МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

ОСНОВИ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ

ЛАБОРАТОРНИЙ ПРАКТИКУМ

Рекомендовано Методичною радою КПІ ім. Ігоря Сікорського
як навчальний посібник для здобувачів ступеня бакалавра за освітніми програмами
«Акустичні електронні системи та технології обробки акустичної інформації»,
«Електронні прилади та пристрої»
спеціальності 171 «Електроніка»

Київ
КПІ ім. Ігоря Сікорського
2021

Основи мікропроцесорної техніки: Лабораторний практикум [Електронний ресурс]: навч. посіб. для студ. спеціальності 171 «Електроніка» / КПІ ім. Ігоря Сікорського; уклад.: Т. О. Терещенко, Л. М. Батрак, Ю. С. Ямненко. – Електронні текстові дані (1 файл: 5,51 Мбайт). – Київ: КПІ ім. Ігоря Сікорського, 2021. – 59 с.

Гриф надано Методичною радою КПІ ім. Ігоря Сікорського (протокол № 04/2021 від 26.04.2021 р.) за поданням Вченої ради Факультету електроніки (протокол № 7 від 13.05.2021 р.)

Електронне мережне навчальне видання

ОСНОВИ МІКРОПРОЦЕСОРНОЇ ТЕХНІКИ
ЛАБОРАТОРНИЙ ПРАКТИКУМ

Укладачі: *Терещенко Тетяна Олександрівна,* д-р техн. наук, проф.
Батрак Лариса Миколаївна, канд. техн. наук, доц.
Ямненко Юлія Сергіївна, д-р техн. наук, проф.

Відповідальний редактор *Клен К.С.,* канд. техн. наук, доц.

Рецензент *Найда С.А.,* д-р. техн. наук, проф., завідувач кафедри акустичних та мультимедійних електронних систем КПІ ім. Ігоря Сікорського

У посібнику наведено методичні вказівки до виконання лабораторних робіт з курсу "Основи мікропроцесорної техніки". Матеріал присвячений вивченню 8-розрядних мікроконтролерів AVR сімейства Mega. Лабораторні роботи спрямовані на вивчення архітектури мікроконтролера, роботи портів введення / виводу, системи таймерів і переривань. Для написання програм застосовується мова асемблера. Розглянуто програмні засоби розробки і налагодження програм. Навчальний посібник сприяє набуттю практичних навичок створення програмного забезпечення мікропроцесорних систем та методів їх налагодження в середовище AVR Studio та емуляції систем на базі мікроконтролера AVR в середовищі Proteus.

ЗМІСТ

ВСТУП..... 4
ЛАБОРАТОРНА РОБОТА 1. Ознайомлення з середовищем AVR Studio та створення простих програм. Виконання арифметичних операцій. 5
ЛАБОРАТОРНА РОБОТА 2. Програмна модель МП AVR. Програмні засоби звертання до різних сегментів пам‘яті AVR процесорів. 9
ЛАБОРАТОРНА РОБОТА 3. Команди умовних переходів..... 13
ЛАБОРАТОРНА РОБОТА 4. Організація часових затримок за допомогою циклів. Емуляція роботи в Proteus..... 17
ЛАБОРАТОРНА РОБОТА 5. Таймери та система переривань МП AVR .. 25
СПИСОК ЛІТЕРАТУРИ 32
Додаток А. ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ AVR STUDIO 4 ... 34
Додаток Б СИСТЕМА КОМАНД AVR..... 45
Додаток В. ДИРЕКТИВИ АСЕМБЛЕРА..... 53
Додаток Г. ЗАГАЛЬНА ХАРАКТЕРИСТИКА СЕРЕДОВИЩА PROTEUS VSM 54

ВСТУП

Метою виконання циклу лабораторних робіт є:

- закріплення і експериментальна перевірка теоретичних положень найважливіших розділів і тем навчального материалу;
- оволодіння інтегрованими програмними комплексами відпрацювання прикладного програмного забезпечення, що спрощують цей процес;

В процесі виконання лабораторних робіт студенти вивчають мову асемблера AVR процесорів, яка відображає архітектуру мікропроцесорної системи. Для грамотного написання програми на мові асемблера потрібно, в загальному, знати архітектуру процесора.

Оптимальною можна вважати програму, яка працює правильно, по можливості швидко і займає малий обсяг пам'яті. Крім того, її легко читати і розуміти; її легко змінити; її створення вимагає мало часу і незначних витрат.

Під час виконання лабораторних робіт студенти відпрацьовують основні етапи створення та налагодження програмного забезпечення мікропроцесорних пристроїв на мові асемблера AVR процесорів .

ЛАБОРАТОРНА РОБОТА 1.

Ознайомлення з середовищем AVR Studio та створення простих програм. Виконання арифметичних операцій.

Мета роботи: Знайомитися з програмним забезпеченням AVR Studio та основами програмування на мові Асемблер. Придбати практичні навички написання, налагодження та виконання програм, написаних мовою асемблеру для програмування арифметичних виразів ініціалізації портів та здійснення операцій введення/виведення для МК AVR.

Порядок виконання роботи

1. Ознайомитися з інтегрованим середовищем розробки та відпрацювання прикладного програмного забезпечення AVR Studio (дод. А).

2. Ознайомитися з інструкціями та регістрами процесора, які дозволяють виконувати арифметичні та логічні дії (дод. Б).

3. Розрахувати вихідні дані для виконання завдання по формулам:

$$a = 9*N + I,\ b = 8*N + \text{I},$$

де N - порядковий номер за журналом; I - індекс групи, (видає викладач).

4. Завантажити програму в AVR Studio 4 наступною послідовностею дій:

New Proect >> Atmega AVR assembler >> Імя Путь (діректорія і імя латинецею) >> Вибір МП Atmega 16 >> Finish (дод. А).

5. Створити пробний проект.

6. Провести компіляцію і вилучити виявлені помилки.

7. Провести покрокове відпрацювання програми з використанням дебагера AVR Studio та заповнити табл. 1.1. Пояснити зміну стану регістрів.

Таблиця 1.1.

I =…	N=…	a=…	b=…
№ кроку	Команда, що виконується	Значення в r16	Опис команди
1			
…			

Програма для ознайомлення з роботою AVR Studio

```
.include "m16def.inc"
.cseg
.equ a=10
.equ b=9
reset :
        ldi r16, a
        ldi r17, b
        inc r16
        lsl r16
        swap r16
        rjmp case_2
case_1:
        add r16, r17
        ori r16,0x0aa
        and r16, r17
        sbr r16, 3
        rjmp reset
case_2:
        cbr r16, 3
        ori r16, 0x55
        and r16, r17
        add r16, r17
rjmp reset
```

Таблиця 1.1

Приклад заповнення

I=1	N=1	a=9*1+1=10	B = 8*1 + 1 = 9
№ кроку	Команда, що виконується	Значення в r16	Опис команди
1	ldi r16, a	10	Завантажити в r16 число 10
2	ldi r17, b	10	Завантажити в r17число 9
3	inc r16	11	Інкрементувати r16
4	lsl r16	22	Логічний зсув вліво r16
5	swap r16	22	Змінити місцями старший та молодший полубайти

Виконання програми злійснюється послідовними операціями Build та Debag (дод. А).

8. Написати програму, що реализує математичні операції та відладити ii у програмному середовище AVR Studio. Змінні X, Y, Z знаходяться в регістрах МП згідно табл. 1.2. Початкові значення X, Y задати самостійно.

9. Пояснити результат, виконавши математичні та логічні операції.

Таблиця 1.2.

№ п.п	Математичне рівняння	X	Y	Z
1.	Z = (200+X – 1) ∨ (Y*2)	R16	R17	R2
2.	Z = (2*·X + 1) – (Y∧ 34)	R1	R2	R3
3.	Z = (25 ⊕ X + 20) - (Y*2)	R4	R5	R6
4.	Z = ((100+X)/2 - (Y∧ 60)	R7	R8	R9
5.	Z = (130 ⊕ X - Y +1) / 2	R8	R9	R10
6.	Z = (X⊕ 22 -Y - 78)*2	R9	R10	R11
7.	Z = ((200+Y) ⊕ X + 1) / 2	R9	R10	R11
8.	Z = ((200⊕X+1) / 2) ∨ 17	R10	R11	R12
9.	Z = (20-Y⊕X + 1)*·2	R13	R14	R15
10.	Z = (((33+Y) /2) + 1) ∨ X	R16	R17	R18
11.	Z = ((X ∨ Y + 1)-250) * 2	R19	R20	R21
12.	Z = (2·X ⊕ 99) ∧ Y	R22	R23	R25
13.	Z = (25 + X /2)∨ 13- Y	R23	R24	R25
14.	Z = (X/2 - Y + 1) ∨ 31	R26	R27	R28
15.	Z = ((X + 10) ∨ Y + 1) * 4	R27	R28	R30
16.	Z = ((X + Y) /2 + 1) ∨ 1	R31	R1	R2
17.	Z = X ⊕ 3 + 4 *Y	R2	R3	R4
18.	Z = (X ∨ 11) – 2*(Y + 1)	R3	R4	R5
19.	Z = (X ∨ 22) /2 - (Y + 2)	R4	R5	R6
20.	Z = X ∨ 33 *4 + 3	R5	R6	R7
21.	Z = (10 ⊕ X * 2) +Y	R6	R7	R8
22.	Z = X ⊕ 54 + 2*Y	R7	R8	R9
23.	Z =(X+Y/2) ∨ 33-1	R8	R9	R10
24.	Z = X/2- (Y∨120 + 1)	R9	R10	R11
25.	Z = 45⊕122 /2-Y+1	R10	R11	R12

Примітка. 1. Множення і ділення безнакові. При діленні не враховувати остачу (результат – тільки ціла частна). 2. Арифметичні дії можна проводити з усіма регістрами R0 - R31, але безпосередні дані можна завантажувати лише в R16 - R31.

Зміст звіту

1. Програма з коментарями.
2. Висновки до лабораторної роботи.

Контрольні питання:

1. Опишіть порядок створення програм в середовищі AVR Studio.
2. Опишіть порядок налагоджування програм в середовищі AVR Studio.
3. Поясніть з якими групами регістрів працює відладчик AVR.
4. Які переваги програм на асемблері?

5. Назвіть область застосування асемблера.

6. Назвіть арифметичні та логічні команди AVR Atmega.

7. Перетворіть на десятковий код наступні двійкові числа:

8. а) 0001; б) 0101; в) 1000; г) 1011; д) 1111; е) 0111.

9. Виконайте додавання чисел 10110011_2 та 11110010_2 з перевіркою результату у десятковій системі.

10. Виконайте віднімання чисел 132_{10} та 77_{10} у двійковому вигляді.

11. Виконайте множення чисел 110110_2 та 111_2 з перевіркою результату у десятковій системі.

12. Вкажіть розрядність результату множення чисел 11011100_2 та 11101111_2.

13. Перетворіть на шістнадцятковий еквівалент числа: а) 1001_2; б) 1100_2; в) 01111110_2; г) 11011011_2.

ЛАБОРАТОРНА РОБОТА 2.

Дослідження програмної моделі МП AVR. Програмні засоби звертання до різних сегментів пам‘яті AVR процесорів

Мета роботи: Поглиблення та закріплення знаннь щодо програмної моделі AVR процесорів. Придбання практичних навичок складання, налагодження і виконання програм, написаних мовою асемблеру для звертання до різних сегментів пам‘яті AVR процесорів.

Порядок виконання роботи

1. Провести ініціалізацію констант X1, X2, X3, які розміщуються в різних сегментах пам'яті мікроконтролера за різними адресами.

2. Виконати задану в завданні математичну обробку.

3. В залежності від результату обробки (парне чи непарне число), вивести інформацію на зазначені в завданні порти мікроконтролера.

4. Варіанти завдань до лабораторної роботи наведені в табл. 2.1.

Таблиця 2.1

№	X1	X2	X3	Y		
	Сегмент, адреса	Сегмент, адреса	Сегмент, адреса	Операція	Реакція – вивод результата на порт	
					Непарний результат	Парний результат
1	ROM, $100	SRAM, $0007	EEPROM, 5	X1+X2∨X3	PORTA	PORTB
2	ROM, 200	EEPROM, 7	SRAM, $0033	X1x(X2•X3)	PORTB	PORTC
3	SRAM, $0070	EEPROM, 55	ROM, 400	X1•X2-X3	PORTC	PORTD
4	SRAM, $001C	ROM, 70	EEPROM, 16	X1⊕(X2+X3)	PORTD	PORTA
5	EEPROM, $1F	ROM, $A00	SRAM, $005E	(X1∨X2)xX3	PORTA	PORTD
6	EEPROM, 2	SRAM, $0065	ROM, $100	X1+X2•X3	PORTC	PORTD
7	ROM, $100	SRAM, $0010	EEPROM, 35	X1∨X2-X3	PORTB	PORTA
8	ROM, $500	EEPROM, 1	SRAM, $005D	X1+(X2⊕X3)	PORTA	PORTB
9	SRAM, $0070	EEPROM, 4	ROM, $200	X1∨(X2xX3)	PORTD	PORTC
10	SRAM, $0001	ROM, $200	EEPROM, 15	X1x(X2-X3)	PORTC	PORTD
11	EEPROM, 2	ROM, $200	SRAM, $0033	X1+X2⊕X3	PORTB	PORTC
12	EEPROM, 6	SRAM, $0100	ROM, $200	(X1+X2)xX3	PORTA	PORTB

Зміст звіту

1. Програма з коментарями

2. Висновки до лабораторної роботи.

Теоретичні відомості

Директива BYTE резервує байти в ОЗУ. Якщо треба мати можливість посилатися на виділену область пам'яті, то директива BYTE повинна мати мітку. Директива приймає один обов'язковий параметр, який вказує кількість виділених байт. Ця директива може використовуватися тільки в сегменті даних. Виділені байти не ініціалізуються (див. дод. В).

```
; резервування байту для X1 в сегменті даних
.dseg
.org 0x100 ;початкова адреса розміщення в сегменті
X1: .byte x1val
```

Директива DB резервує необхідну кількість байт в пам'яті програм або в EEPROM. Якщо треба мати можливість посилатися на виділену область пам'яті, то директива DB повинна мати випереджу мітку. Дана директива може бути розміщена тільки в сегменті програм (CSEG) або в сегменті EEPROM (ESEG). (див. дод. В).

Резервування байту X2 в сегменті кодів

```
.cseg
.org 0x200
X2: .db x2val
```

Резервування байту X3 в додатковому сегменту даних

```
.eseg  ; додатковий сегмент даних для X3
.org 0x006
X3: .db x3val
```

Читання/запис SRAM здійснюється командами

```
sts X1, r16    ; ініціалізує константу в SRAM
lds r17, X1    ; завантажує константу X2 в R17
```

Читання/запис EEPROM здійснюється підпрограмами

```
EERead:
        SBIC    EECR,EEWE     ; Чекаємо закінчення минулої процедури запису
        RJMP    EERead
        LDI     r16,    low(X3)          ; записуємо адресу потрібної комірки
        OUT     EEARL, R16
        LDI     r16,    high(X3)
        OUT     EEARH, R17
        SBI     EECR,EERE     ; Записуємо  біт читання
        IN      R21, EEDR     ; читаємо результат з EEDR
        RET

EEWrite:
        SBIC    EECR,EEWE     ; Чекаємо закінчення минулої процедури запису
        RJMP    EEWrite
        CLI                   ; забороняємо переривання.
        LDI      r16,  low(X3)     ; записуємо адресу потрібної комірки
        OUT     EEARL, R16
        LDI      r16,   high(X3)
        OUT     EEARH, R17
        OUT     EEDR,R21      ; та дані
        SBI     EECR,EEMWE          ; зводимо предохранитель
        SBI     EECR,EEWE           ; записуємо байт
        SEI                         ;дозволяємо переривання
        RET
```

Читання ROM

```
; Завантажуємо константи X2 з FLASH пам'яті
ldi ZH, high(2*X2)    ; завантажує адресу молодшого байту
;константи x3 в регістрову пару Z
ldi ZL, low(2*X2)     ; завантажує адресу старшого байту константи x3 в регістрову
пару Z
lpm     ; завантажує байт з пам'яті програми
```

Контрольні питання

1. Яка організація пам`яті AVR-мікроконтролерів?

2. Наведіть карту розподілу пам`яті.

3. Назвіть особливості організації пам’яті програм AVR мікроконтролерів.

4. Яким чином відбувається адресація пам’яті програм?

5. Назвіть особливості організації пам’яті статичного запам’ятовуючого пристрою.

6. Назвіть призначення та склад регістрів введення виведення. Команди роботи з ними.

7. Назвіть призначення та склад файлового регістру. Наведіть приклади команд роботи з ним.

8. Охарактиризуйте енергонезалежну пам'ять даних. Наведіть приклади команд роботи з нею.

9. Як відбувається ініціалізація області EEPROM?

10. Наведіть фрагмент програми ініціалізації стеку.

11. Які регістри призначені для читання та запису комірок EEPROM?

12. Яке призначення регістру адреси EEPROM?

13. Яке призначення регістру даних EEPROM пам’яті?

14. Яке призначення регістру керування EEPROM - пам’яті?

15. На які сегменти ділиться пам’ять мікроконтролерів AVR?

ЛАБОРАТОРНА РОБОТА 3.

Команди умовних переходів

Мета роботи: Вивчення особливостей адресації в командах умовного і безумовного переходів та Придбання практичних навичок використання команд умовних переходів, налагодження та ініціалізації портів, здійснення операцій введення/виведення.

Порядок виконання роботи

1. Написати програму на мові асемблеру та коментар до кожної команди. Завдання наведено в табл. 3.1.

2. Знайти дві пари чисел для двох гілок розгалуження програми.

3. Визначити час виконання програми та об‘єм пам‘яті програми.

Таблиця 3.1.

№	Завдання
1. .	2.
1.	Виконати операцію додавання по модулю 2 над регістрами R8, R9. У випадку встановлення прапора N результат зберегти в комірці ОЗП за адресою 0FA, інакше записатити в цю комірку число 3F
2.	Виконати операцію віднімання над числами, що знаходяться в регістрах R17 та R18. Результат зберегти в регістрі R17. У випадку переповнення при роботі із знаковими числами замість результату записати число 7FH
3.	Виконати операцію додавання над комірками пам‘яті за адресами 60H та 61H. Результат зберегти в регістрі R20. У випадку наявності переносу замість результату записати число 7FH.
4.	Виконати операцію віднімання над комірками, розташованими у пам‘яті послідовно, починаючи з адреси 6FH. Результат записати у наступну комірку пам‘яті. У випадку займу замість результату записати число 7FH
5.	Виконати операцію додавання над числами, що знаходяться в регістрах R3 та R4. Результат зберегти в регістрі R17. У випадку встановлення прапорця переносу результат збільшити на одиницю, у протилежному випадку – зменшити на одиницю
6.	Виконати операцію додавання над регістрами R16, R17. У випадку встановлення напівпереносу результат зберегти в комірці пам‘яті за адресою 70, в іншому випадку, замість результату записати число 7F.
7.	Виконати операцію додавання над регістрами R16, R17. У випадку встановлення напівпереносу результат зберегти в комірці пам‘яті за адресою 70, в іншому випадку, замість результату записати число 7F.

8.	Виконати операцію віднімання над регістрами R2 та R3. Результат зберегти в регістрі R17. У випадку наявності займу замість результату записати число 7FH
9.	Виконати операцію додавання по модулю два над регістрами R1 та R2. Результат зберегти в регістрі R17. У випадку встановлення прапорця нуля замість результату записати число 7FH
10.	Виконати логічне АБО операцію регістрами R1 та R2. Результат зберегти в регістрі R17. У випадку встановлення прапорця N замість результату записати число 1FH.
11.	Виконати логічну операцію регістрами I R3 та R4. Результат зберегти в регістрі R17. У випадку встановлення прапорця S замість результату записати число 1FH
12.	Виконати логічну операцію регістрами ВИКЛЮЧАЛЬНЕ АБО над R3 та R4. Результат зберегти в регістрі R17. У випадку встановлення прапорця Z замість результату записати число 1FH

Зміст звіту

1. Програма з коментарями
2. Висновки

ТЕОРЕТИЧНІ ВДОМОСТІ

Команди умовних переходів наведено в Додатку Б.

Команди BRNE і BREQ здійснюють перехід залежно від стану Z прапора у регістрі SREG.

Команди BRBS і BRBC працюють з будь-яким бітом SREG.

Крім того, є команди:

SBRC[РРЗП], [номер біту] - пропускає наступну команду, якщо даний біт в РРЗП рівний 0.

SBRS[РРЗП], [номер біту] - пропускає наступну команду, якщо даний біт в РРЗП рівний 1.

SBIC[РВВ], [номер біту] - пропускає наступну команду, якщо даний біт в регістрі вводу-виводу рівний 0.

SBIS[РВВ], [номер біту] - пропускає наступну команду, якщо даний біт в регістрі вводу-виводу рівний 1.

Ці команди можна використовувати лише після команд, що встановлюють прапори.

Приклад Виконати операцію додавання над регістрами R16, R17. У випадку переповнення при роботі із знаковими числами результат зберегти в комірці ОЗП за адресою 0FC, якщо переповнення немає, то замість результату записати число 7F.

```
.include "m16def.inc"
.cseg
  reset :
  ldi R16,0x99 // Завантаження константи в регістр R16
  ldi R17,0x99 // Завантаження константи в регістр R17
  add R16, R17 // Додавання регістрів
  brvs carry; // Команда умовного переходу по прапорцю
  sts $0FC, R16 // Збереження значення регістру в комірці ОЗП
  rjmp ncarry // Команда розгалуження для іншого випадку
  carry:
  ldi R16, 0x7f // Запис числа замість результату
  ncarry: nop
  rjmp reset // Зациклення програми
```

1) Для 1-го випадку з переповненням при R16 = 99, R17 = 99 результат дорівнює R16 = 198.

Для 2-го випадку без переповнення R16 = 1, R17 = 2 результат дорівнює R16 = 3.

Під час відлагодження програми після Build знаходимо об‘єм пам‘яті програми - 22 байта (.cseg used)

2) Час виконання програми знаходимо в режимі Debug: 7 циклів в першому випадку і 12 циклів в другому. Цикл триває при 16 МГц - 1:16000000 = 0,0625 мкс.

Контрольні питання

1. Які групи команд послідовності не впливають на прапорці?

2. Укажіть призначення регістру SREG.

3. Навести приклад команд після яких встановиться прапор Z.

4. Навести приклад послідовності команд після яких скинеться прапор Z.

5. Навести приклад послідовності команд після яких встановиться прапор *N*.

6. Наведіть приклад послідовності команд після виконання яких скинеться прапор *N*

7. Наведіть приклад послідовності команд після виконання яких встановиться прапор *S*

8. Наведіть приклад послідовності команд після виконання яких скинеться прапор *S*

9. Наведіть приклад послідовності команд після виконання яких встановиться прапор *V*

10. Наведіть приклад послідовності команд після виконання яких скинеться прапор *V*

11. Наведіть приклад послідовності команд після виконання яких встановиться прапор *C*

12. Наведіть приклад послідовності команд після виконання яких скинеться прапор *C*.

13. Наведіть приклад послідовності команд після виконання яких встановиться прапор *H*.

14. Наведіть приклад послідовності команд після виконання яких скинеться прапор *H*.

15. Коли застосовується прапор *AF?* Для яких операндів?

ЛАБОРАТОРНА РОБОТА №4.

Організація часових затримок за допомогою циклів.

Емуляція роботи в Proteus

Мета роботи: навчитися досліджувати програмні засоби організації часових затримок за допомогою циклів. Отримати навички роботи в програмному середовищі Proteus.

Порядок виконання роботи

5.1 Написати програму на мові асемблеру та коментар до кожної команди відповідно до завдання (табл. 5.1.). Створити принципову схему мікропроцесорної системи (див. дод. Г).

Таблиця 5.1

Конфігурація схеми

Варіант	Підключення LED	Підключення Button
1.	PA1- LED- Green, PA2- LED- Blue, PA3 - LED- Red	PB0 – BUTTON1, PB1 – BUTTON2, PB2 – BUTTON3
2.	PA4- LED- Green	PB0 – BUTTON1, PB1 – BUTTON2
3.	PA1- LED- Green, PA2- LED- Blue, PA3 - LED- Red	PB0 – BUTTON1, PB1 – BUTTON2, PB2 – BUTTON3
4.	PA4- LED- Green	PB0 – BUTTON1, PB1 – BUTTON2, PB2 – BUTTON3
5.	PA4- LED- Green	PB0 – BUTTON1, PB1 – BUTTON2
6.	PA3 - LED- Red.	Період миготіння змінюється з часом по програмі
7.	PA1- LED- Green, PA2- LED- Blue, PA3 - LED- Red	PB0 – BUTTON1, PB1 – BUTTON2
8.	PA1- LED- Green, PA3 - LED- Red	PB0 – BUTTON1, PB1 – BUTTON2
9.	PA4- LED- Green	PB0 – BUTTON1
10.	PA4- LED- Green	PB0 – BUTTON1
11.	PA4- LED- Green	PB0 – BUTTON1, PB1 – BUTTON2
12.	PA4- LED- Green	PB0 – BUTTON1

5.2. Згідно з завданням по варіанту табл. 5.2. створити HEX файл програми в AVR Studio і завантажити його в проект Proteus (див. дод. Г).

5.3. Відлагодити роботу схему в Proteus і продемонструвати дію кнопок та світлодіодів.

Таблиця 5.2

Вимоги до програми

Варіант	**Час включення/виключення світлодіодів (мс)**	**Функції, що задаються кнопками Button**
1.	LED- Green - 200/200 LED- Blue 200/200 LED- Red - 200/200	При натисканні BUTTON1 – робота миготіння LED- Green, BUTTON2 – робота миготіння LED- Blue , BUTTON3 – робота миготіння LED- Red. Кнопки незалежні, при натисненні всіх – миготять всі світлодіоди.-
2.	LED- Green 500/500	При одночасному натисканні кнопок BUTTON1, BUTTON2 – світлодіод миготить, при віджиманні хоча б однієї кнопки – діод зупинити.
3.	LED- Green, 500/500 LED- Blue, 500/500 LED- Red - 500/500	При натисканні BUTTON1 – миготіння LED- Green,, при віджиманні – зупинити. При натисканні BUTTON2 – миготіння LED- Blue, при віджиманні – зупинити. При натисканні BUTTON3 – миготіння LED- Red, при віджиманні – зупинити.
4.	Період миготіння LED- Green залежить від стану кнопок	При натисканні BUTTON1 – період миготіння LED- Green 250, BUTTON2- 500, BUTTON3 – 1с. При віджиманні – зупинити.
5.	Період миготіння LED- Green залежить від стану кнопок	При одночасному натисканні кнопок BUTTON1, BUTTON2 період миготіння LED- Green 250, при натисканні тільки BUTTON1 – період миготіння- 500, BUTTON2 – період миготіння 1с. Якщо не нажато жодної кнопки – зупинити.
6.	LED- Red Період миготіння змінюється з часом	При включенні період миготіння світлодіоду 0,125 с впродовж 1 сек, через 1 сек – 0,5 с також в продовж 1 сек. Потім процес повторюється
7.	При натисканні BUTTON1 – горять LED- Green та LED- Blue – 1сек, при натисканні BUTTON2- загоряється LED- Red 1 сек.	
8.	LED - Red горить до тих пір, поки не натиснена кнопка BUTTON1. Після натисканні на BUTTON2 LED- Green горить 0,25с.	
9.	При її натисканні BUTTON1 видати серію з п'яти світлових імпульсів за допомогою світлодіода LED-Green наступної тривалості: 0,25 с - 0,5 с - 1 с - 0,5 с - 0,25 с. Пауза між усіма імпульсами постійна і дорівнює 0,5 с. Після видачі серії імпульсів знову очікувати натискання кнопки	
10.	При натисканні на кнопку BUTTON1 протягом 1 секунди, переключити світлодіод LED-Green в протилежний стан. Якщо тривалість натиснення менше 1 секунди, нічого не здійснювати.	
11.	При включенні і двох натиснутих кнопок час включеного/ відключеного стану LED- Green - 200/200	При одночасному натисканні кнопок BUTTON1, BUTTON2 – робота (миготіння світлодіода), при віджиманні BUTTON1 – зміна часу включеного/ відключеного стану на 500/500, при віджиманні BUTTON2 – зміна часу на 1с/1с, при віджиманні двох кнопок – зупин роботи

12.	LED- Green. Миготіння здійснюється імпульсами типу меандо. Період миготіння залежить від кількості натискань кнопки.	При однократному натисненні період миготіння 0,25 с , при 2 кратному – 0,5 с, при трикратному – 1с. Збільшення кількості натиснень більш 3 не впливає на період. При віджиманні кнопки – зупин роботи

5.3. Відлагодити роботу схему в Proteus і продемонструвати дію кнопок та світлодіодів.

Теоретичні відомості

Реалізація затримок

Найпростіший спосіб реалізації затримок оснований на підрахунку тактів в підпрограмі затримки.

Приклад 1.

```
ldi r17,250; Завантажимо в регістр r17 число 250 (кількість повторень циклу)
 Delay:
 dec r17; Зменшуємо на 1 значення r17
 brne Delay; Якщо не 0 то переходимо до мітки Delay
```

Команда декременту виконується за 1 такт, команда умовного переходу - за 2 такти, отже цикл виконується за 3 такти.

Загальна затримка буде 250 * 3 = 750 тактів. Для AVR Mega 16 – це $750/(16 * 10^6)$ м = 47 мкс.

Якщо треба більший час затримки використовують такий прийом, як вкладені цикли. Так для 3 вкладених циклів можна отримати максимальну затримку що дорівнює 255 х 255 х 255 = 16,5 млн тактів.

Приклад 2. Організувати затримку 250 мс. Для спрощення підрахунку вставимо команду NOP перед декрементом, щоб виконання циклу зайняло 4 такти. Тоді один внутрішній цикл виконується за 4/16,000,000=0,00000025 с. Таким чином потрібно 0,25/0,00000025 = 1,000,000 тактів. Представимо 1000000 як 250*250*16. Множники не перевищують 255 для того, щоб можна

було використовувати 8 розрядний регістр. Значення множників визначають початкові значення лічильників для 3 вкладених циклів.

```
Delay250ms:
ldi r17,250    ; Завантажимо в регістр r17 число 16
Cycle3:
 ldi r18,250   ; Завантажимо в регістр r18 число 250
 Cycle2:
 ldi r19,250   ; Завантажимо в регістр r19 число 250
 Cycle1:
 nop           ; Просто тратимо такт впусту
 dec r19       ; Зменшуємо на 1 значення r19
 brne Cycle1 ; Якщо не 0 то переходимо до мітки Cycle1
 dec r18       ; Зменшуємо на 1 значення r18
 brne Cycle2 ; Якщо не 0 то переходимо до мітки Cycle2
 dec r17       ; Зменшуємо на 1 значення r17
 brne Cycle3 ; Якщо не 0 то переходимо до мітки Cycle3
 ret
```

Приклад 3. Написати програму та створити принципову схему мікропроцесорної системи для реалізації послідовної роботи 3 світлодіодів (гірлянда). Підключення світлодіодів наступне: Green - PA1, Blue - PA2, Red - PA3. Час включення/виключення світлодіодів 250 мс. Підключення Button- PB1. При натисканні BUTTON відбувається робота світлодіодів, при віджиманні – зупинка роботи.

В програмі використовується підпрограма Delay250ms з прикладу 1.

```
.include "m16def.inc"
.equ LEDG = 0
.equ LEDB = 1
.equ LEDR = 2
.equ BUT  = 0
.CSEG
.ORG 0x00
start:
                                ;для роботи підпрограм треба стек
    ldi r16, low(RAMEND)  ;ініціалізуємо вказівник стека кінцем оперативної пам’яті,
                                це ;число записане в макрос RAMEND
    out SPL, r16
    ldi r16, high(RAMEND)
    out SPH, r16
  ldi r16, (1<<LEDG) | (1<<LEDB) | (1<<LEDR)        ;світлодіодний порт на вихід
  out DDRA, r16
  clr r16                       ;і скинути в нуль
```

```
    out PORTA, r16

    cbi DDRB, BUT                ;пін кнопки на вхід
    sbi PORTB, BUT               ;пін кнопки підтягнути до +
endless_loop:
                                 ;якшо на кнопці одиниця, то вона не нажата
    sbic PINB, BUT                    ;а якщо нуль стрибаємо через наступну інструкцію
    rjmp endless_loop
    sbi PORTA, LEDG          ;запалюємо зелений
    rcall delay250ms         ;ждемо 200мс
    cbi PORTA, LEDG          ;гасимо зелений
    rcall delay250ms         ;ждемо 200мс
    sbi PORTA, LEDB          ;запалюємо синій
    rcall delay250ms         ;ждемо 200мс
    cbi PORTA, LEDB          ;гасимо синій
    rcall delay250ms         ;ждемо 200мс
    sbi PORTA, LEDR          ;запалюємо червоний
    rcall delay250ms         ;ждемо 200мс
    cbi PORTA, LEDR          ;гасимо червоний
    rcall delay250ms         ;ждемо 200мс
   rjmp endless_loop         ;повторяємо все спочатку
```

Схема в Proteus згідно з умовами прикладу 2 наведена на рис.5. 1

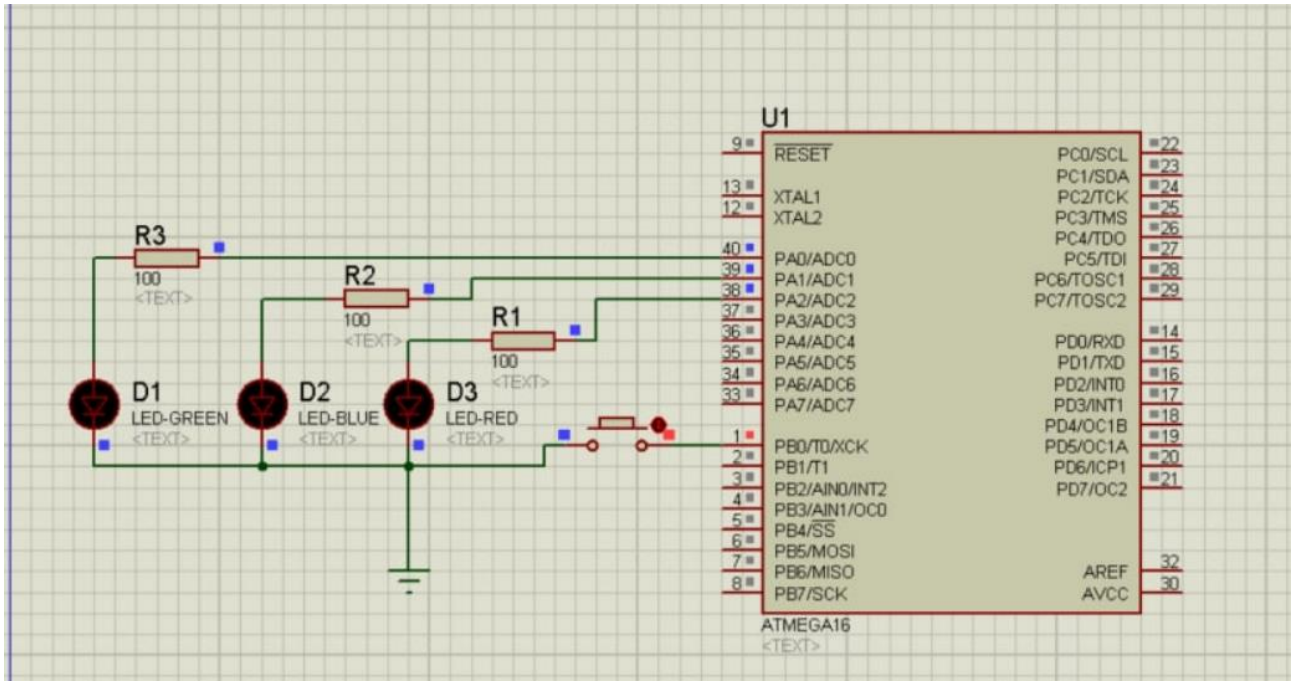


Рис.5.1.

Вікно налаштувань показано на рис. 5.2. (див. дод. Г).

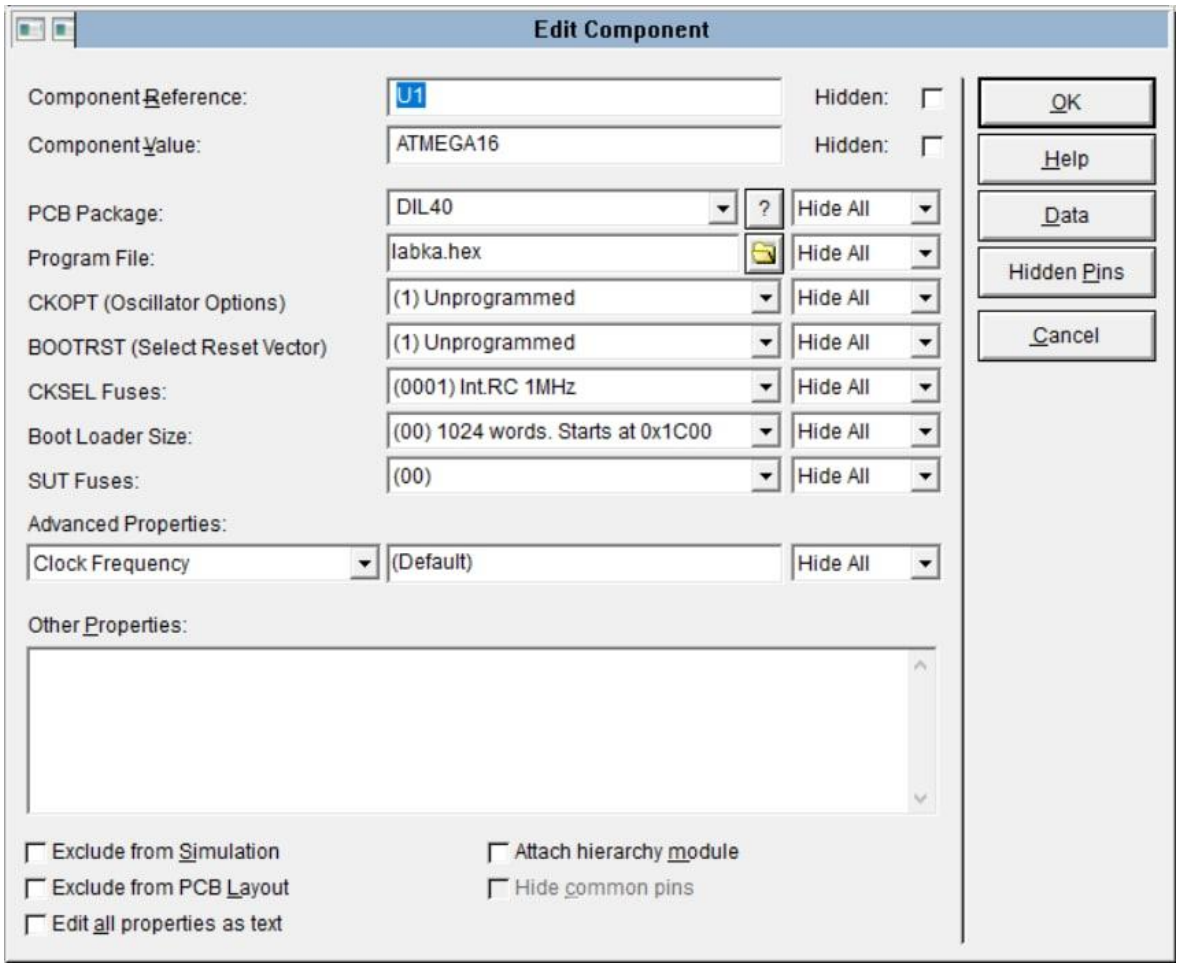


Рис. 5.2.

Рис. 5.3 - 5.5 ілюструють роботу схеми.

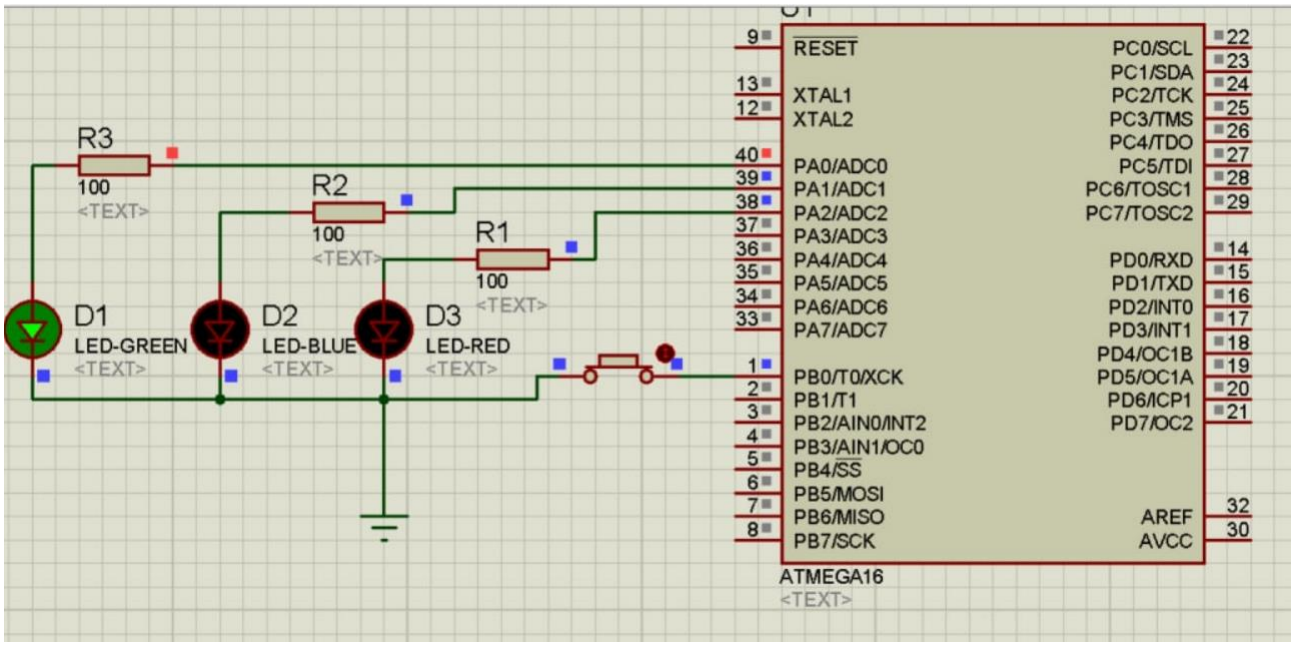


Рис. 5.3

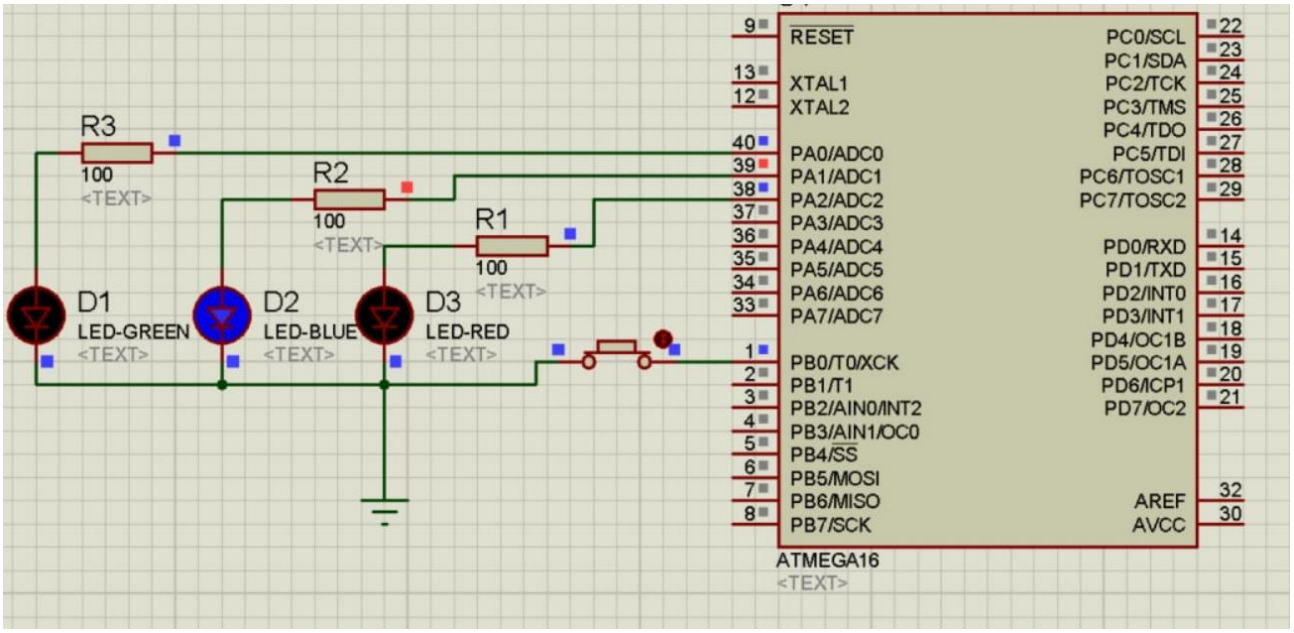


Рис.5.4.

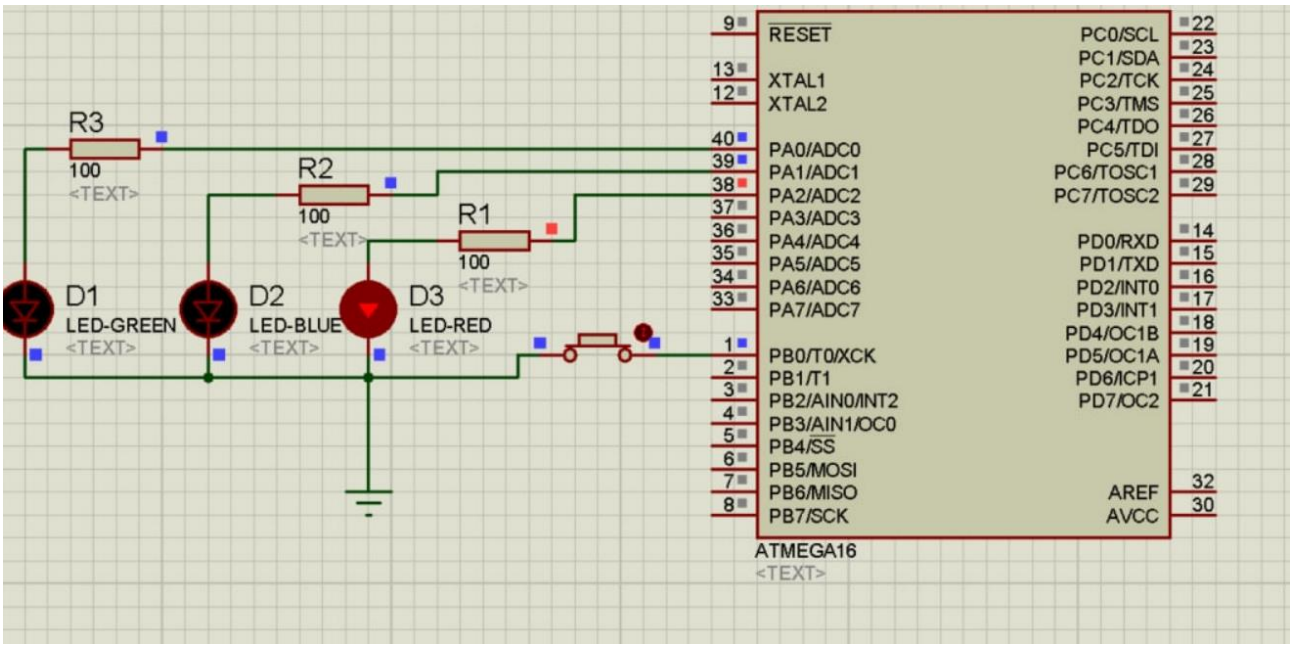


Рис.5.5

Зміст звіту

1. Програма з коментарями
2. Висновки

Контрольні питання

1. Назвіть паралельні порти AVR-мікроконтролерів та режими їх роботи.

2. Наведіть структура однієї лінії порту введення/виведення AVR-мікроконтролерів.

3. Назвіть регістри керування портами введення/виведення AVR-мікроконтролерів та їх призначення.

4. Яким чином здійснюється конфігурування паралельних портів?.

5. Для чого призначенп початкова ініціалізація паралельних портів.

6. Яка роль «підтягувальних» резисторів? Яким чином можна ввімкнути і вимкнути резистори?

7. Як задається альтернативна функція розряду порту?

8. Як розраховується затримка в підпрограмі затримки (вказує викладач)

9. Як підвищити точність програмної затримки?

10. Які мікроконтролери підтримує емулятор електронних пристроїв PROTEUS?

11. Що відображає панель DEVICES в PROTEUS?

12. Де потрібно шукати необхідні елементи, для побудови схеми в PROTEUS?

13. Як задати частоту тактування МК при емуляції в PROTEUS?

14. Як задати необхідний час емуляції в PROTEUS?

ЛАБОРАТОРНА РОБОТА 5.

Дослідження режимів роботи таймерів та системи переривань

Мета роботи: Написання програм формування часових залежностей за допомогою таймерів. Емуляція роботи в Proteus.

Порядок виконання роботи

1. Написати програму на мові асемблеру та коментар до кожної команди відповідно до завдання, наведено в табл. 5.1.

2. Продемонструвати роботу схеми та пояснити результат.

Таблиця 5.1.

№	Завдання
1.	За допомогою переривання від таймера 1 організувати перемикання червоного, жовтого, зеленого світлодіодів як у світлофорі з інтервалом 1с.
2.	перемикати синій і червоний світлодіоди з частотою 4Гц. У це ж час моргати зеленим світлоді із частотою 2Гц у основної програмі.
3.	Мигати синім світлодіодом за допомогою переривання від таймера із частотою 4Гц.
4.	Перемикати жовтий і зелений світлодіоди за допомогою переривання від таймера 0 із частотою 1Гц.
5.	Мигати зеленим світлодіодом за допомогою затримки на таймері 0 0 200 мс, а червоним – із частотою 10Гц за допомогою переривання від таймера 1
6.	Мигати по три рази червоним, потім синім, потім зеленим світлодіодом за допомогою переривання від таймера 0 із частотою 2Гц.
7.	По черзі перемикати світлодіоди за годинниковою стрілкою за допомогою переривання від таймера 1 із частотою 2Гц.
8.	Мигати зеленим світлодіодом за допомогою переривання від таймера 0 із частотою 2 Гц, а . синім світлодіодом за допомогою переривання від таймера 1 із частотою 4 Гц.
9.	Ініціалізувати таймер/лічильник T/C2 на наступний режим роботи: Робота в якості таймера в режимі роботи “CTC”; Константа завантаження -0x0B; Дозволити переривання по співпаданню. Навести діаграму виходного сигналу
10.	Ініціалізувати таймер/лічильник T/C1 (ATmega16) на наступний режим роботи: • Робота в режимі PWM Phase Correct, 12bit; • Коефіцієнт ділення попереднього подільника CLKio/1; □ Константа завантаження регістру порівняння OCR1A-0x1BB; • Забезпечити формування на виводі OC1A інверсного сигналу (Clr OC1A on compare…). Навести діаграму виходного сигналу
11.	Ініціалізувати таймер/лічильник T/C1 на наступний режим роботи: Робота в режимі роботи “Fast PWM 15 біт”; Константа завантаження –регістру порівняння 0xC01;

	Забезпечити формування на виході OC1A прямого сигналу (Set OC1AB on compare…). Навести діаграму виходного сигналу
12.	Ініціалізувати таймер/лічильник T/C2 на наступний режим роботи: Робота в режимі роботи “Fast PWM”; Константа завантаження –регістру порівняння 0xBA; Забезпечити формування на виході OC2 прямого сигналу (Set OC2 on compare…). Навести діаграму виходного сигналу

Зміст звіту

1. Програма з коментарями
2. Висновки до лабораторної роботи.

Теоретичні відомості

Переривання в мікроконтролерах AVR служать для обробки зовнішніх і внутрішніх подій. Структура програми у випадку використання переривань показана в табл. 5.2

Таблиця 5.2

Переривання не задіяні	Переривання задіяні
.CSEG .ORG 0 LDI r16,0xAA ;код програми	.CSEG ;начало сегмента програми .ORG 0 ;адрес $000 кода RJMP reset;обробка reset .ORG 2 ;адрес $002(вектор EXT0INT) RJMP ext0irq; обробка зовнішнього ; переривання 0 ext0irq: … ;команды обробки переривання reti reset: LDI r16, 0xAA ; код програми

Мікроконтролер ATMega 16 може обробляти переривання, що наведені в табл. 5.3.

Під час виклику обробників переривань і процедур, адреса повернення програмного лічильника (звідки далі продовжувати програму) зберігається в стеку. Стек в мікроконтролерах організований програмно, тому всі програми, які використовують переривання і процедури, повинні ініціалізувати стек при початку роботи!

Таблиця 5.3

Адреса вектору	Мітка, яка визначена в M16def.inc (через .EQU)	Опис переривання
$000		Reset
$002	INT0addr	Зовнішнє переривання 0 (вывод 16 = PortD 2)
$004	INT1addr	Зовнішнє переривання 1 (вывод 17 = PortD 3)
$006	OC2addr	Збіг з константою таймера/лічильника 2
$008	OVF2addr	Переповнення таймера/лічильника 2
$00A	ICP1addr	Захоплення таймера/лічильника 1
$00C	OC1Aaddr	Збіг з константою А таймера/лічильника 1
$00E	OC1Baddr	Збіг з константою В таймера/лічильника 1
$010	OVF1addr	Переповнення таймера/лічильника 1
$012	OVF0addr	Переповнення таймера/лічильника 0
$014	SPIaddr	Переривання від SPI
$016	URXCaddr	Переривання від UART: прийом данных закінчено
$018	UDREaddr	Переривання від UART: регістр даних пустий
$01A	UTXCaddr	Переривання від UART: передача даних закінчена
$01C	ADCCaddr	Переривання від ADC: перетворення закінчено
$01E	ERDYaddr	Переривання від EEPROM: готовність
$020	ACIaddr	Переривання від аналогового компаратора
$022	TWIaddr	Переривання від двопровідного послідовного інтерфейсу
$024	INT2addr	Зовнішнє переривання 2 (вивод 3 = PortB 2)
$026	OC0addr	Збіг з константою таймера/лічильника 0
$028	SPMRaddr	Переривання від програмної пам‘яті: завантаження закінчено

Приклад 1. Організувати затримкуа вмикання/вимикання світлодіода на основі таймера Т/С1 (мигання з заданою частотою).

Частота переповнень у цьому режимі визначається за наступною формулою

$$F_{OV} = \frac{f_{CLK} \times N}{0xFFFF + 1 - N_{TCNT1}},$$

де: f_{CLK} - тактова частота генератора мікроконтролера; N- коефіцієнт ділення попереднього подільника (1/1, 1/8, 1/64, 1/256, 1/1024); N_{TCNT1} – константа завантаження лічильного регістру.

Кожне переповнення таймеру стан лінії порту PORTD0 змінюється на протилежний.

```
.include "m16def.inc"
.def temp = r16
.def temp2= r17

.cseg
.org 0x00 ; початковий вектор
 rjmp reset
.org 0x10
rjmp TIM1_OVF ;вектор переривання від переповнення T/C1
reset:ldi temp,low(RAMEND)  ;ініціалізація верхівки стеку
out spl,temp
ldi temp,high(RAMEND)
out sph, temp
ldi temp,0xff ;ініціалізація порту D на вивід
out DDRD,temp
ldi temp,0b00000101 ;  коефіцієнт ділення 1024
out TCCR1B,temp
ldi temp,0b00000100 ; дозволити переривання по перепоненню OVF1 та
встановити його
out TIMSK,temp
out TIFR,temp
ldi temp,0xff ; завнтажити початкове значення в лічильний регістр T/C1
out TCNT1H,temp
ldi temp,0xf0
out TCNT1L,temp

Proga:
sei ; дозвіл переривання
rjmp Proga

TIM1_OVF:
cli
Vix:
ldi temp,0xee ;перезавантажити лічильний регістр T/C1
out TCNT1H,temp
out TCNT1L,temp
sei
reti
ldi temp2,0b00000001 ;перемкнути світлодіод
out PORTD,temp2
rjmp Vix
rjmp reset
```

Приклад 2. Ініціалізувати таймер/лічильник T/C2 на наступний режим роботи:

робота в режимі роботи “Fast PWM”;

константа завантаження –регістру порівняння 0xBA;

забезпечити формування на виході OC2 прямого сигналу (Set OC2 on compare…).

```
.include "m16def.inc"

.cseg
Start:
	;Налаштувати порт PD7 (OC2) на вихід
	ldi r16, (1 << PD7)
	out DDRD, r16
	;Завантажити константу порівняння в регістр OCR2
	ldi r16 , 0xBA
	out OCR2, r16
	;WGM20:1 Waveform Generation Mode (1 << WGM20) | (1 << WGM21) - FastPWM Mode
	;COM20:1 Compare Match Output Mode (1 < COM21) - Clear OC2 on compare match, set OC2 at BOTTOM,
	;(non-inverting mode)
	;CS22:0 Clock Select (1 << CS21) = clkT2S/8 (From prescaler)
	ldi r16, (1<<WGM20) | (1<<COM21)| (1<<WGM21) | (1<<CS21)
	out TCCR2 , r16

Loop:
	rjmp Loop
```

Результат моделювання наведено на рис 5.1 - на виході OC2 формується ШІМ сигнал.

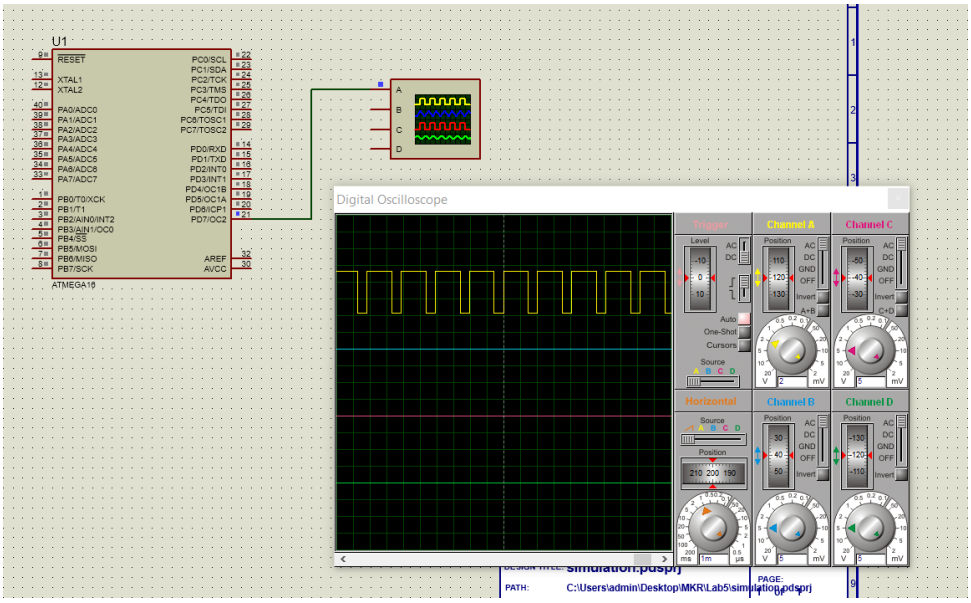


Рис. 5.1

Контрольні питання

1. Яке основне призначення системи переривань мікроконтролера?
2. Які функції можуть виконувати таймери/лічильники мікроконтролерів? Наведіть приклади використання таймерів/лічильників?
3. Що називається перериванням?
4. Які регістри мікроконтролера визначають поточний стан виконання основної програми?
5. Що називається вектором переривання.
6. Які регістри використовують для управління перериваннями в мікроконтролерах AVR?
7. Опишіть структуру регістра GIFR мікроконтролера Atmega 16
8. Опишіть структуру регістра TIMSK мікроконтролера Atmega 16
9. Яке призначення регістра прапорців TIFR?
10. Як змінюється вміст лічильника команд при виникненні переривання?
11. Як виконується команда RETI?
12. Що визначає положення вектора переривання в таблиці векторів переривань?
13. Де може розташовуватися таблиця векторів переривань? Яким чином здійснюється керування розміщенням таблиці векторів переривань?
14. Як здійснити глобальний дозвіл/заборону переривань? Як здійснити дозвіл/заборону окремого переривання?
15. Які регістри використовуються для дозволу/заборони зовнішніх переривань?
16. Скільки таймерів/лічильників в мікроконтролерах AVR Mega16?
17. Які регістри мікроконтролера використовуються для управління таймерами/лічильниками?
18. Опишіть призначення бітів регістра управління TCCR0.

19. Які функції виконує регістр TCNT0 таймера/лічильника T/C0?

20. Які функції регістра OCR0 таймера/лічильника T/C0?

21. Яким чином здійснюється вибір джерела тактового сигналу, а також запуск і зупинка таймерів/лічильників?

22. Як увімкнути режим роботи Normal таймера/лічильника T0? Для чого може використовуватись даний режим роботи?

23. У чому полягає різниця між роботою 8-розрядних та 16-розрядних таймерів/лічильників у режимі Fast PWM?

24. Завдяки чому в режимі Phase and Frequency Correct PWM кожен період сигналу є повністю симетричним?

25. У чому полягає функція сторожевого таймеру?

26. Яким чином задається режим роботи таймера/лічильника T0 (T2)?

27. Чим відрізняється режим роботи CTC від режиму Normal?

28. Для рішення яких завдань краще використання режиму Phase Correct PWM ніж режиму Fast PWM?

29. Які таймери/лічильники можуть працювати в асинхронному режимі?

СПИСОК ЛІТЕРАТУРИ

1. Жуйков В.Я, Терещенко Т.О., Ямненко Ю.С. Заграничний А.В. Електронний підручник "Мікропроцесорна техніка". http://kaf-pe.kpi.ua/?page_id=675, http://ela.kpi.ua/handle/123456789/18969.

2. Терещенко Т. О., Тодоренко В.А., Батрак Л.М., Ямненко Ю. С. Мікропроцесорні пристрої. Навчальний посібник для студентів спеціальності «Електроніка». - К.: НТУУ «КПІ ім. Ігоря Сікорського», 2017. – 244 с.

3. Гребнев В. В. Микроконтроллеры семейства AVR фирмы Atmel – М.: ИП Радио Софт, 2002. – 176 с.

4. Баранов В. Н. Применение микроконтроллеров AVR: схемы, лгоритмы, программы – М.: Издательский дом «ДодэкаXXI», 2004. – 288 с.

5. Белов А.В. Самоучитель разработчика устройств на микроконтроллерах AVR. –М: Наука и техника, 2008. — 544 с.

6. Голубцов М. С., Кириченкова А. В. Микроконтроллеры AVR: от простого к сложному. – [2-е изд. испр. и доп.]. – М.: СОЛОН-Пресс, .2004. – 304 с.

7. Евстифеев А. В. Микроконтроллеры AVR семейства Tiny и Mega фирмы ATMEL – М. : Издательский дом «ДодэкаXXI», 2004. – 560 с.

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

9. Максимов А. Моделирование устройств на микроконтроллерах с помощью программы ISIS из пакета PROTEUS VSM /Радио. — 2005. — № 4, 5, 6. С. 30—33, 31—34, 30—32.

10. МОРТОН Д. Микроконтроллеры AVR. Вводный курс. – М.: Издательский дом «ДодэкаXXI», 2006. – 272 с.

11. Ревич Ю. В. Практическое программирование микроконтроллеров Atmel AVR на языке ассемблера. — 2-е изд., испр. — СПб.: БХВ-Петербург, 2011.— 352 с.

12. Proteus VSM. Пошаговая отладка. [Електронный ресурс]. http://easyelectronics.ru/sistema-modelirovaniya-isis-proteus-bystryjstart.html

13. AVR Ассемблер. Урок 3. Таймер. Мигалка на таймере. AVR Assembler https://www.youtube.com/watch?v=PHDKorunI38

14. DATASheet ATmega16, ATmega16L http://ww1.microchip.com/downloads/en/devicedoc/doc2466.pdf

15. Выбор элементов для схемы - Proteus. Редактор ISIS http://radio-hobby.org/modules/instruction/proteus-redaktor-isis/vybor-elementov-dlya-skhemy.

16. AVR Ассемблер. Урок 2. Порты. Мигалка. AVR Assembler. Lesson 2. Ports. Flasher. https://www.youtube.com/watch?v=SbAX16JI64Y.

17. Работа в Proteus. Часть 1 - Сайт Паяльник https://cxem.net/comp/comp117.php.

18. Цирульник С. М. Застосування програми ISIS пакету Proteus VSM при вивченні курсу «Мікропроцесорна техніка» // Матеріали XIII міжнародної конференції з автоматики управління (Автоматика 2006). — Вінниця: Універсум-Вінниця. — 2007. — С. 526—530.

Додаток А.

ІНТЕГРОВАНЕ СЕРЕДОВИЩЕ РОЗРОБКИ AVR STUDIO 4

AVR Studio 4 - це інтегроване середовище розробки (IDE, Integrated Development Environment), яке дуже зручне для налагодження AVR-додатків в операційних системах Windows 9x / Me / NT / 2000 / XP /. Це середовище пропонує інтерфейс програмного симулятора (імітатора) і внутрисхемного емулятора для восьмирозрядних мікроконтролерів AVR RISC. Крім того, AVR Studio підтримує набір розробника STK500, що дозволяє програмувати AVR-пристрої, а також новий вбудований емулятор JTAG.

Емулятори не є складовою частиною AVR-Studio, тому в методичних вказівках розглядається тільки програмний симулятор.

Після установки AVR Studio за замовчуванням на робочому столі Windows з'явиться відповідний ярлик. У будь-якому випадку, це середовище можна запустити по команді з меню Пуск ►Все программи►Atmel AVR Tools. При першому запуску AVR Studio на екрані з'явиться запрошення (рис. А.1).

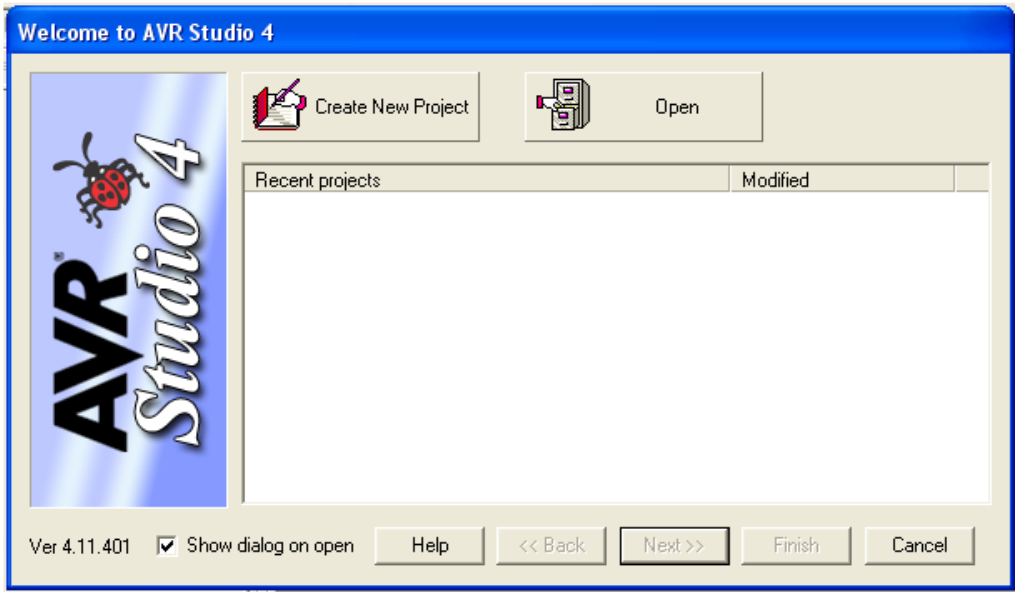


Рис. А.1. Вікно вітання AVR Studio 4

За допомогою цього вікна можна виконати одну з трьох операцій:

- створити новий проект "з нуля" - кнопка **Create New Project**;
- відкрити будь-який файл. cof або. hex з диска - кнопка **Open**;
- відкрити один з проектів, які використовувалися останніми - для цього слід вибрати один з елементів в розташованому нижче списку.

Для прикладу, створимо новий проект. Для цього на екрані запрошення натиснемо кнопку Create New Project. В результаті буде запропоновано вибрати платформу розробки і налагодження програмного забезпечення (рис. 2). В даному випадку нас цікавить інтегрований симулятор **AVR Simulator** і мікроконтролер **Atmega8.**

Тепер можна натиснути **Finish,** щоб перейти до розробки і налагодження програми.

Для того щоб вікно вітання при наступних запусках AVR Studio з'являлося, в ньому слід скинути прапорець **Show this dialog on open**. У такому випадку всі операції по створенню і завантаженні проектів виконуються за допомогою команд меню **File** і **Project**, а вікно вибору емулятора і типу мікроконтролерів, відповідне рис. А.2, відкривається по команді меню **Debug ► Select Platform and Device.**

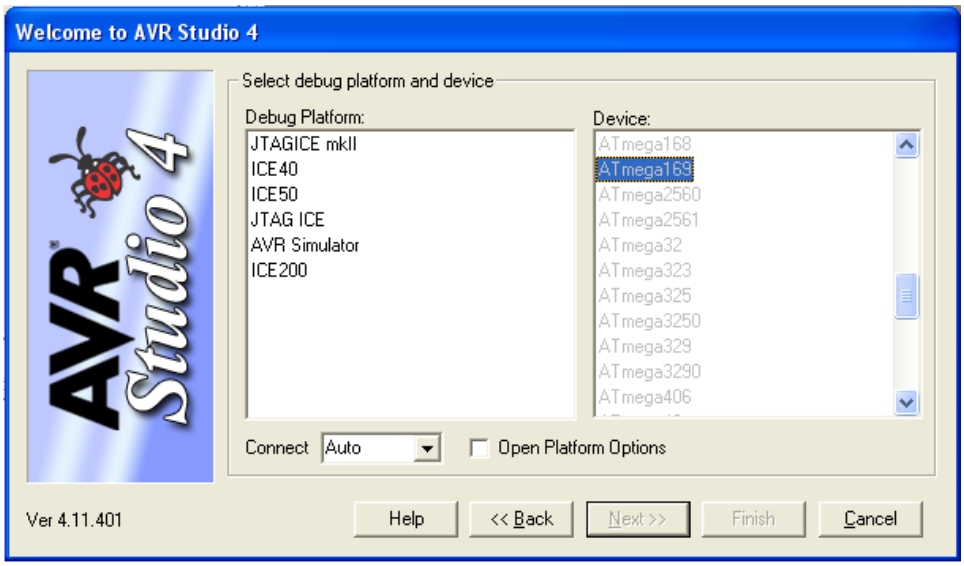


Рис. А.2. Вибір емулятора і типу мікроконтролерів

Головними вікнами AVR Studio є вікно вихідного коду і Workspace (рисунок А.3).

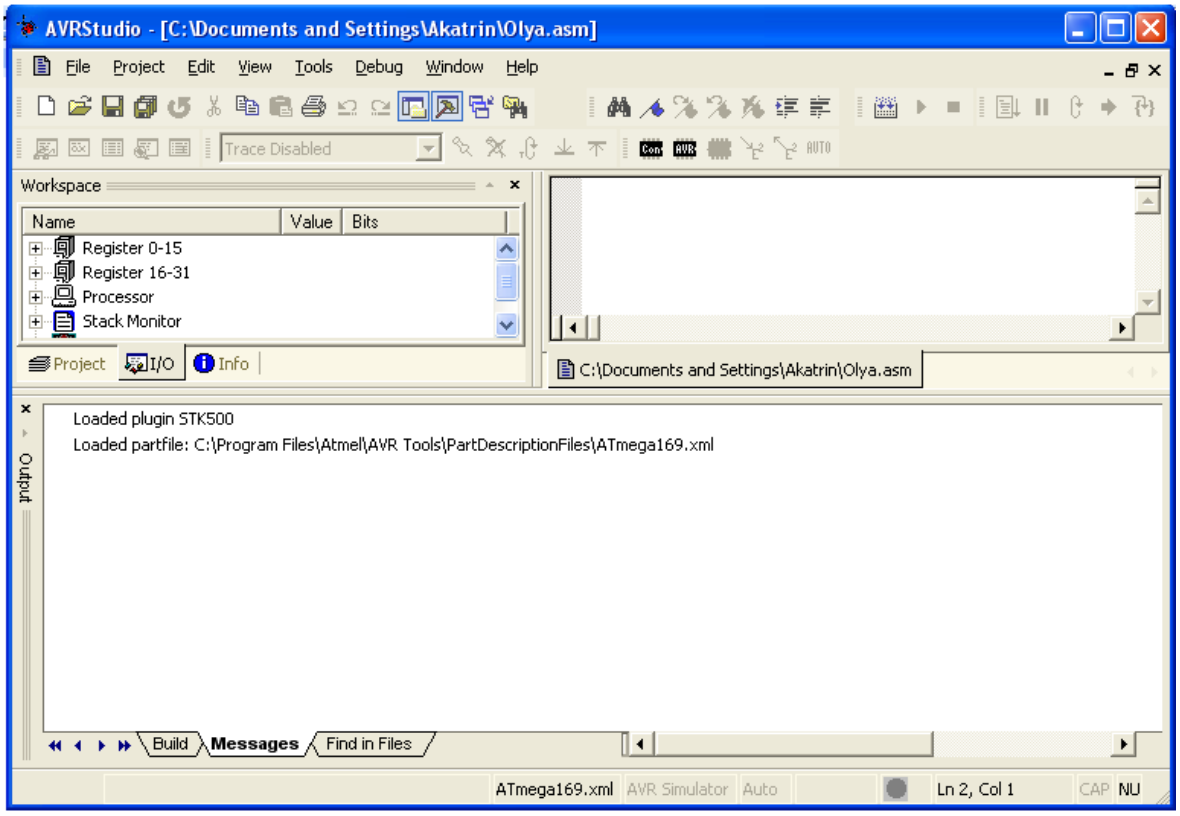


Рис. А.3. Вікно AVR Studio

У вікні початкового коду рядок, що підлягає виконанню першим, позначений жовтою стрілкою. Зміни слід вносити в середовищі **Programmers Notepad**, потім виконувати компіляцію і створення об'єктного файлу, потім - переключитися в **AVR Studio**. Якщо в AVR Studio був відкритий той же файл, то зовнішні зміни будуть розпізнані і на екрані з'явиться пропозиція автоматично оновити вміст вікна вихідного коду. Цю ж операцію можна виконати і вручну за допомогою кнопки панелі інструментів **Reload Object File.**

Вікно Workspace

Вікно Workspace, розташоване на рис. А. 3 зліва від вікна вихідного коду. Надає доступ до 32-м робочим регістрам, до параметрів процесора (лічильник команд, покажчик стека, регістри подвійної довжини X, Y і Z, частота генератора і ін.), до значень стека , а також портів введення / виводу обраного для симуляції або емуляції мікроконтролера. Наприклад, можна відредагувати вміст будь-якого робочого регістра як до, так і в процесі виконання програми. Для цього слід вибрати необхідний елемент у вікні Workspace, двічі клацнути мишею на значенні в поле Value (рис. А.4) і ввести необхідне значення в діалоговому вікні Edit.

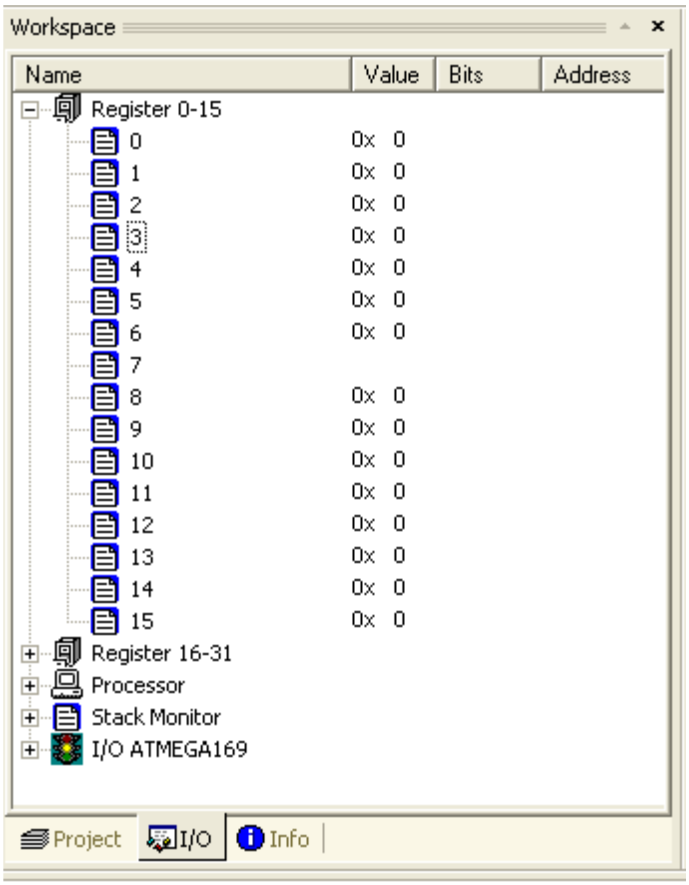


Рис. А.4 Зміна значення в робочому регістрі 3

При цьому значення можна вводити в шістнадцятковому, десятковому, вісімковому або двіййковому форматі, що вибирається за допомогою відповідних перемикачів.

Особливий інтерес на етапі налагодження є галузь параметрів процесора (Рис. А.5). Представлені тут елементи дозволяють переглядати і змінювати вміст програмного лічильника, покажчика стека, лічильника циклів під час виконання програми, регістрів подвійної довжини X, Y і Z, а також частоту кварцового осцилятора і показання таймера відліку часу.

Значення лічильника команд можна змінити за допомогою вікна, представленого раніше на рис. А.2.

При цьому відкриється вікно **Disassembler** з кодом програми на асемблері і поточним буде виділена команда з відповідним адресою (це ж вікно можна відкрити в будь-який момент часу, виконавши команду меню **View ► Disassembler**).

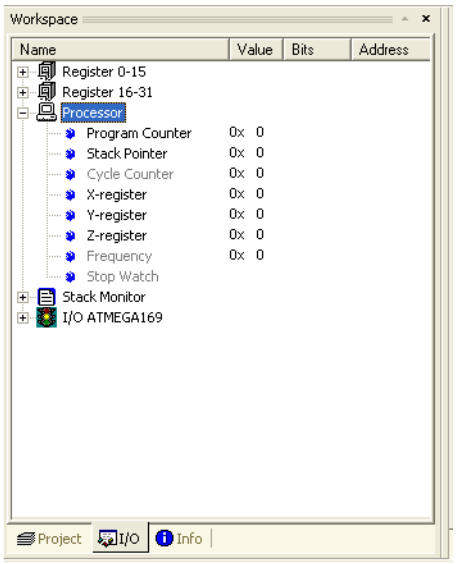


Рис. А.5. Параметри процесора у вікні Workspace

Для скидання в нуль таймера відліку часу слід двічі клацнути мишею на елементі **Stop Watch.** Подібний таймер зручно використовувати для того, щоб з точністю до сотих часток мікросекунд визначати, скільки часу пішло на виконання деякого фрагмента програми.

Найнижчий елемент у вікні **Workspase** дозволяє контролювати роботу пристроїв введення / виводу мікроконтролера, обраного для емуляції (рис. А.6).

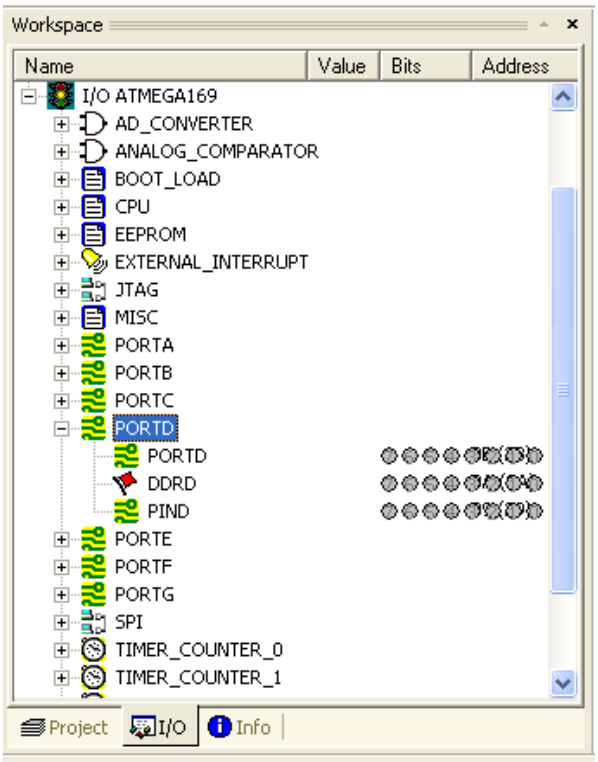


Рис. А.6. Елементи контролю за роботою пристрою введення / виводу мікроконтролера AVR

Наприклад, за допомогою елемента PORTD ми можемо побачити програму в дії. Для цього достатньо запустити її на виконання командою меню **Debug ►Auto Step** (комбінація клавіш <**Alt + F5**>) або відповідною кнопкою панелі інструментів **Debug**.

Команда **Auto Step** відрізняється від команди Run (клавіша <F5>) тим, що в ході виконання програми після кожною кроку оновлюються вікна AVR Studio (в разі команди Run - не оновлюється).

Для того щоб перервати виконання програми, слід виконати команду меню **Debug ► Break** (комбінація клавіш <**Ctrl + F5**>) або натиснути відповідну кнопку панелі інструментів Debug.

Оскільки симулятор виконує програму в віртуальному середовищі, вона працює значно повільніше, ніж в реальному микроконтроллере (особливо, виконання функцій затримки). Враховуйте це обставина при налагодженні програм.

Вікна Memory

Вікно **Memory** дозволяє користувачеві при необхідності контролювати або змінювати деяку область пам'яті мікроконтролера AVR, наприклад, пам'ять програм, пам'ять EEPROM, пам'ять даних, пам'ять введення / виведення. Вікно Memory знаходиться в меню **View** (рис. А.7).

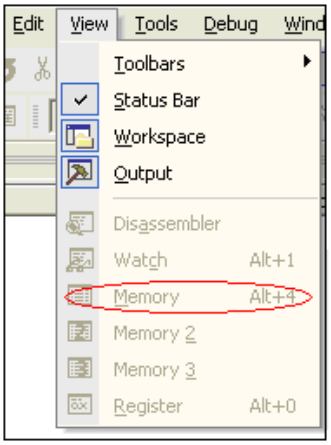


Рис. А.7

Одночасно можуть бути відкриті відразу кілька вікон Memory - для цього використовуються команди меню (рис. А.8). Тип області пам'яті вибирають за допомогою списку, розташованого у верхньому лівому кутку вікна **Memory**. У прикладі на рис. А. 8 для верхнього вікна була обрана пам'ять програм, а для нижнього - пам'ять даних.

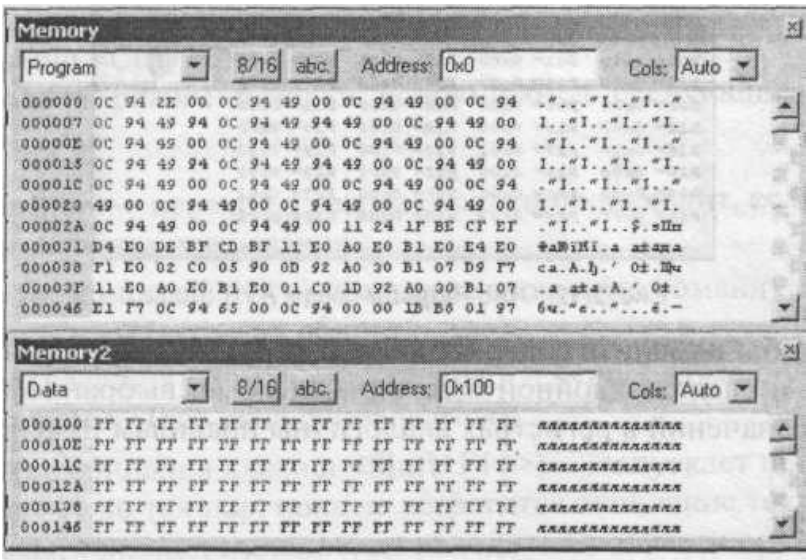


Рис. А.8. Два вікна Memory AVR-Studio

За допомогою кнопки 8/16, розташованої праворуч від списку, а також команд контекстного меню 1 Byte і 2 Byte користувач може перемикатися між режимами розбивки дампа пам'яті на одно- і двухбайтні фрагменти, а за допомогою кнопки abc - відобразити / приховати додаткову колонку з ASCII-значеннями відображеної області пам'яті. Поле Address служить для відображення і введення адреси комірки пам'яті, на якій в даний момент встановлено курсор. Кількість колонок зі значеннями елементів пам'яті вибирається за допомогою списку **Cols** (якщо в цьому списку вибраний елемент **Auto,** то кількість колонок вибирається автоматично залежно від ширини вікна Memory).

Вміст комірки пам'яті у вікні Memory можна легко змінити. Введені символи розцінюються як шістнадцяткові числа, при цьому нове значення потрапляє в комірку пам'яті відразу ж після кожного натискання клавіші. Якщо це небажано (наприклад, в разі застосування емулятора при доступі на запис до регістру UDR приймача UART ініціюється нова передача, хоча в цьому випадку байт ще неповний), то нове значення можна альтернативно вводити в спеціальному діалоговому вікні, яке відкривається за подвійним клацанням миші на відповідній клітинці пам'яті. У цьому випадку вводиться значення записується в комірку тільки тоді, коли у вікні **Edit** буде натиснута кнопка ОК (клавіша <Enter>). Значення, які змінилися з моменту останньої операції налагодження, відображаються червоним шрифтом. Адреси восьмирозрядних комірок пам'яті і ASCII-символи виділені сірим фоном, а адреси 16-розрядних елементів пам'яті - блакитним фоном.

Вікно Register

Вікно відкривається за відповідною командою меню View (або при натисканні комбінації клавіш **<Alt + 0>**).

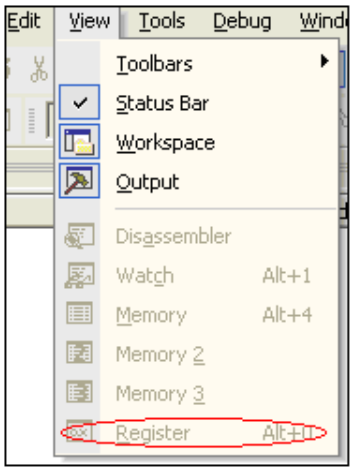


Рис. А.9

Вікно **Register** показує вміст 32-х робочих регістрів мікроконтролера, яке оновлюється після виконання кожної команди (рис. А.10).

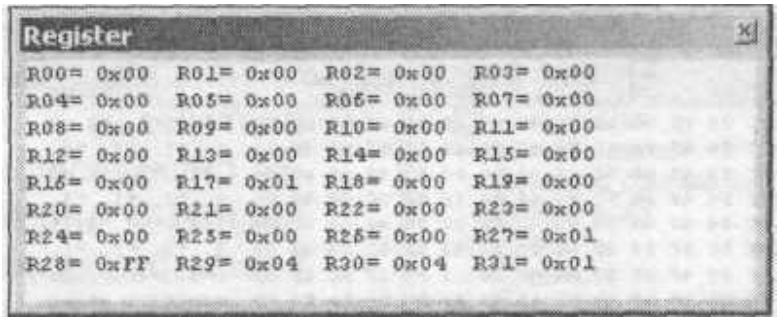


Рис. А.10. Вікно Register AVR Studio

Значення, що змінилися в результаті останньої операції відлагодження (наприклад, при покроковому проході або в результаті досягнення точки зупину) відображаються червоним шрифтом.

Для того щоб змінити вміст регістра, слід зупинити виконання програми і зробити подвійне клацання мишею на обраному регістрі. Правила змін значень в регістрах аналогічні правилам зміни значень в вікнах Memory.

Вікно Watch

Вікно відкривається за відповідною командою меню **View** (або при натисканні комбінації клавіш **<Alt + I>),** рис. А.11.

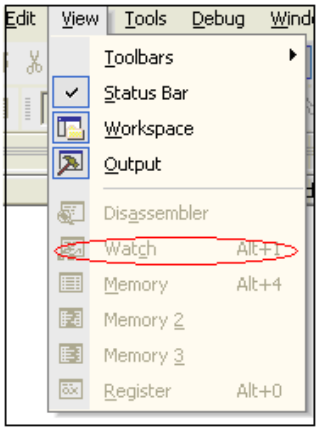


Рис. А.11

Вікно Watch (рис. А.12) служить для відображення типів, значень і адрес таких об'єктів як, наприклад, змінні в програмі на С або на Асемблері. Це вікно складається з чотирьох колонок. У першій зазначено ім'я об'єкта, за яким ведеться спостереження, в другій - тип об'єкта (Integer, unsigned Char і т.д.), у третій - поточне значення, а в четвертій - адреси об'єктів. Користувач може додавати нові об'єкти в вікно Watch по команді контекстного меню **Add Item** або ж по команді контекстного меню **Add to Watch** вікна вихідного коду. В останньому випадку в список Watch буде додано елемент коду, на якому встановлений курсор.

Елементи вікна Watch можна видаляти по одному за допомогою команди контекстного меню **Remove Selected Item** або все відразу по команді контекстного меню **Remove All Items.**

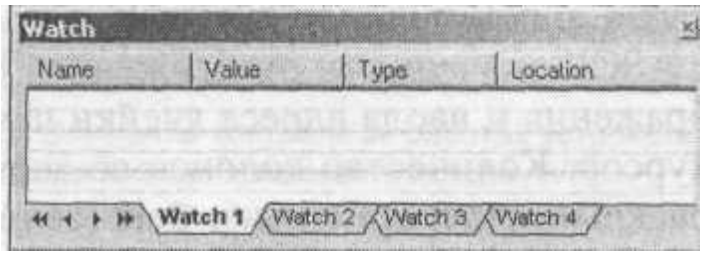


Рис. А.12. Вікно Watch середовища AVR Studio

Налагодження програми

Під налагодженням мають на увазі послідовне виконання програми з контролем вмісту регістрів мікроконтролера (перевірка на низькому рівні) і змінних (перевірка на програмному рівні). Для налагодження програм в AVR Studio використовують команди меню **Debug** і кнопки однойменної панелі інструментів.

Перш, ніж розглянути ці команди має сенс пояснити таке поняття як "точка переривання". Точка переривання (**breakpoin**t) - це рядок вихідного коду, на якій робота

програми припиняється. Таких точок (позначаються коричневим колом зліва від рядка) може бути встановлено стільки ж, скільки ефективних рядків в програмі. Для установки / видалення точки переривання в поточному рядку служить команда меню De**bug ► Toggle Breakpoint** (клавіша <F9>) або відповідна кнопка панелі інструментів Debug. Для видалення всіх розставлених в програмі точок переривання служить команда меню **Debug ► Remove Breakpoints** або кнопка **Clear all breakpoints** панелі інструментів Debug. Для послідовного переходу від однієї точки переривання до іншої використовується команда меню **Debug ► Next Breakpoint** або комбінація клавіш **<Ctrl + F9>.**

Для переходу в режим налагодження використовуються наступні команди меню Debug (рис. А.13).

Рис. А.13. Програми для переходу в режим налагодження

1. Run, Auto Step - перехід в режим налагодження відбувається, якщо зустрічається точка **переривання;**

2. Step Into (клавіша <**F 11**>) - виконує поточну команду з заходом в підпрограми (всі вікна оновлюються); I

3. **Step Over** (клавіша <**F10**>) - виконує поточну команду без заходу в підпрограми (всі вікна оновлюються);

4. **Step Out** (комбінація клавіш <**Shift + F11>)** - запускає програму і виконує її до тих пір, поки не зустрінеться закінчення поточної підпрограми; якщо хід виконання перебуває в області основної програми, то програма буде виконуватися до тих пір, поки не буде зупинена користувачем командою Break або не зустріне точку переривання;

5. **Run To Cursor** (комбінація клавіш <**Ctrl + F10**>) - запускає програму, яка виконується до тих пір, поки не буде досягнута позиція курсору у вікні вихідного коду; якщо зустрічається точка зупинки, то виконання програми не зупиняється; якщо позиція курсора не досягається ніколи, то програма виконується до тих пір, поки не буде зупинена командою Break. Після виконання команди всі вікна оновлюються.

7. Налаштування параметрів імітатора

Для того щоб вибрати імітацію роботи конкретного мікроконтролера, а також його робочу частоту, служить діалогове вікно **Simulator Options** (рис. А.14), яке відкривається по команді меню **Debug ► AVR Simulator Options (**комбінація клавіш <Alt + O>).

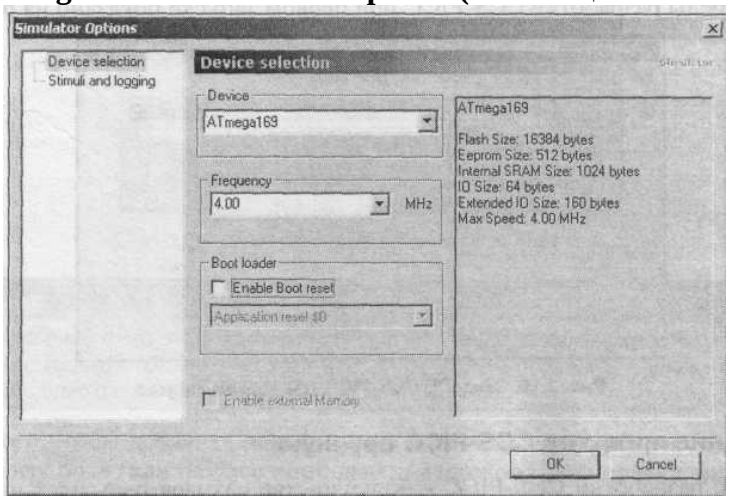


Рис. А.14. Діалогове вікно Simulator Options.

Допустимі параметри обраного мікроконтролера відображаються в текстовому полі, розташованому праворуч.

Послідовність дій для створення проекту

Проект повинен включати основний файл, який містить основну програму. У AVR Studio підпрограми зазвичай записують в окремі файли, які є вкладеними в основний файл. Ці додаткові файли підключаються до основної програми за допомогою директиви "include". Для розробки проекту можна використовувати наступну послідовність операцій.

Виберіть у верхній стрічці меню розділ Project і, відповідно, опцію New Project (створення нового проекту).

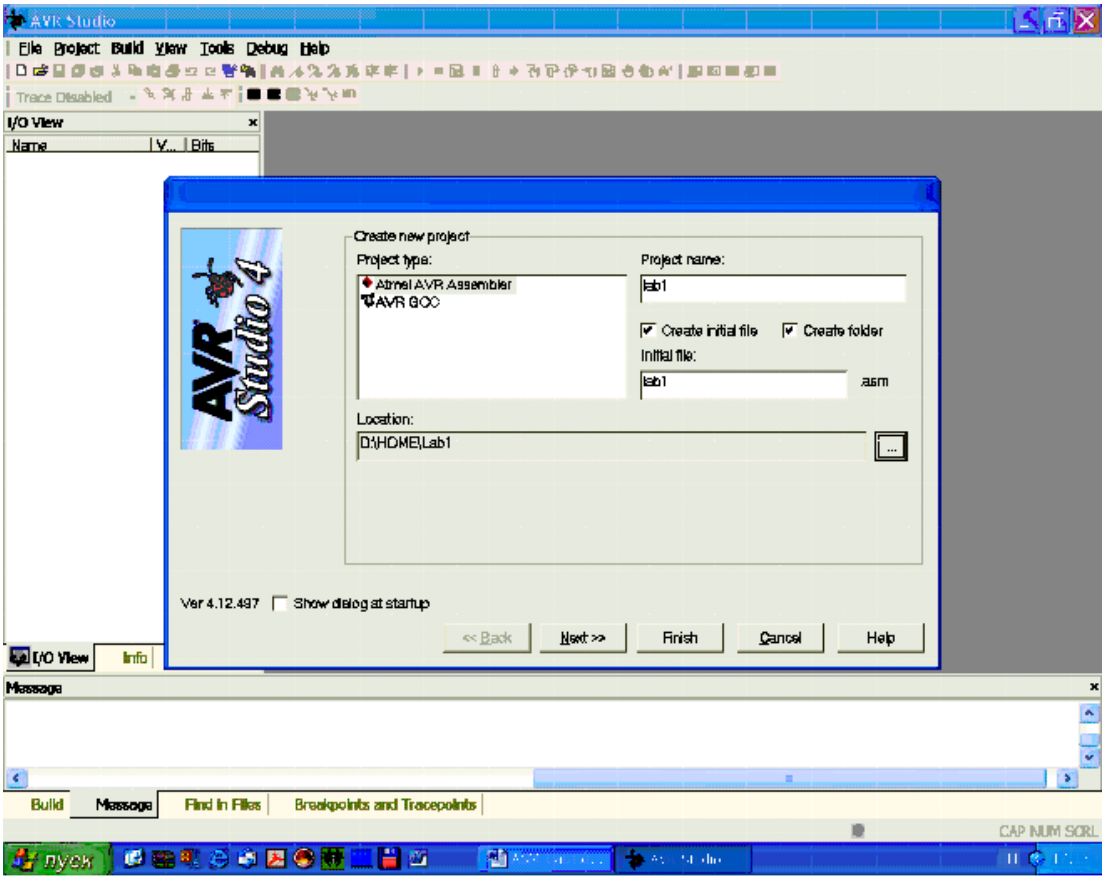


Рис. А.15

На екрані монітора з'явиться вікно **Create new project,** що використовується для встановлення атрибутів проекту. У вікні **Project type** позначте мову програмування - **Atmel AVR Assembler**. У вікні **Project name** визначте ім'я Вашого проекту (Project name). Введіть позначки в вікна створення початкового файлу (**Create initial file**) і створення папки проекту (Create folder). Визначте розміщення (**Location**) папки проекту.

Після натискання кнопки Next з'являється наступне вікно, в якому можливе встановлення платформи відпрацювання програми проекту і типу мікроконтролера. Оберіть в якості платформи AVR Simulator і підключіть мікроконтролер, який використаний в лабораторному макеті, - Atmega16. Закінчується на цьому етапі встановлення атрибутів проекту після натискання кнопки Finish.

Встановлений початковий файл автоматично підключається до проекту як файл асемблера

Для подальшого встановлення атрибутів проекту необхідно перейти в розділі меню Project до опції **Asembler options** (настройка функцій компілятора) (рис. А.16). Визначте

тип **hex**-формату вихідного файлу, який використовується під час програмування мікроконтролера як Intel hex. Встановіть ім'я вихідного об'єктного файлу (наприклад, Project name). Задайте версію компілятора (бажано більш сучасну - Version 2). Можливе підключення до проекту вкладених файлів що знаходяться в Ваших власних бібліотеках. В цьому випадку в розділі Additional Include path необхідно вказати альтернативні шляхи розміщення цих файлів. Зазвичай все вкладені файли проекту бажано розміщувати в папці проекту. Це забезпечує вільне переміщення проекту на інший комп'ютер.

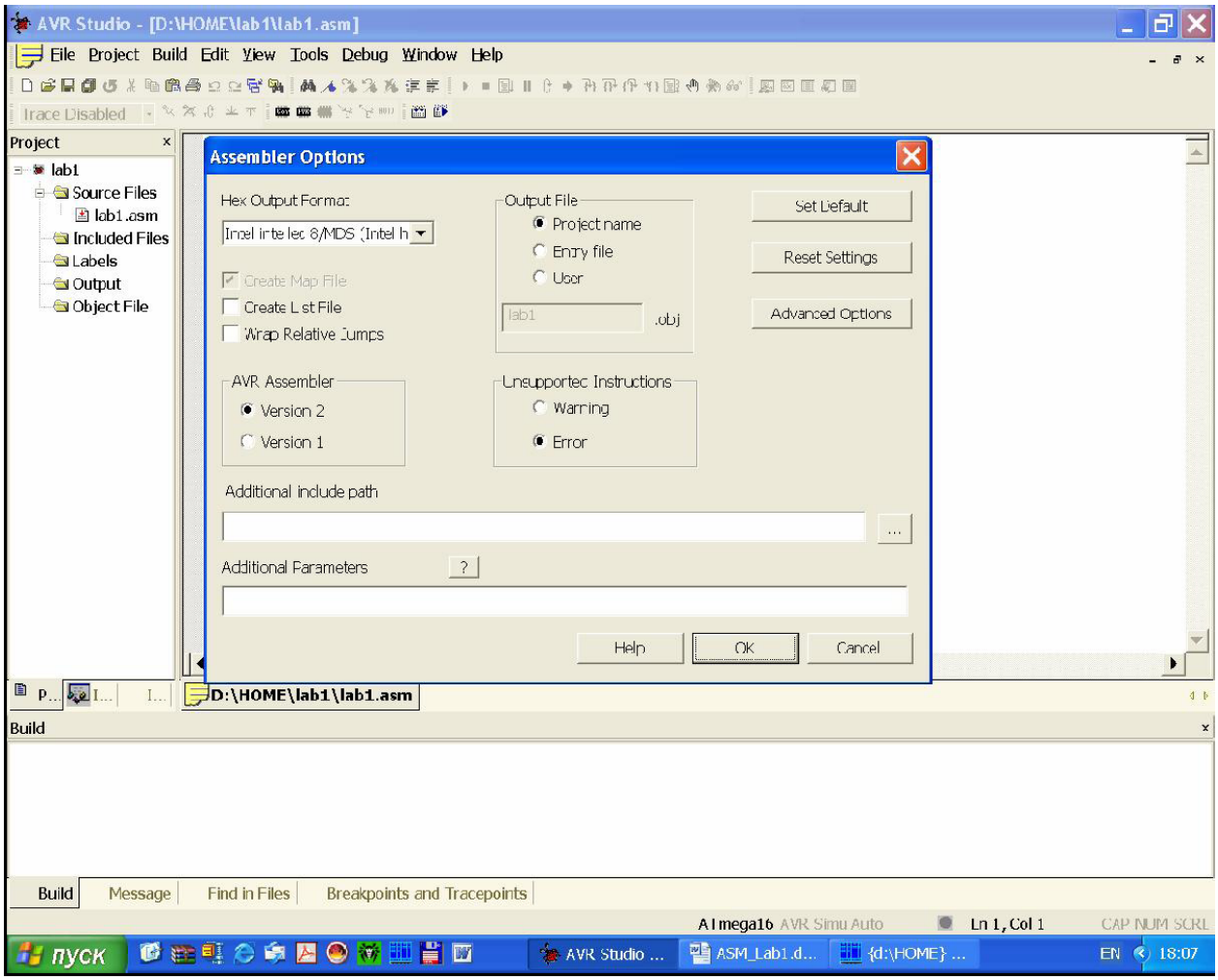


Рис. А.16

Для створення вкладених файлів використовують опцію **New File** розділу меню **File.** Створений файл записують в папку проекту (**Save As**) обов'язково, вказуючи поруч з ім'ям розширення, наприклад **init.asm.**

Для компіляції проекту в розділі меню **Build** викликається опція **Build**. На початку компіляції все відкриті модифіковані файли проекту автоматично записуються. Після вдалої компіляції з використанням компілятора другої версії (**Version 2**), у відповідних розділах вікна проекту **Project** з'являються підключення вкладені файли (**Included files**), використані мітки (**Label**s), вихідні файли для програматора з розширеннями .hex, .eep (Output) але об'єктний файл, який використовується в дебагері (Object file). У нижній частині монітора, в разі відсутності помилок, з'являється повідомлення про використання різних сегментів пам'яті мікроконтролера (рис. А.17).

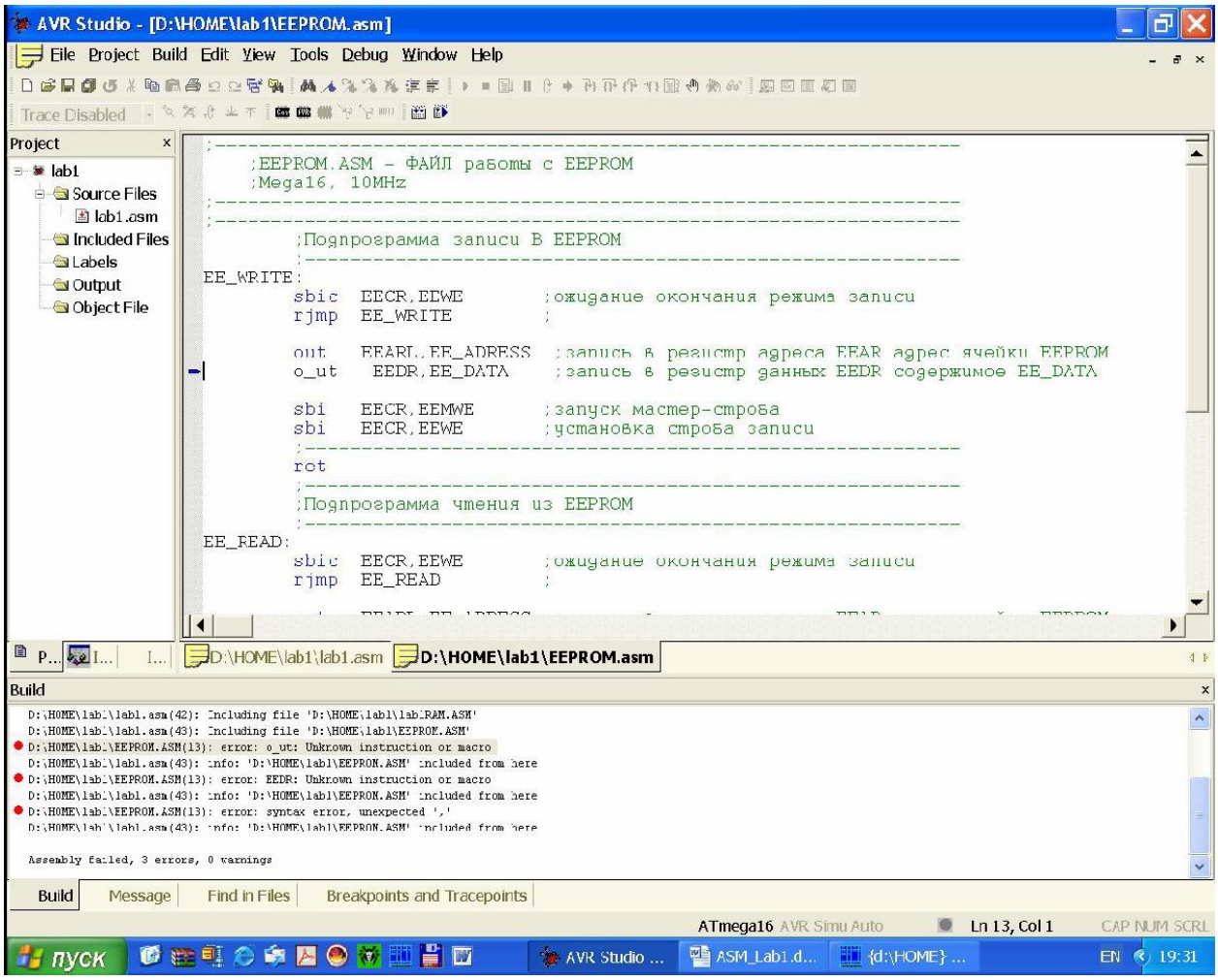


Рис. А.17

Після невдалої компіляції програми у вікні по\відомлень записуються всі виявлення помилок. Помилки відображаються червоною крапкою і вказується їх тип. Якщо рядок, в якій записана знайдена помилка, активізувати за допомогою подвійного натискання лівої кнопки миші, то вікно файлу з вказаною помилкою стане активним і курсор переміститься на початок рядка, яка містить помилку. Якщо ж файл з помилкою не відкритий, то він автоматично відкриється.

Після закінчення роботи проект записують і закривають. Для цього в розділі меню Project послідовно викликають опції Save Project і Close Project.

Додаток Б.

СИСТЕМА КОМАНД AVR

Мікроконтролер AVR має своєму складі 32 регістри. Перша їх половина (R0-R15) не може бути використана в операціях з безпосереднім операндом. У другій половині є специфічні регістрові пари (X, Y і Z) які оперують з 16 даними/адресами.

SREG: Регістр статуса
C: Прапорець перенесення
Z: Прапорець нульового значення
N: Прапорець від'ємного значення
V: Прапорець-покажчик переповнення додаткового коду
S: Для перевірки знаку
H: Прапорець напівперенесення
T: Прапорець пересилання, який використовується командами BLD та BST
I: Прапорець дозволу/заборони глобального переривання

Регістри операнди
Rd: Регістр призначення (і джерело) в Регістровому файлі
Rr: Регістр источник в регістровом файле
R: Результат виконання команди
K: Літерал або байт даних
k: Дані адреса константи для лічильника програм
b: Біт в регістровому файлі або I/O регістр
s: Біт в регістрі статуса
X, Y, Z: Регістр непрямої адресації (X=R27:R26, Y=R29:R28, Z=R31:R30)
P: Адреса введення/виведення порта
q: Зсув за прямою адресацією (6 біт)
SP: Покажчик стека.

https://dfe.karelia.ru/koi/posob/avrlab/avrasm-rus.htm#CSEG%20-%20Code%20segment

АРИФМЕТИЧНІ ТА ЛОГІЧНІ ІНСТРУКЦІЇ

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
ADD	**Rd,Rr**	Додати без перенесення	Rd = Rd + Rr	Z,C,N,V,H,S=	1
ADC	**Rd,Rr**	Додати з перенесенням	Rd = Rd + Rr + C	Z,C,N,V,H,S	1
SUB	**Rd,Rr**	Відняти без прапорця перенесення	Rd = Rd - Rr	Z,C,N,V,H,S	1
SUBI	**Rd,K8**	Відняти константу	Rd = Rd - K8	Z,C,N,V,H,S	1
SBC	**Rd,Rr**	Відняти з перенесенням	Rd = Rd - Rr - C	Z,C,N,V,H,S	1
SBCI	**Rd,K8**	Відняти константу з перенесенням	Rd = Rd - K8 - C	Z,C,N,V,H,S	1
AND	**Rd,Rr**	Виконати логічне І	Rd = Rd & Rr	Z,N,V,S=	1
ANDI	**Rd,K8**	Виконати логічне І з константою К	Rd = Rd & K8	Z,N,V,S	1
OR	**Rd,Rr**	Виконати логічне АБО	Rd = Rd V Rr	Z,N,V,S	1

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
ORI	**Rd,K8**	Виконати логічне АБО з константою К	Rd = Rd V K8	Z,N,V,S	1
EOR	**Rd,Rr**	Виконати виключальне АБО	Rd = Rd EOR Rr	Z,N,V,S	1
COM	**Rd**	Побітна інверсія (перевод в зворотний код)	Rd = $FF - Rd	Z,C,N,V,S	1
NEG	**Rd**	Зміна знаку (додатковий код)	Rd = $00 - Rd	Z,C,N,V,H,S	1
SBR	**Rd,K8**	Встановити біти в регістрі	Rd = Rd V K8	Z,C,N,V,S	1
CBR	**Rd,K8**	Скинути біти у регістрі	Rd = Rd & ($FF - K8)	Z,C,N,V,S	1
INC	**Rd**	Інкрементувати	Rd = Rd + 1	Z,N,V,S	1
DEC	**Rd**	Декрементувати	Rd = Rd -1	Z,N,V,S	1
TST	**Rd**	Перевірити на нуль або від’ємність	Rd = Rd & Rd	Z,C,N,V,S	1
CLR	**Rd**	Скинути регістр	Rd = 0	Z,C,N,V,S	1
SER	**Rd**	Встановити всі біти регістра	Rd = $FF	None	1
ADIW	**Rdl,K6**	Додати константу і слово	Rdh:Rdl = Rdh:Rdl + K6	Z,C,N,V,S	2
SBIW	**Rdl,K6**	Відняти константу із слова	Rdh:Rdl = Rdh:Rdl - K 6	Z,C,N,V,S	2
MUL	**Rd,Rr**	Множення чисел без знака	R1:R0 = Rd * Rr	Z,C	2
MULS	**Rd,Rr**	Множення чисел з знаком	R1:R0 = Rd * Rr	Z,C	2
MULSU	**Rd,Rr**	Множення числа із знаком на число без знака	R1:R0 = Rd * Rr	Z,C	2
FMUL	**Rd,Rr**	Множення дробових чисел без знака	R1:R0 = (Rd * Rr) << 1	Z,C	2
FMULS	**Rd,Rr**	Множення дробових чисел із знаком	R1:R0 = (Rd *Rr) << 1	Z,C	2
FMULSU	**Rd,Rr**	Множення дробового числа із знаком з числом без знака	R1:R0 = (Rd * Rr) << 1	Z,C	

ІНСТРУКЦІЇ РОЗГАЛУЖЕННЯ

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
RJMP	**k**	Перейти за відносною адресацією	PC = PC + k +1	None	2
IJMP	Немає	Перейти за непрямою адресацією **(Z)**	PC = Z	None	2

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
EIJMP	Немає	Розширений відносний перехід на (Z)	STACK = PC+1, PC(15:0) = Z, PC(21:16) = EIND	None	2
JMP	k	Перехід	PC = k	None	3
RCALL	k	Відносний виклик підпрограми	STACK = PC+1, PC = PC + k + 1	None	3/4*
ICALL	Немає	Викликати підпрограму за непрямою адресацією	STACK = PC+1, PC = Z	None	3/4*
EICALL	Немає	Розширений непрямий виклик (Z) (	STACK = PC+1, PC(15:0) = Z, PC(21:16) =EIND	None	4*
CALL	k	Виклик підпрограми	STACK = PC+2, PC = k	None	4/5*
RET	Немає	Повернутись з підпрограми	PC = STACK	None	4/5*
RETI	Немає	Повернутись з підпрограми переривання	PC = STACK	I	4/5*
CPSE	Rd,Rr	Порівняти та пропустити команду якщо рівні	if (Rd ==Rr) PC = PC 2 or 3	None	1/2/3
CP	Rd,Rr	Порівняти	Rd -Rr	Z,C,N,V, H,S	1
CPC	Rd,Rr	Порівняти з урахуванням перенесення	Rd - Rr - C	Z,C,N,V, H,S	1
CPI	Rd,K8	Порівняти з константою	Rd - K	Z,C,N,V, H,S	1
SBRC	Rr,b	Пропустити команду, якщо біт у регістрі скинутий	if(Rr(b)==0) PC = PC + 2 or 3	None	1/2/3
SBRS	Rr,b	Пропустити команду, якщо біт у регістрі якщо біт в регістрі встановлений	if(Rr(b)==1) PC = PC + 2 or 3	None	1/2/3
SBIC	P,b	Пропустити команду, якщо біт в порту скинутий	if(I/O(P,b)==0) PC = PC + 2 or 3	None	1/2/3
SBIS	P,b	Пропустити команду, якщо біт в порту встановлений	if(I/O(P,b)==1) PC = PC + 2 or 3	None	1/2/3

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
BRBC	s,k	Перейти, якщо біт у регістрі статуса SREG очищений	if(SREG(s)==0) PC = PC + k + 1	None	1/2
BRBS	s,k	Перейти, якщо біт у регістрі статуса SREG встановлено	if(SREG(s)==1) PC = PC + k + 1	None	1/2
BREQ	k	Перейти, якщо дорівнює	if(Z==1) PC = PC + k + 1	None	1/2
BRNE	k	Перейти, якщо не дорівнює	if(Z==0) PC = PC + k + 1	None	1/2
BRCS	k	Перейти, якщо прапорець перенесення встановлений	if(C==1) PC = PC + k + 1	None	1/2
BRCC	k	Перейти, якщо прапорець перенесення скинутий	if(C==0) PC = PC + k + 1	None	1/2
BRSH	k	Перейти, якщо дорівнює або більше (без знаку)	if(C==0) PC = PC + k + 1	None	1/2
BRLO	k	Перейти, якщо менше (без знаку)	if(C==1) PC = PC + k + 1	None	1/2
BRMI	k	Перейти, якщо від’ємний результат	if(N==1) PC = PC + k + 1	None	1/2
BRPL	k	Перейти, якщо додатний результат	if(N==0) PC = PC + k + 1	None	1/2
BRGE	k	Перейти, якщо більше або дорівнює (з урахуванням знаку	if(S==0) PC = PC + k + 1	None	1/2
BRLT	k	Перейти, якщо менше (зі знаком)	if(S==1) PC = PC + k + 1	None	1/2
BRHS	k	Перейти, якщо прапорець напівперенесення встановлений	if(H==1) PC = PC + k + 1	None	1/2
BRHC	k	Перейти, якщо прапорець напівперенесення очищений	if(H==0) PC = PC + k + 1	None	1/2
BRTS	k	Перейти, якщо прапорець T встановлений	if(T==1) PC = PC + k + 1	None	1/2
BRTC	k	Перейти, якщо прапорець T скинутий	if(T==0) PC = PC + k + 1	None	1/2

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
BRVS	k	Перейти, якщо переповнення встановлено	if(V==1) PC = PC + k + 1	None	1/2
BRVC	k	Перейти, якщо переповнення скинуто	if(V==0) PC = PC + k + 1	None	1/2
BRIE	k	Перейти, якщо глобальне переривання дозволено	if(I==1) PC = PC + k + 1	None	1/2
BRID	k	Перейти, якщо глобальне переривання заборонено	if(I==0) PC = PC + k + 1	N	

ІНСТРУКЦІЇ ПЕРЕДАЧІ ДАНИХ

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
MOV	Rd,Rr	Копіювати регістр	Rd = Rr	None	1
MOVW	Rd,Rr	Копіювати пару регістрів	Rd+1:Rd = Rr+1:Rr, r,d even	None	1
LDI	Rd,K8	Завантажити константу	Rd = K	None	1
LDS	Rd,k	Завантажити безпосередньо з ОЗП	Rd = (k)	None	2*
LD	Rd,X	Завантажити за непрямою адресацією	Rd = (X)	None	2*
LD	Rd,X+	Завантажити за непрямою адресацією та наступним інкрементом	Rd = (X), X=X+1	None	2*
LD	Rd,-X	Завантажити за непрямою адресацією з попереднім декрементом	X=X-1, Rd = (X)	None	2*
LD	Rd,Y	Завантажити за непрямою адресацією	Rd = (Y)	None	2*
LD	Rd,Y+	Завантажити за непрямою адресацією та наступним інкрементом	Rd = (Y), Y=Y+1	None	2*
LD	Rd,-Y	Завантажити за непрямою адресацією з попереднім декрементом	Y=Y-1, Rd = (Y)	None	2*
LDD	Rd,Y+q	Завантажити за непрямою адресацією із заміщенням	Rd = (Y+q)	None	2*
LD	Rd,Z	Завантажити за непрямою адресацією	Rd = (Z)	None	2*
LD	Rd,Z+	Завантажити за непрямою адресацією та наступним інкрементом	Rd = (Z), Z=Z+1	None	2*

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
LD	Rd,-Z	Завантажити за непрямою адресацією з попереднім декрементом	Z=Z-1, Rd = (Z)	None	2*
LDD	Rd,Z+q	Завантажити за непрямою адресацією із заміщенням	Rd = (Z+q)	None	2*
STS	k,Rr	Завантажити безпосередньо в ОЗП	(k) = Rr	None	2*
ST	X,Rr	Завантажити за непрямою адресацією	(X) = Rr	None	2*
ST	X+,Rr	Завантажити за непрямою адресацією з пост-інкрементом	(X) = Rr, X=X+1	None	2*
ST	-X,Rr	Завантажити за непрямою адресацією з перед-декрементом	X=X-1, (X)=Rr	None	2*
ST	Y,Rr	Завантажити за непрямою адресацією	(Y) = Rr	None	2*
ST	Y+,Rr	Завантажити за непрямою адресацією з пост-інкрементом	(Y) = Rr, Y=Y+1	None	2
ST	-Y,Rr	Завантажити за непрямою адресацією з перед-декрементом	Y=Y-1, (Y) = Rr	None	2
ST	Y+q,Rr	Завантажити за непрямою адресацією з заміщенням	(Y+q) = Rr	None	2
ST	Z,Rr	Завантажити за непрямою адресацією	(Z) = Rr	None	2
ST	Z+,Rr	Завантажити за непрямою адресацією з пост-інкрементом	(Z) = Rr, Z=Z+1	None	2
ST	-Z,Rr	Завантажити за непрямою адресацією з перед-декрементом	Z=Z-1, (Z) = Rr	None	2
ST	Z+q,Rr	Завантажити за непрямою адресацією з заміщенням	(Z+q) = Rr	None	2
LPM	Немає	Завантажити байт з пам’яті програм	R0 = (**Z**)	None	3
LPM	Rd,Z	Завантажити байт з пам’яті програм	Rd = (**Z**)	None	3
LPM	Rd,Z+	Завантажити байт з пам’яті програм з пост-інкрементом	Rd = (**Z**), Z=Z+1	None	3
ELPM	Немає	Розширенне завантаження з пам’яті програм	R0 = (RAMPZ:**Z**)	None	3

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
ELPM	Rd,Z	Розширенне завантаження з пам’яті програм	Rd = (RAMPZ:Z)	None	3
ELPM	Rd,Z+	Розширенне завантаження з пам’яті програм з пост-інкрементом	Rd = (RAMPZ:Z), Z = Z+1	None	3
SPM	Немає	Зберігання в програмній пам’яті	(Z) = R1:R0	None	-
ESPM	Немає	Розширенне зберігання в програмній пам’яті	(RAMPZ:Z) = R1:R0	None	-
IN	Rd,P	Читання порту	Rd = P	None	1
OUT	P,Rr	Запис в порт	P = Rr	None	1
PUSH	Rr	Завантажити регістр у стек	STACK = Rr	None	2
POP	Rd	Витягання регістру із стеку	Rd = STACK		

ІНСТРУКЦІЇ РОБОТИ С БІТАМИ

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
LSL	Rd	Логічно зсунути ліворуч	Rd(n+1)=Rd(n), Rd(0)=0, C=Rd(7)	Z,C,N,V, H,S	1
LSR	Rd	Логічно зсунути праворуч	Rd(n)=Rd(n+1), Rd(7)=0, C=Rd(0)	Z,C,N,V,S	1
ROL	Rd	Циклічний зсув ліворуч через C	Rd(0)=C, Rd(n+1)=Rd(n), C=Rd(7)	Z,C,N,V, H,S	1
ROR	Rd	Циклічний зсув праворуч через C	Rd(7)=C, Rd(n)=Rd(n+1), C=Rd(0)	Z,C,N,V,S	1
ASR	Rd	Арифметичний зсув праворуч	Rd(n)=Rd(n+1), n=0,...,6	Z,C,N,V,S	1
SWAP	Rd	Поміняти тетради місцями	Rd(3..0) = Rd(7..4), Rd(7..4) = Rd(3..0)	None	1
BSET	s	Встановити прапор	SREG(s) = 1	SREG(s)	1
BCLR	s	Скинути прапор	SREG(s) = 0	SREG(s)	1
SBI	P,b	Встановити біт в порту	I/O(P,b) = 1	None	2
CBI	P,b	Скинути біт в порту	I/O(P,b) = 0	None	2
BST	Rr,b	Зберегти біт з регістру в T	T = Rr(b)	T	1
BLD	Rd,b	Завантажити біт із T в регістр	Rd(b) = T	None	1

Мнемоніка	Операнди	Опис	Операція	Прапори	Цикли
SEC	Немає	Встановити прапор перенесення	C =1	C	1
CLC	Немає	Скинути прапор перенесення	C = 0	C	1
SEN	Немає	Встановити прапор від’ємного числа	N = 1	N	1
CLN	Немає	Скинути прапор від’ємного числа	N = 0	N	1
SEZ	Немає	Встановити прапор нуля	Z = 1	Z	1
CLZ	Немає	Скинути прапор нуля	Z = 0	Z	1
SEI	Немає	Встановити прапор глобального переривання	I = 1	I	1
CLI	Нет	Скинути прапор глобального переривання	I = 0	I	1
SES	Немає	Встановити прапор знаку	S = 1	S	1
CLN	Немає	Скинути прапор знаку	S = 0	S	1
SEV	Немає	Встановити прапор переповнення	V = 1	V	1
CLV	Немає	Скинути прапор переповнення	V = 0	V	1
SET	Немає	Встановити прапор T	T = 1	T	1
CLT	Немає	Скинути прапор T	T = 0	T	1
SEH	Немає	Встановити прапор напівперенесення	H = 1	H	1
CLH	Немає	Скинути прапор напівперенесення	H = 0	H	1
NOP	Немає	Немає операції	Нет	None	1
SLEEP	Немає	Встановити режим SLEEP	http://www.gaw.ru/pdf/Atmel/AVR/avr_user_guide.pdf	None	1
WDR	Немає	Скинути сторожовий таймер	http://www.gaw.ru/pdf/Atmel/AVR/avr_user_guide.pdf	None	1

Примітка:*Для операцій доступу до даних кількість циклів вказано за умови доступу до внутрішньої пам'яті даних, і не коректно при роботі із зовнішнім ОЗУ. Для інструкцій CALL, ICALL, EICALL, RCALL, RET і RETI, необхідно додати три цикли плюс по два циклу для кожного очікування в контролерах з PC меншим 16 біт (128KB пам'яті програм). Для пристроїв з пам'яттю програм понад 128KB, додайте п'ять циклів плюс по три цикли на кожне очікування.

Додаток В.

ДИРЕКТИВИ АСЕМБЛЕРА

Директиви асемблера не транслюються безпосередньо в код. Вони використовуються для вказівки положення в програмній пам'яті, визначення макросів, ініціалізації пам'яті і т.д. Список директив наведено в табл.В.1.

Таблиця В.1.

Директива	Опис
BYTE	Зарезервувати байти в ОЗП
CSEG	Програмний сегмент
DB	Визначати байти во флеш або EEPROM
DEF	Призначити регістру символічне ім'я
DEVICE	Визначити пристрій для якого компілюється програма
DSEG	Сегмент даних
DW	Визначити слова у флеш або EEPROM O
ENDM, ENDMACRO	Кінець макросу
EQU	Встановити постійний вираз
ESEG	Сегмент EEPROM
EXIT	Вийти з файлу
INCLUDE	Вкласти інший файл
LIST	Включить генерацію лістингу
LISTMAC	Включити розгортання макросів в лістингу
MACRO	Начало макросу
NOLIST	Вимкнути генерацію лістингу
ORG	Встановити положення в сегменті
SET	Встановити змінний символічний еквівалент виразу

Перед всіма директивами ставлять точку, наприклад .CSEG.

Додаток Г.

ЗАГАЛЬНА ХАРАКТЕРИСТИКА СЕРЕДОВИЩА PROTEUS VSM

Proteus VSM – середовище для проектування та симуляції роботи електронних схем і мікропроцесорних пристроїв. Відмінністю пакета Proteus VSM є можливість моделювання роботи програмованих пристроїв: мікроконтролерів (МК PIC, 8051, AVR, HC11, ARM7/LPC2000 та ін.), мікропроцесорів DSP.

PROTEUS VSM дозволяє моделювати і налагоджувати досить складні пристрої, в яких може міститися кілька МК одночасно і навіть різних сімейств МК в одному пристрої.

PROTEUS складається з двох основних модулів:

- ISIS – графічний редактор принципових схем служить для введення розроблених проектів з подальшою імітацією і передачею для розробки друкованих плат в ARES. До того ж після налагоджування пристрою можна відразу розвести друковану плату в ARES яка підтримує авторозміщення і трасування по вже існуючій схемі;
- ARES – графічний редактор друкованих плат з вбудованим менеджером бібліотек і автотрасувальник ELECTRA з автоматичним розставленням компонентів на друкованій платі.

Для інсталяції Proteus VSM необхідно:

1. Запустити інсталяційний файл;
2. У відкритому вікні вибрати компоненти, які необхідно інсталювати;
3. Виконати налаштування, необхідні для інсталяції.

Основні пункти меню Proteus ISIS

1. Меню File.
2. Меню View.
3. Меню Edit.
4. Меню Library.
5. Меню Tools.
6. Меню Design.
7. Меню Graph.
8. Меню Source.
9. Меню Debug.
10. Меню Template.
11. Меню System.
12. Меню Help.

На рис. Г.1 зображене головне вікно програми. Весь робочий простір програми розділено на декілька основних частин:

- вікно редактора схем (виконується синтез окремих компонентів);
- вікно вибору об’єктів (доступні різні елементи залежно від вибраного режиму);
- панель керування симуляцією (знаходиться у лівому нижньому кутку, містить такі команди: пуск; виконання одного такту, що вмикає симуляцію на час Single Step Time, який задається у розділі головного меню System>Set Animation Options; пауза; стоп).

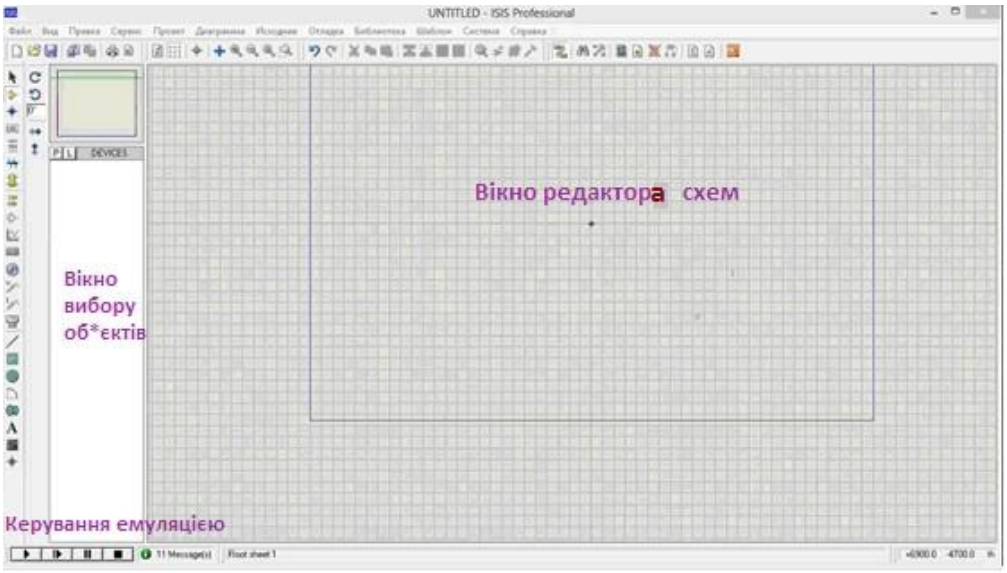


Рис. Г.1. Головне вікно Proteus VSM (ISIS)

Проектування віртуальної моделі електронної схеми та симуляція її роботи в Proteus VSM

Для того, щоб зібрати схему будь-якого пристрою, необхідно підготувати набір елементів, з яких буде складатися ця схема. Для цього переходимо в режим компонентів (рис. Г.2, а) і натискаємо клавішу P, яка знаходиться під вікном перегляду поряд з клавішею L (рис. Г.2, а, б)

а) б)

Рис. Г.2. Панель інструментів

Тоді з’являється менеджер компонентів, який пропонує вибір всіх елементів, що містяться в бібліотеці програми (рис Г.3).

Потрібно користуватися рядком пошуку, який знаходиться у верхньому лівому кутку. Коли потрібний компонент знайдений, подвійне натискання лівої кнопки миші по його назві додасть його до переліку використовуваних компонентів. Наприклад, створимо електронну схему, яка буде складатися зі: світлодіода, резистора, кнопки і блока живлення (батареї). У рядку пошуку (маска) вводимо перший елемент, який будемо додавати до схеми: led-green.

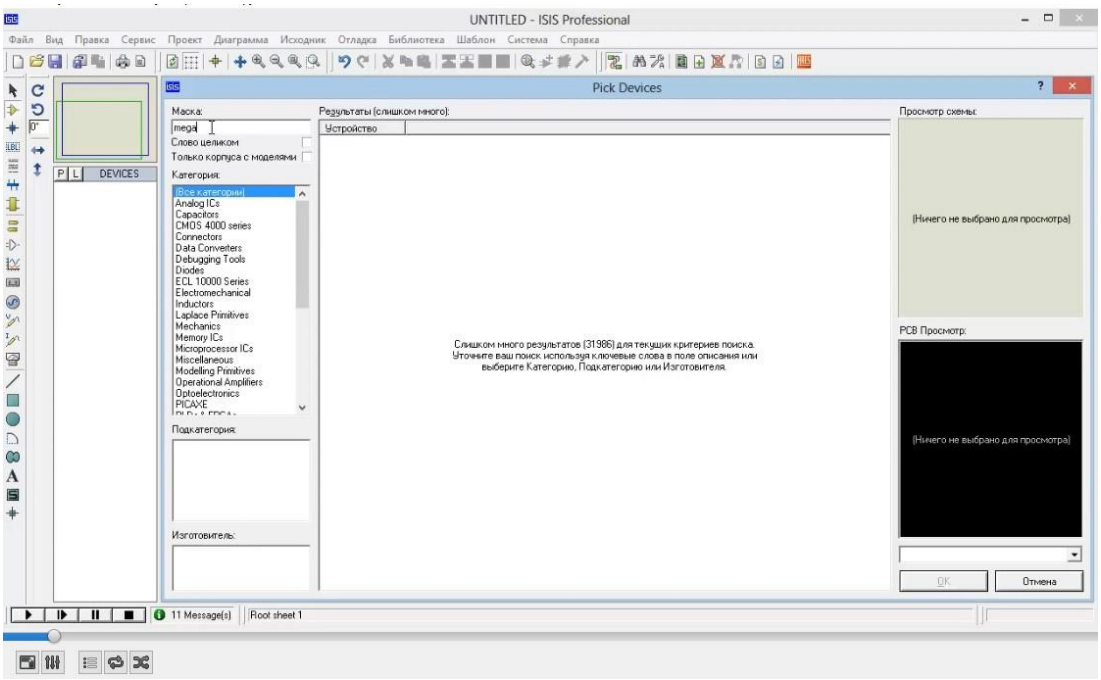


Рис. Г.3. Вікно менеджера компонентів

У списку елементів з’являється світлодіод з такою назвою (рис. Г.4), двічі «клацаємо» по ньому лівою кнопкою миші (ЛКМ).

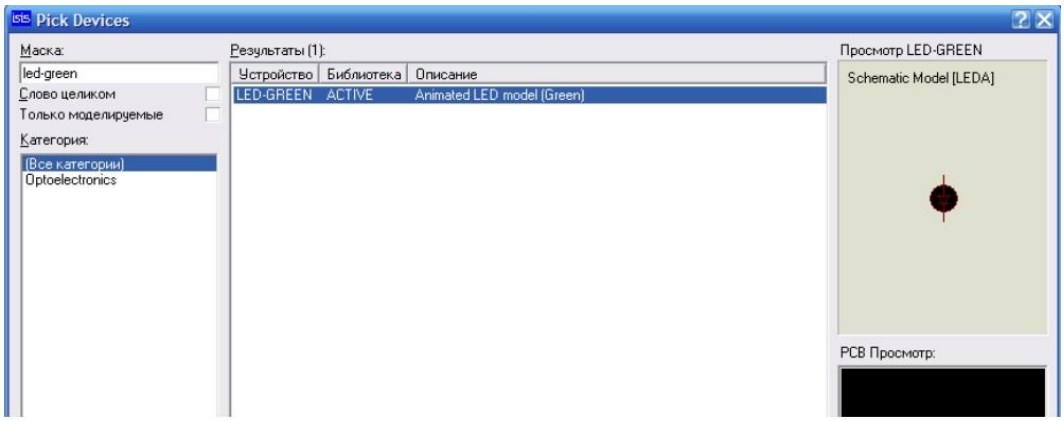


Рис. Г.4 Пошук компонентів

Далі так само робимо пошук за такими назвами: resistor, push button, battery (бібліотека – Active), потім по черзі подвійним натисканням по цих елементах у списку додаємо їх до нашого проекту.

Після того як ми вибрали всі компоненти, натискаємо на кнопку ОК. Тепер вікно вибору об’єктів буде містити всі чотири обрані компоненти і виглядати, як зображено на рис. Г.5.

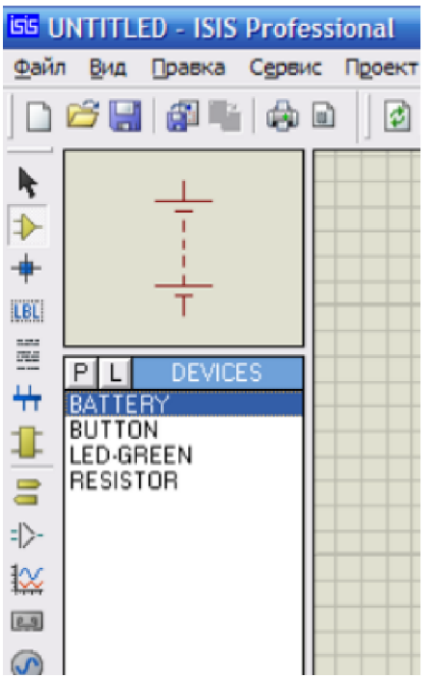
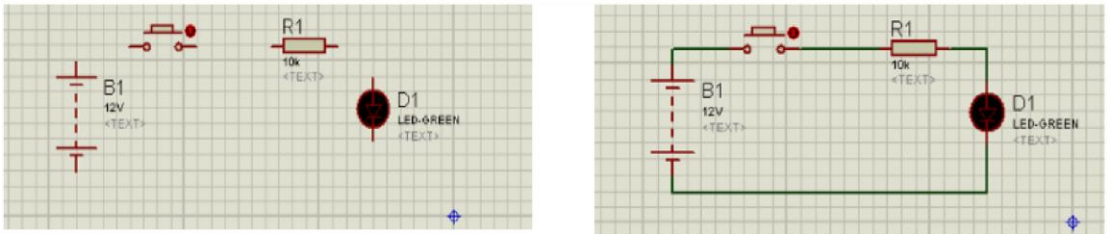


Рис. Г.5. Вікно вибору об’єктів

Для встановлення компонента у вікні редактора схеми його потрібно вибрати зі списку і подвійним натисканням лівої кнопки миші (ЛКМ) встановити у потрібному місці. До того, як встановити компонент на схемі, його можна попередньо розвернути у потрібне положення, що можна контролювати у вікні перегляду. Наприклад, потрібно всі компоненти, які зображено на рис. Г.6, а), з’єднати, як показано на рис. Г.6, б).

Для того, щоб з’єднати два компоненти між собою, для початку потрібно вибрати інструмент **Стрілка**, потім на самій схемі навести курсор миші на кінець елемента, має з’явитися квадратний червоний контур, натиснути і відпустити ЛКМ, вести курсор до контакту з іншим елементом, на кінці іншого елемента також натиснути ЛКМ, щоб завершити побудову з’єднання між двома елементами (рис. Г.6, б).



а) б)

Рис. Г.6 Побудова електричної схеми

Після того як електричне коло створено, змінюємо характеристики наших компонентів. На початку батарея живить схему 12 В, а нам потрібно змінити живлення схеми на 5 В. Для цього подвійно натискаємо ЛКМ на батареї і у полі Voltage встановлюємо 5V.

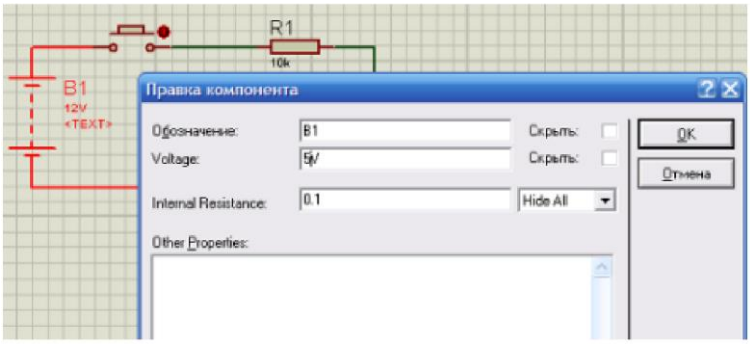


Рис. Г.7. Зміна параметрів живлення

Так само змінюємо опір резистора на 680 Ом.

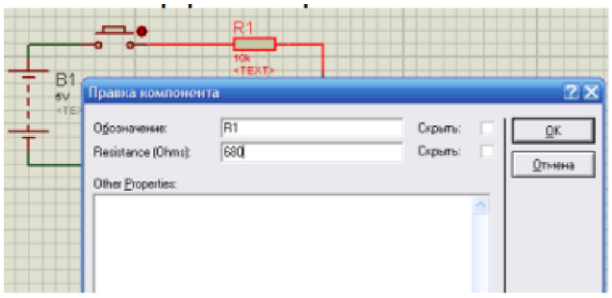


Рис. Г.8. Зміна параметрів опору R1

Потім натискаємо в панелі керування симуляцією кнопку «**Воспроизвести**».

Рис. Г.9. Панель керування симуляцією

Тепер дана схема може віртуально відтворити вмикання та вимикання світлодіода при натисканні на кнопку (рис Г.10). Для цього потрібно просто навести курсор миші на елемент кнопка у схемі та натиснути ЛКМ.

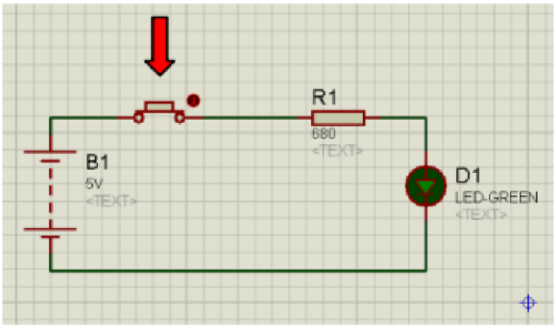


Рис. Г.10. Симуляція роботи світлодіода

Симуляція роботи мікроконтролера за допомогою Proteus VSM

Створіть новий проект у Proteus VSM і накресліть схему, яку зображено на рис. Г.15. У менеджері компонентів знайдіть нижченаведені елементи:

1 світлодіод;

2 мікроконтролер ATMega328P;

3 резистор; 4 заземлення.

Для того, щоб створити елемент «заземлення» (GROUND), перейдіть у панелі інструментів (знаходиться вертикально зліва) на вкладку Terminals Mode.

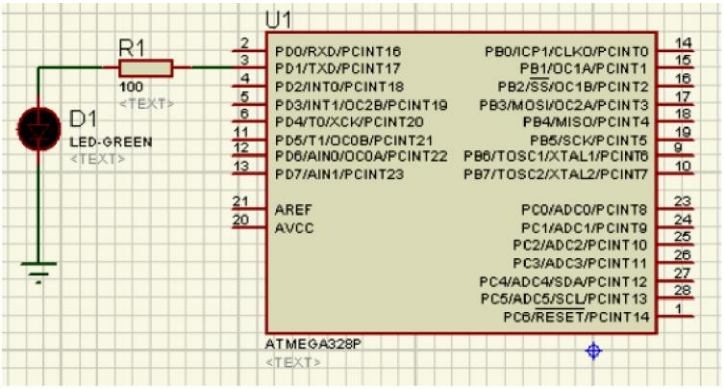


Рис. Г.15. Схема під’єднання

Після з’єднання елементів у електричне коло двічі натисніть на зображення мікроконтролера. Відкриється вікно Edit Component (рис.ºГ.16), в якому у полі Program File вкажіть шлях до файлу BlinkLED.hex. також перевірте, щоб була встановлена частота мікроконтролера така сама, як і у програмному коді, тобто 8 МГц.

Після цього на панелі керування симуляцією натискаємо на **Play**, і якщо все правильно було зроблено, спостерігаємо роботу схеми.

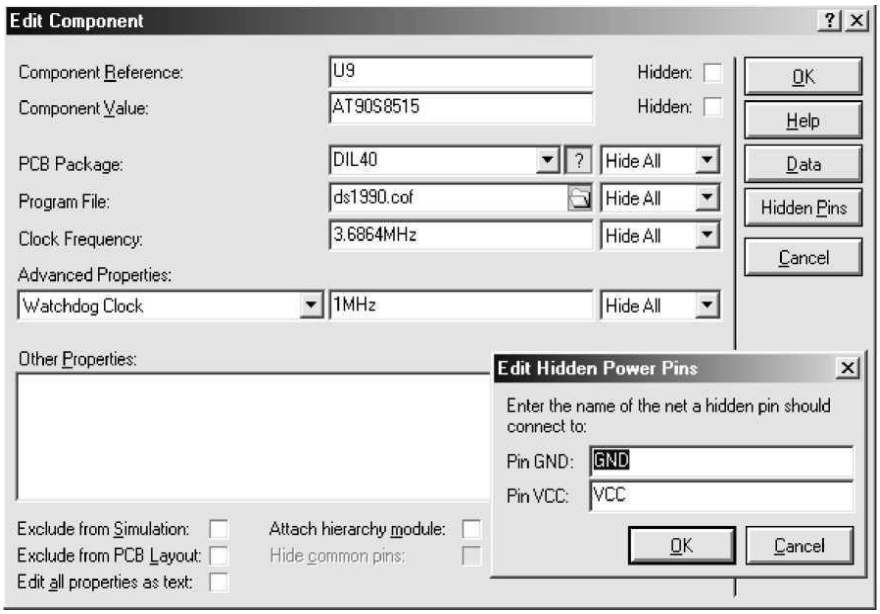


Рис. Г.16. Вікно Edit Component