначається виразом (4.5), а коефіцієнт передачі – виразом (4.6), 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

$$ADC = 5000 \cdot 0{,}2046 = 1023.$$

Нижче на рисунку 4.38 наведено схему моделі після запуску процесу моделювання.

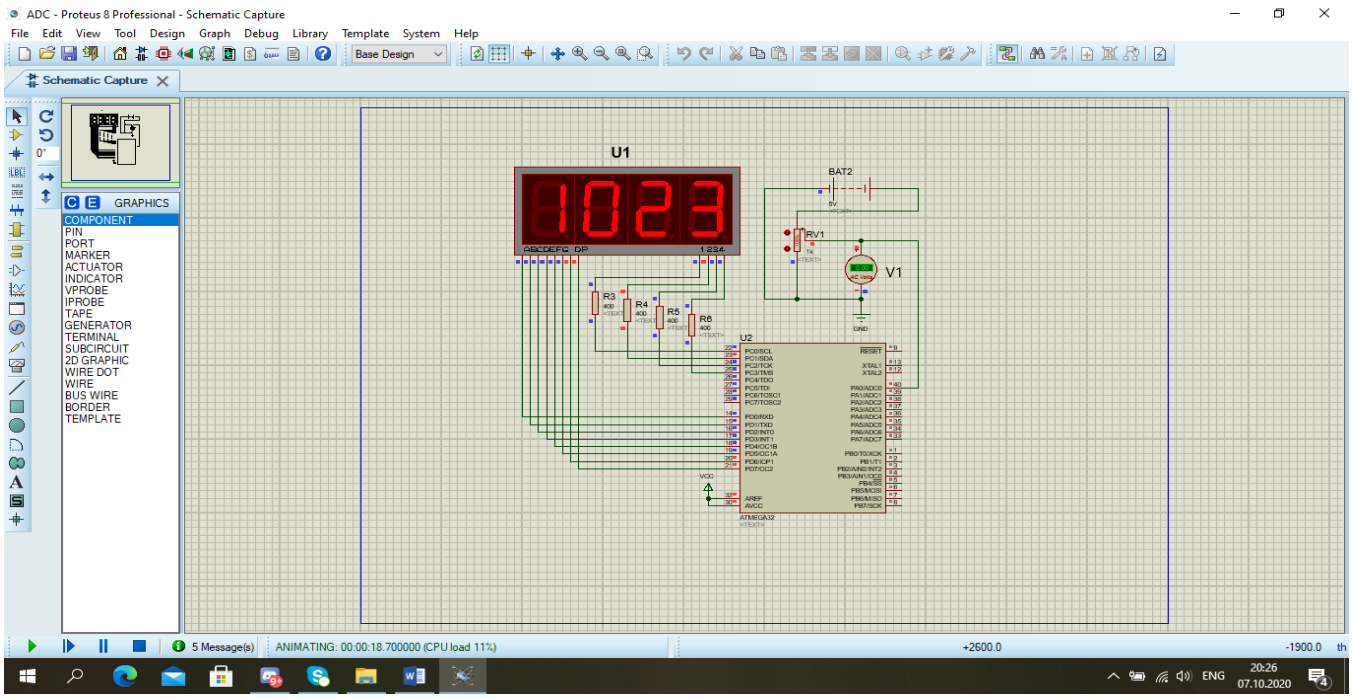


Рисунок 4.38 – Схема моделі АЦП після запуску процесу моделювання

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).

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 – переддільник на 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 C 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 C дорівнюють нулю). При виборі другого індикатора PC0 = 2, інші біти порту C 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 T2 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 T2 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.

4.3.2.2.11 Схема алгоритму роботи моделі

Схему алгоритму роботи моделі наведено на рисунках 4.39…4.42.

Після виконання блоку ініціалізації (блок 1) АЦП запускається в режим безперервного перетворення. Паралельно починає роботу таймер/лічильник T2, 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 4.39.

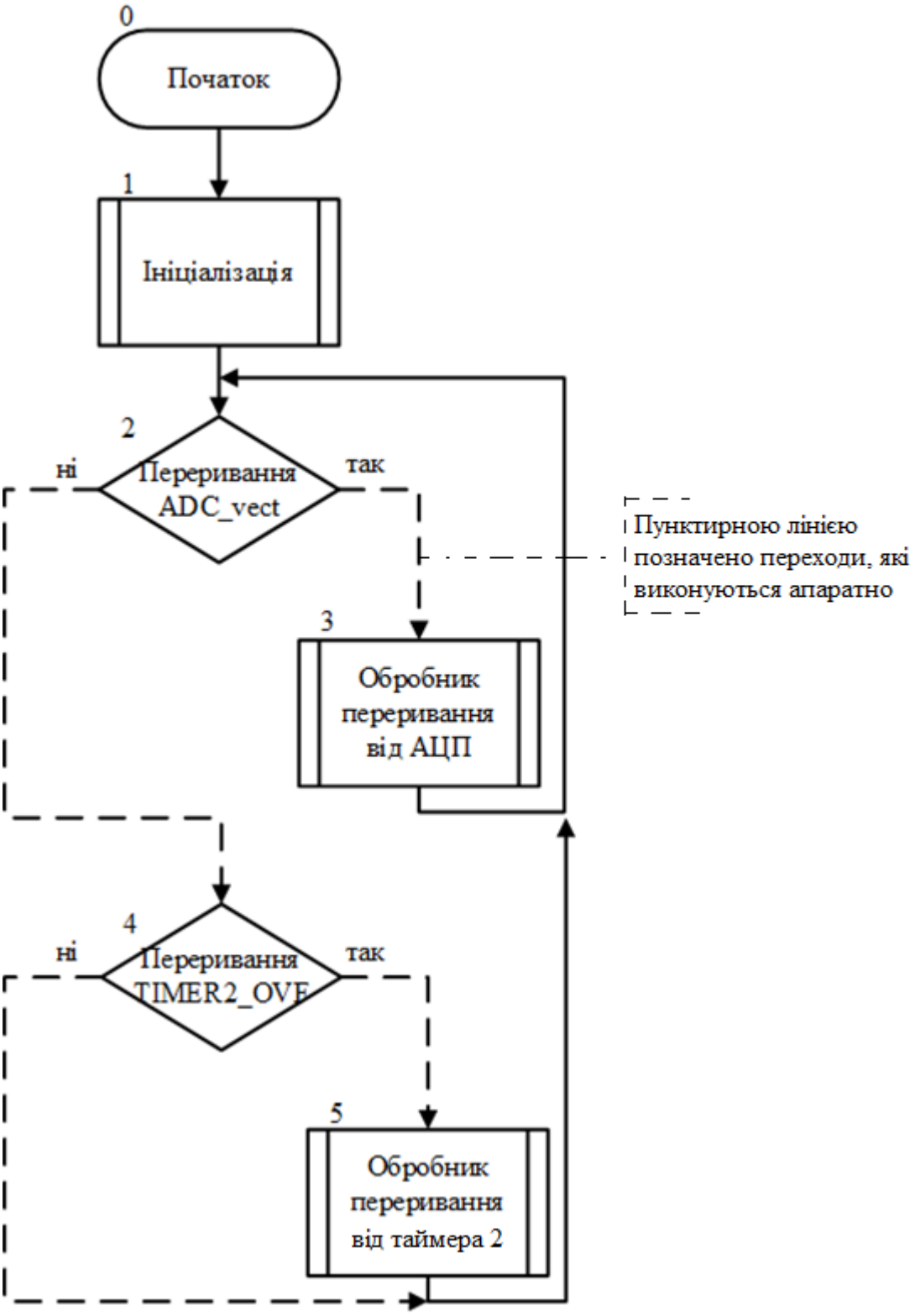


Рисунок 4.39 – Схема загального алгоритму роботи модуля

Схему алгоритму ініціалізації наведено на рисунку 4.40.

Рисунок 4.40 – Схема алгоритму ініціалізації

Схему алгоритму обробки переривання від таймера 2 наведено на рисунку 4.41.

Рисунок 4.41 – Схема алгоритму обробки переривання від таймера 2

Схему алгоритму обробки переривання завершення перетворення АЦП наведено на рисунку 4.42.

Рисунок 4.42 – Схема алгоритму обробки переривання завершення перетворення АЦП

4.3.2.2.12 Робоча програма мовою програмування С

```
// 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 - Обнулення портів семисегментних індикаторів
    DDRC = 0xFF;
    PORTC = 0x00;
    DDRD = 0xFF;
    PORTD = 0x00;
```

```
    // Блок 1.2 – Налаштування Таймера 2
    TIMSK |= (1 << TOIE2); // Дозвіл переривання по таймеру 2
    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); // Блок 1.6   Затримка 50 мс
    }
}

// Блок 2 – Умова переривання від АЦП
ISR (ADC_vect)
{// Блок 3 – Обробник перериванння 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 він за
межами
}
```

Нижче наведено пояснення фрагменту програми виведення на чотири семисегментних індикатори результату перетворення АЦП – ADC, який надходить у десятковому коді згідно з формулою 5.1, що 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.

4.3.2.3 Зміст звіту

Звіт по роботі повинен містити:

– схему моделі;

– схему алгоритму роботи моделі;

– робочу програму;

– формули за потребою.

Контрольні запитання

1. До складу яких моделей AVR-мікроконтролерів входить модуль АЦП? Назвіть його розрядність.
2. Назвіть основні параметри АЦП.
3. Назвіть основні джерела опорної напруги для АЦП.
4. Назвіть режими роботи АЦП.
5. Яку роль у функціональній схемі модуля АЦП виконують: дешифратор, мультиплексор та пристрій вибірки та зберігання?
6. У якому регістрі зберігається результат перетворення?
7. Поясніть як саме відбувається перетворення вхідного сигналу у двійковий код?
8. Яку роль у функціональній схемі модуля АЦП виконує компаратор ?
9. Який принцип перетворення використовується в АЦП? Поясніть особливості цього принципу.
10. Який регістр відповідає за налаштування мультиплексора АЦП?
11. Наведіть та поясніть формати регістрів стану і керування АЦП.
12. Який розряд регістра ADCSRA відповідає за дозвіл роботи АЦП?
13. Як обрати джерело опорної напруги?
14. Як вибрати режим роботи АЦП?
15. За якими перериваннями може відбуватися запуск АЦП?
16. Яку функцію виконує попередній дільник?
17. Як формується тактовий сигнал АЦП?
18. Який регістр використовується для налаштування вхідного мультиплексора?
19. Поясніть часові діаграми роботи АЦП при першому одиночному перетворенні в режимі одиночного перетворення.
20. Поясніть часові діаграми роботи АЦП в режимі наступного одиночного перетворення.
21. Поясніть часові діаграми роботи АЦП в режимі запуску за перериванням.
22. Поясніть часові діаграми роботи АЦП в режимі автоматичного перезапуску для однополярного перетворення.
23. Скільки тактів синхронізації АЦП витрачається на перетворення в режимах: першого одиночного перетворення та наступного одиночного перетворення?
24. Скільки однополярних входів має вхідний мультиплексор модуля АЦП?
25. Як можна програмно врахувати величину зсуву?
26. Чому після перемикання диференціального каналу підсилювача перетворення не повинно стартувати не раніше, ніж через 125 мкс?
27. Назвіть способи вирівнювання результату АЦП.

28. Яке вирівнювання слід використовувати якщо досить точності 8-розрядного значення?
29. Які особливості перемикання каналів при одиночному перетворенні?
30. Які особливості перемикання каналів при перетворенні у режимі автоматичного перезапуску?
31. Що може виступати у ролі джерела опорної напруги?
32. Який вихідний опір повинно мати джерело вхідного сигналу?
33. Яким буде результат перетворення для каналів з однополярним входом?
34. Яким буде результат перетворення для каналів з диференціальним входом?
35. Якою формулою визначається коефіцієнт передачі АЦП?
36. Поясніть функцію перетворення АЦП при зміні однополярного сигналу.
37. Поясніть зв'язок між вхідним сигналом та вихідними кодами для однополярного режиму.
38. Як можна відключити вхідні цифрові буфери на виводах ADC0...ADC15 у випадку, якщо відповідні виводи використаються тільки для зчитування аналогових сигналів?
39. Назвіть методи підвищення точності перетворення АЦП.
40. Чому дорівнює похибка зсуву та як її можна програмно врахувати?
41. Що таке інтегральна нелінійність?
42. Що таке диференціальна нелінійність?
43. Опишіть відмінності в програмах при моделюванні модуля АЦП та цифрового вольтметра.
44. Опишіть особливості програмування мовою С виведення на дисплей результату роботи модуля АЦП та цифрового вольтметра.
45. Як розраховується відносна похибка АЦП від квантування за рівнем?
46. Опишіть схему моделювання модуля АЦП в пакеті PROTEUS 8.6.
47. В який режим програмується модуль АЦП при його моделюванні?
48. Чому дорівнює коефіцієнт передачі модуля АЦП при його моделюванні?
49. В якому вигляді виводиться на індикацію результат моделювання модуля?
50. За допомогою якої математичної операції визначається відповідна цифра при її виведенні на індикатор? Відповідь пояснити.
51. Який таймер/лічильник мікроконтролера використовується при виведенні на семисегментні індикатори?
52. Який сигнал використовується в якості тактового для таймера 2? Відповідь пояснити.
53. Чим відрізняється виведення результату моделювання цифрового вольтметра від моделювання модуля АЦП?

4.3.3 Лабораторна робота №6 Моделювання цифрового вольтметра

Тема: Моделювання цифрового вольтметра

Мета: Користуючись пакетом PROTEUS дослідити моделювання цифрового вольтметра

4.3.3.1 Порядок виконання роботи

– створити модель пристрою в пакеті Proteus 8.6

– розробити схему алгоритму роботи моделі та робочу програму

– створити hex-файл та підключити його до мікроконтролера

– запустити модель та виконати її дослідження згідно методичних вказівок

– зробити відповідні висновки.

4.3.3.2 Стислі теоретичні відомості

4.3.3.2.1 Особливості архітектури модуля АЦП у складі мікроконтролера ATmega32

Основним елементом цифрового вольтметра є аналого–цифровий перетворювач, в якості якого використовується відповідний модуль мікроконтролера ATmega32. Архітектуру цього модуля цього модуля розглянуто у підрозділах 4.3.2.1…4.3.2.9.

4.3.3.2.2 Опис моделі

Робочу модель цифрового вольтметра показано на рисунку 4.43.

З лівого боку моделі зображено 7-сегментний чотирьохпозиційний цифровий дисплей, 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. 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.

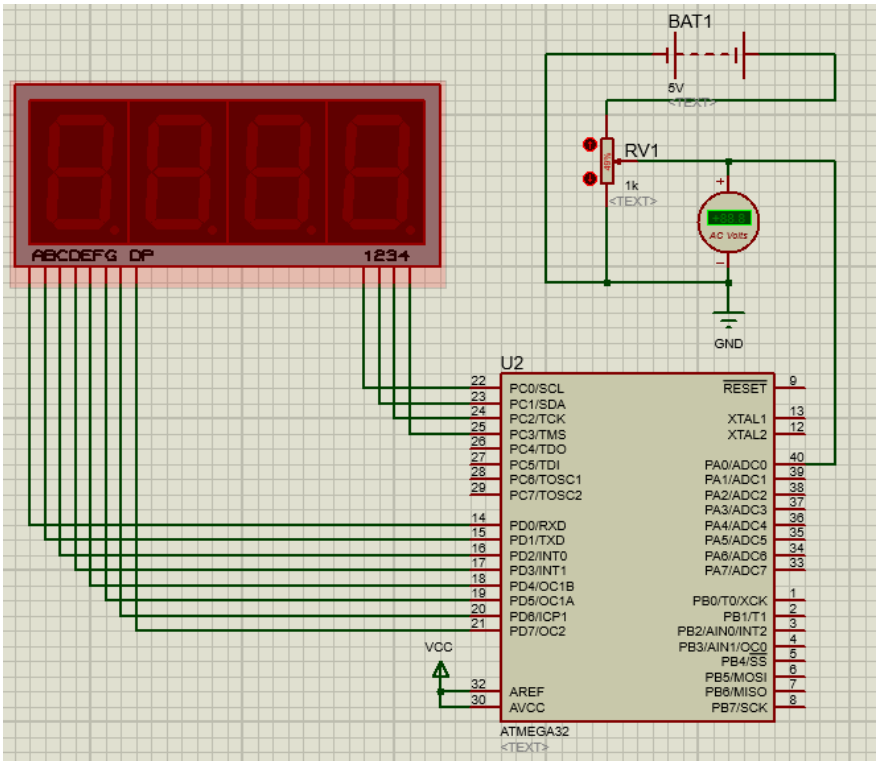


Рисунок 4.43 – Схема моделі цифрового вольтметра

На рисунку 4.44 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 4.26, 4.27.

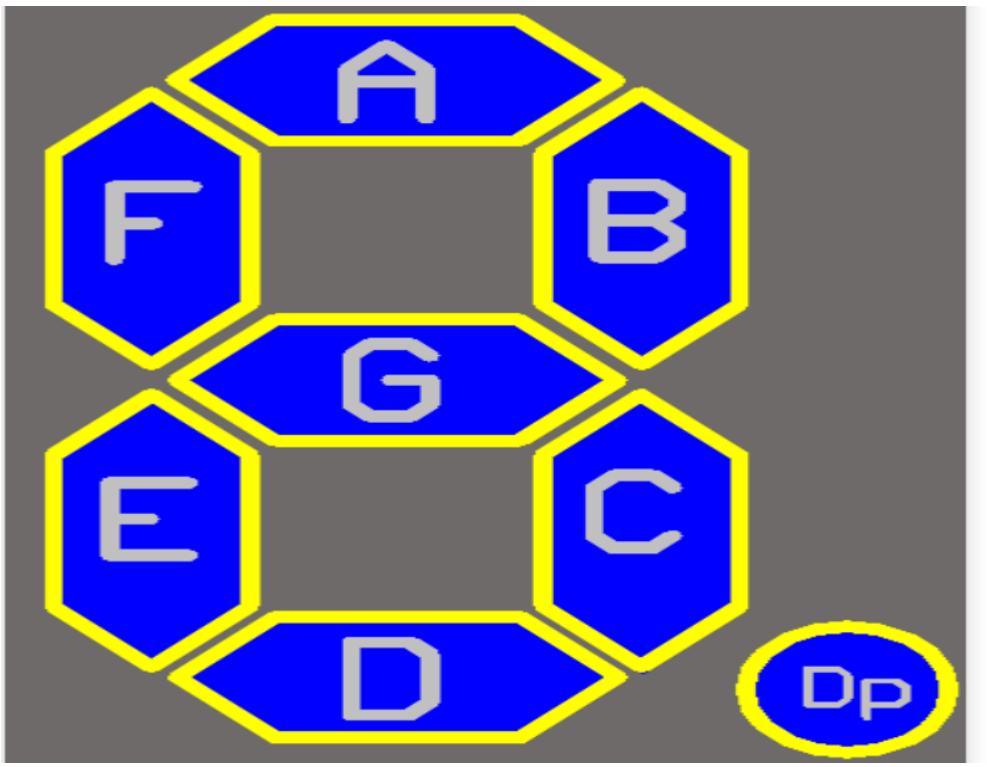


Рисунок 4.44 – 7-сегментний дисплей

Таблиця 4.26 – 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 4.27 – 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 4.45, 4.46 наведено декілька прикладів роботи моделі.

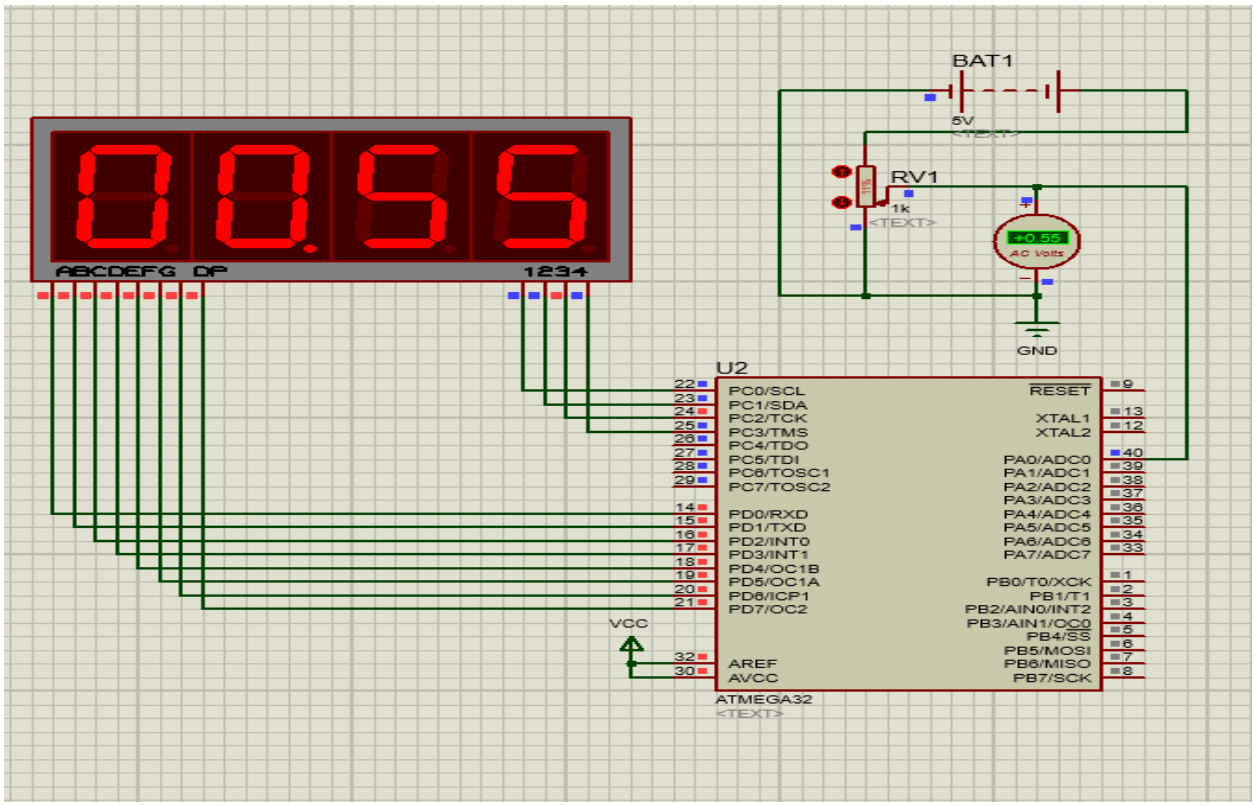


Рисунок 4.45 – Робота цифрового вольтметра при $U_{ВХ}$ = 0,55В

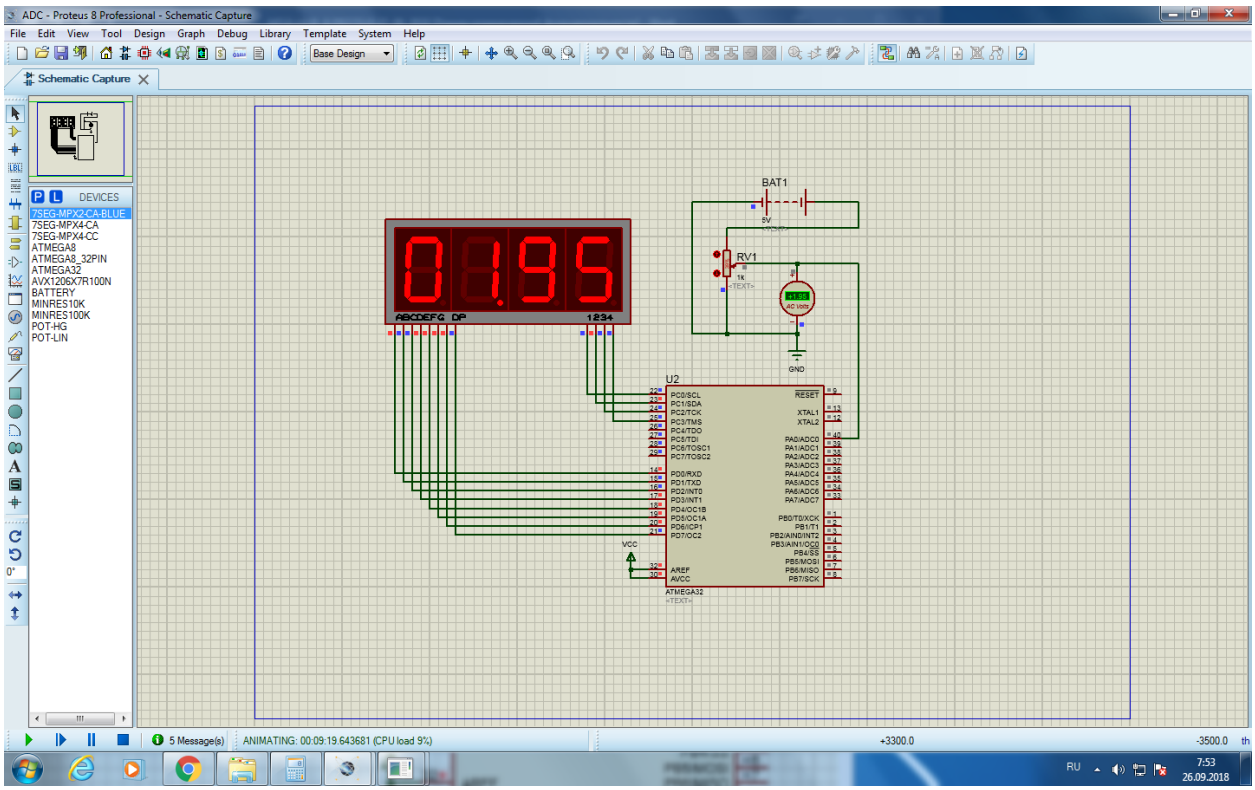


Рисунок 4.46 – Робота цифрового вольтметра при U_{BX} = 1,95В

Схема алгоритму роботи модуля

Схема алгоритму роботи цифрового вольтметра подібна схемі алгоритму роботи модуля АЦП, яку наведено вище на рисунках 4.39…4.42.

Ініціалізація модуля АЦП, таймера Т2 мікроконтролера та робота моделі у більшості виконується аналогічно описанному у підрозділі 4.3.2.10. 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 4.3.2.9 для каналів з однополярним (несиметричним) входом результат перетворення АЦП визначається виразом:

$$ADC = 1023 \bullet U_{IN}/U_{REF},$$

де UIN – значення вхідної напруги, UREF – величина опорної напруги, ADC – десятковий еквівалент двійкового коду на виході АЦП.

Коефіцієнт передачі АЦП

$$К_{ПЕР} = 1023/U_{REF} \text{ [МЗР/мВ]}.$$

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 5.4.10) 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 5.4.10) 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.

Робоча програма мовою С

```
// 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 – Обнулення портів семисегментних індикаторів
   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 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 – 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
    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
}
```

4.3.3.3 Зміст звіту

Звіт по роботі повинен містити:

– схему моделі;

– схему алгоритму роботи моделі;

– робочу програму;

– формули за потребою.

Контрольні запитання

1. Опишіть відмінності в програмах при моделюванні модуля АЦП та цифрового вольтметра.
2. Опишіть особливості програмування мовою С виведення на дисплей результату роботи модуля АЦП та цифрового вольтметра.
3. Як розраховується відносна похибка АЦП від квантування за рівнем?
4. Опишіть схему моделювання модуля АЦП в пакеті PROTEUS 8.6.
5. В який режим програмується модуль АЦП при його моделюванні?
6. Чому дорівнює коефіцієнт передачі модуля АЦП при його моделюванні?
7. В якому вигляді виводиться на індикацію результат моделювання модуля?
8. За допомогою якої математичної операції визначається відповідна цифра при її виведенні на індикатор? Відповідь пояснити.

9. Який таймер/лічильник мікроконтролера використовується при виведенні на семисегментні індикатори?
10. Який сигнал використовується в якості тактового для таймера 2? Відповідь пояснити.
11. Чим відрізняється виведення результату моделювання цифрового вольтметра від моделювання модуля АЦП?

4.3.4 Лабораторна робота №7. 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

4.3.4.1 Порядок виконання роботи

– створити модель пристрою в пакеті Proteus 8.6

– розробити схему алгоритму роботи моделі та робочу програму

– створити hex-файл та підключити його до мікроконтролера

– запустити модель та виконати її дослідження згідно методичних вказівок

– зробити відповідні висновки.

4.3.4.2 Стислі теоретичні відомості

4.3.4.2.1 Опис моделі

Робочу модель дослідження універсального асинхронного приймача–передавача (УАПП) наведено на рисунку 4.47.

Розберемо цю модель докладніше. Зліва ми бачимо вісім кнопок, за допомогою яких можна задавати байт (8 біт) даних, який ми будемо передавати через модуль УАПП.

Кнопки підключено до восьми ліній порту PA мікроконтролера: PA.0, PA.1, … , PA.7. Якщо якась із кнопок відпущена, то через резистори R11...R18 на відповідну лінію порта PA подається логічна 1. Якщо кнопка нажата, то вводиться логічний нуль. Праворуч зображено вісім світлодіодів, які підключено до ліній порта PC: PC.0, PC.1, … , PC.7. Катоди світлодіодів підключено до спільного

провода (землі), а на аноди через резистори R2 … R9, які обмежують струм, із виходів порту PC 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 ми будемо відображати повідомлення, які вводиться та виводяться через модуль УАПП в/з мікроконтролера.

До виводів XTAL1 та XTAL2 підключають кварцовий резонатор частотою, яка визначає тактову частоту генератора мікроконтролера. В нашому прикладі вона дорівнює f_{BQ}=8МГц. Також підключають конденсатори C1 та C2, 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.

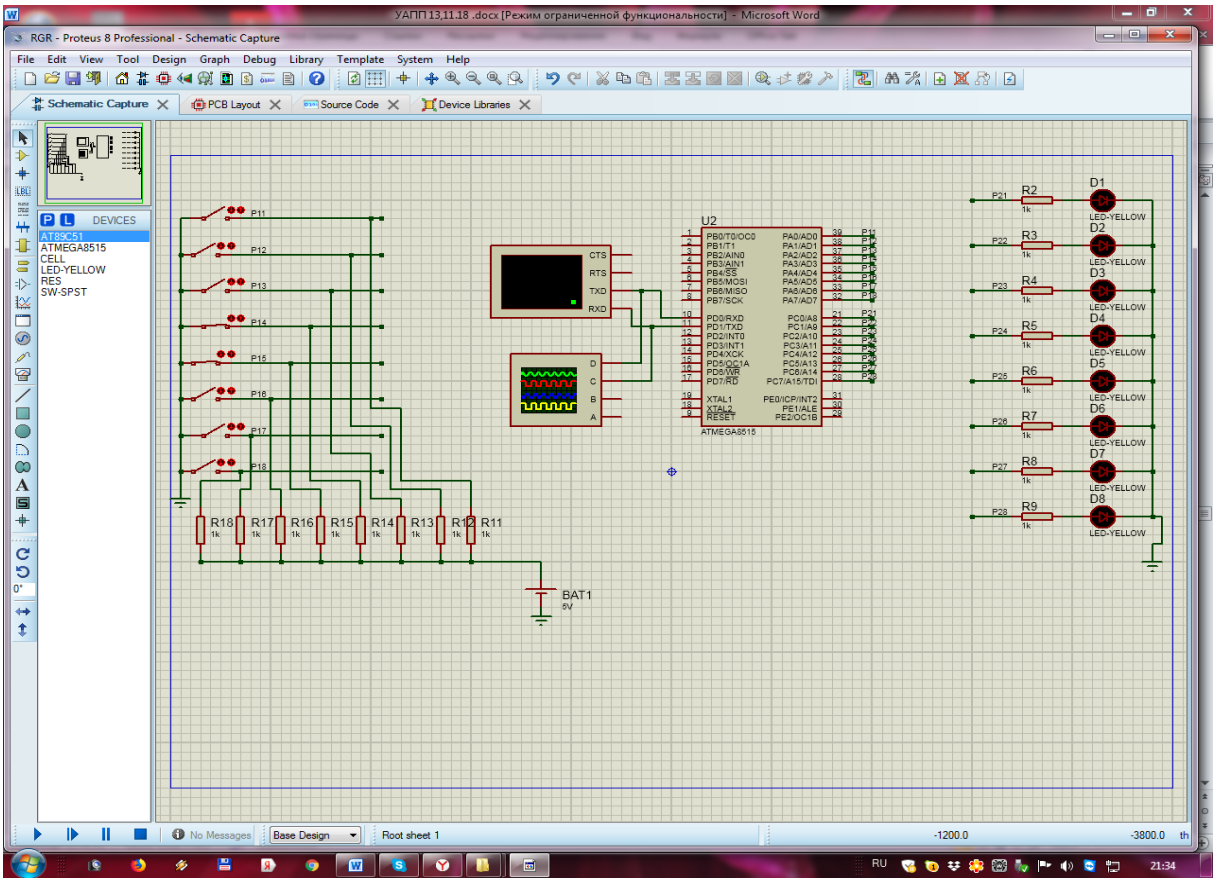


Рисунок 4.47 – Схема робочої моделі

В якості мікроконтролера використано мікросхему ATmega8515. Мікроконтролер побудований за процесорною архітектурою AVR, тобто він вміє виконувати асемблерні команди, які описано цим стандартом.

4.3.4.2.2 Порядок моделювання

1. Перш за все, потрібно пересвідчитись, що Terminal правильно налаштовано. Для цього треба двічі натиснути на нього лівою кнопкою миші, та виставити відповідні налаштування. Нижче наведено приклад для випадку, коли потрібно передавати зі швидкістю 9600 біт/сек: 8 біт даних та один стоп–біт з перевірки на не парність (рисунок 4.48).

Якщо треба використовувати перевірку на парність або непарність, то це треба вказати в опції Parity (рисунок 4.48).

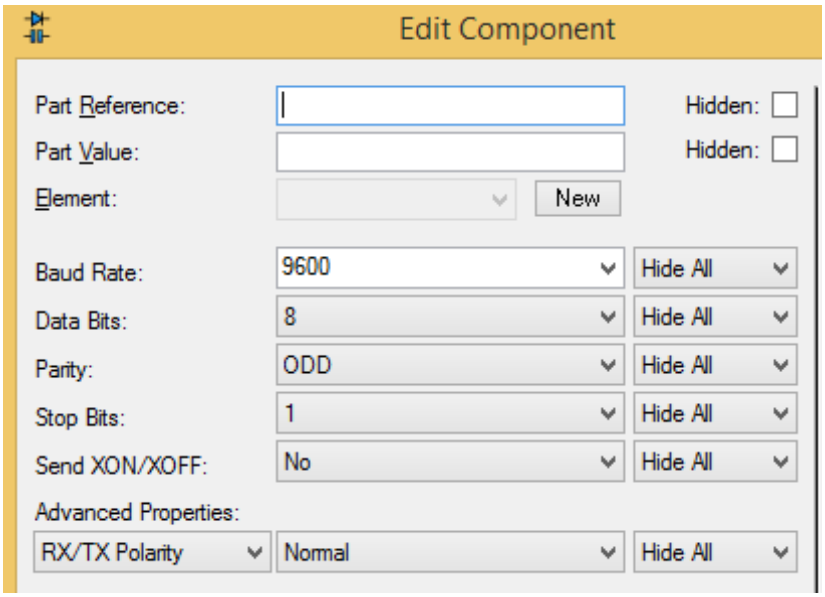


Рисунок 4.48

2. Далі натискаємо Start VSM Debugging (рисунок 4.49).

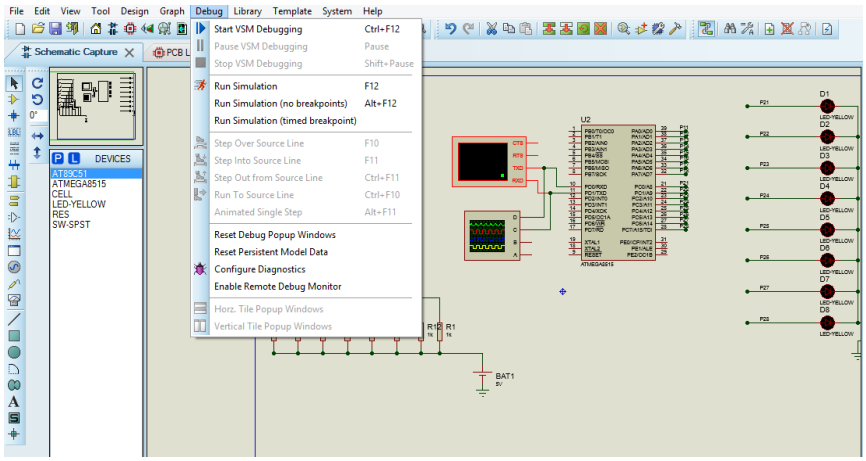


Рисунок 4.49

3. Потім натискаємо на кнопку Pause VSM Debugging

4. Переходимо на вкладку Source Code та ставимо дві точки зупинки. Для цього потрібно лівою кнопкою миші двічі клацнути з лівої сторони двох команд, як показано на рисунку 4.50.

5. Натиснемо Run Simulation (F12).

6. Перейдемо до вікна Virtual Terminal, натиснемо правою кнопкою миші на вікні, та виберемо Hex Display Mode та Echo Typed Characters (рисунок 4.51).

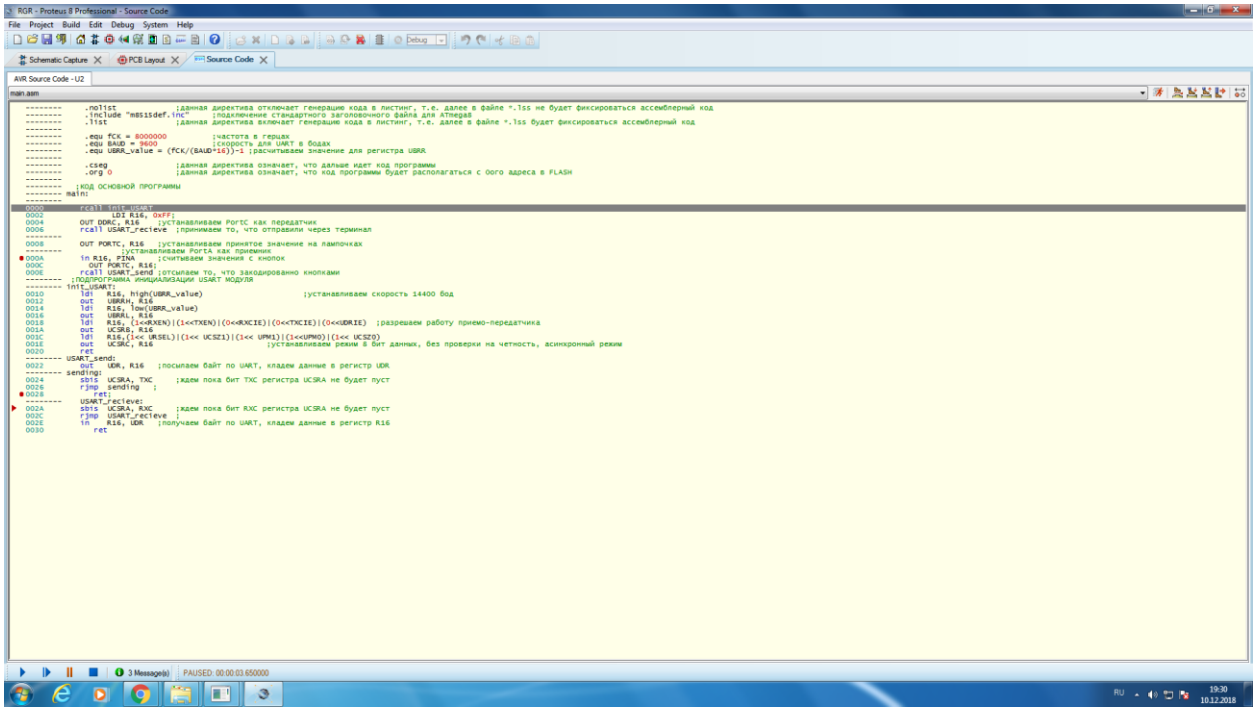


Рисунок 4.50

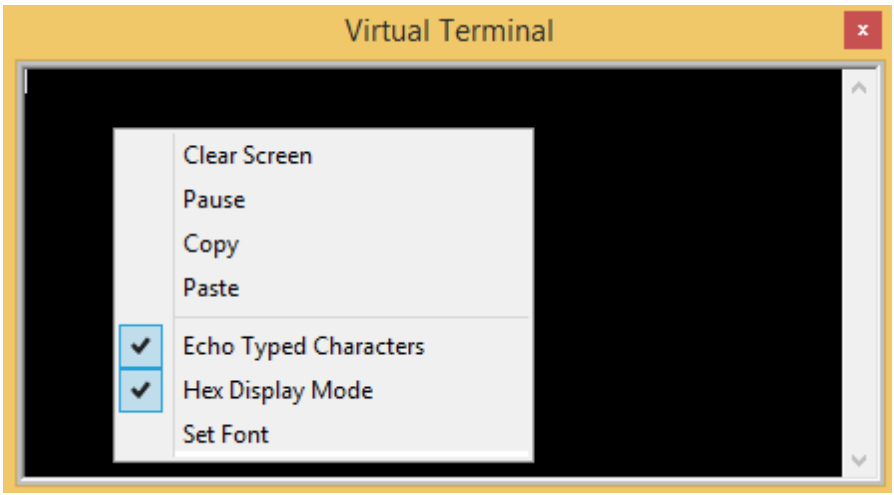


Рисунок 4.51

Якщо вікно Virtual Terminal закрито, то його можна відкрити із вкладки Debug.

7. Далі введемо якийсь символ, котрий хочемо надіслати від віртуального терміналу до мікроконтролера через УАПП. При введенні символу у вікні терміналу повинен стояти курсор. Символ, який ми вводимо з клавіатури, віртуальний термінал перетворює в ASCII–код згідно таблиці, яку наведено у [4].

Введемо, наприклад, цифру 7, ASCII код якої 37 (hex) = 0011 0111 (bin). Програма перейде на рядок, де ми поставили першу точку зупину. Натиснемо F10, зробимо один наступний крок програми, та перейдемо на вкладку зі схемою Schematic Capture. Звернемо увагу на світлодіоди та на індикатори, які на виході порту С (рисунок 4.52).

Світлодіоди та індикатори точно відображають бінарний код символу, який ми відправили (якщо дивитися знизу уверх, а бінарне число читати зліва на право), нуль – лампочка вимкнена, один – лампочка увімкнена (прийнято символ 00110111в = 37h).

8. Далі перевіримо як мікроконтролер через УАПП відправить символ, який ми закодували кнопками: замкнута кнопка – нуль, а розімкнута – одиниця (рисунок 4.53).

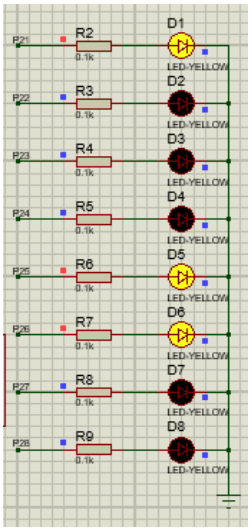


Рисунок 4.52

Якщо дивитися на кнопки знизу уверх, то на рисунку 4.53 набрано число 1110 0111 (bin) = E7 (hex).

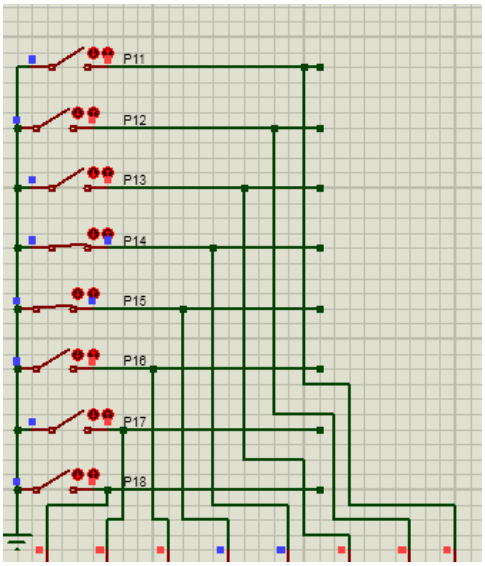


Рисунок 4.53

9 Тепер подивимося на вікно осцилографа. Для того, щоб побачити переданий сигнал на осцилографі, потрібно натиснути кнопку на опції One –Shot (один кадр), як показано на рисунку 4.54.

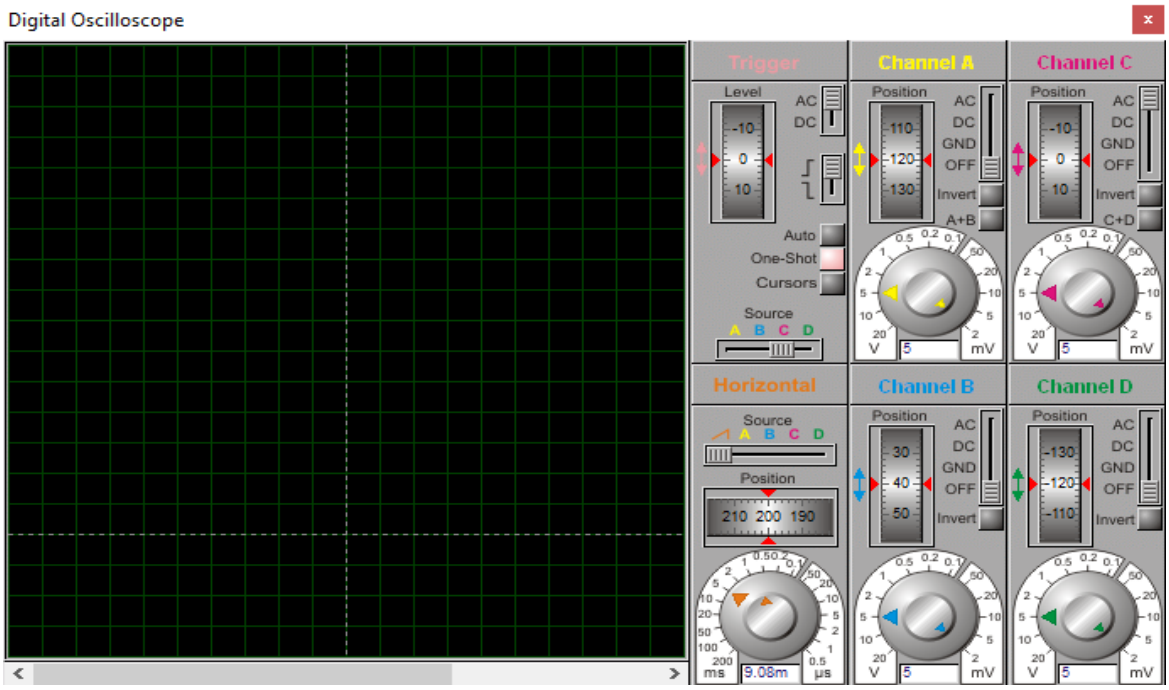


Рисунок 4.54

10 Натиснемо Run Simulation для продовження виконання програми (рисунок 4.49). Програма дійде до другої точки зупину.

11 Розберемо більш детально сигнали, що відображаються на осцилографі (рисунок 4.55).

Осцилограф відображає сигнали за двома промінями: верхній (червоний) – сигнал, який передається від кнопок, нижній (зелений) – сигнал, який приймається від терміналу: 37(hex).

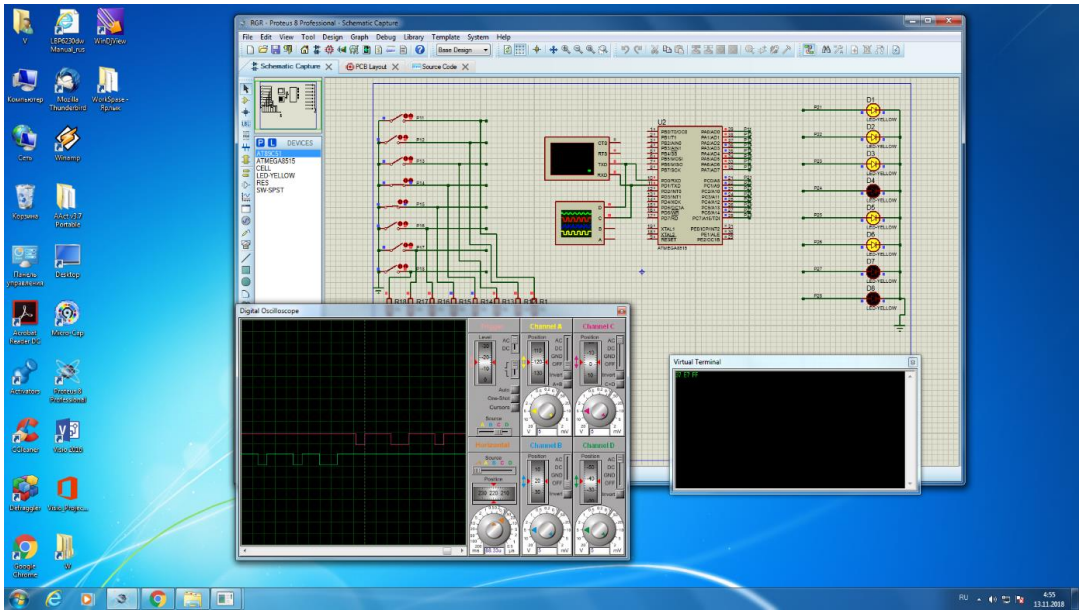


Рисунок 4.55

Розглянемо сигнал, який передається від 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 1110 0111 (bin) = E7 (hex). Далі іде 9–й біт , який дорівнює 0, тому що число одиниць в байті, який було передано – парне (розряд P в регістрі прапорців дорівнює 0). Останній 10–й біт, який дорівнює 1, це стоп–біт.

12 Тепер визначимо отриману швидкість передачі. Для знаходження швидкості передачі (в даному прикладі було обрано швидкість 9600 біт/сек = =9600 пос/сек = 9600 бод) підберемо таку розгортку сигналу на осцилографі, щоб сумістити бічні сторони імпульсу із вертикальними лініями сітки осцилографа.

Для швидкості передачі 9600 біт/сек отримаємо наступну картину (рисунок 4.56).

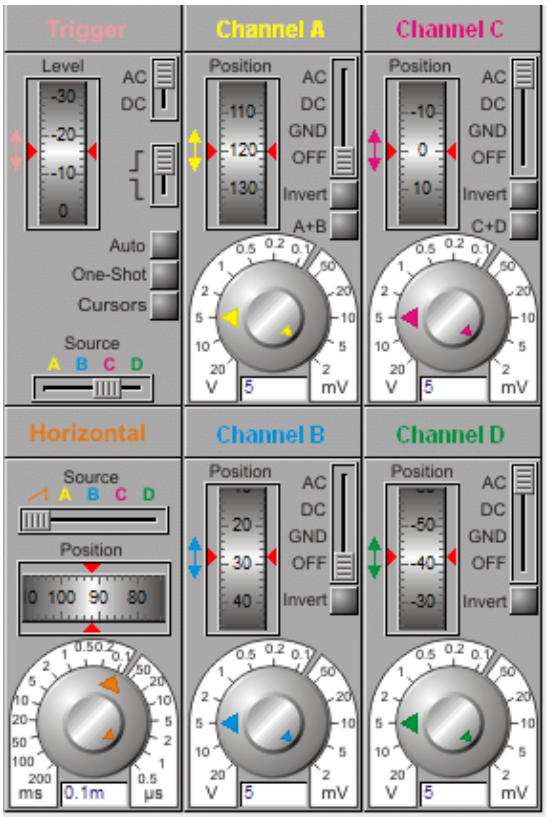


Рисунок 4.56

Як ми бачимо, тривалість одного імпульсу дорівнює 0,104 мс. Отже, якщо розділити один біт на цей час, та помножити на 1000, отримаємо задану швидкість:

$$V = \frac{1}{0.104} * 1000 = 9600 \frac{\text{біт}}{\text{сек}}.$$

Нижче розглянуто ще один приклад (рисунок 4.57, 4.58), коли на кнопках було набрано число 67(hex)=01100111(bin). На верхньому промені осцилографа побачимо: перший нуль – старт- біт, потім 3 одиниці та один 0 – це число 7 на кнопках, далі йде нуль, дві одиниці та нуль – це число 6 на кнопках, а потім на осцилографі бачимо 1 – це біт доповнення до парності, та наступна одиниця – стоп-біт.

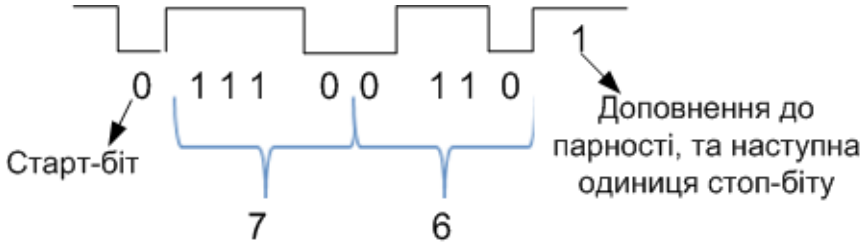


Рисунок 4.57

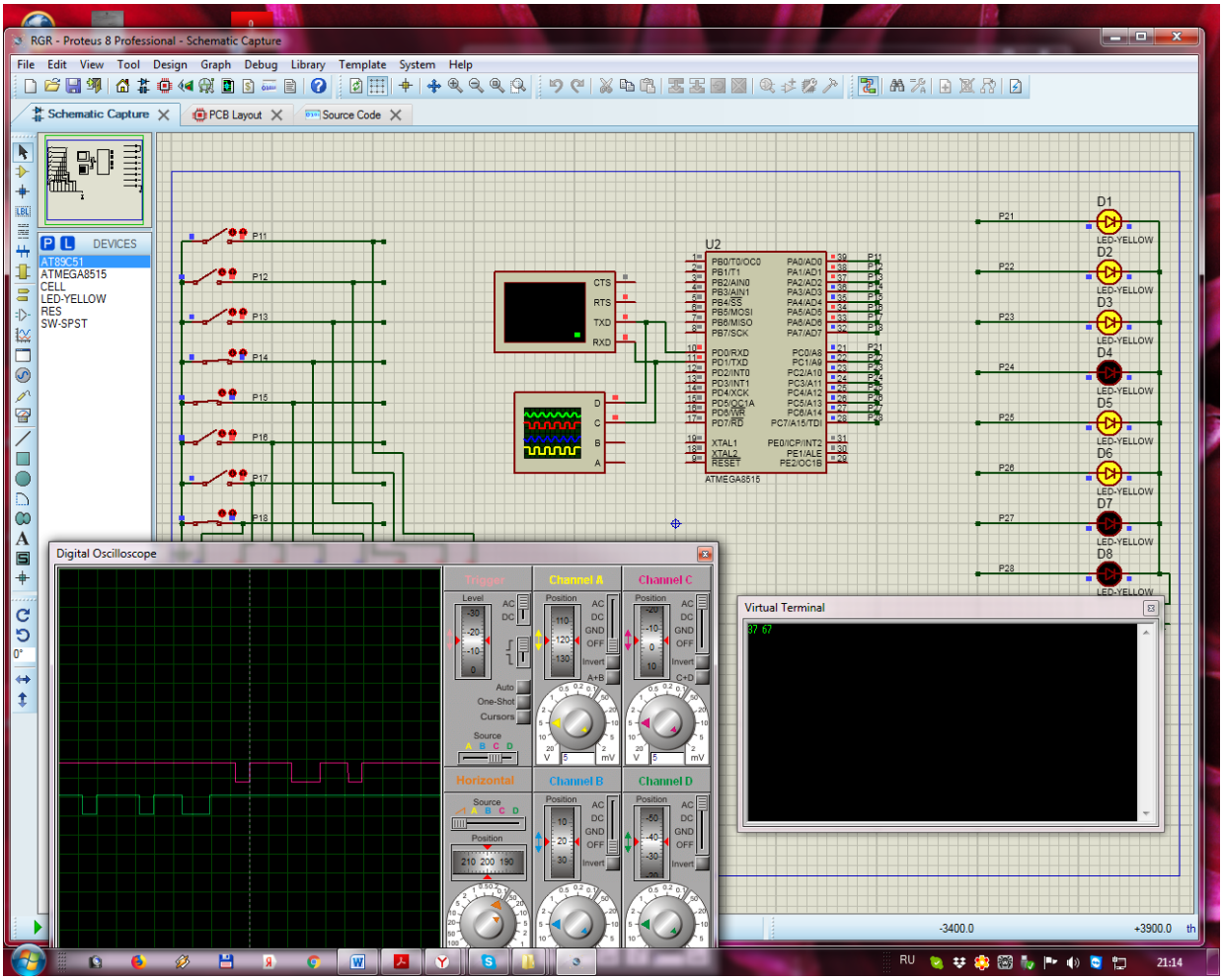


Рисунок 4.58

4.3.4.2.3 Схема алгоритму роботи

На рисунку 4.59 наведено схему алгоритму роботи.

Лістинг робочої програми

Лістинг робочої програми, яку розроблено згідно з алгоритмом, який наведено на рисунку 4.59, представлено надалі. В дужках праворуч біля кожної команди цифрою відмічено номер блоку схеми алгоритму, який реалізує ця команда.

Рисунок 4.59 – Схема алгоритму роботи

Робоча програма мовою Асемблер

```
.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 UART у бодах
.equ UBRR_value = (f_CK/(BAUD*16))-1   ;розраховуємо значення для
                                       регістра  ;UBRR
.cseg                      ;директива означає, що далі іде код програми
.org 0                     ;директива означає, що  код програми у FLASH буде
                           ;розміщено з нульової адреси
; КОД ОСНОВНОЇ ПРОГРАМИ
main:
;ініціалізації з мітки init_USART
rcall init_USART           ; (блок 2) відносний виклик підпрограми
;налаштування порту С на передачу
LDI R16, 0xFF              ; (блок 2) R16← 0xFF
OUT DDRC, R16              ; (блок 2) DDRC← R16
; (блок 3) отримуємо те, що відправив віртуальний термінал
rcall USART_recieve        ; (блок 3) відносний виклик підпрограми з мітки
                                   ; USART_recieve
; (блок 4) виводимо отримане від терміналу на світлодіоди
OUT PORTC, R16             ; (блок 4) PORTC←R16
; (блок 5) налаштовуємо порт А як передавач
LDI R16, 0x00              ; (блок 5) R16←0x00
```

```
   OUT DDRA, R16          ; (блок 5) DDRA← R16
                          ; (блок 6) читаємо стан кнопок
   in R16, PINA           ; (блок 6) R16← PINA
   ; (блок 7) виводимо отримане від кнопок на світлодіоди
   OUT PORTC, R16         ; (блок 7) PORTC←R16
   ; (блок 8)  підпрограма з міткою USART_send відправляє стан кнопок на
   осцилограф та термінал
   rcall USART_send       ; (блок 8) відносний виклик підпрограми з мітки
                                ; USART_ send
   ; (блок 1) ПІДПРОГРАМА ІНІЦІАЛІЗАЦІЇ USART – МОДУЛЯ
   init_USART:
   ; програмуємо швидкість 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
   ; дозволяємо роботу приймача-передавача модуля USART
ldi   R16, (1<<RXEN)|(1<<TXEN)|(0<<RXCIE)|(0<<TXCIE)|(0<<UDRIE)
out   UCSRB, R16                     ; (блок 1) UCSRB←R16
   ; програмуємо передачу 8 біт з перевіркою на парність
   ; у асинхроному режимі
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:
```

```
; чекаємо 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 пропустити наступну команду,
; інакше наступна команда
rjmp  USART_recieve     ; (блок 3) безумовний перехід на мітку
                        ;USART_recieve
in      R16, UDR        ; (блок 3) R16←UDR (завантажуємо в
                        ; регістр  R16 отриманий байт від термінала

ret                     ; (блок 3) повернення з підпрограми
```

4.3.4.3 Зміст звіту

Звіт по роботі повинен містити:

– схему моделі;

– схему алгоритму роботи моделі;

– робочу програму;

– формули за потребою.

Контрольні запитання

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?

4.3.5 Лабораторна робота №8. Моделювання модуля SPI

Тема: Моделювання модуля SPI

Мета: Користуючись пакетом PROTEUS дослідити моделювання модуля SPI

4.3.5.1 Порядок виконання роботи

– створити модель пристрою в пакеті Proteus 8.6

– розробити схему алгоритму роботи моделі та робочу програму

– створити hex-файл та підключити його до мікроконтролера

– запустити модель та виконати її дослідження згідно методичних вказівок

– зробити відповідні висновки.

4.3.5.2 Стислі теоретичні відомості

4.3.5.2.1 Опис моделі

Схему моделі дослідження послідовного периферійного інтерфейсу SPI наведено на рисунку 4.60. Ліворуч на схемі розташовано мікроконтролер, що працює в режимі «Master», а праворуч – мікроконтролер, який працює в режимі «Slave». В якості як веденого, так і ведучого мікроконтролера використано мікроконтролери ATMEGA8. Блок кнопок «Master Data To Send» призначено для задання байта даних для передачі від ведучого до веденого. Усі вісім кнопок цього блоку підключено до ліній порту D. Нижнє положення кнопки відповідає подачі на відповідну лінію порту D мікроконтролера напруги низького рівня, тобто логічного нуля, а верхнє положення кнопки – подачі на відповідну лінію порту напруги високого рівня, тобто логічної одиниці. Резистори R1…R8, що підтягують, призначені для формування напруги високого рівня на відповідній лінії порту мікроконтролера, коли контакти відповідної кнопки розімкнені..

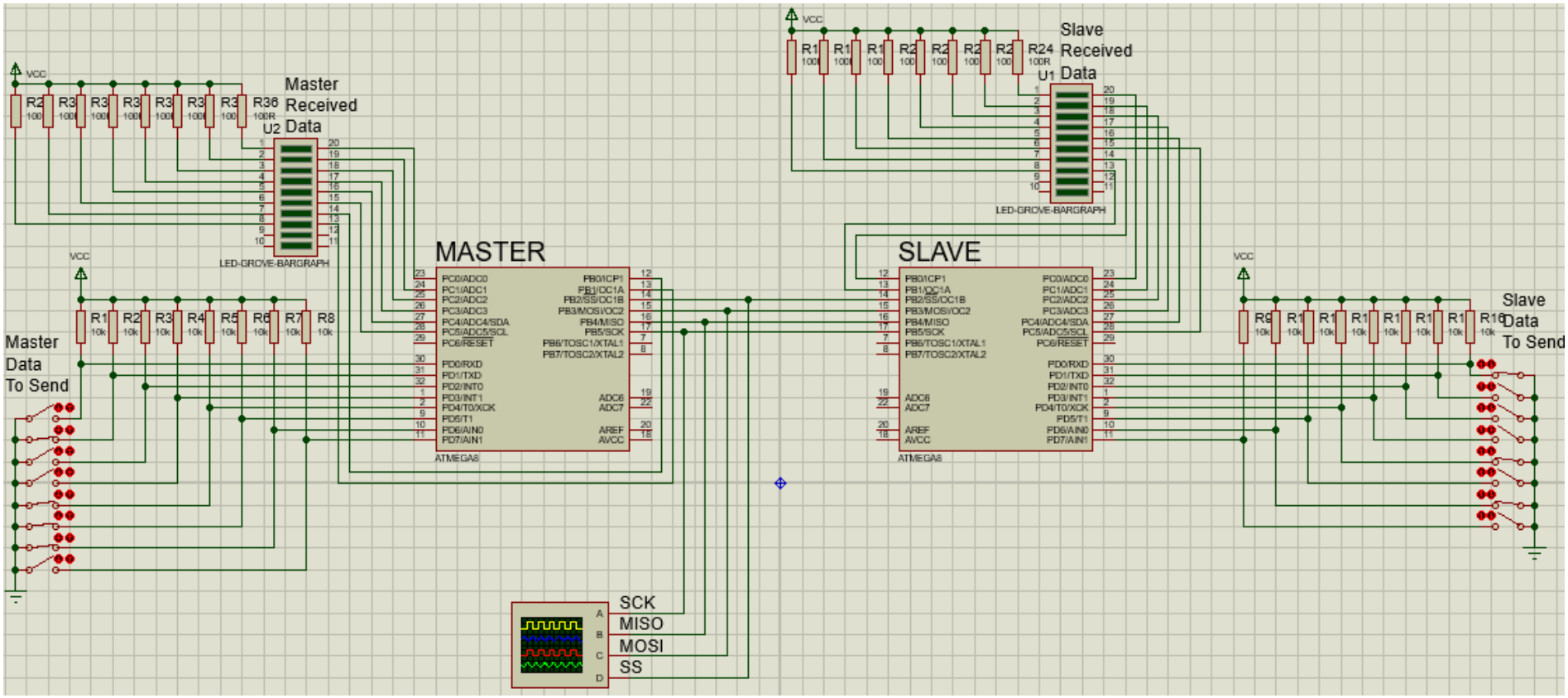


Рисунок 4.60 – Схема робочої моделі

Світлодіодна лінійка U2 «Master Received Data» призначена для відображення байта даних, прийнятого ведучим мікроконтролером від веденого. Верхній за схемою сегмент лінійки відображає молодший біт прийнятого байта даних, а нижній – старший. Світіння сегмента відповідає тому, що було прийнято логічний нуль, тобто натиснутій відповідній кнопці з боку передавача. Катоди сегментів світлодіодної лінійки, що відображають 6 молодших біт прийнятого байта підключено до ліній PC0…PC5 порту C мікроконтролера, а катоди сегментів, що відображають 2 старші біти – до ліній PB0, PB1 порту B. Аноди всіх сегментів світлодіодної лінійки через обмежуючі резистори R25…R32 підключено до лінії живлення +5ВДо ліній SCK, $\overline{SS}$, MOSI та MISO ведучого мікроконтролера підключено відповідні лінії веденого мікроконтролера та входи відповідних каналів віртуального осцилографа.

Підключення веденого мікроконтролера виконано аналогічно. Кнопки «Slave Data To Send» призначені для вводу байта даних для передачі від веденого до ведучого. А світлодіодна лінійка U1 «Slave Received Data» відображає байт даних, прийнятий веденим від ведучого. Резистори R17…R24 обмежують струм через світлодіоди лінійки U1, а резистори R9…R16 призначені для формування напруги високого рівня на відповідних лініях порту D веденого мікроконтролера, коли контакти відповідної кнопки розімкнені.

4.3.5.2.2 Порядок моделювання

Для запуску моделювання натискаємо кнопку «Run simulation» «  ». Після чого кнопками «Master Data To Send» задаємо байт для передачі ведучим МК веденому та одночасно спостерігаємо за відображенням його прийому на лінійці світлодіодів U1 «Slave Received Data». Також спостерігаємо за зміною сигналів, що відображаються у вікні віртуального осцилографа. Далі кнопками «Slave Data To Send» задаємо байт для передачі веденим МК ведучому та одночасно спостерігаємо за відображенням його прийому на лінійці світлодіодів U2 «Master Received Data» та за зміною сигналів, що відображаються у вікні віртуального осцилографа. Задамо, наприклад, байт

«10001101» кнопками «Master Data To Send» для передачі від ведучого мікроконтролера («Master») до веденого («Slave»). Після чого спостерігаємо прийом даного байта веденим мікроконтролером та відображення його прийому на лінійці світлодіодів U1 «Slave Received Data». Відповідні фрагменти схеми моделі наведено на рисунку 4.61. Заданий байт «10001101» від ведучого веденому передається послідовно по лінії MOSI, що видно на відповідному каналі віртуального осцилографа (рисунок 4.62).

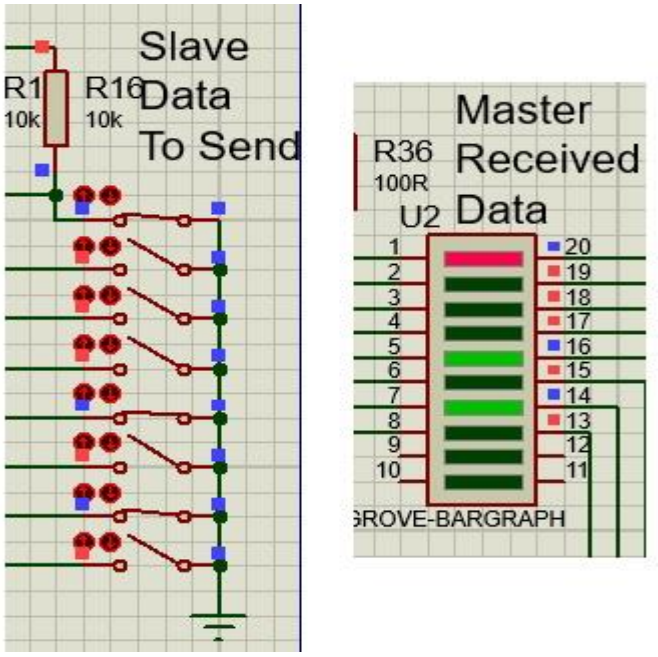


Рисунок 4.61 – Фрагменти схеми моделі для відображення передачі від ведучого до веденого

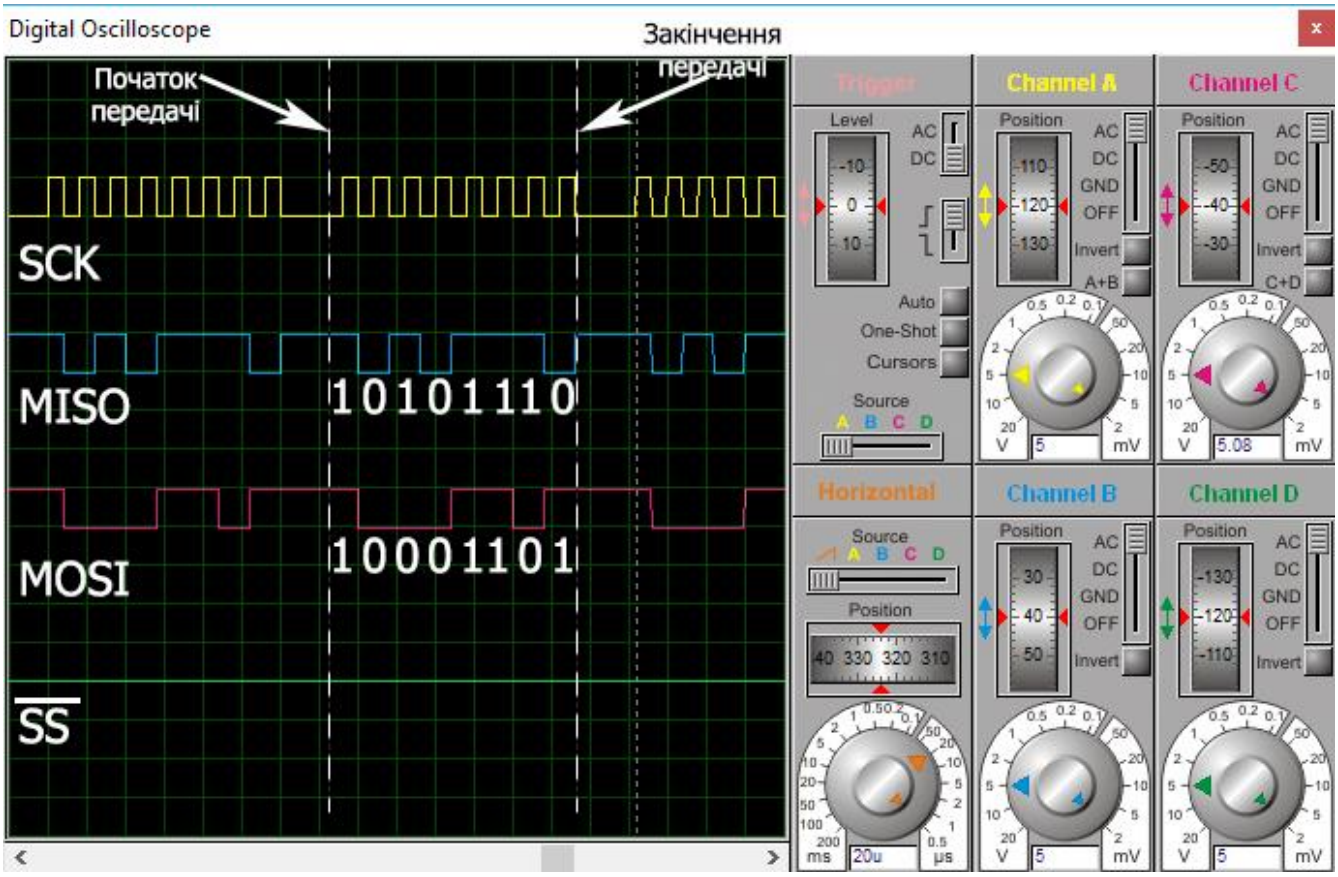


Рисунок 4.62 – Вікно віртуального осцилографа

Кнопками «Slave Data To Send» для передачі від веденого мікроконтролера до ведучого задамо, байт «10101110». Після чого спостерігаємо прийом даного байта ведучим мікроконтролером та відображення його прийому на лінійці світлодіодів U2 «Master Received Data». Відповідні фрагменти схеми моделі наведено на рисунку 4.63 Заданий байт «10101110» від веденого ведучому передається послідовно по лінії MISO, що видно на відповідному каналі віртуального осцилографа (рисунок 4.62).

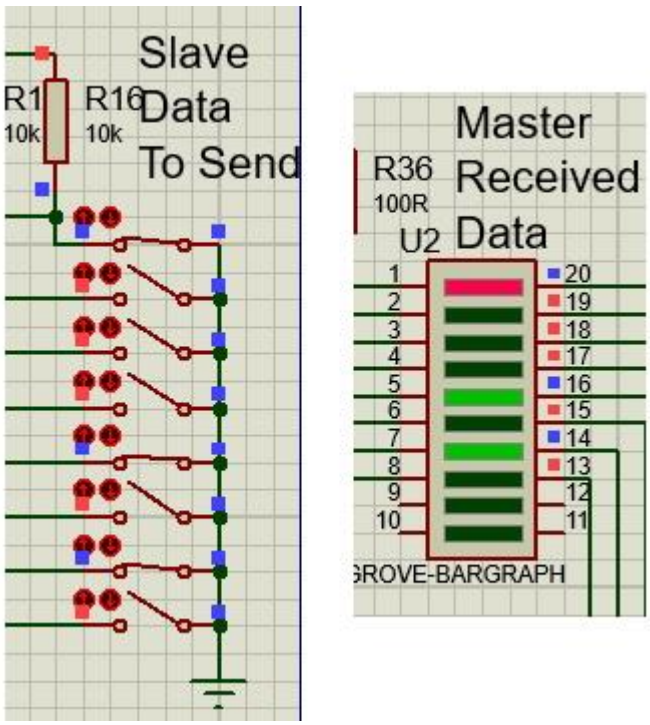


Рисунок 4.63 – Фрагменти схеми моделі для відображення передачі від веденого до ведучого

4.3.5.2.3 Схема алгоритму

На рисунках 4.64 та 4.65 наведено схеми алгоритмів роботи керуючої програми для ведучого та веденого мікроконтролерів відповідно.

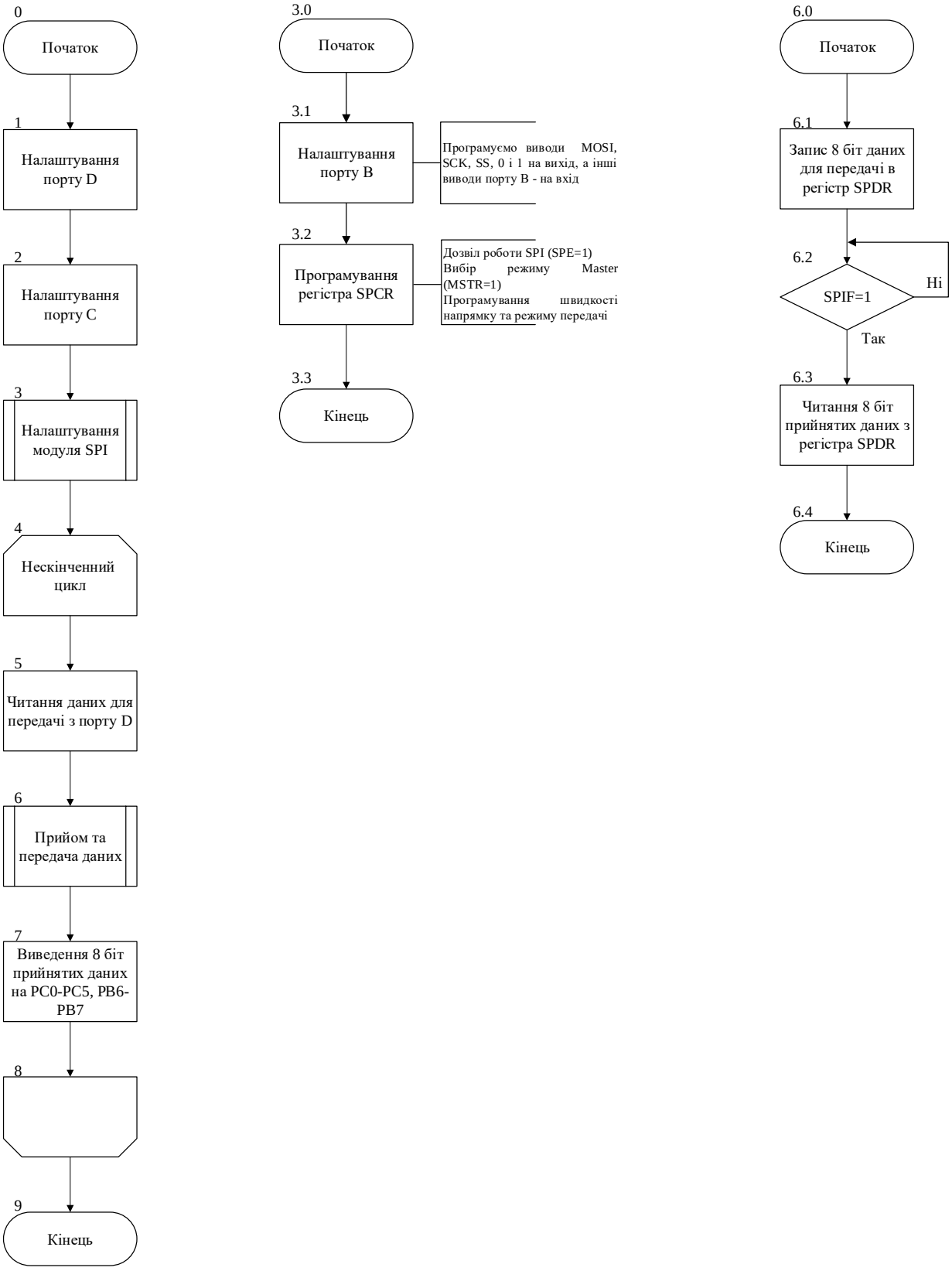


Рисунок 4.64 – Схема алгоритму керуючої програми ведучого МК

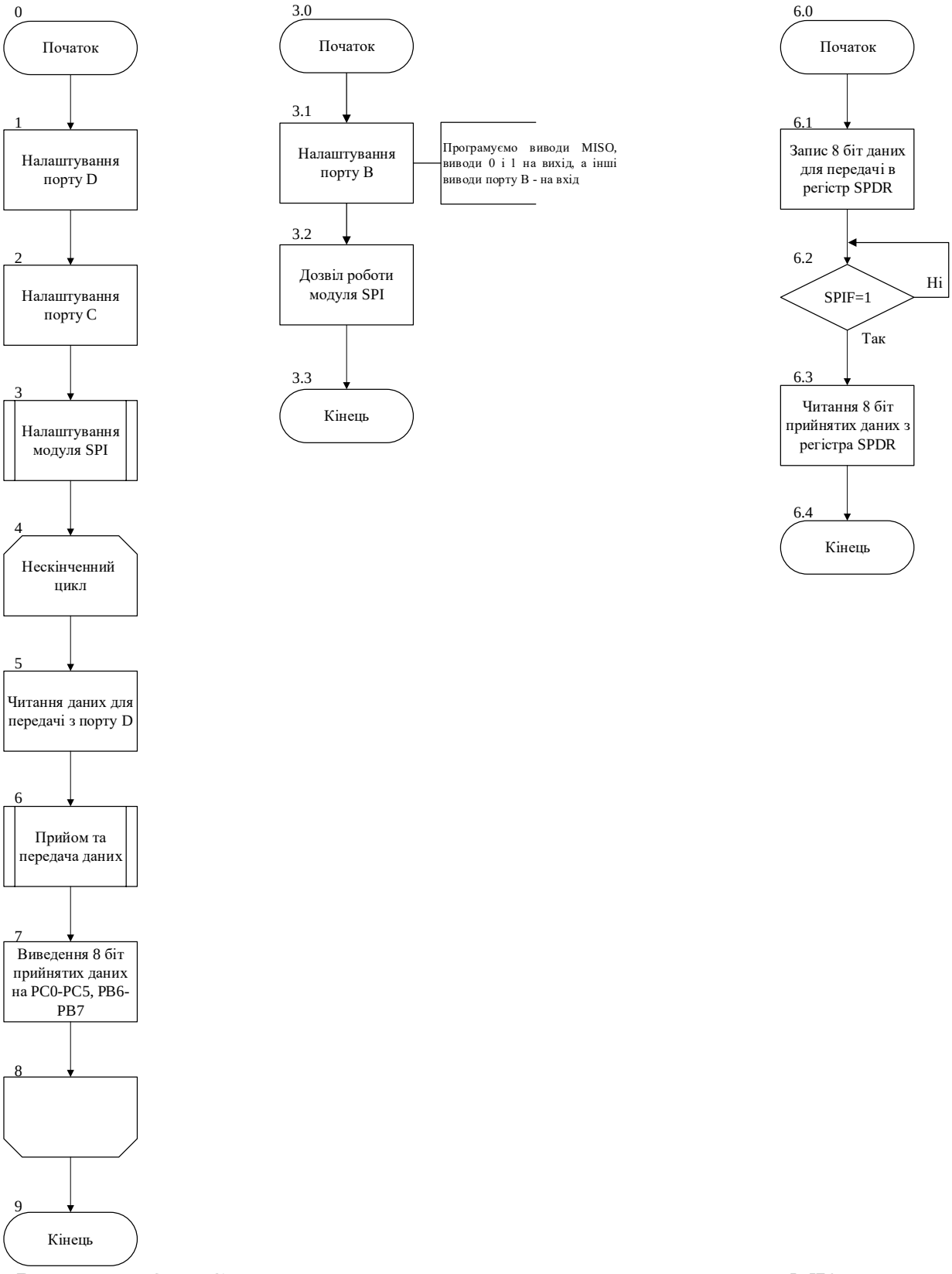


Рисунок 4.65 – Схема алгоритму керуючої програми веденого МК

4.3.5.2.4 Лістинг керуючої програми

Відповідно до схем алгоритмів, наведених на рисунках 4.64 та 4.65, розроблено керуючі програми для ведучого та веденого мікроконтролерів. Лістинги розроблених програм наведено нижче.

SPI Master:

```
#define F_CPU 1000000
#include <avr/io.h>
#include <util/delay.h>

void SPI_MasterInit(void)
{
	DDRB = 0b00101111;                          // 3.1 Програмуємо виводи MOSI та SCK, SS,
                                                    виводи 0 і 1 на вихід, а інші
на вхід
	SPCR = (1<<SPE)|(1<<MSTR)|(1<<SPR0);        // 3.2 Дозвіл роботи SPI, режим Master,
                                                    встановлення швидкості передачі
(fck/16)
}
char SPI_MasterTransmit_Receive(char cData)
{
	char received_Data;
	/* Початок передачі */

	SPDR = cData;                               // 6.1 Запис 8 біт даних для передачі в SPDR

	while(!(SPSR & (1<<SPIF)))                  // 6.2
	;                                           // 6.2 Чекаємо завершення передачі

	received_Data=SPDR;                         // 6.3 Читання прийнятих даних з регістра
SPDR

	return received_Data;
}

int main(void)
{
	DDRD=0x00;                                  // 1 Програмуємо порт D на вхід.
	PORTD=0x00;                                 // 1 Відключаємо внутрішні підтягуючі
                                                        резисторри, оскільки вони є на
схемі.

	DDRC=0xff;                                  // 2 Програмуємо виводи порта C на вихід.
	PORTC=0xff;                                 // 2 Виводи порта C встановлюємо в 1.

	SPI_MasterInit();                           // 3 Налаштування моуля SPI
	char data;

    while (1)                                   // 4
    {
            data = PIND;                        // 5 Читання даних для передачі з порту D

            char r_Data = SPI_MasterTransmit_Receive(data); // 6 Прийом/передача даних
```

```
            PORTC = r_Data;                     // 7 Виведення 6 молодших біт прийнятих
даних на                                                PC0-PC5

            /* Виведення 2 старших біт прийнятих даних на PB6-PB7:*/
            char temp = PORTB;                  // 7
            temp &= 0b11111100;                 // 7
            temp |= ((r_Data>>6) & 0b00000011);// 7
            PORTB = temp;                       // 7
    }                                           // 8
}                                               // 9
```

SPI Slave:

```
#define F_CPU 1000000
#include <avr/io.h>
#include <util/delay.h>

void SPI_SlaveInit(void)
{
      DDRB = 0b00010011;                        // 3.1 Програмуємо виводи MISO, виводи 0 і 1
на                                                    вихід, а інші - на вхід
      SPCR = (1<<SPE);                          // 3.2 Дозвіл роботи модуля SPI
}
char SPI_SlaveTransmit_Receive(char cData)
{
      char received_Data;
      /* Початок передачі */

      SPDR = cData;                             // 6.1 Запис 8 біт даних для передачі в SPDR

      while(!(SPSR & (1<<SPIF))) // 6.2
      ;                                         // 6.2 Чекаємо завершення передачі

      received_Data=SPDR;                       // 6.3 Читання прийнятих даних з регістра
SPDR

      return received_Data;
}

int main(void)
{
      DDRD=0x00;                                // 1 Програмуємо порт D на вхід.
      PORTD=0x00;                               // 1 Відключаємо внутрішні підтягуючі
                                                      резисторри, оскільки вони є на
схемі.

      DDRC=0xff;                                // 2 Програмуємо виводи порта C на вихід.
      PORTC=0xff;                               // 2 Виводи порта C встановлюємо в 1.

      SPI_SlaveInit();                          // 3 Налаштування моуля SPI
      char data;

      while (1)                                 // 4
      {
            data = PIND;                        // 5 Читання даних для передачі з порту D

            char r_Data = SPI_SlaveTransmit_Receive(data);  // 6 Прийом/передача даних

            PORTC = r_Data;                     // 7 Виведення 6 молодших біт прийнятих
даних на                                                PC0-PC5

            /* Виведення 2 старших біт прийнятих даних на PB6-PB7:*/
            char temp = PORTB;                  // 7
```

```
            temp &= 0b11111100;                 // 7
            temp |= ((r_Data>>6) & 0b00000011);// 7
            PORTB = temp;                       // 7
        }                                       // 8
}                                               // 9
```

4.3.5.3 Зміст звіту

Звіт по роботі повинен містити:

– схему моделі;

– схему алгоритму роботи моделі;

– робочу програму;

– формули за потребою.

Контрольні запитання

1. Який режим обміну використовується в модулі SPI: асинхронний чи синхронний? Відповідь пояснити.
2. На яку відстань і з якою швидкістю можна здійснювати обмін даними через інтерфейс SPI?
3. Поясніть структурну схему модуля SPI.
4. В яких режимах може працювати мікроконтролер при обміні даними інтерфейсом SPI?
5. Скільки та які виводи мікроконтролера використовує модуль SPI?
6. Як виконується перепризначення режиму роботи виводів модуля SPI?
7. Поясніть схему з'єднання двох мікроконтролерів інтерфейсом SPI.
8. Які пристрої знаходяться перед входами восьмирозрядних регістрів зсуву ведучого та веденого мікроконтролерів? Яку функцію вони виконують?
9. Який регістр призначено для керування модулем SPI?
10. Який регістр використовується для контролю стану модуля, а також для додаткового керування швидкістю обміну?
11. Як задати режим роботи мікроконтролера в режимі «MASTER»?
12. В який спосіб здійснюється передача даних в модулі SPI?
13. Як визначити кінець передачі байта?
14. За якої умови при передачі даних від ведучого до веденого можлива передача у зворотному напрямі?
15. Що означає те, що у модулі реалізовано одинарну буферизацію при передачі та подвійну при прийомі?
16. Поясніть структура SPI-мережі мікроконтролерів.
17. Як обирається потрібний ведений мікроконтролер в SPI-мережі?

18. Поясніть вплив сигналу $\overline{SS}$ на початок обміну даними в різних режимах роботи модуля.
19. Чи можна до інтерфейсу підключати декілька периферійних пристроїв?
20. Назвіть кількість режимів передачі даних та принцип їх роботи.
21. Що є джерелом тактового сигналу при роботі модуля? Як формується ций сигнал?
22. При яких частотах гарантується функціонування мікроконтролера в режимі «Slave»?
23. З яких етапів складається обмін в режимі SPI?
24. Поясніть схему включення мікроконтролерів в режимі програмування послідовним каналом при використанні інтерфейсу SPI.

4.3.6 Лабораторна робота №9. Моделювання інтерфейсу TWI (I^2C)

Тема: Моделювання інтерфейсу TWI (I^2C)

Мета: Користуючись пакетом PROTEUS дослідити моделювання інтерфейсу TWI (I^2C)

4.3.6.1 Порядок виконання роботи

– створити модель пристрою в пакеті Proteus 8.6

– розробити схему алгоритму роботи моделі та робочу програму

– створити hex-файл та підключити його до мікроконтролера

– запустити модель та виконати її дослідження згідно методичних вказівок

– зробити відповідні висновки.

4.3.6.2 Стислі теоретичні відомості

4.3.6.2.1 Схема моделі

Схему моделювання у пакеті Proteus 6 наведено на рисунку 4.66.

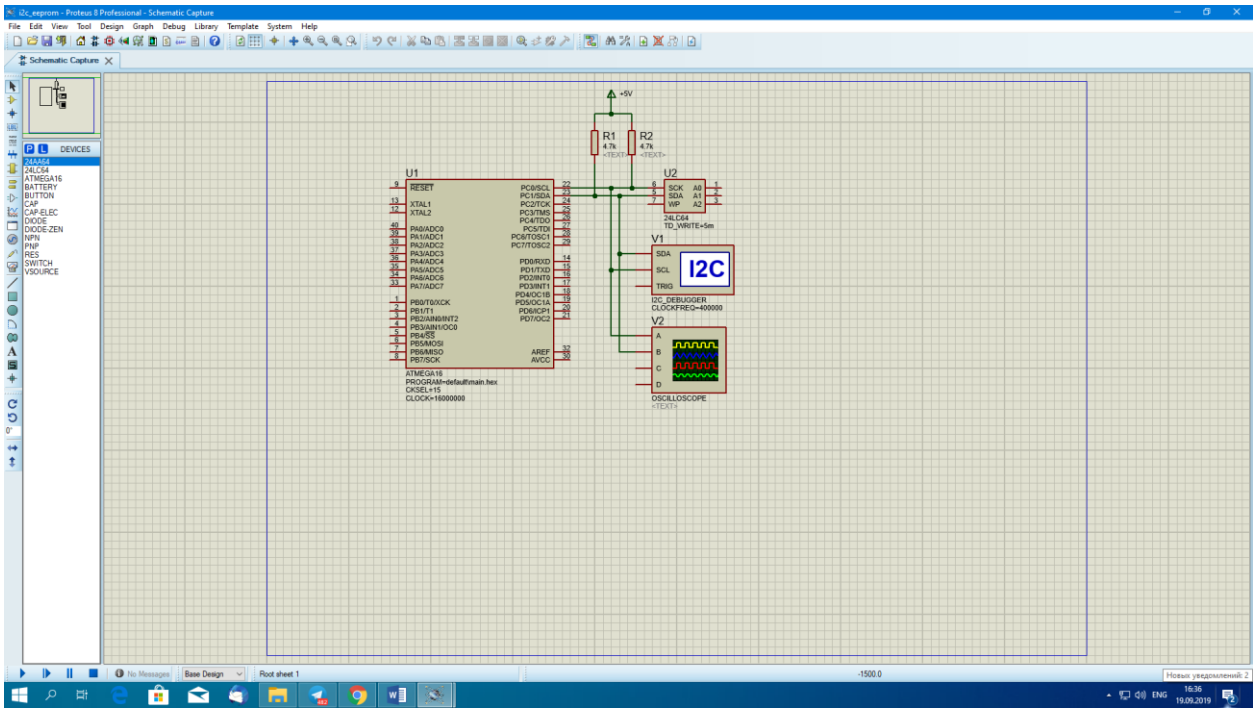


Рисунок 4.66 – Схема моделювання інтерфейсу I^2C у пакеті Proteus 8.6

Опис моделі

У моделі використано мікроконтролер ATmega16. До нього додаються 2 резистори: R1 та R2 для підключення шини I^2C до джерела живлення. З вкладки віртуальні інструменти береться осциллограф, I2C-відлагоджувач та EEPROM-пам'ять даних із вбудованим інтерфейсом I2C – 24LC64. Ця мікросхема має 8 виводів, які, крім живлення і землі, відображено на моделі (U2).

У налаштуваннях мікроконтролера вказано прошивку, частоту тактування: 16 МГц і виставлено CKSEL-фьюзи для кристала. У налаштуваннях I2C-відлагодувача вказано, що тактова частота імпульсів шини дорівнює 400 КГц.

4.3.6.2.2 Налаштування частоти тактових імпульсів лінії SCL

Періодом тактових імпульсів на лінії SCL керує блок генератора швидкості зв'язку ведучого мікроконтролера. Період SCL змінюється шляхом програмування регістра швидкості зв'язку TWI (TWBR) і бітів переддільника у регістрі статусу TWI (TWSR).

Частота SCL визначається за такою формулою:

$$f_{SCL} = f_{CLK} / [16+2(TWBR) \cdot 4TWPS], \qquad (4.7)$$

де:

TWBR – значення регістра швидкості зв'язку TWI;

TWPS – значення бітів переддільника в регістрі статусу TWI (TWSR);

f_{CLK} – тактова частота ведучого мікроконтролера;

f_{SCL} – частота тактових імпульсів лінії SCL.

Налаштування TWBR потрібно, тому що ведена мікросхема навчена обмінюватися даними на певній частоті. Наприклад, якщо в Proteus, ввести пошук I^2C MEM, то побачите велику кількість відповідних мікросхем пам'яті, для яких в основному вказані частоти 100 і 400КГц. Для обраної в моделі

мікросхеми EEPROM-пам’яті тактова частота імпульсів шини дорівнює 400 КГц.

Підставляючи у формулу частоти f_{CLK} та f_{SCL}, ми зможемо знайти оптимальне значення для регістра TWBR.

TWPS – це 2-бітове число [TWPS1: TWPS0], перший біт – це розряд TWPS0, другий – TWPS1 в регістрі статусу TWSR. Задаючи значення переддільника швидкості зв’язку 4TWPS, ми можемо зменшити значення TWBR, тому що TWBR має довжину вісім біт, а відповідно максимальне значення 255. Але зазвичай це не потрібно, тому TWPS задається 0 і 4 TWPS = 1. Набагато частіше, навпаки, важливе значення нижнього діапазону регістра TWBR. Якщо TWI працює у ведучому режимі, то значення TWBR повинно бути не менше 10. Якщо значення TWBR менше 10, то ведучий пристрій шини може генерувати не коректні сигнали на лініях SDA і SCL під час передачі байта [2].

Таблиця 4.28

TWPS1	TWPS0	$TWPS_{10}$	Значення переддільника 4TWPS
0	0	0	1
0	1	1	4
1	0	2	16
1	1	3	64

Таблиця 4.29

Частота МК, МГц	TWBR	TWPS	Частота SCL, КГц
16.0	12	0	400
16.0	72	0	100
14.4	10	0	400
14.4	64	0	100
12.0	52	0	100
8.0	32	0	100
4.0	12	0	100
3.6	10	0	100

Таким чином для налаштування TWI необхідно:

- завдати значення переддільника [TWPS1: TWPS0] у регістрі статусу TWSR;
- завдати швидкість зв’язку у регістрі TWBR.

4.3.6.2.3 Запуск процесу моделювання

На рисунку 4.67 наведено стан моделі після запуску процесу моделювання (після натискання клавіши «START»).

Як бачимо, що в PORTA записалося число 0b0000’1010 або 0x0A або 10, тобто змінна uint8_t byte містить вірне значення після читання пам’яті 24LC64 за адресою 0x19.

Нижче наведено пояснення окремих етапів обміну TWI (I2C)-шиною, які відображено у вікні I^{2}C Debug (рисунок 4.68), та часових діаграм обміну між мікроконтролером та EEPROM-пам’яттю, отриманих на екрані осцилографа (рисунок 4.69).

Як видно з цих рисунків, результати моделювання підтверджують теоретичні відомості, які наведено на рисунку 4.70. Адреса веденого пристрою в нашому прикладі дорівнює: 0x19, а байт даних, який спочатку записується, а потім зчитується за цією адресою, дорівнює: 0x0A.

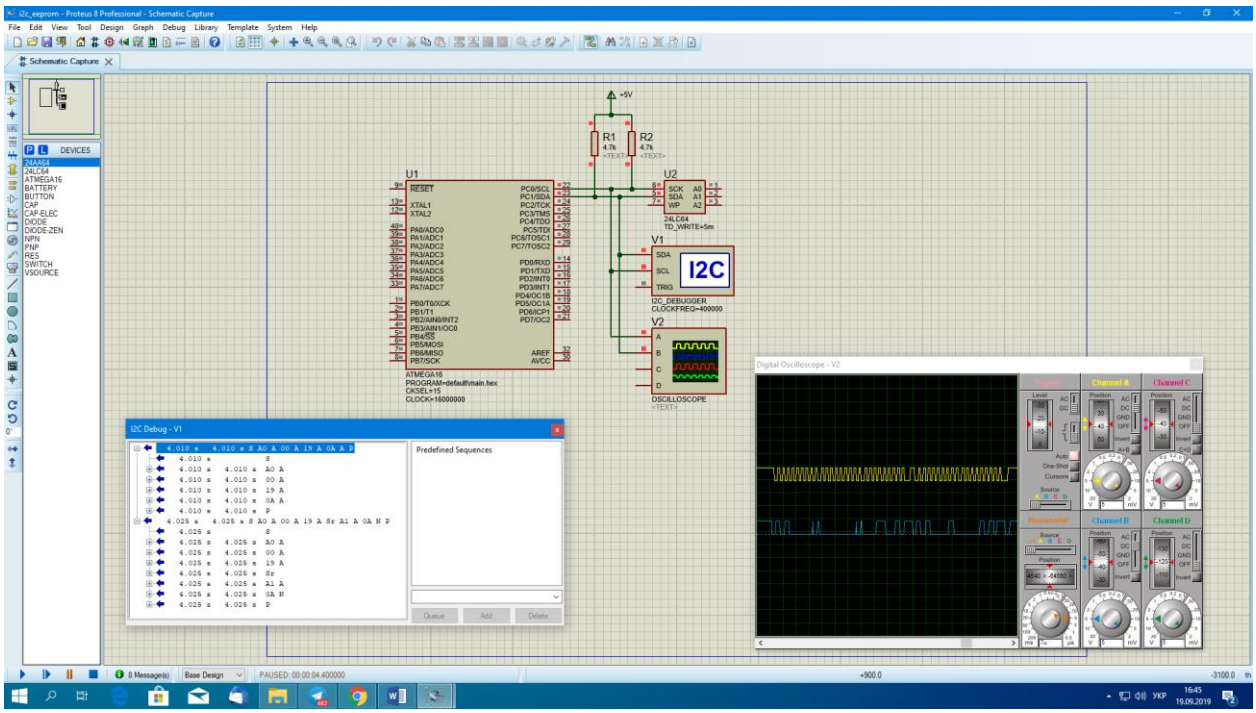
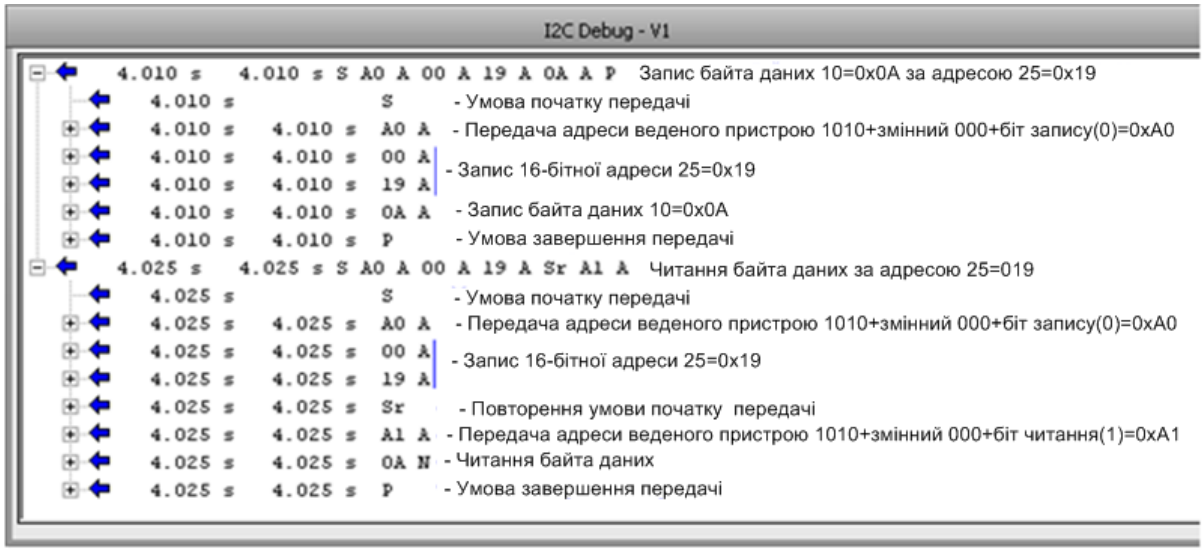


Рисунок 4.67 – Стан моделі після запуску процесу моделювання



A – підтвердження від ведучого або веденого, N – відсутність підтвердження

Рисунок 4.68 – Пояснення окремих етапів обміну шиною TWI (I2C)

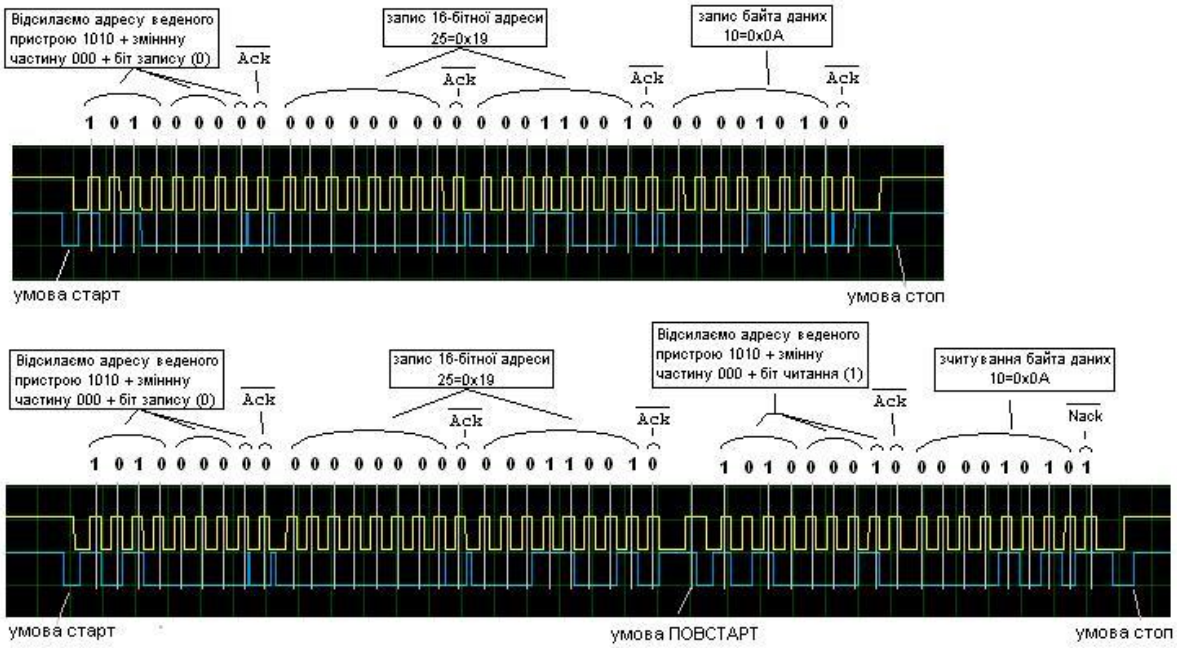


Рисунок 4.69 – Пояснення часових діаграм обміну між мікроконтролером та EEPROM-пам’яттю, отриманих на екрані осцилографа

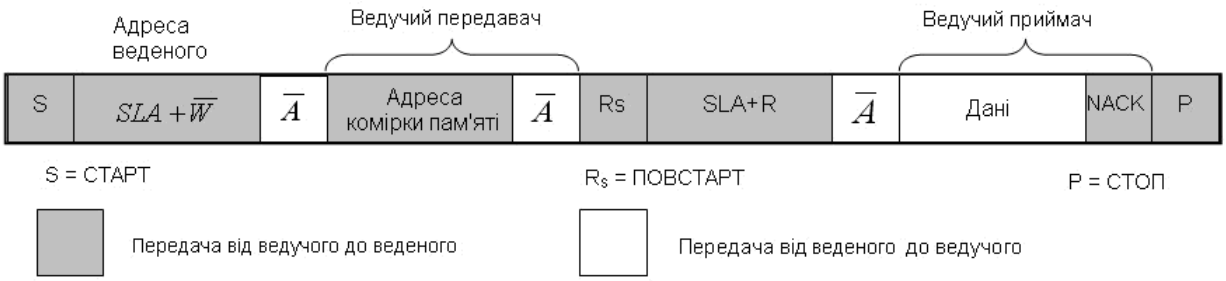


Рисунок 4.70 – Приклад звернення до зовнішньої пам’яті даних типу EEPROM

4.3.6.2.4 Схема алгоритму роботи

Схема алгоритму роботи моделі наведено на рисунках 4.71…4.73.

Згідно з описом роботи моделі, який наведено вище, спочатку МК передає веденому пристрою (EEPROM-пам’яті) байт даних 0x10 за адресою 0x19, а потім читає вміст комірки EEPROM-пам’яті з тією самою адресою. Отримання в цьому випадку значення 0x10 говорить про те, що модель працює правильно. Через те, що МК моделі спочатку працює як ведучий передавач, а потім, як ведений приймач, а EEPROM-пам’ять спочатку виконує функцію веденого приймача, а потім ведучого передавача, нижче на рисунках 4.71…4.73 наведено схеми алгоритмів роботи МК та EEPROM-пам’яті у цих режимах.

Рисунок 4.71 – Схема алгоритму роботи моделі

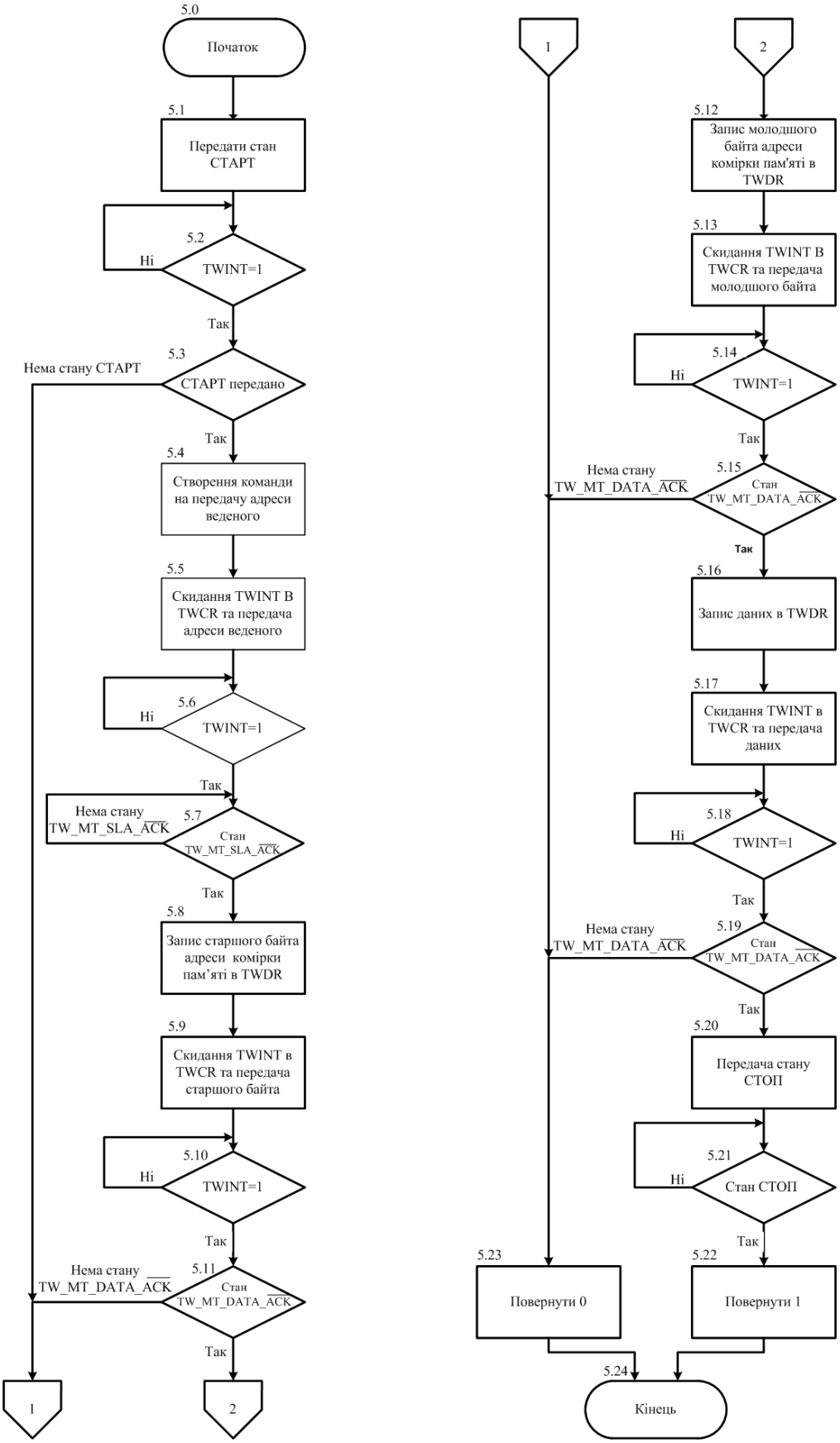


Рисунок 4.72 – Схема алгоритму запису у EEPROM-пам’ять

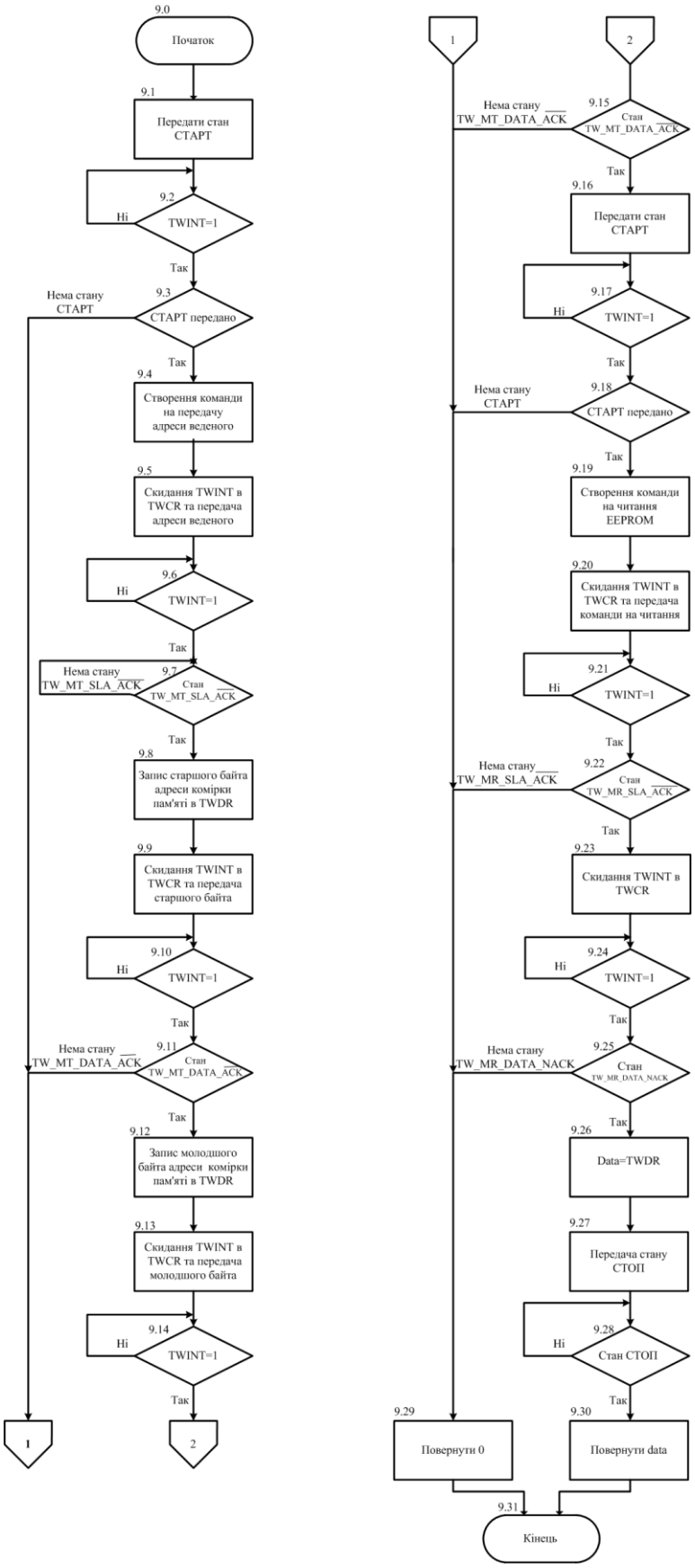


Рисунок 4.73 – Схема алгоритму читання EEPROM-пам’яті

Робоча програма

Всі функції (ініціалізація TWI, читання, запис зовнішньої пам’яті) винесено в окремі файли i2c_eeprom.c і i2c_eeprom.h.

Зміст i2c_eeprom.h

```
#define false 0
#define true 1

#define slaveF_SCL 400000 //400 Khz

#define slaveAddressConst 0b1010
#define slaveAddressVar 0b000

#define READFLAG 1 // read
#define WRITEFLAG 0 // write

void eeInit(void);
uint8_t eeWriteByte(uint16_t address,uint8_t data);
uint8_t eeReadByte(uint16_t address);

                                    // Master
#define TW_START                0x08 //Було сформовано стан СТАРТ
#define TW_REP_START            0x10 //Було сформовано стан ПОВСТАРТ
// Master Transmitter
#define TW_MT_SLA_ACK               0x18 //було передано стан SLA+W і
                                            //прийнято підтвердження ACK
#define TW_MT_SLA_NACK          0x20 //Було передано пакет SLA+W і
                                        //прийнято непідтвердження NACK
#define TW_MT_DATA_ACK          0x28 //Було передано пакет даних і
                                        //прийнято підтвердження ACK
#define TW_MT_DATA_NACK     0x30 //Було передано пакет даних і
                                        //прийнято непідтвердження NACK
#define TW_MT_ARB_LOST          0x38 //Втрата пріоритету при передачі
                                        //пакета адреси або даних
// Master Receiver
#define TW_MR_ARB_LOST          0x38 //Втрата пріоритету при передачі
                                        //пакета адреси або даних
#define TW_MR_SLA_ACK           0x40 //Було передано пакет SLA+R і
                                            //прийнято підтвердження ACK
#define TW_MR_SLA_NACK          0x48 //Було передано пакет SLA+R і
                                        //прийнято непідтвердження NACK
#define TW_MR_DATA_ACK          0x50 //Було прийнято байт даних і
                                        //передано підтвердження ACK
#define TW_MR_DATA_NACK         0x58 //Було прийнято байт даних і
                                        //передано не підтвердження NACK
// Slave Transmitter
#define TW_ST_SLA_ACK       0xA8 //Було прийнято SLA+R з власною
                                    // адресою і послано підтвердження ACK
#define TW_ST_ARB_LOST_SLA_ACK      0xB0  //Втрата пріоритету в
```

```
                                                    //режимі ведучого
   //під час передачі SLA+R/W
#define TW_ST_DATA_ACK          0xB8 //Було передано байт даних і
                                                    //передано підтвердження ACK
#define TW_ST_DATA_NACK         0xC0   //Було передано останній байт
                                                  //даних і отримано
                                                    //непідтвердження NACK
#define TW_ST_LAST_DATA     0xC8 //Було передано останній байт даних
                                                   // і отримано підтвердження ACK
// Slave Receiver
#define TW_SR_SLA_ACK     0x60 //Було прийнято SLA+W з власною
                                                 //адресою і послано підтвердження ACK
#define TW_SR_ARB_LOST_SLA_ACK    0x68 //Втрата пріоритету в
                                                    //режимі ведучого
//під час передачі SLA+R/W
#define TW_SR_GCALL_ACK       0x70 //Було прийнято загальний виклик і
                                                  //послано підтвердження ACK
#define TW_SR_ARB_LOST_GCALL_ACK    0x78 //Втрата пріоритету в
                                                    //режимі ведучого
    //під час передачі SLA+R/W
#define TW_SR_DATA_ACK          0x80 //Пристрій уже адресовано: було
                                                    //прийнято байт даних і
                                                   //послано підтвердження ACK
#define TW_SR_DATA_NACK         0x88 //Пристрій уже адресовано:
                                                  //було прийнято байт даних і
                                                    //послано непідтвердження NACK
#define TW_SR_GCALL_DATA_ACK  0x90 //Пристрій уже адресовано
                                                    //(загальний виклик):
                                                     //було прийнято байт даних і
                                                    //послано підтвердження ACK
#define TW_SR_GCALL_DATA_NACK 0x98 //Пристрій уже адресовано
                                                     //(загальний виклик): було прийнято
//байт даних та послано //непідтвердження NACK
#define TW_SR_STOP     0xA0 //Було виявлено стан СТОП або ПОВСТАРТ
                                      //у той час, коли пристрій було
                                          //адресовано в якості веденого
// Misc
#define TW_NO_INFO              0xF8 //Немає інформації TWINT = 0
#define TW_BUS_ERROR            0x00 //Помилка на шині в результаті
//некоректного формування стану //СТАРТ або СТОП
```

Лістинг файлу i2c_eeprom.c

При написанні програми мовою С у змінні: TW_MT_SLA_ACK; TW_MT_DATA_ACK та TW_MR_SLA_ACK входить скорочення ACK (acknowledge) – підтвердження. Це скорочення відображає отримання сигналу низького рівня (підтвердження) після передачі адреси або даних (рис. 4.72, 4.73).

Щоб підкреслити активний низький рівень цього сигналу на рисунках 4.72, 4.73 він помічений інверсією. Але при написанні програм мовою С використання інверсії заборонено. Тому названі вище змінні мають скорочення ACK без інверсії.

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

#include "i2c_eeprom.h"

void eeInit(void)
{
   /* Налаштування генератора швидкості зв'язку */
   TWBR = (F_CPU/slaveF_SCL - 16)/(2 * /* TWI_Prescaler= 4^TWPS */1);

/*
Якщо TWI працює у ведучому режимі, то значення TWBR повинно бути не менше 10. Якщо значення TWBR менше 10, то ведучий пристрій шини може генерувати некоректні сигнали на лініях SDA і SCL під час передачі байта.*/

   if(TWBR < 10)
      TWBR = 10;

   /*
  Налаштування переддільника у регістрі статусу блоку управління.
  Очищаються біти TWPS0 і TWPS1 регістра статусу, встановлюючи тим самим, значення переддільника = 1.
  */
   TWSR &= (~((1<<TWPS1)|(1<<TWPS0)));
}

uint8_t eeWriteByte(uint16_t address,uint8_t data)
{
   do
   {
        TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN);  //5.1 Передача стану
                                                //СТАРТ

        while(!(TWCR & (1<<TWINT))); //5.2 Очікування встановлення TWINT

        if((TWSR & 0xF8) != TW_START)         //5.3 Перевірка регістру стану
                                               //5.3 return false;

TWDR = (slaveAddressConst<<4) +  //5.4 Створення команди на
(slaveAddressVar<<1) + (WRITEFLAG);  //5.4 передачу

TWCR=(1<<TWINT)|(1<<TWEN);  //5.5 Скидання TWINT в TWCR та
                            // передача адреси веденого
```

```
while(!(TWCR & (1<<TWINT))); //5.6 Очікування встановлення TWINT

   }while((TWSR & 0x18) != TW_MT_SLA_ACK);   //5.7 Перевірка стану
                                                    //5.7 TW_MT_SLA_ACK
 TWDR=(address>>8);           //5.8 Завантаження старшого
                              //байта адреси комірки пам'яті у TWDR

TWCR=(1<<TWINT)|(1<<TWEN);          //5.9 Скидання TWINT та
                                    //передача старшого байта адреси
                                    //комірки в EEPROM

 while(!(TWCR & (1<<TWINT)));   //5.10 Очікування встановлення TWINT


if((TWSR & 0x28) != TW_MT_DATA_ACK)      //5.11 Перевірка стану
                                         //TW_MT_DATA_ACK

return false;                            //5.23

  TWDR=(address);                  //5.12 Завантаження молодшого
                                   // байта адреси комірки пам'яті у TWDR

  TWCR=(1<<TWINT)|(1<<TWEN);           //5.13 Скидання TWINT та
                                       // передача молодшого байта
                                       //адреси комірки в EEPROM
  while(!(TWCR & (1<<TWINT)));    //5.14 Очікування встановлення TWINT
    if((TWSR & 0x28) != TW_MT_DATA_ACK)      //5.15 Перевірка стану
                                             // TW_MT_DATA_ACK
          return false;                      //5.23

                /***** Передача даних в EEPROM ********/


    TWDR=(data);                     //5.16 Завантаження даних у TWDR

    TWCR=(1<<TWINT)|(1<<TWEN);           //5.17 Скидання TWINT та
                                         // передача даних в EEPROM
    while(!(TWCR & (1<<TWINT)));  //5.18 Очікування встановлення TWINT

    if((TWSR & 0x28) != TW_MT_DATA_ACK)      //5.19 Перевірка стану
                                             // TW_MT_DATA_ACK
return false;                                //5.23


TWCR=(1<<TWINT)|(1<<TWEN)|(1<<TWSTO);//5.20 Передача умови СТОП
    while(TWCR & (1<<TWSTO));      //5.21 Очікування передачі умови СТОП

    return true;                     //5.22
}
```

```
uint8_t eeReadByte(uint16_t address)
{
   uint8_t data; // Змінна, до якої записується прочитаний байт
                 /***** Встановлення зв'язку із веденим ********/
   do
   {
      TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN);  //9.1 Передача стану СТАРТ

      while(!(TWCR & (1<<TWINT)));   //9.2 Очікування встановлення TWINT

      if((TWSR & 0xF8) != TW_START)         //9.3 Перевірка регістру стану
         return false;                      //9.29

         //9.4 Створення команди на передачу
      TWDR = (slaveAddressConst<<4) +(slaveAddressVar<<1) + WRITEFLAG;

      TWCR=(1<<TWINT)|(1<<TWEN);    //9.5 Скидання TWINT в TWCR та
                                    // передача адреси веденого
      while(!(TWCR & (1<<TWINT)));  //9.6 Очікування встановлення TWINT

   }while((TWSR & 0x18) != TW_MT_SLA_ACK);    //9.7 Перевірка стану
                                              // TW_MT_SLA_ACK

                 /*****Передача адреси читання********/
   TWDR=(address>>8);                  //9.8 Завантаження старшого
                                       // байта адреси комірки EEPROM у TWDR
TWCR=(1<<TWINT)|(1<<TWEN);             //9.9 Скидання TWINT та
 //передача старшого байта адреси комірки EEPROM
   while(!(TWCR & (1<<TWINT)));    //9.10 Очікування встановлення TWINT

   if((TWSR & 0x28) != TW_MT_DATA_ACK)        //9.11 Перевірка стану
                                              // TW_MT_DATA_ACK
return false;                                 //9.29
   TWDR=(address);                     //9.12 Завантаження молодшого байта
                                       // адреси комірки пам'яті у TWDR
   TWCR=(1<<TWINT)|(1<<TWEN);  //9.13 Скидання TWINT та передача
                               // молодшого байта адреси комірки
                               // в EEPROM
   while(!(TWCR & (1<<TWINT)));   //9.14 Очікування встановлення TWINT

    if((TWSR & 0xF8) != TW_MT_DATA_ACK)   //9.15 Перевірка стану
                                          //TW_MT_DATA_ACK
      return false;                       //9.29

                 /*****Перехід у режим читання********/
/* Знову зв'язок з веденим, тому що раніше було передано адресний пакет (slaveAddressConst
<<4) + (slaveAddressVar <<1) + WRITEFLAG, щоб записати адресу читання байта даних.
Тепер потрібно перейти у режим читання, щоб прочитати байт даних), для цього передаємо
новий пакет          (slaveAddressConst <<4) + (slaveAddressVar <<1) + READFLAG .*/
```

```
   //Повторення умови початку передачі
   TWCR=(1<<TWINT)|(1<<TWSTA)|(1<<TWEN);     //9.16 Передача стану       //Повторний
старт
   while(!(TWCR & (1<<TWINT)));                          //9.17 Очікування TWINT=1

   if((TWSR & 0x10) != TW_REP_START)              //9.18 Перевірка регістру стану
      return false;                                        //9.29

  //9.29 Створення команди на читання
   TWDR = (slaveAddressConst<<4) + (slaveAddressVar<<1) + READFLAG;

         //Передача
TWCR=(1<<TWINT)|(1<<TWEN);           //9. 20 Скидання TWINT та передача
                                                  //адреси веденого
   while(!(TWCR & (1<<TWINT)));          //9. 21 Очікування встановлення TWINT

/* Перевірка, що знайшовся ведений з адресою 1010’000 і він готовий до читання */
   if((TWSR & 0x40) != TW_MR_SLA_ACK)        //9. 22 Перевірка стану
                                                        // TW_MR_SLA_ACK
 return false;                                          //9. 29


                  /*****Прийом  байта даних********/

/* Початок прийому даних шляхом очищення прапорця переривання TWINT. Байт, який буде
прийнятий, записується у регістр TWDR.*/
   TWCR=(1<<TWINT)|(1<<TWEN);                  //9.23 Скидання TWINT та
                                                        //встановлення TWEN=1

   //Чекання закінчення прийому.
   while(!(TWCR & (1<<TWINT)));                //9.24 Чекання закінчення прийому

/* Перевірка регістра статусу. За протоколом, прийом даних повинен закінчуватися без
підтвердження з боку ведучого (TW_MR_DATA_NACK = 0x58)*/
   if((TWSR & 0x58) != TW_MR_DATA_NACK)  //9.25 Перевірка регістра статусу
      return false;                                       //9.29

   data=TWDR;                              //9.26 Надання змінній data значення TWDR

TWCR=(1<<TWINT)|(1<<TWEN)|(1<<TWSTO);  //9.27 Встановлення умови СТОП

   while(TWCR & (1<<TWSTO));    //9.28 Чекання встановлення умови СТОП

   return data;                             //9.30 Повернення прочитаного байта
}                                                 //9.31
```

Створення проекту в AVR STUDIO

У налаштуваннях проекту вказують ATmega16 та підключають файли i2c_eeprom.c та i2c_eeprom.h (рис. 4.74).

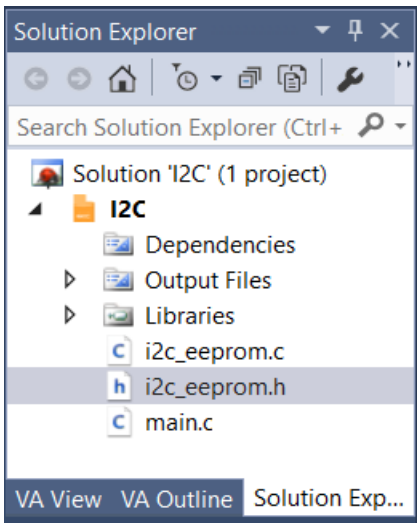


Рисунок 4.74 – Структура проекту

Лістинг файлу main.c

```
#define F_CPU 16000000UL
#include <avr/io.h>                                    //1
#include "util/delay.h"
#include "i2c_eeprom.h"
int main(void)
{
        uint8_t byte = 10;                             //2
        uint16_t address = 25;                         //2

/*Треба почекати, тому що віртуальний осцилограф у Proteus включається із затримкою */
        _delay_ms(300);                                //3

        //Налаштування TWI
        eeInit();                                      //4

        // Запис байта даних 10 = 0xA0 за адресою 25 = 0x19
        // Якщо запис успішний, то повернеться true
        if(eeWriteByte(address, byte))                 //5 ; 6
        {
                // Очищення змінної
                byte = 0;                              //7

                // Чекаєння 15 с, щоб візуально розрізнити осцилограми
                _delay_ms(15);                         //8

                // Читання байта, розташованого за адресою 25 = 0x19
```

```
            byte = eeReadByte(address);          //9

            // Перевірка виведення цього байта у порт виведення (PORTA)

            PORTA = byte;                        //10
        }
}                                                //11
```

4.3.6.3 Зміст звіту

Звіт по роботі повинен містити:

– схему моделі;

– схему алгоритму роботи моделі;

– робочу програму;

– формули за потребою.

Контрольні запитання

1. Якою фирмою було розроблено інтерфейс I^2C та для чого він призначений?
2. Чим відрізняється інтерфейс I^2C від інтерфейсу TWI?
3. З яких двох ліній складається шина інтерфейсу I^2C (TWI) та як вони повінні бути піключені в мережі?
4. Поясніть призначення функції «монтажне АБО/І» та як вона використовується в I^2C (TWI)-мережі?
5. Наведіть та поясніть структурну схему типової мережі, яка використовує для обміну даними шину I^2C (TWI).
6. Опишіть роботу пристроїв, які використовують для обміну інтерфейс I^2C (TWI).
7. Опишіть чотири можливі режими обміну даними інтерфейсом I^2C (TWI).
8. Поясніть призначення станів «СТАРТ» і «СТОП» шини TWI. Яким чином вони формуються?
9. У чому полягає задача розподілу пріоритетів при підключенні до шини TWI декількох ведучих пристроїв? Як вона вирішується?
10. Опишіть формат адресного пакету та пакету даних.
11. Наведіть та поясніть структурну схему типового модуля TWI.
12. Що задає контролер швидкості передачі?

13. Які два вузли входять до складу блоку шинного інтерфейсу? Які їх функції?
14. Що перевіряє блок контролю адреси?
15. При виникненні яких подій блок керування здійснює формування запиту на переривання?
16. Назвіть та опишіть використання регістрів керування TWI-модулем.
17. На чому базується взаємодія прикладної програми з модулем TWI?
18. З яких трьох частин складається будь-який етап взаємодії прикладної програми з модулем TWI? Поясніть цю взаємодію.
19. Назвіть деякі пристрої, які підтримують інтерфейс TWI.
20. У яких режимах може працювати модуль TWI, реалізований у мікроконтролерах сімейства Mega?
21. Опишіть роботу модуля TWI у режимі «Ведучий передавач».
22. Опишіть роботу модуля TWI у режимі «Ведучий приймач».
23. Опишіть роботу модуля TWI у режимі «Ведений приймач».
24. Опишіть роботу модуля TWI у режимі «Ведений передавач».
25. Які можливості надає використання стану «ПОВСТАРТ»?
26. Що трапиться, якщо адресація пристрою відбудеться при перебуванні мікроконтролера у «сплячому» режимі?
27. За якими сценаріями може розвиватися процес арбітражу?
28. З якою швидкістю може виконуватися обмін інформацією в I^2C (TWI)-мережі?
29. На яку відстань може передаватися інформація в I^2C (TWI)-мережі?
30. Скільки ведучих пристроїв може одночасно бути I^2C (TWI)-мережі?
31. Опишіть схему моделювання інтерфейсу TWI у пакеті Proteus 8.6.
32. Опишіть схему алгоритму роботи моделі.
33. Опишіть схему алгоритму запису в EEPROM-пам`ять.
34. Опишіть схему алгоритму читання EEPROM-пам`яті.
35. Поясніть окремі етапи обміну шиною TWI (I2C), які отримано у вікні I2C Debug після запуску моделювання.
36. Поясніть часові діаграми обміну між мікроконтролером та EEPROM-пам`яттю, отриманих на екрані осцилографа після запуску моделювання.

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. : ил.

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. : ил.

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 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.: ил.

7 Евстифеев А.В. Микроконтроллеры AVR семейств Tiny и Mega фирмы ATMEL – М.: Додэка, 2005 – 560 с.: ил.

8 Евстифеев А. В. Микроконтроллеры AVR семейства Mega. Руководство пользователя. – М.:Издательский дом «Додэка-XXI», 2007. – 592 с.: ил.

9 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.

10 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

11 Голубцов М.С., Кириченкова А.В. Микроконтроллеры AVR: от простого к сложному. – М.: Солон-Пресс, 2005.

12 Трамперт Вольфган. AVR-RISC микроконтроллеры. – Перевод с немецкого. – Киев.: МК – Пресс, 2006.

13 Баранов В.Н. Применения микроконтроллеров AVR: схемы, алгоритмы, программы. – М.: Додэка, 2004.

14 Ревич Ю. В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера. – 2-е изд., испр. – СПб.:БХВ-Петербург,2011.

15 Кравченко А. В. 10 практических устройств на AVR-микроконтроллерах. Книга 1 – М.: Издательский дом «Додэка-XXI», К. «МК-Пресс», 2008.

16 Кравченко А. В. 10 практических устройств на AVR-микроконтроллерах. Книга 2 – К. «МК-Пресс», СПб.: «КОРОНА-ВЕК», 2009.

17 Кравченко А. В. 10 практических устройств на AVR-микроконтроллерах. Книга 3 – К. «МК-Пресс», СПб.: «КОРОНА-ВЕК», 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