Міністерство освіти і науки України
ЧЕРНІГІВСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ

МІКРОКОНТРОЛЕРИ СІМЕЙСТВА MCS-51 В ЗАДАЧАХ ОБРОБКИ ІНФОРМАЦІЇ ТА КЕРУВАННЯ

Методичні вказівки до виконання лабораторних робіт з дисципліни «Мікропроцесорні пристрої керування та обробки інформації» для студентів напрямів підготовки 6.050802 «Електронні пристрої та системи» та 6.050902 «Радіоелектронні апарати»

Затверджено на засіданні кафедри
промислової електроніки.
Протокол № 8 від 30.05.2014 р.

Чернігів – 2014

Мікроконтролери сімейства MCS-51 в задачах обробки інформації та керування. Методичні вказівки до виконання лабораторних робіт з дисципліни «Мікропроцесорні пристрої керування та обробки інформації» для студентів напрямів підготовки 6.050802 «Електронні пристрої та системи» та 6.050902 «Радіоелектронні апарати».Укл.: Войтенко В.П., Хоменко М.А. – Чернігів: ЧНТУ, 2014. – 71 с.

Укладачі: Войтенко Володимир Павлович, кандидат технічних наук, доцент
Хоменко Максим Анатолійович, кандидат технічних наук, доцент

Відповідальний за випуск: Денисов Юрій Олександрович, доктор технічних наук, професор, завідувач кафедри промислової електроніки

Рецензент: Вервейко Олександр Іванович, кандидат технічних наук, доцент кафедри інформаційних комп‘ютерних систем Чернігівського національного технологічного університету

Зміст

Перелік умовних скорочень

МК – мікроконтролер.
МК51 – мікроконтролери сімейства MCS-51.
МП – мікропроцесор (мікропроцесорний).
МПС – мікропроцесорна система.
ОЗП – оперативний запам‘ятовуючий пристрій.

Вступ

Мікроконтролери сімейства MCS-51 (МК51) впродовж багатьох років залишаються актуальними завдяки їх широкому застосуванню в раніше розроблених електронних системах, накопиченому прикладному програмному забезпеченню та підтримці виробниками радіоелектронних компонентів (наприклад, *Atmel, NXP, Analog Devices*). Велике значення має розвинуте методичне забезпечення та досвід викладання особливостей MCS-51.

Дані методичні вказівки призначені для самостійної підготовки студентів до виконання лабораторних робіт з дисциплін «Мікропроцесорні пристрої керування та обробки інформації», а також «Мікропроцесорна техніка». Лабораторні роботи виконуються на спеціалізованому навчально-відлагоджувальному стенді *EV8031/AVR* і присвячені вивченню архітектури та практичному дослідженню характерних особливостей мікропроцесорних систем на базі мікроконтролерів сімейства MCS-51.

До складу кожного розділу входять основні відомості та деякі особливості підключення периферійних модулів у складі стенда. Процес підготовки до виконання роботи потребує обов'язкової самостійної роботи з рекомендованою літературою та лекційним матеріалом. Після теоретичної частини наведено список контрольних питань, за допомогою яких оцінюється ступінь опанування теоретичного матеріалу і готовність студента до виконання лабораторної роботи. Далі наводиться перелік завдань, які потрібно вирішити в ході виконання роботи.

Результати виконання робіт оформлюються у вигляді звітів на аркушах формату *A4*. Звіт повинен містити таку інформацію:

1) титульний аркуш встановленого зразку з повною назвою роботи і підписом студента, що виконав звіт;

2) короткі відомості про об'єкт вивчення (*15…20* рядків), в тому числі – схему підключення певного периферійного модуля;

3) відповіді на контрольні запитання;

4) схему програми для МК і стислий опис алгоритму її дії;

5) текст програми для МК з коментарями;

6) висновки щодо отриманих результатів.

Пункти 1 – 4 оформлюються і показуються викладачеві на початку заняття, що є умовою допуску до виконання роботи в навчальній лабораторії. Весь звіт надається безпосередньо на захист роботи.

1 Лабораторна робота №1. Налаштування робочого місця для програмування МК

Мета роботи: вивчити побудову, можливості лабораторного стенда та ознайомитися з основами програмування МП, налагодженням і виконанням програм мовою С.

1.1 Призначення і склад лабораторного стенду „МПТ”

Лабораторний стенд „МПТ” призначений для отримання навичок програмування на асемблері та мовами високого рівня мікроконтролера ATMEL AT89C52 (прототип – Intel MCS-51), а також вивчення допоміжних і інтерфейсних мікросхем, застосовуваних в МПС:

– паралельного програмуємого інтерфейсу (ППІ) i8055A;

– годинника реального часу з послідовним інтерфейсом Dallas Semiconductor DS1307;

– цифрового датчика температури Dallas Semiconductor DS1621/ DS1631;

– пам‘яті з послідовним доступом на 256 байт Atmel AT24C02.

Стенд „МПТ” містить персональний комп‘ютер та навчально-відлагоджувальний пристрій (НВП) EV8031/AVR (Рисунок 1.1). Персональний комп‘ютер використовується для створення вхідного програмного модуля, його трансляції та відлагодження за допомогою крос-засобів. Завантаження відлагодженої програми у НВП EV8031/AVR виконується за допомогою послідовного інтерфейсу RS-232C через COM-порт персонального комп‘ютера.

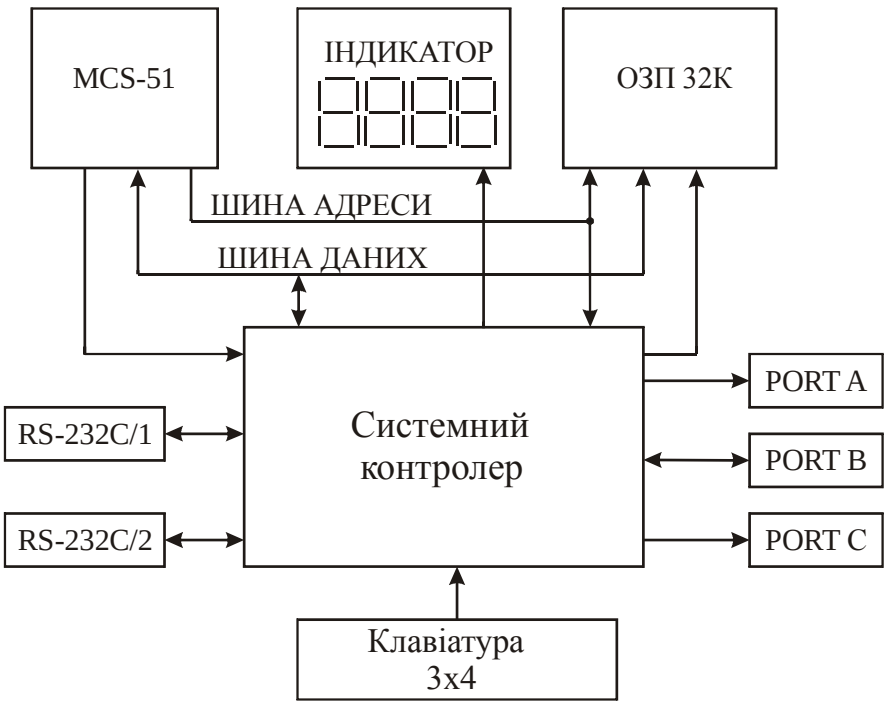

Рисунок 1.1 – Структура НВП *EV8031/AVR*

НВП являє собою відлагоджувальну мікроЕОМ, до системної шини

якої підключені в якості зовнішніх пристроїв мікросхеми, що використовуються в лабораторних роботах. НВП містить наступні вузли:

RS232C/1 – послідовний порт для зв‘язку з персональним комп‘ютером;

RS232C/2 – послідовний порт для зв‘язку з периферійними пристроями;

PORTA, PORTB, PORTC – паралельні приймачі/передавачі.

Логіку НВП EV8031/AVR реалізовано на програмуємій логічній мікросхемі EPM7128STC100. Системний контролер керує режимами роботи, формує керуючі сигнали для ОЗП, регістрів защілки, семисегментного світлодіодного індикатора та клавіатури.

На схемі розміщення складових частин НВП (Рисунок 1.2) позначені наступні елементи НВП:

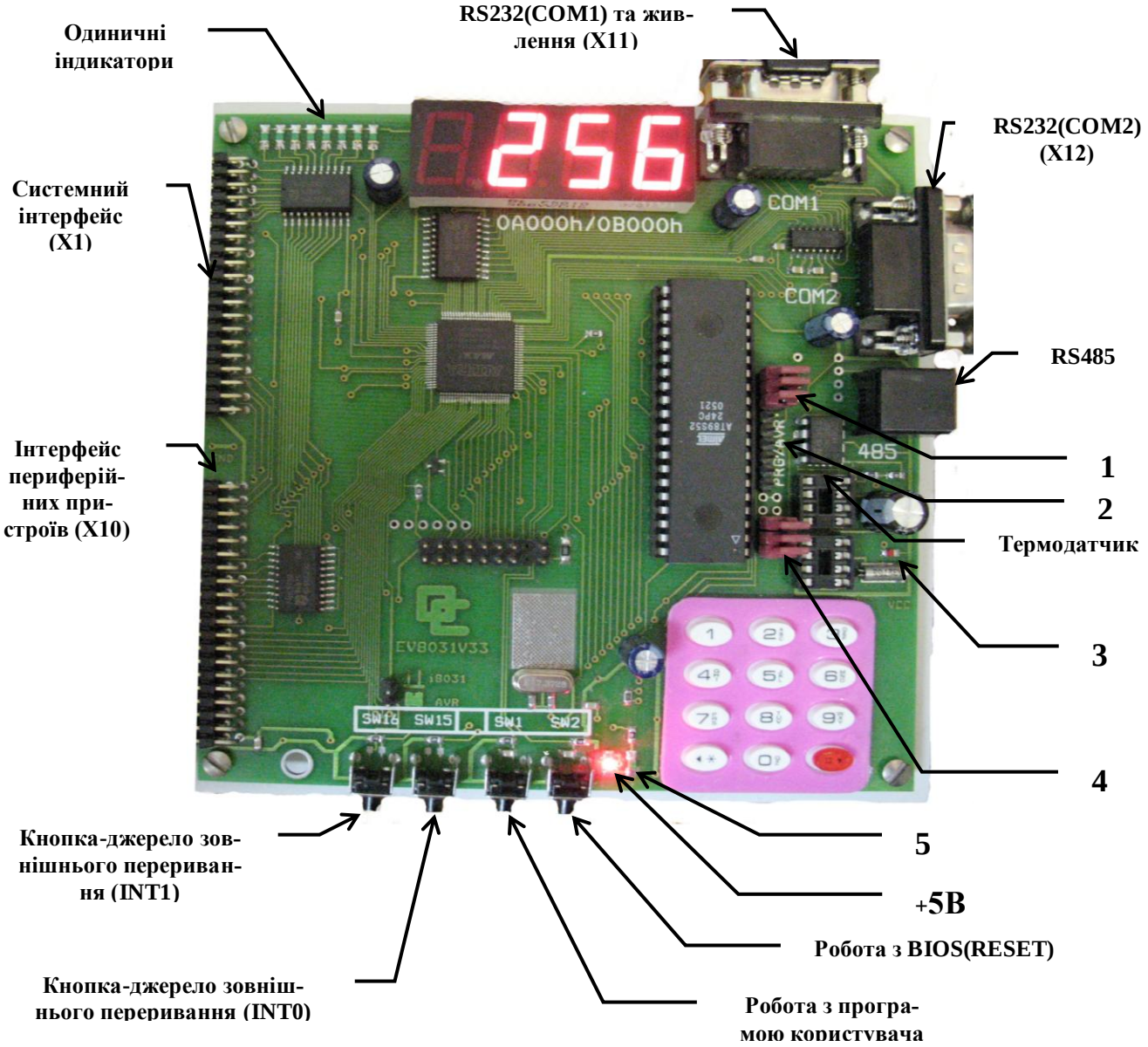

Рисунок 1.2 – Схема розміщення складових частин НВП EV8031/AVR

X1 – системний інтерфейс з повним адресним простором MCS-51;

X10 – інтерфейс розширення для підключення зовнішніх пристроїв з використанням паралельного приймача/передавача;

X11 – інтерфейс послідовного порту COM1 для зв‘язку НВП з комп‘ютером;

X12 – інтерфейс послідовного порту COM2 для зв‘язку НВП з іншими пристроями, які мають стандартний порт RS232C;

1 – перемички з’єднання ліній T0, INT0, INT1 (згори донизу);

2 – роз’єм для програмування МК ATMEL AVR (X3);

3 – індикатор роботи мікросхеми-годинника;

4 – перемички з’єднання ліній P1.2, P1.1, P1.0 (згори донизу);

5 – індикатор роботи програми користувача HL1 (зелений).

До системної шини НВП *EV8031/AVR* може бути під‘єднано одну з кількох модифікацій плат розширення. Базова плата розширення призначена для проведення лабораторних робіт, пов‘язаних з аналого-цифровим та частотним перетворенням, а також з обробкою дискретних сигналів.

Базова плата розширення №1 (Рисунок 1.3) містить набір приладів, які дозволяють засвоїти основні прийоми програмування периферії МПС.

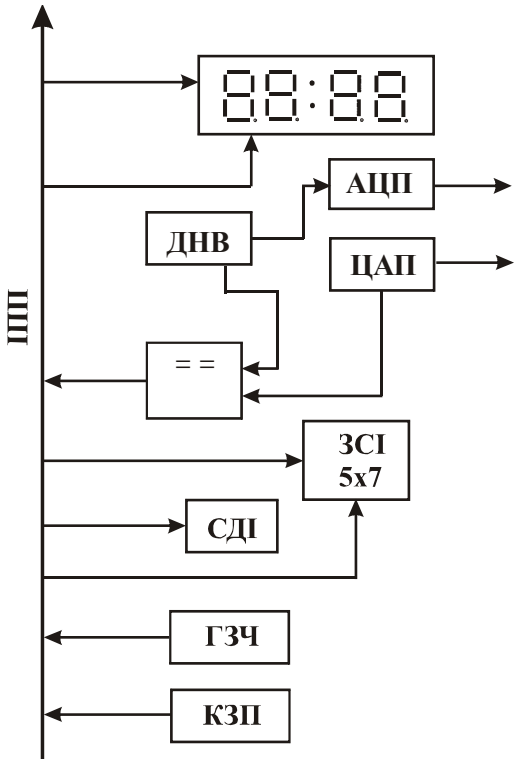

Рисунок 1.3 – Структура плати розширення №1 НВП *EV8031/AVR*

На рисунку позначені наступні елементи плати розширення:

8888 – 4-розрядний індикатор з динамічною індикацією;

ІПП – інтерфейс периферійних пристроїв;

ЦАП – цифроаналоговий перетворювач;

СДІ – світлодиодні індикатори;

ЗСІ – світлодіодний знакосинтезуючий матричний індикатор 5x7;

ГЗЧ – генератор із змінною частотою сигналу;

КЗП – кнопки запиту переривання;

ДНВ – джерело напруги, що вимірюється;

ШД – шина даних.

Після увімкнення стенда або скидання МК (DD1) стартує програма-завантажувач, яка знаходиться у резидентній Flash-пам‘яті МК AT89C52. Ця програма виконує ініціалізацію послідовного приймача/передавача МК, перевіряє наявність і ємність пам‘яті даних.

Пам‘ять ОЗП обсягом 32 Кб розподілено на дві частини по 16 Кб. Одна частина використовується в якості пам‘яті програм для завантаження програм користувача. Інша частина є для МК звичайною зовнішньою пам‘яттю даних. Під час завантаження вся зовнішня пам‘ять 32К б відображується в адресний простір в якості пам‘яті даних (Рисунок 1.4).

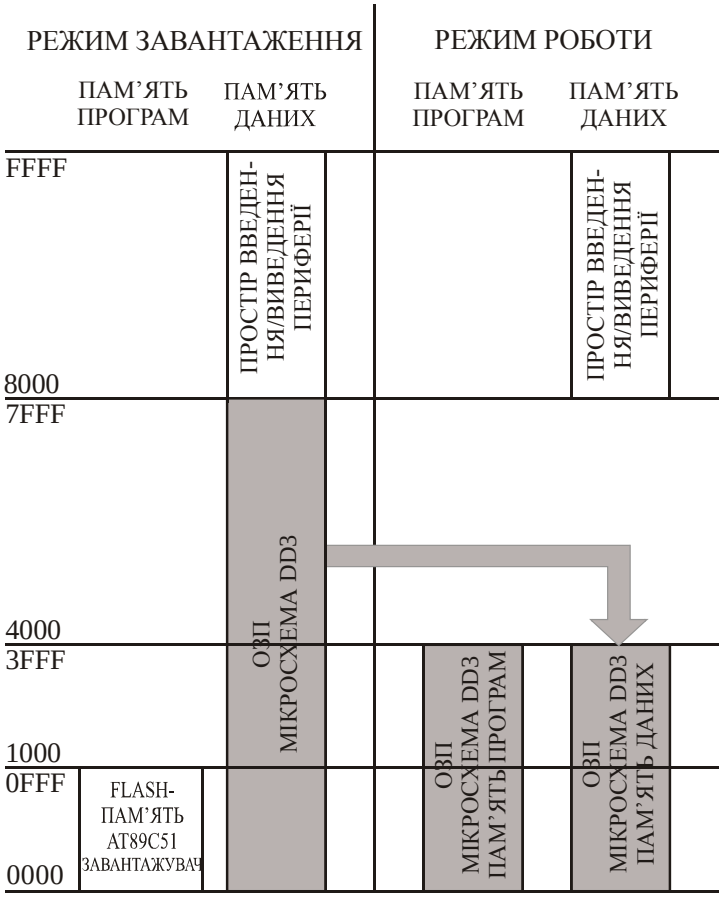

Рисунок 1.4 – Розподіл пам‘яті НВП *EV8031/AVR*

Звертання процесора до периферійних пристроїв НВП EV8031/AVR реалізовано, як адресація до комірок зовнішньої пам‘яті даних в адресному просторі від 8000h до FFFFh. Сигнали вибору периферійних пристроїв формуються дешифратором адреси у складі мікросхеми системного контролера DD4. Адресний простір пристроїв введення/виведення наведений в таблиці 1.1.

Чотирирозрядний семисегментний світлодіодний індикатор підключений до системного контролера, який автоматично виконує динамічну регенерацію зображення та декодування двійкового коду в код семисегмент-

ного індикатора. Індикатор починає працювати одразу після подання живлення. Контролер індикатора містить два восьмирозрядних регістри, вміст яких відображується на індикаторі. Вміст регістра з адресою 0xA000 відображується на двох лівих розрядах, вміст регістра з адресою 0xA001 (0xB000) – на двох правих розрядах у вигляді шістнадцяткових цифр.

Керування десятковими крапками та гасінням здійснюється через регістр DC_REG (0xA004). Біти DP3...DP0 керують десятковими крапками. Запис «1» у певний розряд вмикає десяткову крапку. Біти BL3...BL0 керують гасінням розрядів індикатора. Запис «1» у ці біти призводить до гасіння відповідного розряду індикатора.

Стан стовпчика матриці клавіатури зчитується з комірки з базовою адресою 0x9000, біти 3...0. Відповідний стовпчик вибирають «нулем» в розрядах адреси A2..A0. Тобто адреса 0x9006 вибирає перший стовпчик, адреса 0x9005 – другий стовпчик, адреса 0x9003 – третій стовпчик. Признак натиснутої клавіші зчитується як «нуль» у відповідному розряді.

В НВП EV8031/AVR можна завантажити програму користувача у форматі *Intel-HEX*. При цьому використовується спеціальна програма-завантажувач *eval32.exe.* Щоб розпочати процес завантаження, наприклад, файлу *ind.hex*, треба в командному рядку `DOS` (або `Total Commander`) набрати:

```
EVAL32.EXE -hs -com 1 9600 IND.HEX
```

Під час завантаження програми з персонального комп‘ютера в НВП на екрані відеомонітора відображуються дані, що передаються. Ці ж дані відображуються на індикаторі НВП HG1. Світлодіод HL1 засвічується. Після передавання останнього байта програми завантажена програма запускається автоматично.

Зупинка запущеної програми та перехід в режим очікування на прийом даних з персонального комп‘ютера можливі за натисканням кнопки SW2. При цьому світлодіод HL1 погасне. Завантаження нової програми можливе в будь-який момент роботи раніше завантаженої програми.

При постійній практичній роботі з НВП дуже зручним рішенням буде налаштування файлової асоціації *Windows* таким чином, аби автоматично при виборі файлу з розширенням “`hex`” запускався допоміжний пакетний файл *DOS* (тобто з розширенням “`bat`”), в якому був би присутній всього лише один рядок виду:

```
EVAL32.EXE -hs -com 1 9600 %1
```

Таке налаштування зроблене в навчальній лабораторії, а отже за кликом миші по `hex`-файлу розпочинається завантаження програми у НВП.

Таблиця 1.1 **– Адресний простір пристроїв введення/виведення НВП EV8031/AVR**

Адреса	**Тип циклу**	**B7**	**B6**	**B5**	**B4**	**B3**	**B2**	**B1**	**B0**	**Ім'я**
Порти периферійних пристроїв										
8xx0	Запис	[Порт A]								PA_REG
8xx1	Запис	[Порт B]								PB_REG
8xx2	Запис	[Порт C]								PC_REG
8xx3	Запис	x	x	x	x	x	TRISC	x	x	TRIS
РКІ										
8xx4	Запис	Регістр команд РК-індикатора								LCD_CMD
8xx5	Запис	Регістр даних РК-індикатора								LCD_DATA
Послідовний порт										
9xxx	Читання	CTS	DSR	DCD	RI	KL3	KL2	KL1	KL0	US_REG
Cxx0	Запис	x	x	X	x	DTR	RTS	CFG1	CFG0	UC_REG
CFG1	CFG0									
0	0	RS-232				COM1, X11				
0	1	RS-232				COM2, X12				
1	0	RS-485				Прийом, X13				
1	1	RS-485				Передача, X13				
Індикатор и світлодіоди										
Axx0	Запис	[Регістр індикатора 0] – ліві знакомісця								DISPLAY[0]
Axx1	Запис	[Регістр індикатора 1] – праві знакомісця								DISPLAY[1]
Axx2	Запис	<зарезервовано>								DISPLAY[2]
Axx3	Запис	<зарезервовано>								DISPLAY[3]
Axx4	Запис	DP3	DP2	DP1	DP0	BL3	BL2	BL1	BL0	DC_REG
Axx5	Запис	<зарезервовано>								EDC_REG
Axx6	Запис	LED7	LED6	LED5	LED4	LED3	LED2	LED1	LED0	LED_REG
Керування роботою										
Axx7	Запис	x	x	X	x	x	x	x	RUN	SYS_CTL
Сумісні регістри										
Bxx0	Запис	[Регістр індикатора 1]								DISPLAYB
F000	Запис									DAC nCS7

1.2 Робота в інтегрованому середовищі розробки програмного забезпечення *Eclipse*

НВП EV8031/AVR являє собою найпростішу ЕОМ, котра дозволяє виконувати програми, які підготовлені на персональному комп'ютері та завантажені в пам'ять програм МК51. Програмування на Асемблері [2] до-

зволяє отримати компактні програми з високою швидкістю виконання задач управління периферійними пристроями. Проте тексти програм на Асемблері виходять громіздкими та погано читаються. Іншим недоліком є те, що програма, написана на Асемблері, не може бути перенесена на інший тип мікропроцесора. До того ж широка номенклатура МК, які застосовуються в задачах обробки інформації та керування, змушує програміста занадто багато часу витрачати на засвоєння нових систем команд. Тому, хоча один рядок операторів мови високого рівня (Pascal, C...) здебільшого включає декілька команд мікропроцесора на Асемблері, програмування мікроконтролерів саме мовами високого рівня набуло поширення.

На даний час існує чимало компіляторів С, пристосованих під *MCS51*. В навчальній лабораторії застосовується пакет програм оптимізуючого ANSI-C компілятора *SDCC* (*Small Device C Compiler*) з відкритим кодом, який можна налаштувати для декількох 8-розрядних МК [9].

Процедура написання програм на С має ряд особливостей. Для прикладу розглянемо просту задачу виведення числа „2014” на семисегментний дисплей НВП. Складання програми починається з обмірковування алгоритму та розробки схеми програми. Наступним етапом є написання вхідного тексту програми та компіляції її у файл, який можна завантажити у НВП. На практиці в сучасних умовах ця процедура виконується за допомогою програми-транслятора на персональному комп’ютері.

Для того, щоб виконати в навчально-відлагоджувальному пристрої *EV8031/AVR* свій перший проект для *MCS51*, що написаний мовою *C*, застосуємо універсальне середовище для розробки програмного забезпечення (*IDE*) *Eclipse*. Воно має декілька переваг порівняно з „фірмовими” *IDE* або безпосереднім застосуванням *SDCC*:

1) Зручність роботи завдяки графічному представленню інформації у єдиній сучасній наочній оболонці замість командного рядка та використання декількох програм.

2) Можливість нарощування шляхом підключення додаткових плагінів для різних МК. Наразі можна працювати з *MCS51, AVR, ARM* та ін.

3) Безкоштовність без обмеження розміру вхідного файлу та терміну використання.

4) Покращення організації робочого місця, спрощення доступу до власних розробок. Універсальність і відсутність потреби у встановленні та засвоєнні декількох програм: одне й те ж середовище для кількох проектів для різних МК.

Наступні кроки дозволять Вам підготувати робоче місце та завантажити шістнадцятковий файл для виконання на *EV8031/AVR*.

1. По-перше, треба скачати та встановити Java-машину [8].

2. Скачати *IDE Eclipse*, яке підтримує розробку програмного забезпечення мовами C/C++ для певної операційної системи, тобто містить *C/C++ Development Toolkit* (*CDT*) [7]. У навчальній лабораторії, наприклад, застосовується версія *Juno Service Release 2*. Поточний архівний файл мо-

жна знайти тут: Z:\SOFT\speciality\Microcontrollers\MCS51\eclipse-cpp-juno-SR2-win32.zip.

3. Установити *IDE Eclipse*. Для цього достатньо скопіювати директорію *eclipse* з архіву зі всім її вмістом на місце установки. В нашому випадку це зроблено в E:\Program Files\.

4. Скачати та встановити компілятор *SDCC* [9].

5. Скачати плагін для *IDE Eclipse*, що підтримує компілятор *SDCC* [11]. Поточний архівний файл можна знайти тут: Z:\SOFT\speciality\Microcontrollers\MCS51\net.sourceforge.eclipsesdcc-1.0.0-win32.x86.zip.

6. Додати плагін до *IDE Eclipse*. Для цього достатньо скопіювати все, що знаходиться в директоріях features (в нашому випадку – net.sourceforge.eclipsesdcc.feature_1.0.0) та plugins (у нас – net.sourceforge. eclipsesdcc.win32_1.0.0, а також net.sourceforge. eclipsesdcc_1.0.0) архіву зі всім вмістом у відповідні (однойменні) директорії установки *IDE Eclipse*. В нашому випадку це зроблено в E:\Program Files\.

7. Запустити *IDE Eclipse*. В навчальній лабораторії це можна зробити, використовуючи піктограму *IDE Eclipse* в рядку меню *Total Commander* (підказка, що випливає, *Universal IDE*). Налаштування передбачає запуск файлу E:\Program Files\eclipse\eclipse.exe. Тобто, той же результат буде, якщо в командному рядку набрати:

eclipse.exe

8. Під час першого завантаження *IDE Eclipse* з'явиться діалогове вікно **Workspace Launcher**, в якому буде запропоновано обрати робочу директорію. У лабораторії обрано D:\students. В подальшому можна легко обрати свій робочий простір безпосередньо з вікна **Workspace Launcher**, в яке можна увійти за допомогою меню **File > Switch Workspace >Other…** Ми обрали D:\students\4_kurs\Brigade_1. При цьому можливе перезавантаження *IDE Eclipse*.

9. Під час першого завантаження *IDE Eclipse* з'явиться вікно-запрошення **Welcome**, яке треба уважно дослідити, щоб ознайомитися з можливостями середовища.

10. Після закриття вікна **Welcome** Ви опинитеся безпосередньо в лабораторії *IDE Eclipse* (**Workbench**), де й будуть утворюватися та зберігатися різноманітні програмні проекти.

11. До початку створення свого першого проекту для *SDCC* треба перевірити, чи правильно встановлений відповідний плагін. Це можна зробити, скориставшись меню **Help/Help Contents**. Якщо все в порядку, Ви побачите рядок **Eclipse SDCC compiler plugin**. Прочитайте та зробіть письмовий переклад пунктів допомоги **Before you begin** та **Getting started**. Вигляд вікон програми та послідовність дій може відрізнятися в залежності від встановленої версії *IDE Eclipse*, проте суть роботи з проектом залишається тою ж самою.

12. Утворіть свій перший проект наступним чином. З рядка меню оберіть **File > New > C Project...**

13. У вікні проекту (Рисунок 1.5) лівою кнопкою миші укажіть тип проекту **MCS51 family (SDCC)**. Підвіконце **Toolchains** змінить вигляд. Якщо зняти вибір **Use Default Location**, у Вас з'явиться можливість створити проект в іншій директорії, ніж була задана раніше (через кнопку **Browse**). Поки що нічого міняти не будемо, а у віконці **Project name** наберіть ім'я Вашого першого проекту, наприклад, **Test1**. Аби не мати зайвого клопоту, не використовуйте кирилиці, проміжків та спеціальних символів в іменах проектів, файлів та директорій (зі всіма піддиректоріями) під час своєї роботи з мікроконтролерами! Це стосується не тільки *MCS51*.

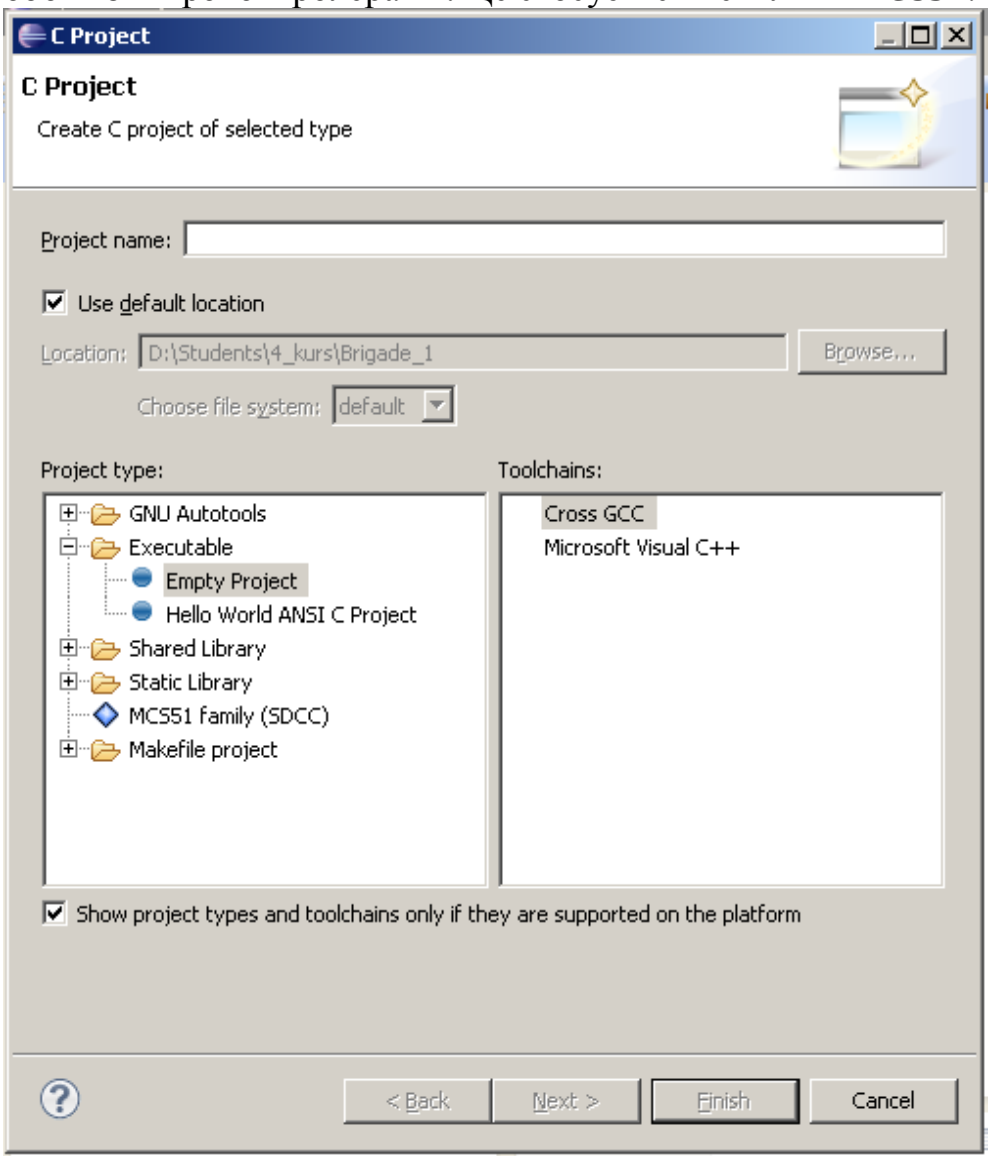

Рисунок 1.5 – Вікно проекту Eclipse

14. Натиснувши *Next*, ми потрапимо у вікно вибору конфігурації проекту (Рисунок 1.6). Тут можна побачити, що **Eclipse SDCC compiler plugin** підтримує дві версії проекту: **Release** та **Debug**. Нас поки що цікавить *Release*-версія, проте залишимо обидва вибори, просто натиснувши кнопку **Finish**.

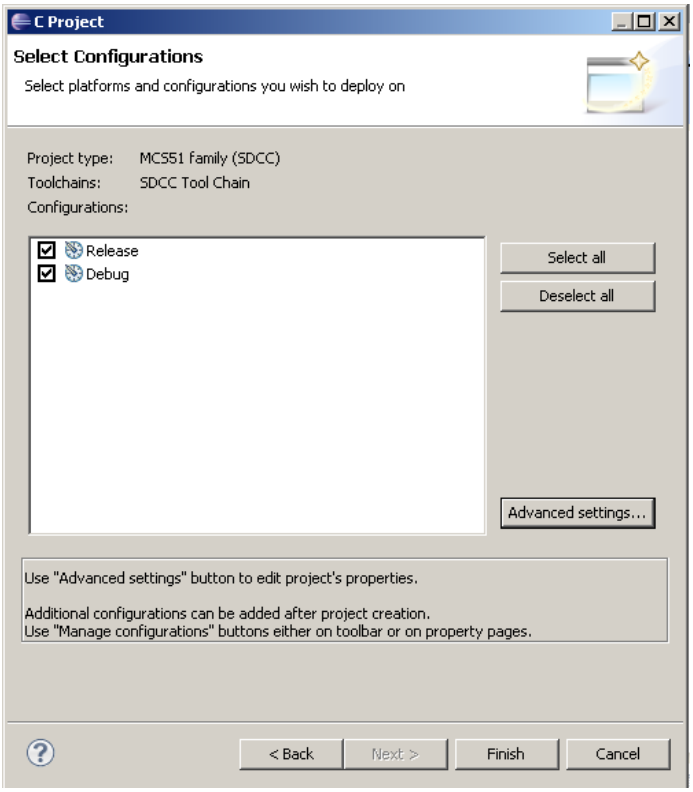

Рисунок 1.6 – Вікно вибору конфігурації проекту Eclipse

15. У вікні *IDE Eclipse* в розділі **Project Explorer** тепер присутній наш проект, який розкриємо подвійним кликом миші.

16. Далі можна (задля кращої структуризації) утворити директорію для зберігання вхідних файлів проекту за допомогою або головного, або контекстного меню (з використанням клику правою клавішею миші), вказавши **New > Source Folder**. Будемо зберігати вхідні файли, наприклад, в директорії Sources. Після введення її імені треба натиснути **Finish.**

17. На наступному етапі треба утворити вхідний файл за допомогою, наприклад, контекстного меню **New>Source File**. У вікні вхідного файлу (Рисунок 1.7) вводимо, наприклад, ім'я **main**. Прочитайте повідомлення, яке з'явилося у полі коментарю. Як тільки ви додасте розширення файлу **.c**, все стане потрібним чином, і до проекту буде доданий файл, де можна ввести власну програму. Цей пункт можна сумістити з попереднім.

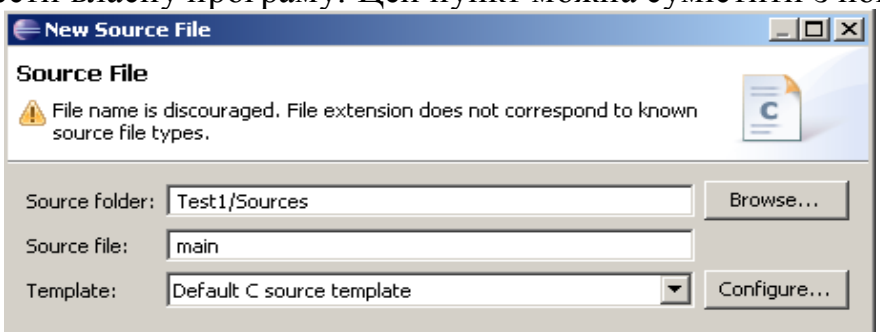

Рисунок 1.7 – Вікно вибору вхідного файлу Eclipse

18. В центральному вікні редактора *IDE Eclipse* (Рисунок 1.8) розк-

рийте заголовок файлу, який створений автоматично (клик по „+”), та скоригуйте його (вкажіть автора, номер лабораторної роботи тощо). Як бачимо, *IDE Eclipse* не тільки вказує на помилки, але й пропонує методи вирішення. Якщо в коментарях Ви будете використовувати інші, окрім англійської, мови, можна додати словники користувача для перевірки правопису через меню **Window>Preferences>General>Editors>Text Editors>Spelling.**

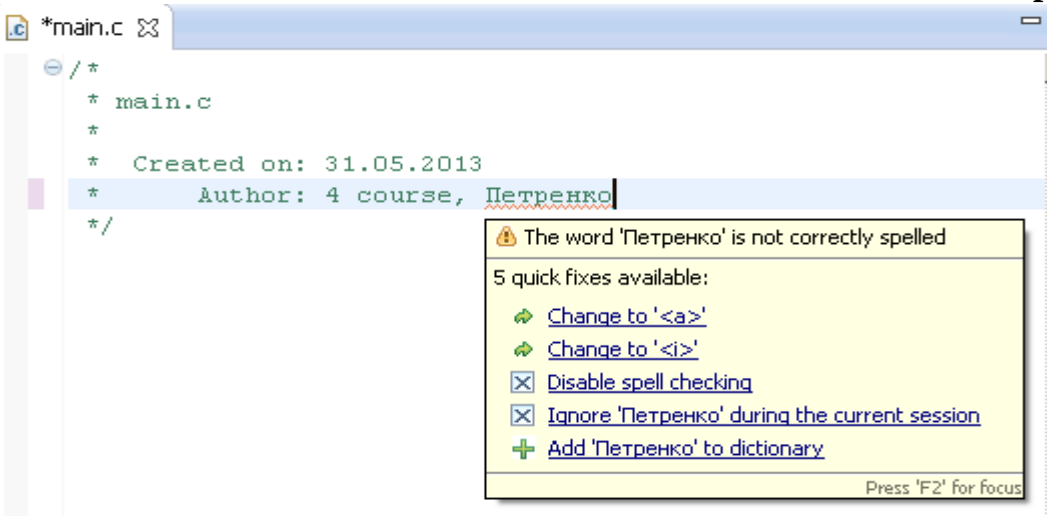

Рисунок 1.8 – Вікно редактора Eclipse

Там же можна зробити й інші налаштування перевірки правопису у коментарях (Рисунок 1.9), чи відключити перевірку правопису взагалі.

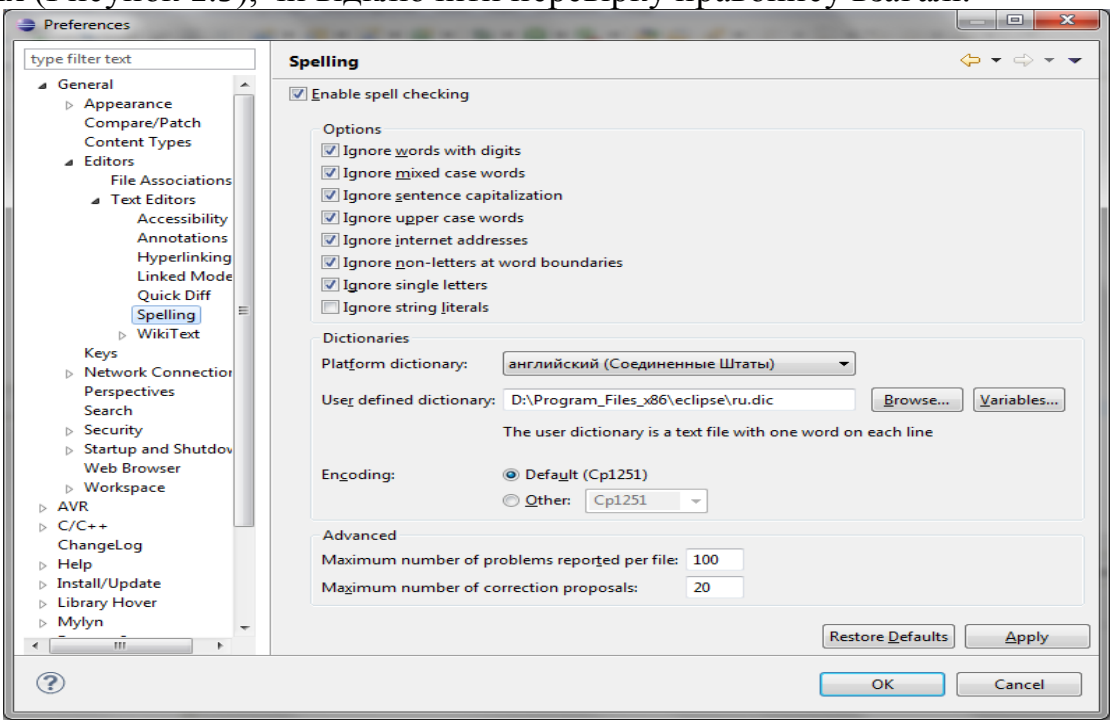

Рисунок 1.9 – Налаштування перевірки правопису

19. Перш, ніж почати вводити текст програми, необхідно зробити деякі налаштування проекту. Для цього заходимо в меню **Pro-**

ject>Properties>C/C++ Build>Settings та обираємо пункт **Directories**, де за допомогою кнопки **Add** додаємо шляхи до бібліотечних заголовкових файлів компілятора SDCC (Рисунок 1.10). Потім обираємо пункт **Libraries,** де у вікні **Library search path** за допомогою кнопки **Add** (зелений +) додаємо шляхи до бібліотек компілятора SDCC (Рисунок 1.11). Натискаємо кнопку OK, підтвердивши, таким чином, зроблені налаштування.

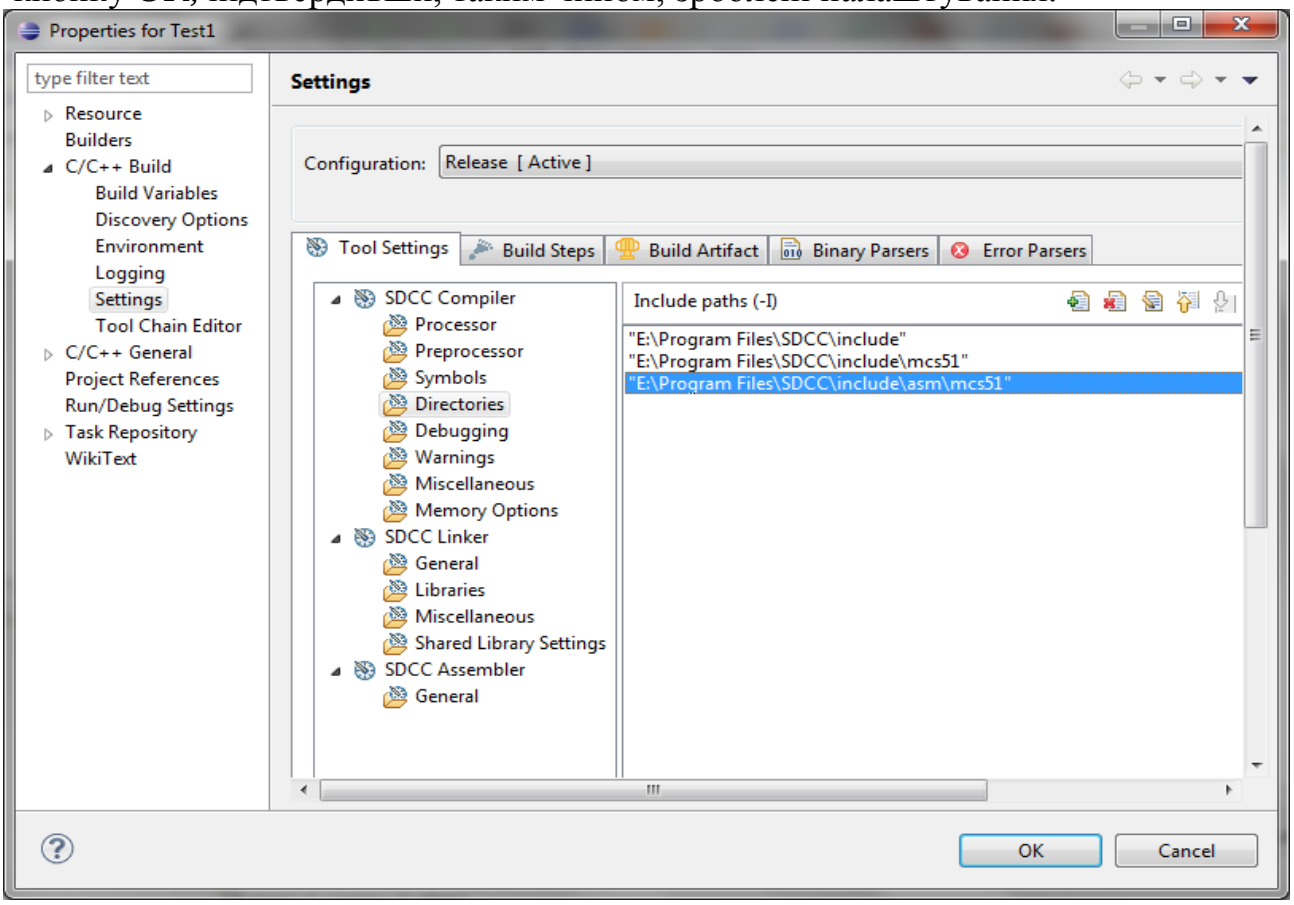

Рисунок 1.10 – Додання шляхів до бібліотечних заголовкових файлів

20.Далі поступово вводимо текст програми. За умовчанням в *IDE Eclipse* установлено **Project>Build Automatically**. Тобто при кожному введенні відбувається автоматична побудова проекту зо всіма плюсами та мінусами такого налаштування. З одного боку, ми одразу бачимо майбутні проблеми. З іншого – це сповільнює процес підготовки тексту, що актуально для ПЕОМ з малим обсягом оперативної пам'яті. Тому цю опцію можна відключити, а запускати побудову проекту вручну.

Введіть рядок визначення регістра, який відповідає за виведення інформації на два лівих знакомісця семисегментного індикатора:

```
volatile __xdata __at (0xA000) unsigned char LEFT_DISPLAY; // left part
```

21.Ліворуч у цьому рядку з'явиться позначка (?), яка вказує на синтаксичну помилку. Проте ця помилка – з точки зору стандартного *ANSI C*. Оскільки ми використовуємо конкретний компілятор для конкретного мік-

роконтролера, треба спиратися на рекомендації [9]. Проглянувши документацію на *SDCC*, дійдемо висновку, що для того, щоб модуль перевірки синтаксису не вважав це за помилку, треба до нашої програми підключити бібліотечний файл **<lint.h>**. Для цього вище за попередній набираємо наступний рядок:

```
#include <lint.h>
```

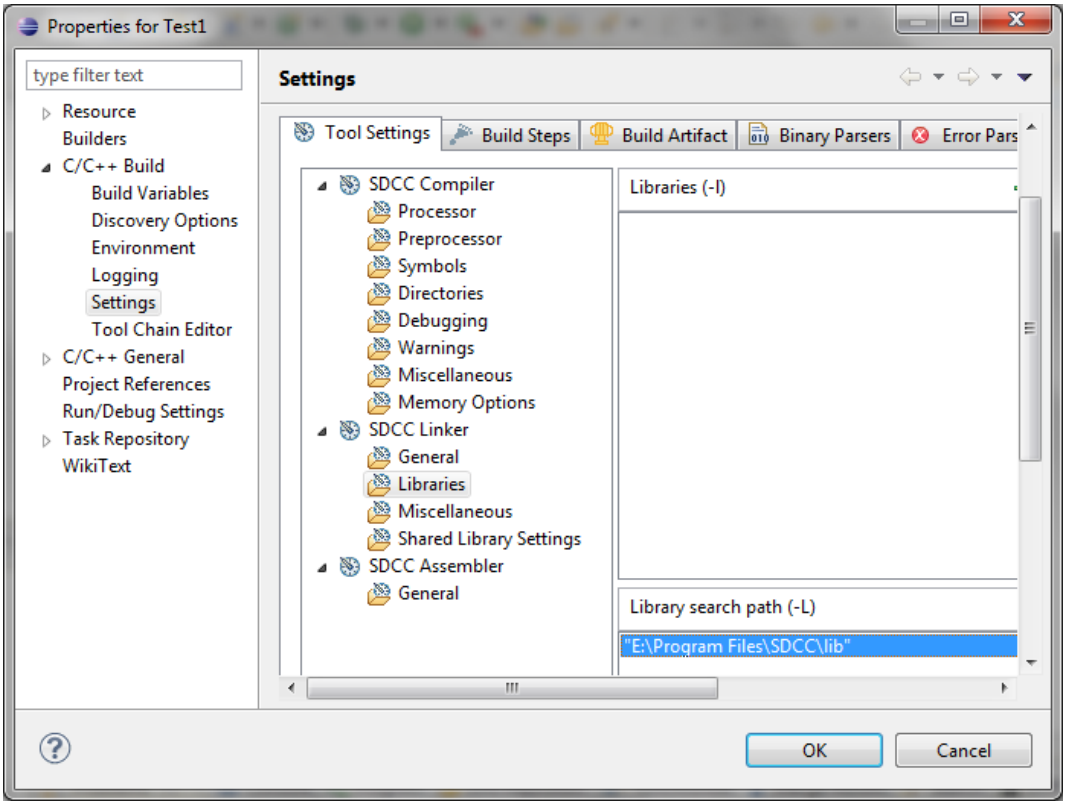

Рисунок 1.11 – Додання шляхів до бібліотек компілятора

Після цього позначка (?) повинна зникнути. Якщо цього не сталося, збережіть файл, натиснувши на кнопку з піктограмою дискети на панелі інструментів.

Введемо інші рядки програми:

```
volatile __xdata __at (0xB000) unsigned char RIGHT_DISPLAY; // right part
volatile __xdata __at (0xA006) unsigned char LED_REG = 0; // leds on board
void main(void)
{while(1)
      {
            LED_REG =0b01010101;
            LEFT_DISPLAY =0x20;
            RIGHT_DISPLAY =0x14;
      }
}
```

22.Виконаємо побудову проекту, вибравши **Project>Build All (Ctrl-B)**. Можливі зауваження (**Warnings**), які могли залишитися від попередніх невдалих спроб, треба просто видалити з вікна **Problems**.

23.Якщо активовано **Project>Build Configurations>Set**

Active>Release, у вікні **Project Explorer** з'явиться папка **Release** з вкладеними файлами та папками (Рисунок 1.12). Ці файли потрібно дослідити та з'ясувати їхнє призначення.

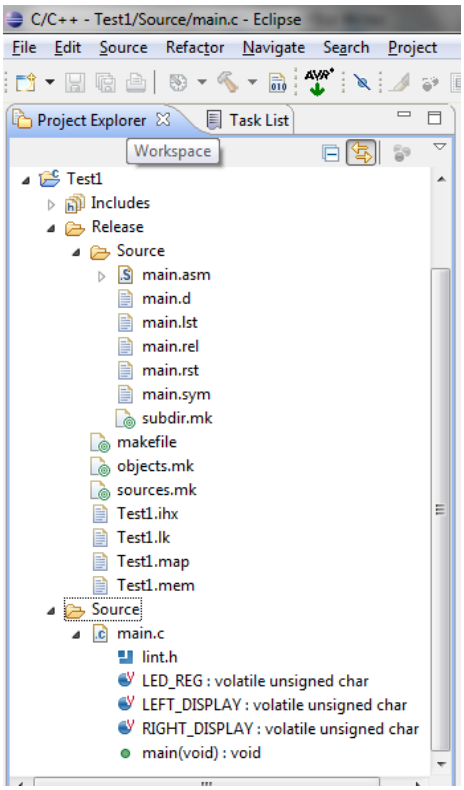

Рисунок 1.12 – Вікно Project Explorer

24.Найбільше зараз нас цікавить файл, який має таке ж ім'я, що й проект і розширення **ihx**. Оскільки в лабораторних ПЕОМ зроблено відповідне налаштування файлової асоціації, подвійне натискання лівою кнопкою миші по цьому файлу призведе до завантаження його безпосередньо до навчально-відлагоджувального пристрою *EV8031/AVR*.

25.Якщо на семисегментному дисплеї *EV8031/AVR* Ви побачите **„2014"**, а на лінійці одиничних світлодіодів – інформаційну модель **01011010**, то Вас можна привітати з першим успішним проектом для мікроконтролера *MCS51* мовою *C*.

1.3 Контрольні питання

1. Навести призначення та склад лабораторного стенду „МПТ".
2. Навести призначення та склад базової плати розширення НВП.
3. Навести розподіл пам'яті НВП *EV8031/AVR*.
4. Які програмні та апаратні ресурси використовуються для завантаження програми користувача в НВП EV8031/AVR?
5. Яка частота задаючого генератора f_Q в стенді? Чому дорівнює тривалість одного такту T_T мікроконтролера?
6. Які основні особливості мови програмування С та її відмінності від мови Асемблер?
7. Що являє собою система команд мікропроцесора ?

8. Для чого використовується файл *lint.h*?
9. Як увімкнути нумерацію рядків на екрані редактора Eclipse?

1.4 Хід роботи

1.4.1 Підготовчі стадії

Поза навчальною лабораторією „Електронні системи” треба зробити наступне:

1. Ознайомитися із стендом "*МПТ*" та особливостями його складових частин.

2. Оформити першу частину звіту з виконання лабораторної роботи, в якій зазначити прізвища студентів бригади та дати письмові відповіді на контрольні питання (п. *1.3*).

1.4.2 Етап розробки програмного забезпечення

1. Виконайте дії, описані в розділі 1.2.

2. Змініть вид інформаційної моделі, що виводиться, користуючись індивідуальним завданнями (1.6, пп. 1 – 9).

3. Відповідно до отриманого варіанту завдання (Таблиця 1.1) розробити схему, вхідний текст та відлагодити програму для *EV8031/AVR*.

1.5 Вимоги до звіту по роботі

Звіт про виконання роботи повинний містити:

1. Титульний аркуш встановленого в університеті зразку.

2. Письмові відповіді на контрольні питання та п. 11 розділу 1.2.

3. Схему програми вирішеної задачі з функціональною схемою підключення застосованих індикаторів та кнопок.

4. Текст програми з докладними коментарями, які відповідають схемі програми.

5. Висновки по роботі, в яких треба зазначити, чи досягнута мета даної лабораторної роботи. Що саме конкретно (перерахувати по пунктах) дозволяє досліджувати лабораторний стенд „МПТ”?

1.6 Орієнтовні варіанти завдань

1. Вивести число „51” на два праві знакомісця дисплея НВП.
2. Вивести число „28” на два ліві знакомісця дисплея НВП.
3. Вивести число „2015” на всі знакомісця дисплея НВП.
4. Вивести цифру „5” на крайнє праве та цифру „2” на крайнє ліве знакомісця дисплея НВП.
5. Вивести цифру „4” на середнє праве та цифру „5” на середнє ліве знакомісця дисплея НВП.
6. Вивести цифру „1” на крайнє праве та цифру „7” на середнє ліве знакомісця дисплея НВП.
7. Увімкнути десяткову крапку на другому знакомісці праворуч.

8. Увімкнути десяткові крапки на двох крайніх знакомісцях.
9. Увімкнути десяткові крапки на двох середніх знакомісцях.

Таблиця 1.2 **– Варіанти завдань для розробки програми**

№ вар.	1 етап	Інформаційна модель	2 етап	
				Натискання SW15 (права)
1	«Вогонь, що біжить» з одним СВД праворуч	●●●☼●●●● →	Натискання SW16 (ліва) – зміна напряму «руху». Відпускання SW16 – повернення напряму «руху». Одночасне натискання SW16 та SW15 – гасіння СВД	Праві ЗМ – інкремент 0,5с 00…FF
2	«Вогонь, що біжить» з одним СВД ліворуч	●●●●●☼●● ←		Ліві ЗМ – інкремент 0,5с 00…FF
3	«Тінь, що біжить» з одним СВД праворуч	☼☼●☼☼☼☼☼ →		Праві ЗМ – декремент 0,5с FF…00
4	«Тінь, що біжить» з одним СВД ліворуч	☼☼☼☼☼☼●☼☼ ←		Ліві ЗМ – декремент 0,5с FF…00
5	«Вогонь, що біжить» з двома СВД праворуч	●●☼☼●●●● →		Праві ЗМ – інкремент 1с 00…99
6	«Вогонь, що біжить» з двома СВД ліворуч	●●●●☼☼●● ←		Ліві ЗМ – інкремент 1с 00…99
7	«Тінь, що біжить» з двома СВД праворуч	☼☼●●☼☼☼☼ →		Праві ЗМ – декремент 1с 99…00
8	«Тінь, що біжить» з двома СВД ліворуч	☼☼☼☼☼☼●●☼☼ ←		Ліві ЗМ – декремент 1с 99…00
9	«Вогонь, що біжить» з двома (трьома) СВД праворуч	☼●●☼●●☼● →		На індикаторі – інкремент 0,2с 0000…9999
10	«Вогонь, що біжить» з двома (трьома) СВД ліворуч	●●☼●●☼●● ←		На індикаторі – інкремент 0,5с 0000…9999
11	«Тінь, що біжить» з двома (трьома) СВД праворуч	☼☼●☼☼●☼☼ →		На індикаторі – декремент 0,2с 9999…0000
12	«Тінь, що біжить» з двома (трьома) СВД ліворуч	●☼☼●☼☼●☼ ←		На індикаторі – декремент 0,5с 9999…0000

2 Лабораторна робота №2. Вивчення системи переривань та таймерів-лічильників у мікроконтролерах MCS-51

Мета роботи: практично дослідити принципи організації переривань та можливості резидентних таймерів-лічильників у мікроконтролерах сімейства MCS-51.

2.1 Особливості системи переривань в MCS-51

Переривання – це спосіб асинхронного обміну між ядром МП та периферійним пристроєм, призначення якого – скорочення простоювання (очікування, холостого ходу) МП. Якщо певне переривання дозволене (розблоковано), і сталася подія, що асоційована з цим перериванням, основна виконуєма програма тимчасово призупиняється, а поточна адреса лічильника команд зберігається у стеку. Далі розпочинається підпрограма обробки переривання, яка обов'язково повинна завершуватися командою RETI, що забезпечує повернення адреси перерваної раніше основної програми з стеку. Таким чином, механізм переривань можна порівняти з асинхронним викликом підпрограми у момент, який визначає джерело запиту. Адреса першої команди обробки переривання (вектор, Рисунок 2.1) визначається конкретним джерелом запиту.

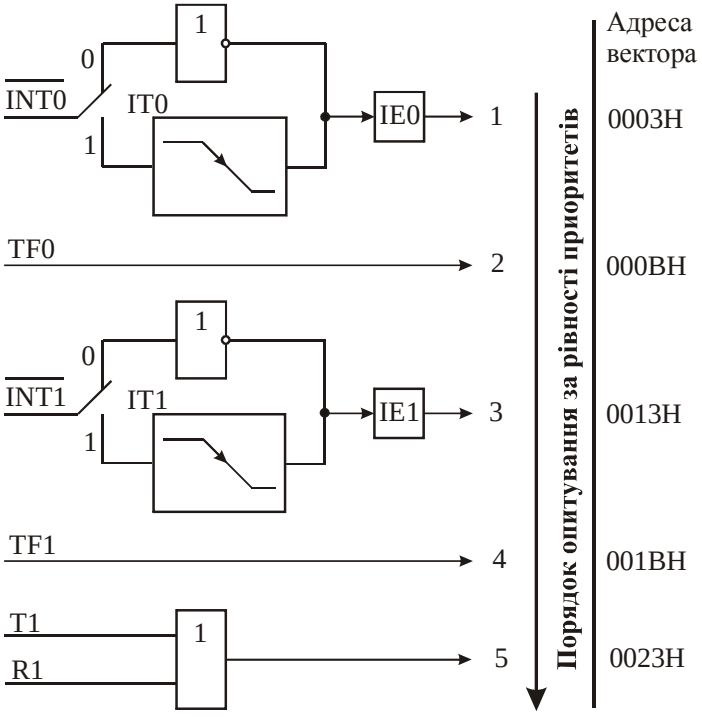

Рисунок 2.1 – Схема переривань в MCS-51

Зовнішні переривання INT0 та INT1 в MCS-51 можуть бути викликані рівнем або переходом сигналу з 1 в 0 на входах МК в залежності від значень керуючих бітів IT0 та IT1 в регістрі спеціальних функцій TCON. Від зовнішніх переривань встановлюються прапори IE0 та IE1 в регістрі TCON, що ініціює виклик відповідної підпрограми обслуговування переривання. Скидання цих прапорів виконується апаратно тільки в тому випа-

дку, коли переривання було викликане по переходу (зрізу) сигналу. Якщо ж переривання викликане рівнем вхідного сигналу, то скиданням прапору IE керує відповідна підпрограма обслуговування переривання шляхом впливу на джерело переривання з метою зняття їм запиту.

Прапори запитів переривань від таймерів TF0 і TF1 скидаються автоматично при передачі керування підпрограмі обслуговування. Прапори запитів переривань від послідовного порту RI та TI встановлюються UART апаратно, проте скидатися повинні програмою. Переривання можуть бути викликані або відмінені програмою, оскільки усі перераховані прапори програмно доступні.

У блоці регістрів спеціальних функцій є два регістри, призначені для керування режимом переривань і рівнями пріоритету. Формати цих регістрів, що мають символічні імена IE и IP, описано в таблиці 2.1 і 2.2.

Таблиця 2.1 – Регістр масок переривань IE

Символ	Розряд	Ім'я та призначення
EA	IE.7	Зняття блокування переривань. Скидається програмно для заборони усіх переривань незалежно від стану IE4-IE0
-	IE.6, 5	Не використовується
ES	IE.4	Біт дозволу переривання від UART. Установка/скидання програмою для дозволу/заборони переривань від TI, RI
ET1	IE.3	Біт дозволу переривання від таймера 1. Установка/скидання програмою для дозволу/заборони переривань від таймера 1
EX1	IE.2	Дозвіл зовнішнього переривання 1. Установка/скидання програмою для дозволу/заборони переривання від INT1
ET0	IE.1	Дозвіл переривання від таймера 0. Працює аналогічно IE.3
EX0	IE.0	Дозвіл зовнішнього переривання 0. Працює аналогічно IE.2

Таблиця 2.2 – Регістр пріоритетів переривань IP

Символ	Розряд	Ім'я та призначення
-	IP.7-5	Не використовується
PS	IP.4	Пріоритет UART. Установка/скидання програмою для призначення перериванню від UART вищого/нижчого пріоритету
PT1	IP.3	Пріоритет таймера 1. Установка/скидання програмою для призначення перериванню від таймера 1 вищого/нижчого пріоритету
PX1	IP.2	Пріоритет зовнішнього переривання 1. Установка/скидання програмою для призначення перериванню INT1 вищого/нижчого пріоритету
PT0	IP.1	Пріоритет таймера 0. Працює аналогічно IP.3
PX0	IP.0	Пріоритет зовнішнього переривання 0. Працює аналогічно IP.2

Можливість програмної установки/скидання будь-якого керуючого біту в цих двох регістрах робить систему переривань виключно гнучкою.

Прапори переривань опитуються в кожному машинному циклі. Ранжирування переривань за пріоритетами виконується протягом наступного

машинного циклу. Система переривань формує апаратно виклик LCALL відповідної підпрограми обслуговування, якщо вона не заблокована однією з умов:

- в даний момент обслуговується запит переривання рівного або більш високого рівня пріоритету;
- поточний машинний цикл – не останній в циклі команди, що виконується;
- виконується команда RETI або будь-яка команда, пов'язана зі звертанням до регістрів IE або IP.

Примітка. *Якщо прапор переривання був встановлений, але за жодною з перерахованих умов не отримав обслуговування та до моменту закінчення блокування вже був скинутий, запит переривання губиться.*

За апаратно сформованим кодом команди LCALL система переривань розміщує у стеку вміст програмного лічильника PC і завантажує в PC адресу вектора переривання відповідної підпрограми обслуговування. За цією адресою повинна бути розміщена команда безумовного переходу JMP до початкової адреси підпрограми обслуговування переривання. Ця підпрограма у разі необхідності повинна починатися командами запису в стек PUSH слова стану програми PSW, акумулятора A, розширювача акумулятора B, покажчика даних DPTR тощо та закінчуватися командами повернення з стеку POP. Підпрограми обслуговування переривання обов'язково завершуються командою RETI, за якою в програмний лічильник перезавантажується з стеку збережена адреса повернення в основну програму. Команда RET також повертає керування, але при цьому не знімає блокування переривання.

2.2 Особливості таймерів/лічильників в MCS-51

В складі МК MCS-51 є регістрові пари з символічними іменами TH0, TL0 и TH1, TL1, на основі яких функціонують два незалежних програмно-керуємих 16-бітних таймера/лічильника подій (T/C0 и T/C1). Під час роботи в якості таймера вміст T/C інкрементується в кожному машинному циклі, тобто через кожні 12 періодів резонатора. При роботі в якості лічильника вміст T/C інкрементується під впливом переходу з 1 в 0 зовнішнього вхідного сигналу, який подається на відповідний (T0, T1) вхід МК. Опитування сигналів виконується в кожному машинному циклі. Оскільки на розпізнання переходу потрібно два машинних цикли, максимальна частота підрахунку вхідних сигналів дорівнює 1/24 частоти резонатора. На тривалість періоду вхідних сигналів обмежень згори нема. Для гарантованого прочитання вхідного зчитуваного сигналу він повинен утримувати значення 1 як мінімум протягом одного машинного циклу.

Для керування режимами роботи та для організації взаємодії таймерів із системою переривань використовуються два регістри спеціальних функцій TMOD і TCON, опис яких наведений в таблицях 2.3 та 2.4. Для обох T/C режими роботи 0, 1 и 2 однакові. Режими 3 для T/C0 и T/C1 відмінні.

Режим 0. Переведення будь-якого Т/С в цей режим робить його 8-розрядним таймером, до входу якого підключений 5-бітний переддільник частоти на 32. В цьому режимі таймерний регістр має розрядність 13 біт. При переході із стану "всі одиниці" в стан "всі нулі" встановлюється прапор переривання від таймера TF1. Вхідний синхросигнал таймера 1 дозволений (поступає на вхід Т/С), коли керуючий біт TR1 встановлений в 1 і або керуючий біт GATE (блокування) дорівнює 0, або на зовнішній вхід запиту переривання INT1 поступає рівень 1.

Таблиця 2.3 – Регістр режиму роботи таймера/лічильника TMOD

Символ	Розряд	Ім'я та призначення
GATE	TMOD.7 для Т/С1 TMOD.3 для Т/С0	Керування блокуванням. Якщо біт встановлений, таймер/лічильник "х" дозволений доти, доки на вході "INT x" високий рівень і біт керування "TRx" встановлений. Якщо біт в скинутий, Т/С дозволений одразу після встановлення біту керування "TRx"
C/T	TMOD.6 для Т/С1 TMOD.2 для Т/С0	Біт вибору режиму таймера або лічильника подій. Якщо біт скинутий, працює таймер від внутрішнього джерела сигналів синхронізації. Якщо біт встановлений, працює лічильник зовнішніх сигналів на вході "Tx"
M1	TMOD.5 для Т/С1 TMOD.1 для Т/С0	Режим роботи (див. таблицю 4.4)
M0	TMOD.4 для Т/С1 TMOD.0 для Т/С0	

Таблиця 2.4 – Режими роботи таймерів/лічильників

M1	M0	Режим роботи
0	0	"TLx" працює як 5-бітний переддільник
0	1	16-бітний таймер/лічильник. "THx" и "TLx" включені послідовно
1	0	8-бітний автоперезавантажуваний таймер/лічильник. "THx" зберігає значення, що має бути завантажено в "TLx" кожного разу за переповненням
1	1	Таймер/лічильник 1 зупиняється. Таймер/лічильник 0: TL0 працює як 8-бітний таймер/лічильник, а його режим визначають керуючі біти таймера 0. TH0 працює тільки як 8-бітний таймер, а його режим визначають керуючі біти таймера 1

Встановлення біта GATE в 1 дозволяє використовувати таймер для вимірювання тривалості імпульсного сигналу, що подається на вхід запиту переривання.

Режим 1. Робота будь-якого Т/С в цьому режимі така ж, як і в режимі 0, за виключенням того, що таймерний регістр має розрядність 16 біт.

Режим 2. В цьому режимі роботу організовано таким чином, що переповнення (перехід із стану "всі одиниці" в стан "всі нулі") 8-бітного лічильника TL1 призводить не тільки до встановлення прапору TF1, але й

автоматично перезавантажує в TL1 вміст старшого байту (TH1) таймерного регістра, який попередньо був заданий програмним шляхом. Перезавантаження залишає вміст TH1 незмінним. В режимі 2 T/C0 и T/C1 працюють абсолютно однаково.

Режим 3. В цьому режимі T/C0 и T/C1 працюють по-різному. T/C1 зберігає незмінним свій поточний вміст. Іншими словами, ефект такий же, як і при скиданні керуючого біта TR1 в нуль. В цьому режимі TL0 и TH0 функціонують як два незалежних 8-бітних лічильника. Роботу TL0 визначають керуючі біти T/C0 (C/T, GATE, TR0), вхідний сигнал INT0 та прапор переповнення TF0. Роботу TH0, який може виконувати тільки функції таймера (підрахунок машинних циклів МК), визначаю біт керування TR1. При цьому TH0 використовує прапор переповнення TF1.

Таблиця 2.5 – Регістр керування/статусу таймера TCON

Символ	Розряд	Ім'я та призначення
TF1	TCON.7	Прапор переповнення таймера 1. Встановлюється апаратно при переповненні таймера/лічильника. Скидається при обслуговуванні переривання апаратно
TR1	TCON.6	Біт керування таймера 1. Встановлюється/скидається програмою для пуску/зупинки таймера/лічильника
TF0	TCON.5	Прапор переповнення таймера 0. Встановлюється апаратно. Скидається при обслуговуванні переривання
TR0	TCON.4	Біт керування таймера 0. Встановлюється/скидається програмою для пуску/зупинки таймера/лічильника
IE1	TCON.3	Прапор фронту переривання 1. Встановлюється апаратно, коли детектується зріз зовнішнього сигналу INT1. Скидається при обслуговуванні переривання
IT1	TCON.2	Біт керування типом переривання 1. Встановлюється/ скидається програмно для специфікації запиту INT1 (зріз/низький рівень)
IE0	TCON.1	Прапор фронту переривання 0. Встановлюється за зрізом сигналу INT0. Скидається при обслуговуванні переривання
IT0	TCON.0	Біт керування типом переривання 0. Встановлюється/ скидається програмно для специфікації запиту INT0 (зріз/низький рівень)

Режим 3 використовується у випадках, коли потрібна наявність додаткового 8-бітного таймера або лічильника подій. Можна вважати, що в режимі 3 МК має в своєму складі три таймери/лічильники. В цьому випадку, якщо T/C0 використовується в режимі 3, T/C1 може бути або увімкнений, або вимкнений, або переведений у свій власний режим 3, або може бути використаний послідовним портом в якості генератора частоти передачі, або, врешті-решт, може бути використаний в будь-якому застосуванні, де не потрібне переривання.

2.3 Формування часової затримки таймером

Недолік програмного способу реалізації часової затримки – це нераціональне використання ресурсів МК: під час формування затримки МП-ядро практично простоює та не може вирішувати ніяких задач формування закону керування об‘єктом. В той же час резидентні апаратні засоби дозволяють реалізувати часові затримки на фоні основної програми роботи.

На вхід таймера/лічильника (Т/С) можуть поступати сигнали синхронізації з частотою $f_Q/12$ (Т/С в режимі таймера) або сигнали від зовнішнього джерела (Т/С в режимі лічильника). Обидва ці режими можна використати для формування затримок. Якщо використати Т/С в режимі таймера повного формату (*16 біт*), можна отримати затримки в діапазоні (1…65536) T_T.

2.4 Особливості програмування периферійних пристроїв та застосування переривань в SDCC

Будь яка програма для МК керує не тільки програмними об'єктами (даними), а й апаратними ресурсами МК (порти введення-виведення, переривання, таймери-лічильники, послідовний інтерфейс тощо). Перш, ніж використовувати будь-які апаратні ресурси, їх необхідно налаштувати на відповідний режим роботи. Первинна процедура налаштування апаратних засобів називається *ініціалізацією*. Ініціалізація виконується шляхом запису відповідних кодів у регістри спеціальних функцій МК. Однією з цілей даної роботи є набуття навичок по роботі з регістрами МК в разі програмування мовою С. Перш, ніж перейти до написання програми, виконаємо наступні підготовчі дії:

1. Запустимо **Eclipse** та відкриємо попередній проект (**Test1**). У вікні **Project Explorer** встановимо курсор на папку **Test1**, натиснемо праву кнопку миші та у контекстному меню оберемо пункт **Copy** (те ж саме можна зробити за допомогою сполучення клавіш **Ctrl+C**).

2. Натиснемо на порожньому місці у вікні **Project Explorer**, а потім натиснемо сполучення клавіш **Ctrl+V** (або команду **Paste** з контекстного меню)**,** після чого з'явиться діалогове вікно (Рисунок 2.2).

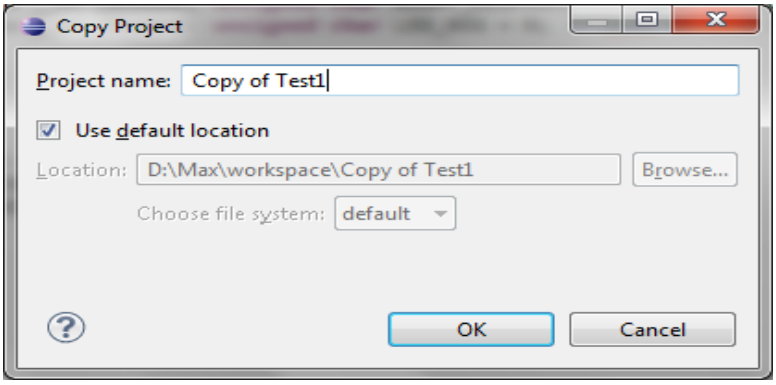

Рисунок 2.2 – Діалогове вікно копіювання проекту

3. Змінимо вміст поля **Project name** на **Interrupts_and_SFR** та

натиснемо **ОК.** Отримаємо новий проект, який буде містити усі налаштування, файли і папки точнісінько такі, як і у попередньому проекті-шаблоні (Рисунок 2.3). Якщо ви забули перейменувати проект, а залишили ім'я за умовчанням, це завжди можна зробити пізніше. Для цього треба обрати проект лівою кнопкою миші та натиснути **F2** (або обрати команду **Rename** з контекстного меню).

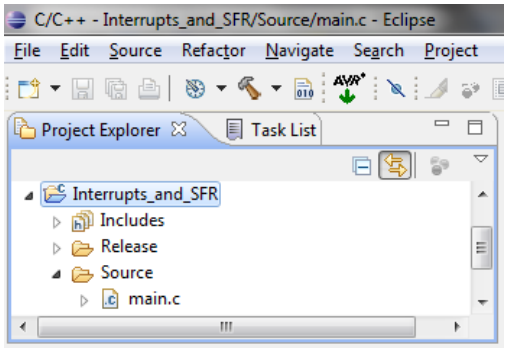

Рисунок 2.3 – Вміст нового проекту

Тепер перейдемо до формування структури проекту. Для того, щоб підтримувати порядок, а файли з вихідними текстами не утворювали сміт-ник, структура папок для зберігання файлів має відповідати їх призначен-ню або змісту. Для цього у папці з назвою **Source** утворимо нову папку з назвою **inc. У** цій папці будемо зберігати усі наші **header**-файли (файли за-головків із розширенням ***.h**). У папці **Source** будемо зберігати файли з ви-хідними текстами мовою С (файли з розширенням ***.c**).

4. Оберемо папку **Source** та з контекстного меню введемо команду **New>Folder**. У діалоговому вікні, що з'явилося в рядку **Folder name** вводимо ім'я папки **inc** (Рисунок 2.4) та натискаємо кнопку **Finish**.

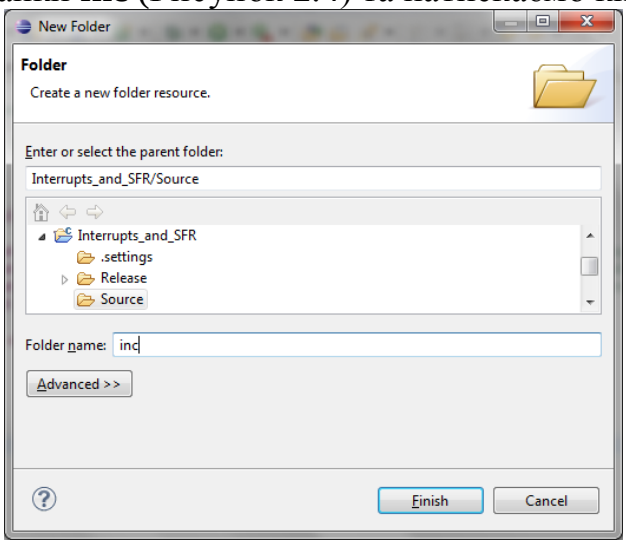

Рисунок 2.4 – Діалогове вікно нової папки

5. Оскільки у папці **inc** будуть зберігатися **header**-файли, у налаштуваннях проекту (**ALT-Enter**, знаходячись на папці проекту) треба вказати путь до цієї папки. Як це зробити, описано у пункті 1.2.19

попередньої роботи. Проте цього разу, замість абсолютних шляхів, будемо використовувати відносні. Оберемо кнопку **Workspace...** у діалоговому вікні **Add directory path** та вкажемо папку **inc** (Рисунок 2.5).

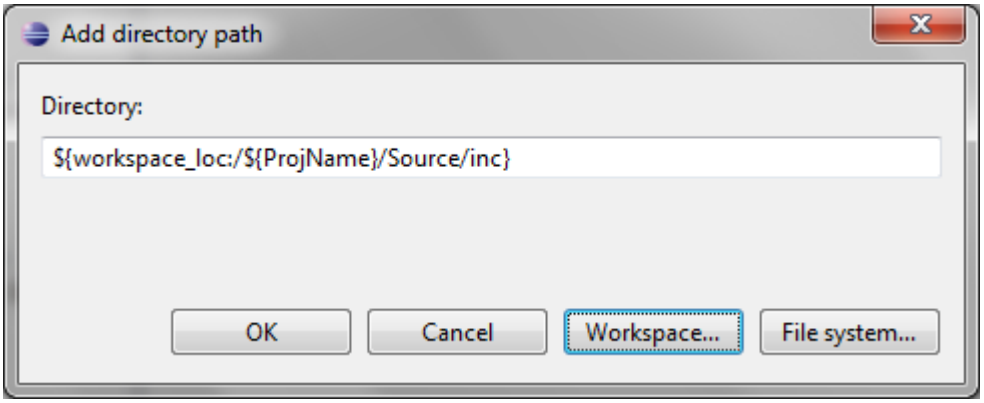

Рисунок 2.5 – Діалогове вікно додавання путі до папки

6. Створимо дуже важливий **header**-файл, який буде описувати адреси пристроїв **НВП EV8031**. Для цього у вікні **Project Explorer** обираємо папку **inc,** викликаємо контекстне меню, з якого обираємо **New>Header File**. У рядку **Header file** вводимо ім'я **EV8031.h** (рисунок 10) та натискаємо кнопку **Finish.**

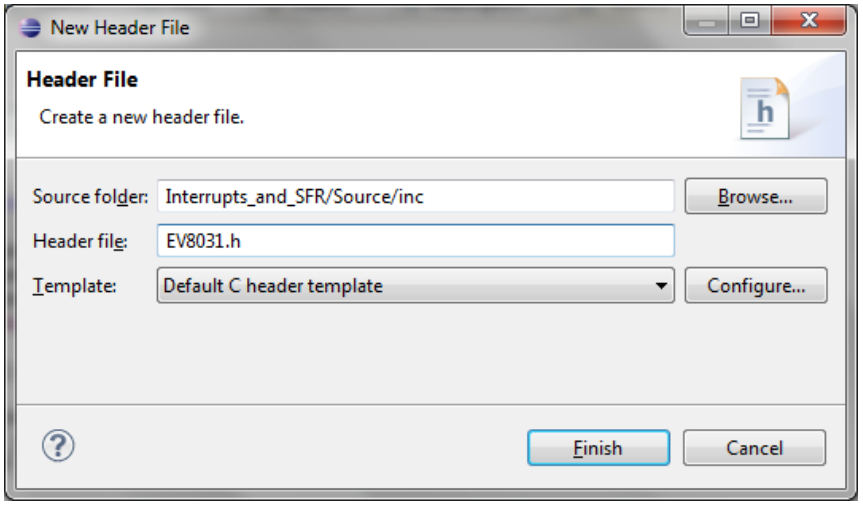

Рисунок 2.6 – Діалогове вікно створення нового файлу заголовка

7. Відкриємо щойно створений файл для редагування. Як можна побачити, новий файл не порожній, а містить певну конструкцію з директив препроцесора С. Вони унеможливлюють багатократне підключення цього файлу і запобігають помилкам повторного визначення об'єктів. Запишемо наступний текст у цьому файлі в середині конструкції з директив препроцесора С:

```
#include <lint.h>

volatile __xdata __at (0xA000) unsigned char LEFT_DISPLAY; // left part
volatile __xdata __at (0xB000) unsigned char RIGHT_DISPLAY; // right part
volatile __xdata __at (0xA006) unsigned char LED_REG = 0; // leds on board
```

8. Зберігаємо файл (сполучення клавіш **Ctrl+S**). Зазначені вище рядки, окрім самого верхнього, треба видалити з файлу **main.c**. Підключимо наш **header** до файлу **main.c**, для чого наберемо рядок:

```
#include "EV8031.h"
```

Довідка.
Під час набору тексту в **Eclipse** можна користуватися такою зручною функцією як автодоповнення, що викликається комбінацією клавіш **Ctrl+Space** (Рисунок 2.7).

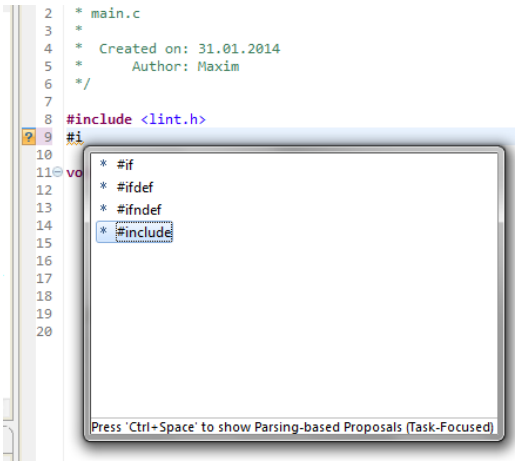

Рисунок 2.7 – Автодоповнення

Врешті-решт файли **EV8031.h** та **main.c** повинні виглядати наступним чином (Рисунок 2.8):

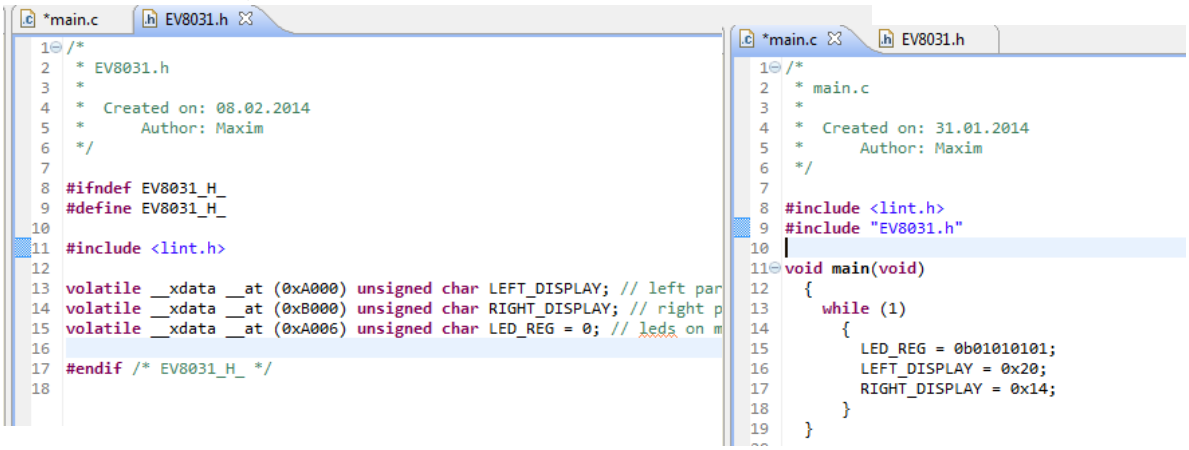

Рисунок 2.8 – Початок файлу опису адрес пристроїв **НВП** EV8031.h

На вкладці з файлом **main.c** перед ім'ям файлу зображена зірочка. Це означає, що ми внесли правки і не зберегли файл. Тому, якщо ми зараз спробуємо виконати команду **Build,** в побудові проекту буде задіяний файл **main.c** без урахування останніх правок. Це зовсім не те, на що ми очікуємо. Отже, перш ніж виконати побудову проекту, треба зберегти усі файли, які були змінені. Цей процес можна автоматизувати, якщо у меню **Window>Preferences>Workspace** встановити галочку напроти пункту

Save automatically before build (Рисунок 2.9)

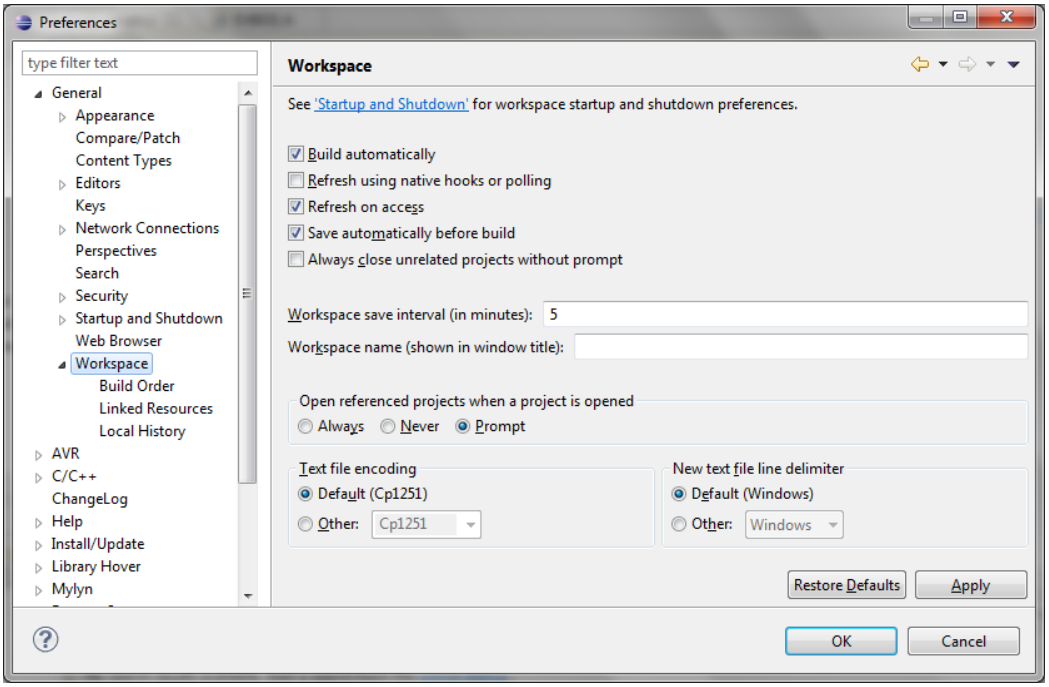

Рисунок 2.9 – Автозбереження файлу перед компіляцією

9. Побудуємо проект. Якщо все зроблено вірно, компіляція і збірка проекту повинні пройти без помилок.

10. Щоб почати працювати з регістрами МК, у програмі на С необхідно підключити **header** з визначенням імен усіх регістрів і бітів МК. Такі файли для різних МК є у бібліотеці компілятора **SDCC.** У **НВП EV8031** встановлено МК фірми **ATMEL AT89S52**. Отже, до тих вихідних файлів, де будемо використовувати імена регістрів МК, необхідно підключати файл **at89x52.h** (у нашому випадку це файл **main.c**):

```
#include <at89x52.h>
```

Слід зазначити, що файл **at89x52.h** був попередньо модифікований: усі адреси регістрів у ньому були взяті в дужки, що необхідно для коректної роботи директив з файлу **lint.h.** Якщо доведеться працювати з іншим МК, треба буде підключати інший **header** файл з бібліотеки **SDCC**. Не забудьте попередньо відредагувати його відповідно до зазначеного вище правила.

11. Розробимо програму, що буде блимати усіма світлодіодами з частотою 1 Гц. Для реалізації затримки використаємо таймер-лічильник 0 та переривання. Перше, що ми зробимо – це виконаємо ініціалізацію таймера-лічильника, переривань та регістра світлодіодів. Для цього наберемо наступний текст на початку функції **main()**:

```
// Ініціалізація світлодіодів
LED_REG = 0b11111111;
// Ініціалізація таймера
TMOD = 0b00000010;
TL0 = 0;
TH0 = 0;
```

```
// Ініціалізація переривань
IE = 0b10000010;
// Запуск таймера 0
TR0 = 1;
```

Далі видалимо усе зайве, що залишилось у файлі від попереднього проекту, який був використаний у якості шаблону при створенні даного проекту. Тепер файл **main.c** має виглядати наступним чином (Рисунок 2.10):

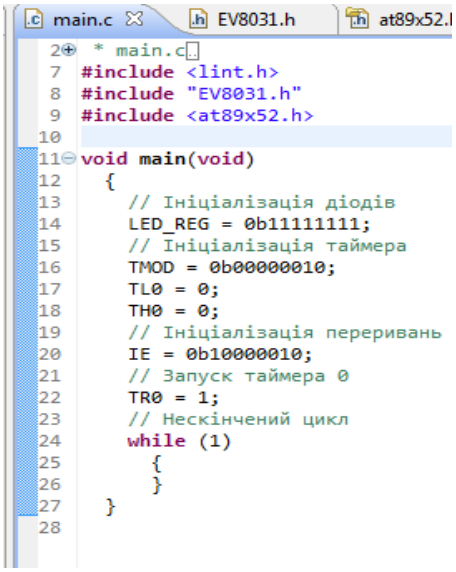

Рисунок 2.10 – Файл main.c з ініціалізацією периферії

12. Напишемо функцію обробки переривання від таймера-лічильника 0. Для цього утворимо два нових файли. Один – у папці **Source –** це файл з вихідним текстом підпрограми обробки переривання. Дамо йому назву **interrupts.c** Другий файл створимо у папці **inc**. Це буде файл з прототипом функції переривання. Назвемо його **interrupts.h**.

13. У файлі **interrupts.c** введемо наступні рядки:

```
#include <at89x52.h>
#include <stdint.h>
#include "EV8031.h"

void Timer0Interrupt (void) __interrupt(TF0_VECTOR)
	{
		static uint16_t i = 1200;
		static uint8_t leds = 0b11111111;
		if (--i == 0)
			{
				leds= ~leds;
				LED_REG = leds;
				i = 1200;
			}
	}
```

Зверніть увагу: для того, щоб вказати компілятору, що функція, яку ми створили, буде функцією обробки переривання, використано ключове слово **__interrupt** з параметром **TF0_VECTOR**. Він задає номер вектора переривання (макрос **TF0_VECTOR** визначений у файлі **at89x52.h**, як і усі

інші вектори переривань для даного МК).

Довідка.

Щоб подивитись, де і як задана будь-яка змінна, функція або макрос, необхідно перемістити курсор на ім'я цього об'єкту у тексті програми та натиснути клавішу **F3**. Курсор переміститься до місця у файлі, де визначено цей об'єкт. Якщо об'єкт визначено у іншому файлі, цей файл буде автоматично відкрито у редакторі.

14. У файлі **interrupts.h** введемо наступні рядки:

```
#include <lint.h>
#include <at89x52.h>

void Timer0Interrupt (void) __interrupt(TF0_VECTOR);
```

У файлі **main.c** додамо наступний рядок:

```
#include "interrupts.h"
```

Після цього файли **main.c, interrupts.h, interrupts.c** матимуть вигляд (Рисунок 2.11):

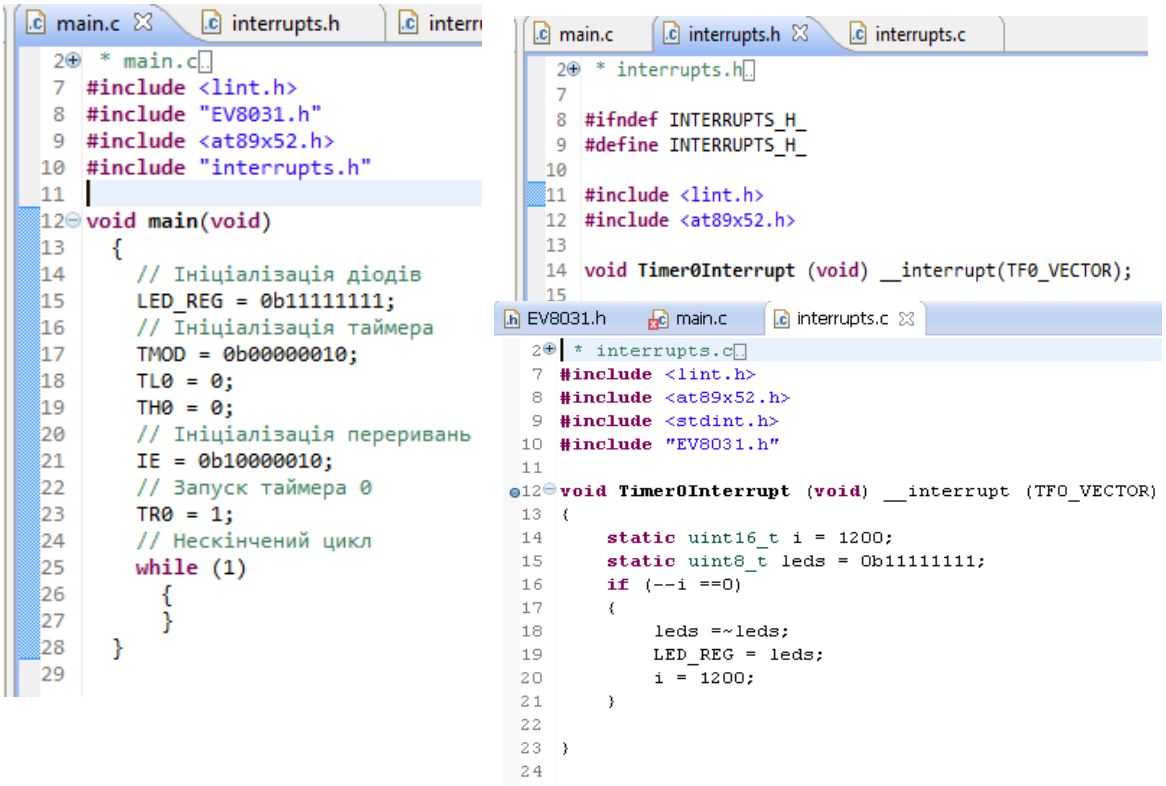

Рисунок 2.11 – Файли проекту main.c, interrupts.h, interrupts.c

15.Виконаємо побудову проекту та завантажимо отриманий файл з розширенням **ihx** у **НВП.** Якщо усі 8 світлодіодів стенду будуть вмикатися і вимикатися з частотою 1 Гц, Вас можна привітати з успішним опануванням методів роботи з регістрами та перериваннями мікроконтролера MCS51 засобами мови С.

2.5 Контрольні питання

1. Нарисуйте схему переривань в MCS-51. Перерахуйте та схарактеризуйте типи переривань.
2. Для чого потрібні регістри масок та пріоритетів переривань?
3. Як перевести МК в режим зниженого енергоспоживання?
4. Схарактеризуйте режими роботи таймера/лічильника в MCS-51.
5. Як за допомогою таймера з використанням та без переривання згенерувати затримку?
6. Як за допомогою таймера можна виміряти тривалість імпульсу?

2.6 Хід роботи

2.6.1 Частина перша

1. Вивчіть особливості системи переривань MCS-51: номенклатура, узагальнена структура, функціональні можливості та застосування [5].
2. Оформіть першу частину звіту з виконання лабораторної роботи, в якій дайте письмові відповіді на контрольні питання 1…3 (п. 2.5).
3. Виконайте дії підрозділу 2.4.
4. Безпосередньо у викладача отримайте допуск до виконання роботи в лабораторії та варіант завдання.
5. Перевірте програму на стенді і продемонструйте її викладачеві.

2.6.2 Частина друга

1. Вивчіть особливості таймерів-лічильників MCS-51: номенклатура, узагальнена структура, функціональні можливості, застосування [5].
2. Оформіть другу частину звіту з виконання лабораторної роботи, в якій дайте письмові відповіді на контрольні питання 4…6 (п. 2.5).
3. Безпосередньо у викладача отримайте допуск до виконання роботи в лабораторії та варіант завдання.
4. Розробіть алгоритм, складіть програму (згідно з варіантом), перевірте її роботу програми на стенді і продемонструйте викладачеві.

2.7 Вимоги до звіту по роботі

Звіт повинен містити функціональну схему підключення застосованих апаратних засобів до МК з вказівкою конкретних виводів, завдання, алгоритми і коментовані тексти розроблених програм. Необхідно також навести опис функцій та їхніх параметрів, блоків даних. Зробити висновки щодо досягнутої мети роботи.

2.8 Орієнтовні варіанти завдань

1. Варіанти завдань повністю відповідають таблиці 1.1, але обробка натискань на кнопки здійснюється за допомогою переривань.
2. Варіанти завдань відповідають попередньому пункту, але часові затримки реалізовані за допомогою переривань від таймерів-лічильників.

3 Лабораторна робота №3. Формування та обробка функцій часу в МПС на базі MCS-51

Мета роботи: навчитися формувати дискретні електричні сигнали та вимірювати їхні часові параметри за допомогою резидентних засобів мікроконтролерів сімейства MCS-51.

3.1 Короткі відомості про вимірювання часових параметрів

В системах автоматичного керування часто доводиться вимірювати такі величини, як частота f, період T, тривалість τ, зсув фаз φ. Для цього застосовують перетворення частота-код. В залежності від того, який саме параметр потрібно виміряти, використовують різні підходи.

При вимірюванні *частоти* f_x підраховують кількість імпульсів вхідного сигналу N_x протягом *фіксованого* проміжку часу T_0 (Рисунок 3.1).

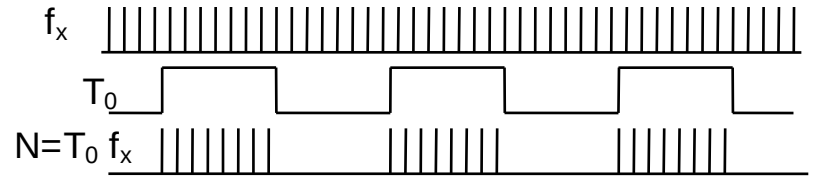

Рисунок 3.1 – Вимірювання частоти підрахунком імпульсів за фіксований час

Цей метод можна застосовувати для вимірювання високих частот (більше *100 Гц*). Верхня межа обмежена швидкодією елементів схеми та розрядністю лічильників.

Інший метод заснований на вимірюванні *періоду* T_x повторення шляхом підрахунку кількості імпульсів *відомої* частоти f_0 (Рисунок 3.2).

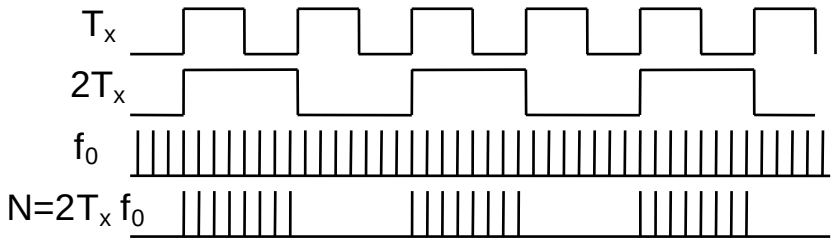

Рисунок 3.2 – Вимірювання періоду підрахунком імпульсів відомої частоти

Зазвичай виконують підрахунок імпульсів фіксованої частоти f_0 за інтервал, що дорівнює або кратний періоду, що вимірюється.

Таким же чином вимірюють *тривалість імпульсу* τ_x. Відмінність полягає лише в тому, що при вимірюванні тривалості імпульсу немає потреби ділити частоту на два.

При вимірюванні *зсуву фаз* часовий інтервал τ_φ формують шляхом кон‘юнкції сигналів f_{1x} та f_{2x}. Отриманий інтервал часу вимірюють мето-

дом, що описаний вище (Рисунок 3.3).

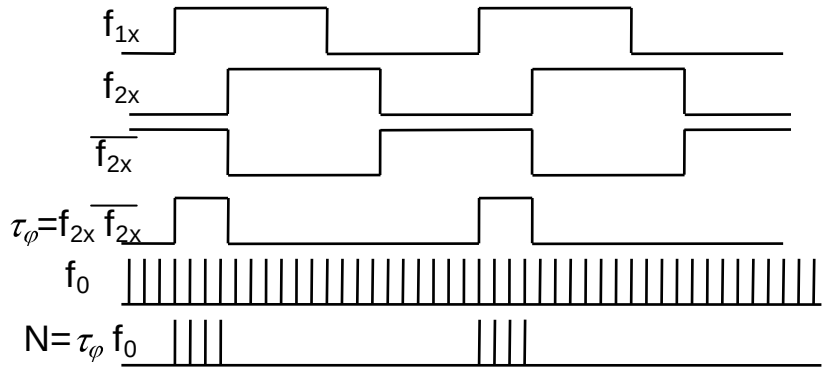

Рисунок 3.3 – Вимірювання зсуву фаз підрахунком імпульсів відомої частоти

В лабораторному стенді „МПТ” за наявності плати розширення до входу МК *T0* можна підключити генератор фіксованої частоти (ГФЧ), а до входу *T1* – генератор з частотою, яку можна змінювати (ГЗЧ, Рисунок 3.4).

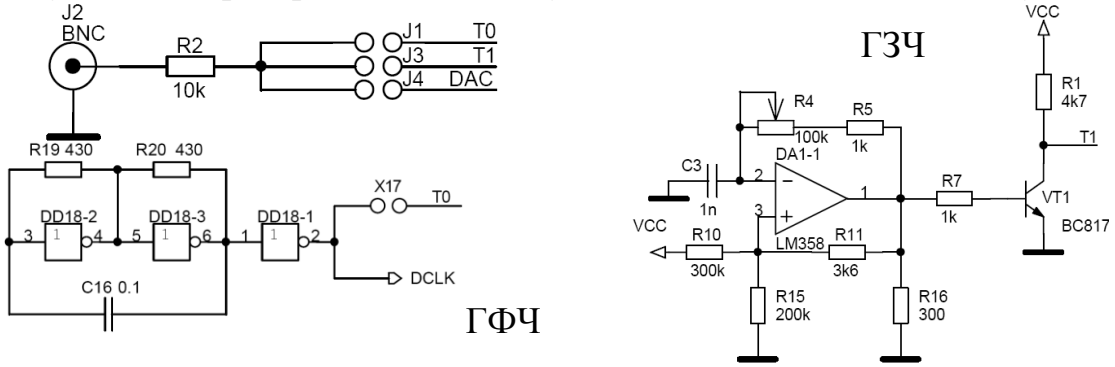

Рисунок 3.4 – Схемотехніка генераторів у стенді «МПТ»

Сигнали цих генераторів можна спостерігати на осцилографі, підключеному до роз‘єднувача BNC. Для спостереження сигналу на виводі МК *T0* потрібно замкнути перемичку *J1*, а для *T1* – перемичку *J3*.

Резидентні таймери/лічильники МК призначені для підрахунку зовнішніх подій, для реалізації програмно-керуємих часових затримок, виконання функцій завдання часу МК. Опис цих блоків наведено в розділі 3 даних методичних вказівок.

У Прикладі 3.1 показано, яким чином можна виміряти частоту одного з генераторів та відобразити результат на дисплеї.

Приклад 3.1 **– Вимірювання та відображення частоти в кГц**

```
; /*
 * main.c
 * Вимірювання частоти в стенді EV8031/AVR
*/
#include <lint.h>
#include "EV8031.h"
#include <at89x52.h>
#include "interrupts.h"
// Прототип функції переведення у двійково-десятковий код у форматі: XX.XX (кГц)
```

```
uint16_t uint_to_number (uint16_t number);

void main(void)
	{
		uint16_t nx = 0, freq = 0;
		// Ініціалізація індикатора
		DC_REG = 0b00100001;
		LEFT_DISPLAY = 0;
		RIGHT_DISPLAY = 0;
		// Ініціалізація таймерів
		TMOD = 0b00100101;
		TL0 = 0;
		TH0 = 0;
		TL1 = 0;
		TH1 = 0;
		// Ініціалізація переривань
		IE = 0b10001000;
		// Запуск таймерів
		TR0 = 1;
		TR1 = 1;
		// Нескінчений цикл
		while (1)
			{
				if (nx != Nx) // Якщо нове значення відрізняється, то:
				{
				nx = Nx;  // присвоїти поточну кількість імпульсів;
				freq = uint_to_number (nx); // перевести виміряне зна-
чення частоти у двійково-десятковий код;
				DC_REG = (freq >> 12) ? 0b00100000 : 0b00100001; // якщо
перша цифра дорівнює 0, встановити крапку, погасити перше знакомісце;
				LEFT_DISPLAY = freq>>8; /вивести дробну частину частоти;
				RIGHT_DISPLAY = freq;   // вивести цілу частину частоти;
				}
			}
	}

uint16_t uint_to_number (uint16_t number)
	{
		uint16_t result = 0;
		result |= (number / 10000) << 12;
		number %= 10000;
		result |= (number / 1000) << 8;
		number %= 1000;
		result |= (number / 100) << 4;
		number %= 100;
		result |= (number / 10);
		return (result);
	}

 /* Заголовковий файл переривань
 * interrupts.h
*/

#ifndef INTERRUPTS_H_
#define INTERRUPTS_H_

#include <lint.h>
#include <at89x52.h>
#include <stdint.h>

extern uint16_t Nx;
```

```
void Timer1Interrupt (void) __interrupt(TF1_VECTOR);

#endif /* INTERRUPTS_H_ */


 /* Файл з функцією обслуговування переривань
 * interrupts.c
*/
#include <at89x52.h>
#include <stdint.h>
#include "interrupts.h"

uint16_t Nx = 0;

void Timer1Interrupt(void) __interrupt(TF1_VECTOR)
      {
      static uint16_t i = 2400;
      if (--i == 0)
            {
            TR0 = 0; // Зупинити обидва таймери.
            TR1 = 0;
            Nx = ((uint16_t)TH0 << 8) | (uint16_t)TL0; // Зберегти отриману кіль-
кість імпульсів.
            TL0 = 0; // Обнулити вміст обох таймерів.
            TH0 = 0;
            TL1 = 0;
            i = 2400; // Початкове значення програмного лічильника часу.
            TR0 = 1;  // Запустити обидва таймери.
            TR1 = 1;
            }
      }
```

3.2 Контрольні питання

1. Нарисуйте функціональну схему вимірювання частоти.
2. Опишіть методи частотного і часового перетворення.
3. Визначте параметри частотного перетворення.
4. Схарактеризуйте фактори, що впливають на похибку частотного перетворення.
5. Дайте визначення поняттю «роздільна здатність частотного перетворення».
6. Чим відрізняється вимірювання періоду від вимірювання частоти?
7. Що таке характеристика та нелінійність частотного перетворення?
8. Чим відрізняються апаратні реалізації частотного та часового перетворення від програмно-апаратних реалізацій?
9. Наведіть приклади практичного застосування частотного та часового перетворення.
10. Поясніть призначення окремих рядків Прикладу 3.1.

3.3 Хід роботи

1. Вивчіть особливості вимірювання часових параметрів імпульсних сигналів.

2. Оформіть першу частину звіту з виконання лабораторної роботи, в якій дайте письмові відповіді на контрольні питання 1…9 (п. 3.2).

3. Безпосередньо у викладача отримайте допуск до виконання роботи в лабораторії та варіант завдання.

4. Розробіть алгоритм, складіть програму (згідно з варіантом), перевірте її роботу програми на стенді і продемонструйте викладачеві.

Таблиця 3.2 **– Варіанти завдань для розробки програми**

	Зміст індивідуального завдання
1	Виміряти частоту T0 та відобразити на дисплеї.
2	Виміряти частоту T1 відносно ГФЧ та відобразити на дисплеї.
3	Виміряти частоту T1 відносно тактової частоти процесора та відобразити на дисплеї.
4	Виміряти різницю частот ГФЧ і ГЗЧ та відобразити на дисплеї.
5	З використанням внутрішніх Т/С забезпечити плавне загоряння числа 55 та відобразити на дисплеї.
6	Почергово відображувати 48 на другому – третьому знакомісці дисплея та на першому – четвертому знакомісці з інтервалом 1 с. Часові інтервали формувати за допомогою ГФЧ.
7	Підрахувати та відобразити на дисплеї кількість натискань SW15 за 10 с. Часові інтервали формувати за допомогою Т/С МК.
8	Відобразити на другому знакомісці дисплея числа від 1 до 9 протягом 1 с з паузою між відображеннями 1 с. Часові інтервали формувати за допомогою Т/С МК.
9	При натисканні кнопки SW16 включити секундомір та відобразити на дисплеї. Часові інтервали формувати за допомогою Т/С МК.
10	Відображати числа, що мерехтять на чотирьох розрядах дисплея з різним часом відображення паузи: 0.25, 0.5, 0.75, 1 с. Часові інтервали формувати за допомогою Т/С МК.
11	Підрахувати кількість натискань на кнопку SW16 за 5 с. Часові інтервали формувати за допомогою Т/С МК.
12	Виміряти час між натисканнями кнопок SW15 и SW16. Час в секундах відобразити на дисплеї.
13	При натисканні на кнопку SW15 відобразити в секундах час її натискання на дисплеї.
14	Виміряти період T0 та відобразити на дисплеї.
15	Виміряти період T1 та відобразити на дисплеї.

4 Лабораторна робота №4. Дослідження методів побудови генераторів сигналів на MCS-51

Мета роботи: навчитися формувати різноманітні електричні сигнали за допомогою засобів мікропроцесорної техніки.

4.1 Теоретичні відомості про методи побудови генераторів сигналів на мікроконтролері

В сейсмології, медицині, автоматиці, зв'язку та інших областях техніки застосовуються періодичні сигнали різної форми з періодом повторення від десятків до сотень мілісекунд та більше. Формування таких сигналів за допомогою аналогових та цифрових мікросхем з жорсткою логікою досить складне, оскільки кожна форма сигналу потребує свого схемного рішення.

Від цього недоліку вільні генератори на мікропроцесорах (ГМП). Розглянемо принцип роботи такого генератора. Апаратно його можна розділити на 2 частини: цифрову й аналогову. Завданням цифрової частини є генерація періодичного сигналу в цифровій формі, тоді як аналогова частина перетворює цифрові коди в сигнал необхідної форми.

Отримання періодичного сигналу в цифровій формі зводиться до обчислення його значень в кількох точках періоду T і циклічному повторенні цих обчислень. Для довільного сигналу $U=f(t)$ необхідно обчислити значення U_i при зміні аргументу t в проміжку $0...T$ з кроком дискретизації dT. Кількість відліків K на період сигналу T при цьому складе: $K=T/dT$. Завдавшись кількістю відліків на один період та змінюючи розмір кроку dT, можна отримати змінний період. Значення відліків для кожного типу сигналу доцільно попередньо обчислити, а потім записати у вигляді таблиці у пам'ять. При цьому обчислення функції замінюється пошуками відповідного елемента таблиці. Потрібно зазначити, що завдання функції у вигляді таблиці скорочує ємність пам'яті та підвищує швидкодію генератора.

Таким чином, алгоритм роботи ГМП може містити такі блоки:

– введення типу сигналу й значення періоду. При цьому код типу сигналу визначає потрібну таблицю, а код значення періоду – необхідний розмір затримки;

– циклічне виведення вмісту таблиці на зовнішній пристрій аналого-цифрового перетворення з необхідною затримкою при виведенні кожного елементу таблиці.

Визначення потрібної розрядності.

Вибір розрядності, що визначає крок квантування за рівнем, диктується завданою точністю представлення цифрового відліку. Мінімальна величина кроку при представленні чисел в МП у формі з фіксованою комою складе одиницю молодшого розряду. Таким чином помилка при заміні точного значення функції її цілочисельним значенням з округленням в більшу чи меншу сторону не перевищить $0,5$ одиниці молодшого розряду.

Для МК MCS-51 слід прямувати до однобайтового представлення чисел, переходячи до двохбайтового лише при необхідності, оскільки в останньому випадку ускладнюється програмне забезпечення, а також – і технічна реалізація ГМП.

Для пилкоподібного сигналу при *K=16* зміна значень цифрових відліків відбувається точно за лінійним законом з кроком, що дорівнює *16*. Таблиця вихідних відліків має вигляд:*0, 16, 32, 48, 64, 80, 96, 112, 128, 144, 160, 176, 192, 208, 224, 240*. При цьому помилка в цифровому представленні пилкоподібного сигналу відсутня.

Для синусоїдного сигналу помилка виникає при формуванні практично кожного вихідного відліку. Вихідні відліки обчислюються за формулою *U*=]*128 sinωt*[*+128*, де][– округлення до найближчого цілого значення. В даній формулі використовується весь діапазон чисел від *0* до *255*. Максимальна помилка тут складає *2,6%*. Зменшити похибку можна за рахунок зменшення амплітуди сигналу *U*. Синусоїдний сигнал можна представити, наприклад, *16* відліками. Максимальна відносна похибка в цьому випадку не перевищить *0,8%*.

У НВП *EV8031/AVR* застосовано восьмирозрядний ЦАП DD2 *AD7801* (Рисунок 4.1).Стан компаратора можна зчитати з виводу порту P1.7 МК. Про завершення циклу перетворення також може свідчити світіння світлодіоду, який підключений до виходу компаратора. Доступ до ЦАП здійснюється як до комірки зовнішнього ОЗП за адресою *0F000h*.

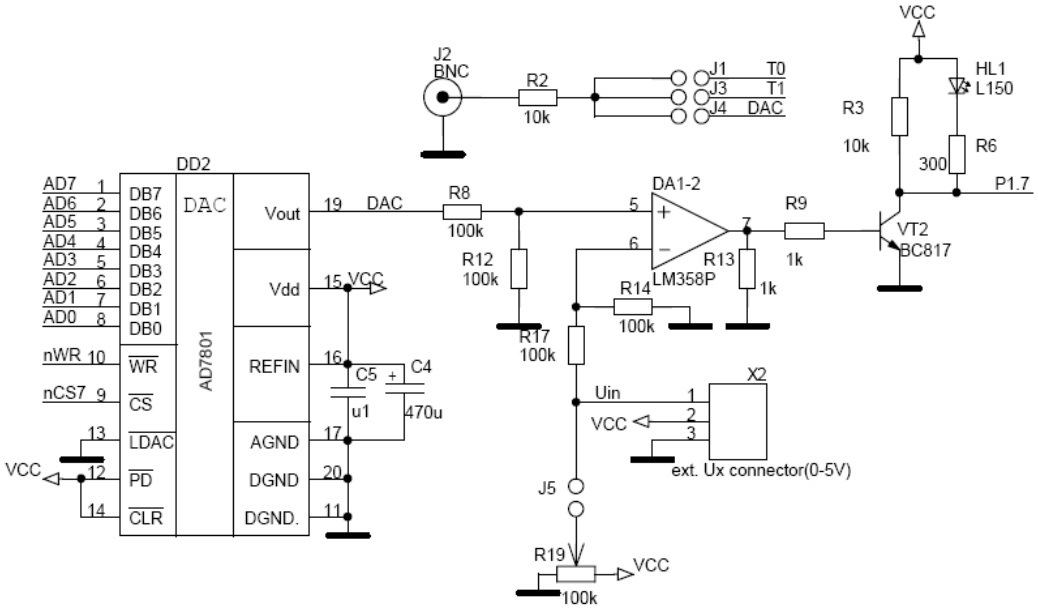

Рисунок 4.1 – Підключення ЦАП у НВП EV8031/AVR

4.2 Паралельні порти введення/виведення MCS-51

Усі чотири порти (P0-P3) призначені для введення або виведення інформації побайтно. Кожний порт містить керуєму регістр-защілку, вхідний

буфер і вихідний драйвер.

Вихідні драйвери портів 0 и 2, а також вхідний буфер порту 0 використовуються при звертанні до зовнішньої пам‘яті. При цьому через порт 0 в режимі часового мультиплексування спочатку виводиться молодший байт адреси, а потім видається або приймається байт даних. Через порт 2 виводиться старший байт адреси в тих випадках, коли розрядність адреси дорівнює 16 біт.

Усі виводи порту 3 можуть бути використані для реалізації альтернативних функцій, наведених в табл. 4.5. Ці функції можуть бути задіяні шляхом запису 1 у відповідні біти регістра-защілки (P3.0-P3.7) порту 3.

Порт 0 є двонапрямленим, а порти 1-3 – квазідвонапрямленими. Кожна лінія портів може бути використана незалежно для введення або виведення.

По сигналу RST в регістри-защілки усіх портів автоматично записуються одиниці, налаштовуючи їх тим самим на режим введення.

Усі порти можуть бути використані для організації введення/виведення інформації по двонапрямленим лініям передачі. Однак порти 0 и 2 не можуть бути використані з цією метою у випадку, коли система має зовнішню пам‘ять, зв‘язок з якою організується через спільну розділяєму шину адреси/даних, яка працює в режимі часового мультиплексування.

Звертання до портів введення/виведення можливе з використанням команд, що оперують з байтом, окремим бітом, довільною комбінацією бітів.

Таблиця 4.1 – Альтернативні функції порту P3

Символ	Розряд	Ім‘я та призначення
RD	P3.7	Читання. Активний сигнал низького рівня формується апаратно при звертанні до зовнішньої пам‘яті даних
WR	P3.6	Запис. Активний сигнал низького рівня формується апаратно при звертанні до зовнішньої пам‘яті даних
T1	P3.5	Вхід таймера/лічильника 1 або тест-вхід
T0	P3.4	Вхід таймера/лічильника 0 або тест-вхід
INT1	P3.3	Вхід запиту переривання 1. Сприймається сигнал низького рівня або зріз
INT0	P3.2	Вхід запиту переривання 0. Сприймається сигнал низького рівня або зріз
TXD	P3.1	Вихід передавача послідовного порту в режимі UART. Вихід синхронізації в режимі регістра зсуву
RXD	P3.0	Вхід приймача послідовного порту в режимі UART. Введення/виведення даних в режимі регістра зсуву

При цьому в тих випадках, коли порт є одночасно операндом і місцем призначення результату, пристрій керування автоматично реалізує спеціальний режим, який зветься “читання-модифікація-запис”. Цей режим звертання передбачає введення сигналів не з зовнішніх виводів порту, а з його регістра-защілки, що дозволяє виключити неправильне зчитування раніше виведеної інформації. Цей механізм звертання до портів реалізова-

ний у командах:

- ANL – логічне І, наприклад, ANL P1,A;
- ORL – логічне АБО, наприклад, ORL P2,A;
- XRL – виключаюче АБО, наприклад, XRL P3,A;
- JBC – перехід, якщо в адресуємому біті одиниця, та наступне скидання біта, наприклад, JBC P1.1, LABEL;
- CPL – інверсія біта, наприклад, CPL P3.3;
- INC – інкремент порту, наприклад, INC P2;
- DEC – декремент порту, наприклад, DEC P2;
- DJNZ – декремент порту й перехід, якщо його вміст не дорівнює нулю, наприклад, DJNZ r, LABEL;
- MOV PX.Y,C – передача біта перенесення в біт Y порту X;
- SET PX.Y – установка біта Y порту X;
- CLR PX.Y – скидання біта Y порту X.

У прикладі 4.1. масив, що зберігає вибірки сигналу, визначений з використанням ключового слова **__code** яке вказує компілятору SDCC, що цей масив треба зберігати у пам'яті програм, а не у пам'яті даних (дивись додаток А).

Приклад 4.1 **– Генерація сигналу виду t^2 з періодом 495 мкс**

```
/*
 * main.c
 * Генерація сигналу з періодом 495 мкс
 */
#include <lint.h>
#include <at89x52.h>
#include "interrupts.h"

void main(void)
	{
		// Ініціалізація таймера
		TMOD = 0b00100000;
		TL1 = 218;
		TH1 = 218;
		// Ініціалізація переривань
		IE = 0b10001000;
		// Запуск таймера
		TR1 = 1;
		// Нескінчений цикл
		while (1)
			{
			}
	}
/*
 * interrupts.h
 */

#ifndef INTERRUPTS_H_
#define INTERRUPTS_H_

#include <lint.h>
#include <at89x52.h>

void Timer1Interrupt (void) __interrupt(TF1_VECTOR);
```

```
#endif /* INTERRUPTS_H_ */
/*
 * interrupts.c
 */
#include <at89x52.h>
#include <stdint.h>
#include "EV8031.h"

__code uint8_t sample[8] = {0, 4, 16, 36, 64, 100, 144, 196};

void Timer1Interrupt(void) __interrupt(TF1_VECTOR)
	{
		static uint8_t i = 0;
		DAC =sample[i++ & 0b00000111]; // Виведення чергового відліку сигналу
у ЦАП
	}
```

Поглянемо, яким чином компілятор перетворив код функції переривання у асемблерний текст (Рисунок 4.2)

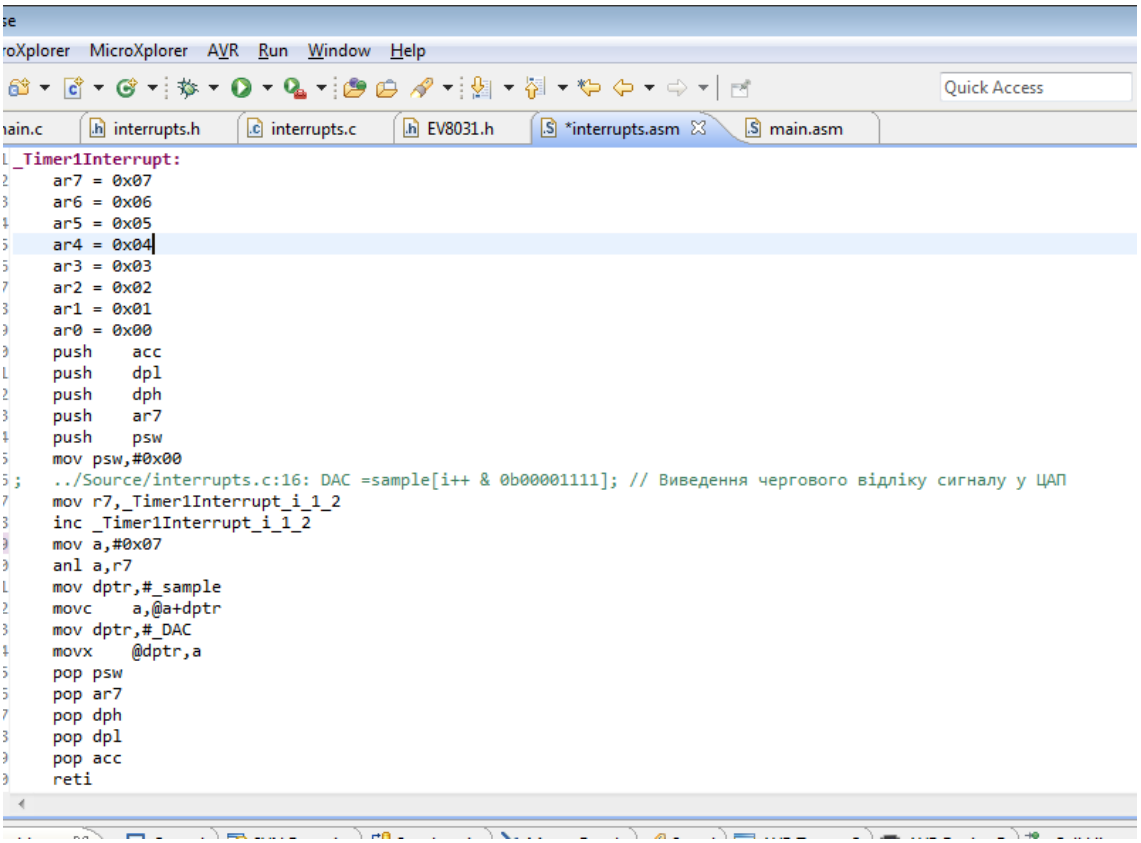

Рисунок 4.2 – Асемблерний текст функції переривання

Як бачимо, функція переривання починається (пролог) зі збереження у стеку вмісту регістрів, що використовуються у перериванні (команди `push`) і закінчується (епілог) відновленням стану цих регістрів зі стеку (команди `pop`) та командою `reti`. Ці команди потребують часу на виконання кожного разу, коли відбувається виклик переривання. Іноді (якщо регістри, що використовуються у перериванні, не використовуються у ос-

новній програмі) пролог і епілог можна пропустити або скоротити, що дозволяє скоротити час реакції на переривання. Проте робити це слід дуже обережно, оскільки це – потенційний спосіб “вистрелити собі у ногу”.

Програма генерації сигналу, аналогічна тій, що у прикладі 4.1, але зі збільшеною кількістю вибірок (це стало можливим за рахунок підвищення швидкості реакції на переривання), наведена у прикладі 4.2.

Приклад 4.2 **– Генерація сигналу виду t^2 з періодом 495 мкс (по таблиці з 16 значень)**

```
/*
 * main.c
 * Генерація сигналу з періодом 495 мкс
 */
#include <lint.h>
#include <at89x52.h>
#include "interrupts.h"

void main(void)
	{
		// Ініціалізація таймера
		TMOD = 0b00100000;
		TL1 = 237;
		TH1 = 237;
		// Ініціалізація переривань
		IE = 0b10001000;
		// Запуск таймера
		TR1 = 1;
		// Нескінчений цикл
		while (1)
			{
			}
	}
/*
 * interrupts.h
 */

#ifndef INTERRUPTS_H_
#define INTERRUPTS_H_

#include <lint.h>
#include <at89x52.h>

void Timer1Interrupt (void) __interrupt(TF1_VECTOR);

#endif /* INTERRUPTS_H_ */

/*
 * interrupts.c
 */
#include <at89x52.h>
#include <stdint.h>
#include "EV8031.h"

__code uint8_t sample[16] = {0, 1, 4, 9, 16, 25, 36, 49, 64, 81, 100, 121, 144,
169, 196, 225};

void Timer1Interrupt(void) __interrupt(TF1_VECTOR) __naked
	{
		static uint8_t i = 0;
```

```
        DAC =sample[i++ & 0b00001111]; // Виведення чергового відліку сигналу
у ЦАП
        __asm__("reti");
}
```

У прикладі 4.2 ключове слово **__naked** вказує компілятору, що для даної функції не треба створювати пролог та епілог. Якщо якісь регістри все ж треба зберігати, а потім відновити, то для цього можна використовувати асемблерні вставки **__asm__("команда асемблера");**. Результати трансляції функції обробки переривання з прикладу 4.2 наведені на рисунку 4.3.

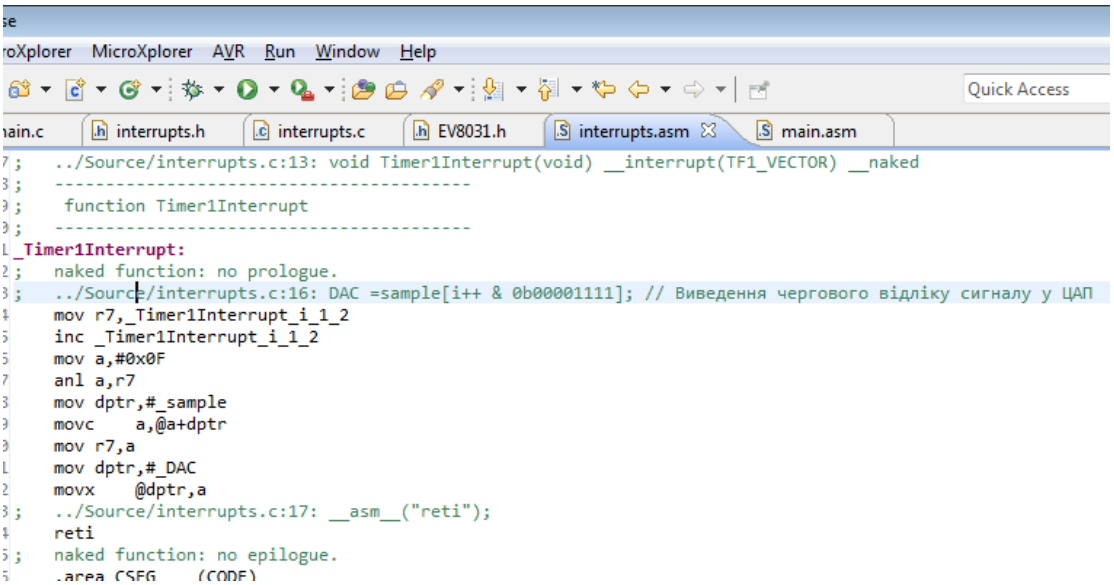

Рисунок 4.3 – Асемблерний текст функції переривання без прологу та епілогу

4.3 Контрольні питання

1. Перерахуйте відомі Вам методи побудови генераторів електричних сигналів та наведіть приклади сфер їх застосування.

2. Вкажіть особливості побудови та функціонування портів введення/виведення MCS-51.

3. Наведіть часові діаграми роботи MCS-51 при виконанні наступних команд:

`Mov A,R3;` `Movx A,@DPTR;` `Movx @DPTR,A`

4.4 Хід роботи

1. Ознайомитися з теоретичними відомостями щодо принципів побудови генераторів сигналів на мікроконтролері.

2. Дайте письмові відповіді на контрольні запитання.

3. Безпосередньо у викладача отримайте допуск до виконання роботи в лабораторії та варіант завдання.

3. Утворіть новий проект (наприклад, `МПТ_lab4`), розробіть алго-

ритм і складіть програму відповідно до отриманого завдання.

5. Перевірте роботу розробленої програми в реальному часі на стенді *EV8031/AVR* та продемонструйте її викладачу.

6. Завершіть звіт та захистіть роботу.

4.5 Вимоги до звіту по роботі

Звіт про виконання роботи повинен містити:

1. Титульний аркуш встановленого в університеті зразку.

2. Письмові відповіді на контрольні питання.

3. Зміст завдання зі схемою підключення мікроконтролера, а також схему програми вирішеної задачі.

4. Розподіл пам'яті для розміщення основної програми, підпрограм (за їхньої наявності) і блоків даних, розподіл регістрів мікроконтролера.

5. Вхідний текст програми, з обов'язковими коментарями.

6. Висновки, в яких треба зазначити, чи досягнута мета даної лабораторної роботи. Що саме конкретно (перерахувати по пунктах) повинна містити програма формування електричних сигналів за допомогою зовнішнього ЦАП?

4.6 Орієнтовні варіанти завдань

Розробити програму формування періодичних сигналів за допомогою ЦАП на платі розширення. Вибір типу сигналу виконувати кнопками *SW15, SW16* за перериванням. Номер сигналу відображати на дисплеї.

Таблиця 4.2 – Варіанти завдань для розробки програми

Вар.	**Сигнал №1**	**Сигнал №2**	**Період, мкс**
1	$\sin \omega t$	at	*256*
2	t^2	$\lvert\sin \omega t\rvert$	*512*
3	*пилкоподібний*	$1/t$	*1024*
4	$-\lvert\sin \omega t\rvert$	*спадаюча пилка*	*2048*
5	e^t	*прямокутний (τ=25мс, q=4)*	*1024*
6	$-t^2$	*трапеція (200 мкс/100 мкс)*	*512*
7	*наростаюча пилка*	$-e^t$	*256*
8	$\ln t$	*Напівперіод. випрямл.* $\sin \omega t$	*2048*
9	*Напівп.випрямл.-* $\sin \omega t$	$-\ln t$	*4096*
10	e^{-t}	*AIM* $\sin \omega t$	*8192*
11	*трикутний (10/20 мс)*	e^{-t2}	*32768*

5 Лабораторна робота №5. Дослідження методів введення аналогової інформації в МПС на базі MCS-51

Мета роботи: практично дослідити методи введення аналогової інформації в мікропроцесорну систему на базі мікроконтролерів сімейства MCS-51.

5.1 Короткі відомості про побудову АЦП

МК є приладом, в основі якого лежать операції з цифровими даними, тобто такими, що подані двійковими кодами. Проте існує багато джерел, інформація на виході яких являє собою аналогову величину. Це, наприклад, різноманітні датчики параметрів. Існує багато випадків, коли об‘єкт керування є аналоговим за своєю суттю і потребує аналогового сигналу для керування. Застосування МП техніки і в цих випадках дає чимало переваг, але актуалізується питання зв‘язку МП з аналоговими пристроями. Для виведення інформації з МПС на аналогові об‘єкти керування зазвичай використовуються цифро-аналогові перетворювачі та широтно-імпульсні модулятори. Для реалізації процедури введення аналогового сигналу в МПС використовуються аналогово-цифрові перетворювачі (АЦП).

АЦП застосовуються в вимірювальних системах та обчислювальних комплексах для узгодження аналогових джерел сигналів з цифровими пристроями обробки й представлення результатів вимірювання.

Існують різні методи побудови АЦП. Їх розрізнюють за складністю реалізації, завадостійкістю, швидкодією.

В системах, де основним критерієм є швидкодія, застосовують АЦП паралельного перетворення. Проте АЦП цього типу досить складні в реалізації. Для n-розрядного АЦП необхідно *2n-1* компараторів і паралельний подільник напруги, який виробляє *2n-1* рівнів квантування.

Для реалізації систем з високою завадостійкістю застосовують АЦП, що інтегрують. Такий АЦП складається з двох перетворювачів. Напруга, що підлягає вимірюванню, перетворюється в тривалість імпульсу, а далі тривалість імпульсу перетворюється в цифровий код.

Найпоширенішими є АЦП, побудовані на базі цифро-аналогового перетворювача (ЦАП, Рисунок 5.1).

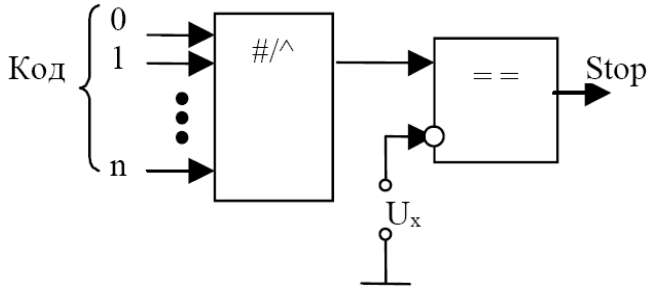

Рисунок 5.1 – Схема АЦП на базі ЦАП

Код формується або лічильником (реалізація засобами «жорсткої логіки»), або програмно (якщо АЦП працює в складі МПС). Вхідний код перетворюється в аналоговий сигнал за допомогою ЦАП. Напруга з виходу ЦАП поступає на один з входів компаратора. На інший вхід подається напруга, що вимірюється U_x. В момент, коли напруга ЦАП буде рівною вимірюваній, компаратор сформує сигнал *'Stop'*, який свідчить про завершення циклу вимірювання.

При формуванні коду застосовують різні алгоритми. Найпростішим алгоритмом є *порозрядне врівноваження*. При такому підході код змінюється від мінімального шляхом прирощення одиниці молодшого розряду доти, доки напруга ЦАП не зрівняється з напругою, що вимірюється. Недолік порозрядного врівноваження – мала швидкодія.

Для скорочення часу перетворення застосовують метод *половинних наближень*. Врівноважування починається з старшого розряду. В цьому розряді встановлюється одиниця та читається стан компаратора. Якщо напруга ЦАП більша за вимірювану, розряд скидається, а якщо менша, розряд зберігає свій стан. Далі таким же чином оброблюється наступний розряд. Перетворення закінчується тоді, коли будуть оброблені всі розряди.

В системах слідкування за якимось параметрами часто необхідно безперервно зчитувати стан датчика. Малий час перетворення в цьому випадку можна забезпечити за рахунок застосування *слідкуючого АЦП*. Суть даного алгоритму полягає в тому, що спочатку код формується методом половинних наближень. А після порівняння з напругою, що вимірюється, АЦП відстежує *зміну напруги*. Якщо напруга росте, код порозрядно підвищується доти, доки напруга ЦАП не зрівняється з вимірюваною, і навпаки.

У НВП *EV8031/AVR* АЦП побудований на мікросхемах *AD7801* (восьмирозрядний ЦАП) та *LM358* (операційний підсилювач в якості компаратора) (Рисунок 4.1). Стан компаратора можна зчитати з виводу порту P1.7 МК. Про завершення циклу перетворення також може свідчити світіння світлодіоду, який підключений до виходу компаратора. Доступ до ЦАП здійснюється як до комірки зовнішнього ОЗП за адресою *0F000h*.

Напруга що вимірюється *Ux* формується потенціометром *R19* (якщо встановлено перемичку *J5*), або джерело сигналу підключається до роз'єднувача *X2* на платі розширення №1.

В розширеній комплектації НВП *EV8031/AVR* поставляється з інтегральним десятибітним АЦП з паралельним інтерфейсом *AD7813*. Доступ до АЦП *AD7813* здійснюється як до комірки зовнішнього ОЗП за адресою *0E000h*.

Далі наведено приклад програми аналого-цифрового перетворення.

***Приклад 5.1* – Текст програми аналого-цифрового перетворення**

```
// main.c
#include <lint.h>
#include <at89x52.h>
#include "interrupts.h"
#include "EV8031.h"
```

```
uint8_t ADC_sample(void); // Прототип функції аналого-цифрового перетворення.

Void
main(void)
{
  uint8_t adc = 0;
  //  Ініціалізація дисплея
  DC_REG = 0b00000011;
  // Ініціалізація таймера
  TMOD = 0b00100000;
  TL1 = 0;
  TH1 = 0;
  // Ініціалізація переривань
  IE = 0b10001000;
  // Запуск таймера
  TR1 = 1;
  // Нескінчений цикл
  while (1)
    {
      adc = ADC_sample(); // Отримати чергову вибірку з АЦП.
      if (time_flag)        // Якщо вже час виводити результат на екран то:
        {
          RIGHT_DISPLAY = adc; // 1. Вивести на екран результат перетворення;
          time_flag = 0;       // 2. Скинути прапорець.
        }
    }
}
// Функція, що повертає результат аналого-цифрового перетворення.
uint8_t ADC_sample(void)
{
  uint8_t sample = 255, i;
  do
    {
      sample++;
      DAC = sample;        // Вивести значення змінної sample у ЦАП.
      for (i = 0; i < 20; i++)
        ;                  // Трохи почекати.
    }
  while (P1_7 && (sample != 255)); // Якщо компаратор видає 1 та змінна sample не
дорівнює 255, повторити цикл інакше
  return sample;             // повернути результат аналого-цифрового перетворен-
ня.
}

// interrupts.h

#ifndef INTERRUPTS_H_
#define INTERRUPTS_H_

#include <lint.h>
#include <at89x52.h>
#include <stdint.h>

void Timer1Interrupt (void) __interrupt(TF1_VECTOR);
extern uint8_t time_flag;

#endif /* INTERRUPTS_H_ */
```

```
// interrupts.c

#include <at89x52.h>
#include <stdint.h>
#include <lint.h>

uint8_t time_flag = 0;

void Timer1Interrupt(void) __interrupt(TF1_VECTOR)
	{
		static uint16_t i = 0;
		if (++i >= 600 )
			{
				i = 0;
				time_flag = 1;
			}
	}
```

5.2 Контрольні питання

1. Наведіть методи, типи та алгоритми аналого-цифрового перетворення.
2. Дайте визначення наступних статичних параметрів АЦП: дискретизація, квантування, роздільна здатність, диференційна нелінійність, відхилення коефіцієнта перетворення, зсув нуля.
3. Дайте визначення наступних динамічних параметрів АЦП: час перетворення, час циклу перетворення, максимальна частота перетворення, апертурний час.
4. Перелічіть основні фактори, що впливають на похибку АЦП.
5. Яким чином будуються АЦП за допомогою мікросхем ЦАП?
6. Який алгоритм АЦП реалізовано в прикладі 5.1?

5.3 Хід роботи

1. Ознайомтеся з теоретичними відомостями щодо принципів побудови та зі схемою модуля АЦП у НВП *EV8031/AVR*.

2. Дайте письмові відповіді на контрольні запитання.

3. Безпосередньо у викладача отримайте допуск до виконання роботи в лабораторії та варіант завдання.

3. Утворіть новий проект (наприклад, `МПТ_lab5`), розробіть алгоритм і складіть програму відповідно до отриманого завдання.

5. Перевірте роботу розробленої програми в реальному часі на стенді *EV8031/AVR* та продемонструйте її викладачу.

6. Завершіть звіт та захистіть роботу.

5.4 Орієнтовні варіанти завдань

Таблиця 5.1 – Варіанти завдань для розробки програми

№	*Алгоритм для реалізації*
1	Порозрядне врівноважування з індикацією результату на дисплеї.
2	Половинні наближення з індикацією результату на дисплеї.
3	Слідкуючий з початковим порозрядним врівноважуванням з індикацією результату на дисплеї.
4	Слідкуючий з початковим половинним наближенням з індикацією результату на дисплеї.
5	За натисканням кнопки SW15 запустити АЦП половинних наближень з індикацією результату на дисплеї.
6	За натисканням кнопки SW15 запустити АЦП порозрядного врівноважування з індикацією результату на дисплеї.
7	За натисканням кнопки SW15 запустити АЦП слідкуючого типу з початковим половинним наближенням та індикацією результату на дисплеї.
8	За натисканням кнопки SW16 запустити АЦП слідкуючого типу з початковим половинним наближенням та індикацією результату на дисплеї.
9	За натисканням кнопки SW16 запустити АЦП половинних наближень з індикацією результату на дисплеї.
10	За перериванням INT0 запустити АЦП порозрядного врівноважування з індикацією результату на дисплеї за натисканням кнопки SW16
11	За натисканням кнопки SW15 запустити АЦП слідкуючого типу з початковим половинним наближенням та індикацією результату на дисплеї за натисканням кнопки SW16
12	За натисканням SW16 запустити АЦП слідкуючого типу з початковим половинним наближенням та індикацією результату на дисплеї за натисканням SW15

6 Лабораторна робота №6. Дослідження принципів побудови систем автоматичного керування на базі мікроконтролерів

Мета роботи: виявити можливості мікроконтролерів сімейства MCS-51 виконувати завдання в контурі системи автоматичного керування.

Примітка. Виконання цієї роботи потребує підключення плати розширення №2 замість плати розширення №1.

6.1 Короткі відомості про побудову систем автоматичного керування

Системи автоматичного керування (САК) є квінтесенцією електронних систем і сьогодні визначають обличчя науково-технічного прогресу. Велику цікавість викликають *цифрові* системи автоматичного керування завдяки високій точності, стабільності параметрів та повторюваності характеристик. Використання МК в САК надає додаткових переваг, які пов'язані з гнучкістю алгоритму роботи та можливістю суттєвого зниження габаритів, енергоспоживання, вартості. Одночасне підвищення надійності електронної системи дозволяє зробити однозначний висновок про суттєво вищий технічний рівень продукції с застосуванням МК.

Структура типової цифрової САК (Рисунок 6.1) містить блок задання (БЗ), цифровий регулятор (ЦР), об'єкт керування (ОК), датчики параметрів об'єкта (ДПО), інтерфейси з об'єктом керування (ІОК) та з датчиком параметрів об'єкта (ІДО).

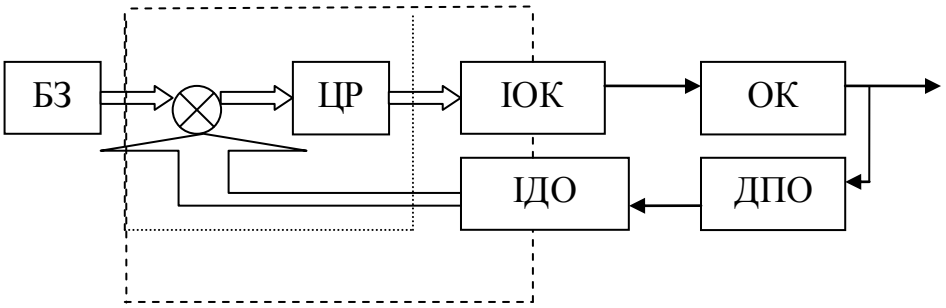

Рисунок 6.1 – Структура цифрової САК

Пунктиром показане типове використання МП в САК в якості блоку різності та ЦР. Розвинута периферія сучасних МК дає можливість поширити застосування засобів МП-техніки також на пристрої зв'язку з ОК та ДПО (штрихова лінія, Рисунок 6.1). Вміле використання вбудованих у кристал МК апаратних засобів може суттєво покращити техніко-економічні характеристики САР.

БЗ призначений для формування задання на керування у вигляді цифрового коду. В якості БЗ може використовуватися цифровий задатчик САК (клавіатура, паралельний інтерфейс), імпульсний задатчик (енкодер, послідовний інтерфейс) або аналоговий задатчик (на базі потенціометра та

АЦП). Схеми підключення дискретних задатчиків САК у стенді «МПТ» показано нижче (Рисунок 6.2).

Матричну клавіатуру 3x4 SW3-SW14 підключено до шини даних МК за допомогою мікросхеми буфера DD1 74245 (АП6). За один раз можна прочитати стан тільки одного стовпчика клавіатури.

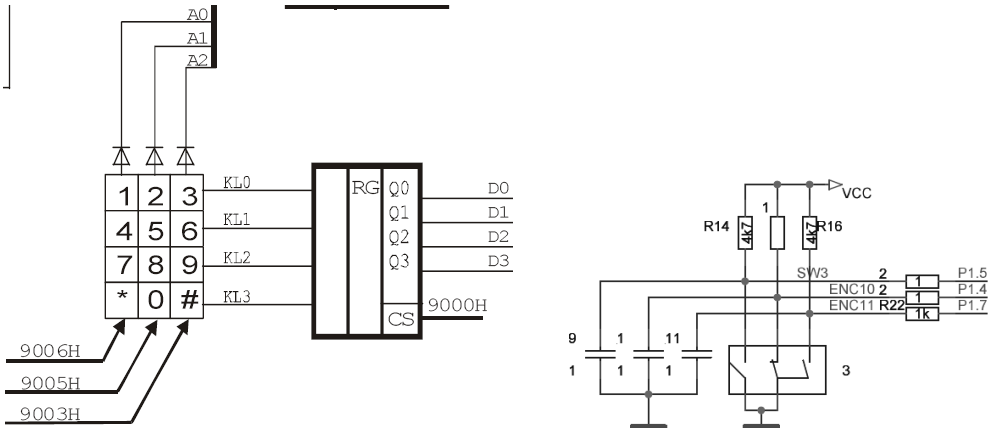

Рисунок 6.2 – Дискретні задатчики САК у стенді «МПТ»

Тобто, щоб опитати всі клавіші, потрібно тричі прочитати буфер. Щоб опитати стовпчик (клавіші «1», «4», «7», «*»; «2», «5», «8», «0» або «3», «6», «9», «#»), необхідно виставити на відповідну лінію адреси (A0, A1, A2 для першого, другого та третього стовпчика, відповідно) рівень логічного нуля, а на інших лініях – рівень логічної одиниці. Це можна здійснити, прочитавши стан буфера, що підключений до шини даних МК, як три доступні для читання комірки пам‘яті з адресами 9006h, 9005h та 9003h. Якщо кнопку натиснуто, відповідний біт у зчитаному байті буде дорівнювати нулю, якщо не натиснуто – то одиниці.

Енкодер міститься на платі розширення №2 і зовні нагадує звичайний поворотний потенціометр. Проте, на відміну від останнього, енкодер – значно надійніший сучасний радіоелектронний компонент, за допомогою якого простіше ввести інформацію в МК. Замість потреби в АЦП інформація про кут повернення ручки енкодера безпосередньо в імпульсному вигляді поступає на цифрові (логічні) входи МК. При повертанні ручки за годинниковою стрілкою сигнал (перехід з одиниці до нуля) з‘являється спочатку на виході ENC10 (підключений до P1.4 МК), а потім вже – на ENC11 (P1.7). При повертанні ручки в зворотному напрямі все виходить навпаки. Для надійності роботи в програмі треба передбачити усунення дзвону контактів. Отже, енкодер дозволяє контролювати напрям повороту ручки. Кожний перехід логічного рівня відбувається внаслідок зсуву на певний кут, в залежності від конкретного типу застосованого енкодера. Підрахунок логічних переходів забезпечує введення в МК певної величини завдання. Цікаво, що при натисканні на ручку задіється ще один цифровий вихід енкодера (Рисунок 6.2, SW3). Отже, цей зручний пристрій дозволяє реалізувати

меню вибору сигналу, що уводиться оператором в САК через порт P1.5.

В якості об’єктів керування в стенді «МПТ» передбачені мініатюрний електричний двигун постійного струму та лампа розжарювання (Рисунок 6.3).

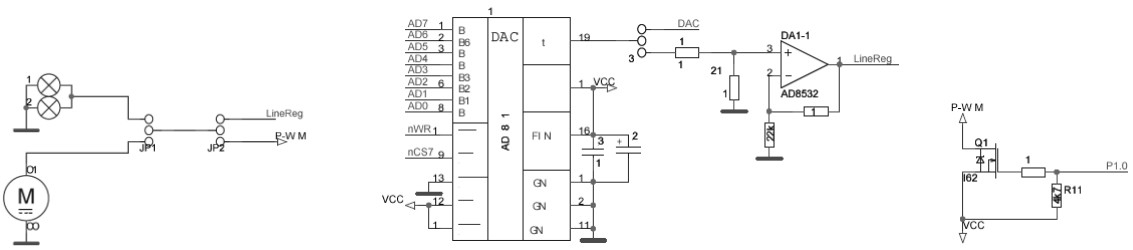

Рисунок 6.3 – Об’єкти керування у стенді «МПТ»

Вибір конкретного ОК здійснюється за допомогою перемички JP1 на платі розширення №2. Інша перемичка JP2 дозволяє використовувати аналоговий або широтно-імпульсний інтерфейс з ОК. В першому випадку МК взаємодіє з ОК через цифро-аналоговий перетворювач (DAC) та операційний підсилювач. В другому випадку логічний сигнал з лінії порту МК P1.0 безпосередньо подається на затвор силового польового транзистора з *p*-каналом. Встановлення логічного 0 призводить до відпирання ключа і подання майже всієї напруги живлення (V_{cc}=5В) на об’єкт керування.

Для кожного з об’єктів керування в стенді «МПТ» передбачені датчики параметрів ОК (Рисунок 6.4): безконтактний датчик на ефекті Холла (*U2*) та світлодіод для двигуна, а також датчик температури з послідовним інтерфейсом TMP03 (*U1*) для лампи розжарювання [10]. Датчик Холла являє собою компактну інтегральну схему, в складі якої міститься власне датчик і схема формування вихідного сигналу. Цей сигнал підключений до входу лічильника подій T0, що дає можливість використати резидентні апаратні засоби МК для визначення періоду обертання вала ротора.

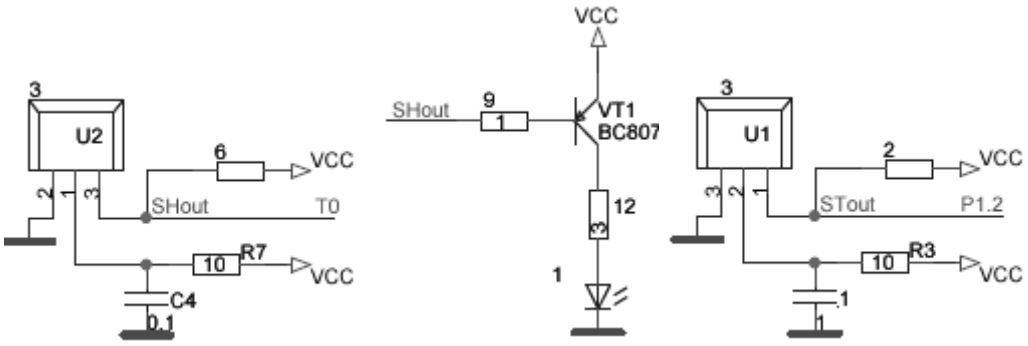

Рисунок 6.4 – Датчики параметрів об’єктів у стенді «МПТ»

На виході датчика температури TMP03 формується прямокутний періодичний сигнал з номінальною частотою 35 Гц (±20%) при +25°C (Рисунок 6.5).

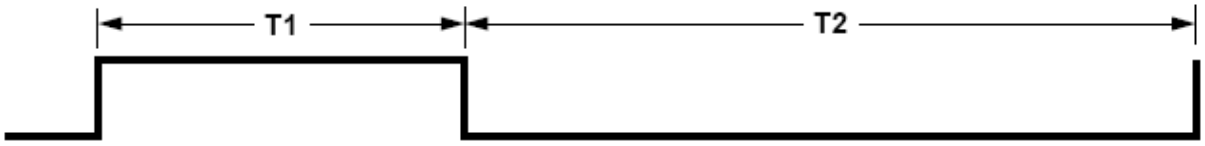

Рисунок 6.5 – Форма сигналу на виході датчика TMP03

Цей сигнал, що подається на вхід P1.2 МК, можна декодувати, використовуючи наступну формулу:

$$\text{Температура } (°\text{C}) = 235 - \left(\frac{400 \times T_1}{T_2} \right). \quad (6.1)$$

Інтервали часу T1 (логічна одиниця) та T2 (логічний нуль) можна виміряти за допомогою таймерів/лічильників МК, а далі виконати обчислення відповідно до (6.1) за допомогою програми. Оскільки обидва інтервали послідовні, то можна застосувати єдиний тактовий генератор. Тоді внаслідок ділення отримуємо коефіцієнт, що не залежить від точного значення чи нестабільності ані тактової частоти МК, ані частоти задаючого генератора самого датчика. Єдине, тактова частота таймера має бути достатньо високою та стабільною для забезпечення потрібної роздільної здатності вимірювання [10].

Програма взаємодії TMP03 з МК наведена у прикладі 6.1. Програма моніторить вихід TMP03 та вмикає і вимикає лічильники для вимірювання шпаруватості. Інтервал, де вихід має високий рівень, вимірюється за допомогою Timer 0, а низький – Timer 1. Наприкінці програми результати розміщуються в регістрах спеціальних функцій (SFR) з 08Ah по 08Dh.

Приклад 6.1 **– Програма взаємодії TMP03 з MCS-51**

```
; Тестування інтерфейсу TMP03 з MCS-51 з використанням timer 0
; та timer 1 для вимірювання шпаруватості на вході P1.2.
; Ця програма складається з трьох кроків:
; 1. Очистити регістри таймерів та чекати переходу з 0 до 1.
; 2. Коли P1.2=1, стартує timer 0. Цикл перевірки P1.2.
; 3. Коли P1.2=0, timer 0 зупиняється, а timer 1 стартує.
; Програма зациклюється, доки P1.2 знову не стане 1. Це
; зупиняє timer 1. Величини T1 та T2 зберігаються в SFR
; з 8Ah по 8Dh (з TL0 по TH1).
READ_TMP03: mov A,#00 ; Спочатку -
            mov TH0,A ; очистка
            mov TH1,A ; лічильників
            mov TL0,A ;
            mov TL1,A ;
WAIT_LO: jb P1.2,WAIT_LO ; Чекаємо переходу TMP03 у 0 -\_
            mov A,#11H ; Підготовка до старту
            mov TMOD,A ; timer0
WAIT_HI: jnb P1.2,WAIT_HI ; Чекаємо переходу у 1  _/-
; Timer 0 рахує, коли вихід TMP03 є високим (1)
            setb TCON.4 ; Старт timer 0
WAITTIMER0: jb P1.2,WAITTIMER0
```

```
            clr TCON.4 ; Виключити timer 0
; Timer 1 рахує, коли вихід TMP03 є низьким (0)
            setb TCON.6 ; Старт timer 1
WAITTIMER1: jnb P1.0,WAITTIMER1
            clr TCON.6 ; Зупинити timer 1
            mov A,#0H ; Підготувати заборону таймерів
            mov TMOD,A
            ret
```

Коли викликається підпрограма READ_TMP03, лічильні регістри очищуються. Програма встановлює 16-бітний режим лічильників та чекає одиниці на виході TMP03. Коли вихід порту повертається у 1, стартує Timer 0. Таймер продовжує роботу, а програма перевіряє порт. Коли вихід TMP03 стає низьким, Timer 0 зупиняється, а Timer 1 стартує. Timer 1 працює, доки вихід TMP03 не стане високим, і в цей момент обмін з TMP03 завершується. Коли підпрограму закінчено, вміст таймерів зберігається в їхніх відповідних регістрах SFR, а температуру TMP03 можна обчислити програмно.

Оскільки МК працює асинхронно відносно TMP03, між переключенням виходу датчика та запуском таймера завжди існує затримка. Ця затримка може варіювати від нуля до часу виконання тієї команди, що розпізнає перехід. Команди переходу по значенню біта порту MCS-51 (JB та JNB) виконуються за 24 такти опорного генератора. При тактуванні 12 МГц це призводить до невизначеності у 2 мкс для кожного переходу на виході TMP03. Найгіршій случай має місце, коли T1 на 4 мкс коротше за справжню величину, а T2 на 4 мкс триваліше. Для читання при +25°C (“кімнатна температура”) номінальна помилка, що викликана затримкою в 2 мкс складає тільки близько ±0.15°C.

Оскільки вихідні сигнали наявних датчиків є логічними, для сполучення з лініями портів МК жодних додаткових інтерфейсних вузлів не потрібно. Світлодіод, підключений до датчика Холла, дозволяє візуально перевіряти наявність обертання вала ротора електродвигуна.

Таким чином, стенд «МПТ» дає можливість досліджувати застосування різноманітних програмно реалізованих цифрових регуляторів як в розімкнених, так і в замкнених САК.

6.2 Елементи теорії ПІД-регуляторів

Метою алгоритму ПІ-регулятора є визначення сигналу керування ОК (для двигуна постійного струму – це напруга статора), при якій контролюємий вихідний сигнал ОК (частота обертання ротора) досягне заданого значення (бажана частота обертання, яку задано користувачем). ПІ – це скорочення від "пропорційний та інтегральний". Ці два терміни описують окремі елементи регулятора:

- *пропорційна частина*, в якій виконується множення результуючого сигналу непогодження (різниці виміряного вихідного сигналу ОК та заданого значення) на постійну величину, яка носить назву *коефіцієн-*

та передачі пропорційної частини. Пропорційна частина визначає короткотермінову поведінку регулятора, оскільки вона задає, з якою силою треба реагувати регулятору на зміну заданих значень;

- інтегральна частина, яка додає довготермінову точність регулятору. В даній частині регулятора береться добуток суми всіх попередніх сигналів непогодження на постійну величину, яка носить назву *коефіцієнта передачі інтегральної частини*. Попередні значення сигналу непогодження для обчислення суми зберігаються в пам‘яті та оновлюються доки значення непогодження не дорівнює нулю. Це допомагає регулятору зменшити різницю між виміряним вихідним значенням і заданим, проте знижується швидкодія та стійкість замкненої системи.

Іноді крім пропорційної та інтегруючої частин додають третю складову – диференціальну. В цьому випадку регулятор називають ПІД (пропорційно-інтегрально-диференціальний). Застосування диференціальної частини дозволяє підвищити швидкодію контуру регулювання, але при цьому також пропускаються шуми та знижується стабільність замкненого контуру. Крім того, Д-компонент складний в налаштуванні.

Ідеалізоване рівняння неперервного ПІД-регулятора має вигляд

$$u(t) = K\left[e(t) + \frac{1}{T}\int_0^t e(\tau)\,d\tau + T_D\frac{de(t)}{dt}\right], \qquad (6.2)$$

де K – коефіцієнт передачі, T – постійна інтегрування, T_D – постійна диференціювання.

Ці три параметри підбирають в процесі настроювання регулятора таким чином, щоб максимально наблизити алгоритм роботи системи до бажаного вигляду. Найбільш поширені критерії якості САК визначаються характером перехідного процесу.

При побудові регулятора на базі МК вхідні та вихідні змінні регулятора необхідно дискретизувати у часі (з деяким кроком T_0) та квантувати за рівнем (тобто перетворити в цифрову форму, наприклад – за допомогою АЦП). При цьому рівняння ПІД-регулятора має бути перетворене в різницеве за допомогою заміни похідних кінцевою різницею, а інтеграла – кінцевою сумою. В залежності від обраного методу переходу від неперервних операторів до їхніх дискретних аналогів виникає декілька різних рівнянь, що описують дискретні ПІД-регулятори. При застосуванні метода прямокутників для заміни інтеграла кінцевою сумою отримаємо:

$$u(k) = K\left[e(k) + \frac{T_0}{T}\sum_{i=0}^{k} e(k-1) + \frac{T_D}{T_0}\left[e(k) - e(k-1)\right]\right], \qquad (6.3)$$

де $k = 0,1,\ldots\frac{t}{T_0}$ – порядковий номер відліку дискретного часу.

Недоліком такого представлення рівняння регулятора є необхідність пам‘ятати значення відхилень *e(k)* для всіх моментів часу від початку процесу регулювання.

Цей недолік можна обійти, якщо для обчислення поточного значення керуючої змінної *u(k)* використовувати її попереднє значення *u(k-1)* та поправковий член. Для отримання такого рекурентного алгоритму достатньо відняти з рівняння (6.3) наступне рівняння:

$$u(k-1) = K\left[e(k-1) + \frac{T_0}{T}\sum_{i=0}^{k-1} e(k-1) + \frac{T_D}{T_0}\left[e(k-1) - e(k-2)\right]\right], \quad (6.4)$$

В результаті отримаємо:

$$u(k) - u(k-1) = q_0 e(k) + q_1 e(k-1) + q_2 e(k-2), \quad (6.5)$$

де

$$q_0 = K\left(1 + \frac{T_D}{T_0}\right), \quad (6.6)$$

$$q_1 = -K\left(1 + 2\frac{T_D}{T_0} - \frac{T_0}{T}\right), \quad (6.7)$$

$$q_2 = K\frac{T_D}{T_0}, \quad (6.8)$$

Таким чином, для обчислення поточного значення керуючого впливу *u(k)* на ОК достатньо зберігати в пам‘яті тільки величини *u*(*k-1*), *e*(*k*), *e*(*k-1*), *e*(*k-2*), тобто величини

$$u(k) = u(k-1) + \Delta u(k), \quad (6.9)$$

$$\Delta u(k) = q_0 e(k) + q_1 e(k-1) + q_2 e(k-2). \quad (6.10)$$

Таким чином, алгоритм роботи ПІД-регулятора може бути представлений у вигляді (6.9), (6.10), (6.6) – (6.8).

При переході від неперервних операторів до дискретних виникає похибка, величина якої пропорційна остаточному члену ряда Тейлора функції *e*(*t*). Тому отримані дискретні рівняння можна вважати еквівалентними неперервним тільки за умови, що *e*(*t*) змінюється слабо в межах такту дискретизації.

Однак за допомогою апарату z-перетворення можна показати, що основні властивості ПІД-регулятора зберігаються і при великих кроках дискретизації, якщо параметри регулятора q_0, q_1, q_2 вибирати не на основі параметрів його неперервного аналога (6.6) – (6.8), а незалежно від них, методами параметричної оптимізації, обравши необхідний критерій якості оптимізації, виходячи з цілей регулювання. Такт дискретизації вибирають аналогічно.

6.3 Контрольні питання

1. Нарисуйте функціональну схему цифрової САК.
2. Опишіть принцип роботи датчика Холла.
3. Наведіть класифікацію САК.
4. Визначте основні характеристики та параметри САК.
5. Охарактеризуйте відомі Вам методи настроювання ПІД-регуляторів.

6. За допомогою програми *Matlab Simulink* промоделюйте перехідні процеси в замкненій САК з ПІД-регулятором. Кількісні значення параметрів ОК та регулятора задайте приблизно.

6.4 Хід роботи

1. Вивчіть особливості побудови та функціювання цифрових САК.

2. Оформіть першу частину звіту з виконання лабораторної роботи, в якій дайте письмові відповіді на контрольні питання 1…5 (п. 6.3).

3. Безпосередньо у викладача отримайте допуск до виконання роботи в лабораторії та варіант завдання.

4. Розробіть алгоритми, складіть програми (згідно з варіантом), перевірте у симуляторі і продемонструйте їх викладачеві.

5. Уточніть кількісні значення параметрів ОК та регулятора (для замкненої системи) за результатами експериментів.

6. Перевірте роботу відлагодженої програми на стенді і продемонструйте її викладачеві.

6.5 Орієнтовні варіанти завдань

Таблиця 6.1 – Частина перша: дискретні задатчики САК

Варіант	*Зміст індивідуального завдання*
1	Підрахувати та відобразити на світлодіодах HL1-HL8 у двійковому коді кількість натискань на клавішу «5».
2	Реалізувати опитування клавіатури. Номер клавіші відобразити на правому знакомісці дисплея.
3	Реалізувати опитування клавіатури. Номер клавіші послідовно відображати в кожному знакомісці дисплея.
4	За натисканням клавіші «1» запускати вогонь що біжить на світлодіодах HL1-HL8 та засвітити число 1 на дисплеї. За натисканням клавіші «2» плавно засвітити число 2.
5	За натисканням клавіші «3» запустити секундомір з відображенням секунд на дисплеї. При відпусканні клавіші запускати “тінь, що біжить” на світлодіодах HL1-HL8.
6	Реалізувати опитування клавіатури. Номер клавіші відображати позиційним кодом на світлодіодах, а значення – на дисплеї.
7	Реалізувати програму введення чотиризначного числа з клавіатури та відображення його на дисплеї.
8	Реалізувати опитування клавіатури після 2-х натискань на SW16. Номер клавіші відобразити на лівому знакомісці дисплея.
9	Після натискання SW15 запустити програму введення трьохзначного числа з клавіатури та відображення його на дисплеї.
10	Реалізувати програму підрахунку імпульсів з енкодера з відображенням даних в десятковому коді на дисплеї.
11	Рахувати імпульси з енкодера та відображати на дисплеї. Після натискання на ручку запустити «вогонь, що біжить».
12	Рахувати імпульси з енкодера та відображати на дисплеї. Після натискання на ручку запустити «тінь, що біжить» на світлодіодах.
13	Рахувати імпульси з енкодера та відображати на дисплеї. Після натискання на ручку обнулити показання.
14	Рахувати імпульси з енкодера та відображати на дисплеї. Після натискання на ручку запустити програму опитування клавіатури з відображенням натиснутої клавіші на дисплеї.
15	Рахувати імпульси з енкодера та відображати на дисплеї. Після натискання на ручку запустити програму введення чотиризначного числа з клавіатури та відображення його на дисплеї.

Таблиця 6.2 – Частина друга: – Цифрові регулятори в САК

	Зміст індивідуального завдання
1	Реалізувати розімкнену САК з наступними характеристиками: 1) ОК – двигун; 2) задатчик частоти обертання – клавіатура; 3) відображення коду задання – на світлодіодах HL1-HL8; 4) відображення періоду обертання – на дисплеї.
2	Реалізувати розімкнену САК з наступними характеристиками: 1) ОК – двигун; 2) задатчик частоти обертання – енкодер; 3) відображення коду задання – на світлодіодах HL1-HL8; 4) відображення періоду обертання – на дисплеї.
3	Реалізувати розімкнену САК з наступними характеристиками: 1) ОК – лампа розжарювання; 2) задатчик температури – клавіатура; 3) відображення коду завдання температури – на світлодіодах; 4) відображення температури – на дисплеї.
4	Реалізувати розімкнену САК з наступними характеристиками: 1) ОК – лампа розжарювання; 2) задатчик температури – енкодер; 3) відображення коду завдання температури – на світлодіодах; 4) відображення температури – на дисплеї.
5	Реалізувати замкнену САК з наступними характеристиками: 1) ОК – двигун; 2) задатчик частоти обертання – клавіатура; 3) відображення коду задання – на світлодіодах HL1-HL8; 4) відображення періоду обертання – на дисплеї.
6	Реалізувати замкнену САК з наступними характеристиками: 1) ОК – двигун; 2) задатчик частоти обертання – енкодер; 3) відображення коду задання – на світлодіодах HL1-HL8; 4) відображення періоду обертання – на дисплеї.
7	Реалізувати замкнену САК з наступними характеристиками: 1) ОК – лампа розжарювання; 2) задатчик температури – клавіатура; 3) відображення коду завдання температури – на світлодіодах; 4) відображення температури – на дисплеї.
8/	Реалізувати замкнену САК з наступними характеристиками: 1) ОК – лампа розжарювання; 2) задатчик температури – енкодер; 3) відображення коду завдання температури – на світлодіодах; 4) відображення температури – на дисплеї.

Рекомендована література

1. Грищук Ю.С. Мікропроцесорні пристрої: навч. посібник / Ю.С.Грищук. – Харків : НТУ „ХПІ”, 2008. – 348 с. (Бібл. ЧДТУ)

2. Мікроконтролери сімейства MCS-51. Методичні вказівки до виконання лабораторних робіт з дисциплін «Мікропроцесорна техніка» та «Мікропроцесорні пристрої керування та обробки інформації» для студентів напряму підготовки 0908 – "Електроніка". – Чернігів: ЧДТУ, 2008. – 68с.

3. Однокристальные микроЭВМ/ А.В.Боборыкин, Г.П.Липовецкий, Г.В.Литвинский и др. М.: МИКАП, 1994. 400 с.

4. Система команд МК MCS-51. Див. на: \\Inel\archive\Kources\4_Курс\Мікропроцесорна_Техніка_Intel_Mcs-51\ASM-51doc_Система команд мікроконтролерів MCS-51.mht

5. Сташин В.В., Урусов А.В., Мологонцева О.Ф. Проектирование цифровых устройств на однокристальных микроконтроллерах. М.: Энергоатомиздат, 1990. 224 с.

6. AT89C52. 8-bit Microcontroller with 8K Bytes Flash. Datasheet. Rev. 0313H–02/00. © Atmel Corporation 2000. (Див. на: \\Inel\archive\Kources\4_Курс\Мікропроцесорна_Техніка_Atmel\MCS-51 architecture\doc0313.pdf)

7. Eclipse IDE for C/C++ Developers. (Доступно на: http://eclipse.org/)

8. Java (Доступно на: http://www.java.com)

9. SDCC Compiler User Guide (Доступно на: http://sdcc.sourceforge.net/)

10. Serial Digital Output Thermometers TMP03/TMP04. Datasheet. (Див. на: \\Inel\archive\Kources\4_Курс\Мікропроцесорні пристроі керування та обробки інформації\Datasheet\Thermo_tmp03_4.pdf)

11. sourceforge.eclipsesdcc (Доступно на: http://sourceforge.net/projects/eclipse-sdcc)

Додаток А
Деякі особливості компілятора SDCC

А.1 Розширення мови С для класів пам'яті (Storage Class Language Extensions)

У доповнення до ANSI класів пам'яті SDCC дозволяє наступні специфічні класи пам'яті MCS51:

А.1.1 __data / __near

Це – заданий за умовчанням клас пам'яті для малої моделі (Small). Змінні, що об'явлені з цим класом пам'яті, будуть розподілені в прямо адресуємій частині внутрішньої оперативної пам'яті 8051, наприклад:

```
__data unsigned char test_data;
```

Запис 0x01 у цю змінну сгенерує асемблерний код:

```
75*00 01  mov _test_data,#0x01
```

А.1.2 __xdata / __far

Змінні, що об'явлені з цим класом пам'яті, будуть розміщені у зовнішній оперативній пам'яті. Це – заданий за умовчанням клас пам'яті для моделі великого обсягу, наприклад:

```
__xdata unsigned char test_xdata;
```

Запис 0x01 у цю змінну сгенерує асемблерний код:

```
90s00r00  mov  dptr,#_test_xdata
74 01          mov  a,#0x01
F0         movx  @dptr,a
```

А.1.3 __idata

Змінні, що об'явлені з цим класом пам'яті, будуть розподілені в непрямо (indirectly) адресуємій частині внутрішньої оперативної пам'яті 8051, наприклад:

```
__idata unsigned char test_idata;
```

Запис 0x01 у цю змінну сгенерує асемблерний код:

```
78r00 mov r0,#_test_idata
76 01 mov @r0,#0x01
```

Зауважимо, що доступ до перших 128 байтів `idata` фізично означає звертання до тієї ж оперативної пам'яті класу `data`. У оригінальних 8051 було 128 байтів пам'яті `idata`. У більшості сучасних пристроїв є 256 байтів `idata` пам'ять. Стек розміщений в пам'яті `idata`.

А.1.4 __pdata

Сторінковий (paged) доступ `xdata` є настільки ж прямим, як і використання інших способів адресації 8051. Зазвичай дані розміщуються на початку `xdata` та мають максимальний обсяг 256 байтів. В наступному прикладі у змінну `pdata` записується `0x01`.

Зауважимо, що доступ до `pdata` фізично означає звертання до пам'яті класу `xdata`. Старший байт адреси визначається портом P2 (або окремим регістром спеціальних функцій для деяких варіантів 8051). Це – заданий за умовчанням клас пам'яті для моделі середнього (Medium) обсягу, наприклад:

```
__pdata unsigned char test_pdata;
```

Запис 0x01 у цю змінну сгенерує асемблерний код:

```
78r00          mov r0,#_test_pdata
74 01          mov a,#0x01
F2         movx @r0,a
```

Якщо використовується опція `--xstack`, область пам'яті `pdata` йде за областю пам'яті `xstack`, а сума їхніх обсягів обмежена 256 байтами.

A.1.5 __code

«Змінні», що об'явлені з цим класом пам'яті, будуть розміщені в пам'яті програм:

```
__code unsigned char test_code;
```

Доступ для читання цієї змінної сгенерує асемблерний код:

```
90s00r6F  mov dptr,#_test_code
E4        clr a
93        movc a,@a+dptr
```

Можливий ефективний доступ до індексованих масивів типу char в пам'яті програм:

```
__code char test_array[] = {'c','h','e','a','p'};
```

Доступ для читання цього масиву з використанням 8-бітного беззнакового індексу сгенерує асемблерний код:

```
E5*00     mov a,_index
90s00r41  mov dptr,#_test_array
93        movc a,@a+dptr
```

A.1.6 __bit

Це – специфікатор і типу даних, і класу зберігання. Коли змінну об'явлено як біт, вона розміщується у пам'яті 8051 з бітовою адресацією, наприклад:

```
__bit test_bit;
```

Запис 1 у цю змінну сгенерує асемблерний код:

```
D2*00     setb _test_bit
```

Пам'ять з бітовою адресацією складається з 128 бітів, які розміщуються у пам'яті даних з комірки 0x20 по 0x2f.

Крім цього специфічного для 8051 класу пам'яті більшість архітектур підтримують bitfields4 ANSI-C. Відповідно до ISO/IEC 9899 `bit` та `bitfields` без явно прописаного модифікатора розглядаються, як беззнакові.

A.1.7 __sfr / __sfr16 / __sfr32 / __sbit

Як і ключове слово `bit`, `__sfr` / `__sfr16` / `__sfr32` / `__sbit` позначають і

тип даних, і клас пам'яті. Вони використовуються для того, щоб описати регістри спеціальних функцій та спеціальні бітові змінні 8051, наприклад:

```
__sfr __at (0x80) P0; /* регістр спеціальних функцій P0 з адресою 0x80 */
/* 16-бітний регістр спеціальних функцій таймера 0, старший байт якого розміщений за адресою 0x8C, а молодший – 0x8A */
     __sfr16 __at (0x8C8A) TMR0;
__sbit __at (0xd7) CY; /*CY (прапор перенесення CarrY flag) */
```

Регістри спеціальних функцій, які розміщені за адресами, що діляться на 8, підтримують адресацію бітів з адресами `__sbit`, специфікованими всередині цих регістрів.

Комбінації 16- та 32-розрядних регістрів спеціальних функцій, що потребують певного порядку доступу, краще не об'являти, використовуючи `__sfr16` або `__sfr32`. Хоча SDCC зазвичай звертається до них, розпочинаючи з молодшого байту (LSB), це не гарантується.

Зауважимо, що, якщо використовуватиметься заголовковий файл, який був написаний для іншого компілятора, розширення класів пам'яті `sfr` / `sfr16` / `sfr32` / `sbit` вірогідно буде не сумісним. Наприклад, синтаксис `sfr P0 = 0x80;` компілюється без попередження SDCC щодо присвоєння `0x80` змінній з іменем `P0`. Тим не менш з файлом !compiler.h можливо написати файли заголовків, які можна було б використати для різних компіляторів (див. підрозділ 6.1, 3]).

A.2 Обробка переривань

A.2.1 Загальна інформація

SDCC за допомогою деяких розширених ключових слів дозволяє складати процедури обробки переривань мовою С.

```
void timer_isr (void) __interrupt (1) __using (1)
{
...
}
```

Опціональна цифра після ключового слова `__interrupt` – це номер переривання, яке обслуговує дана функція. Коли ця цифра наявна, компілятор вставить виклик цієї функції в таблицю векторів переривань для визначеного номера переривання.

Якщо Ваш проект складається з кількох вхідних файлів, процедури обробки переривань можуть бути розміщені в будь-якому з них, але прототип функції обробки ПОВИНЕН бути присутнім, або включеним у файл, який містить функцію `main`. Опціональне (специфічне для 8051) ключове слово `__using` може використовуватися, щоб змусити компілятор застосувати указаний банк регістрів під час генерації коду цієї функції.

Процедури обробки переривань можуть призвести до деяких досить цікавих помилок програмування.

A.2.1.1 Загальна пастка переривань: змінну не задекларовано, як volatile

Якщо процедура обробки переривання змінює змінні, до яких звертаються інші функції, ці змінні мають бути оголошені, як volatile. Див. http://en.wikipedia.org/wiki/Volatile variable.

A.2.1.2 Загальна пастка переривань: неатомарний доступ

Якщо доступ до цих змінних не є атомарним (тобто процесору потрібна більше, ніж одна машинна команда для доступу, який може бути перерваний), переривання має бути заблокованим під час доступу до змінної, щоб уникнути порушення даних.

Доступ до 16- або 32-розрядних змінних у 8-розрядному процесорі, певна річ, є не атомарним і має бути захищеним шляхом заборони переривань. Проте, навіть за умови використання 8-розрядних змінних, не завжди можна забезпечити коректну роботу програми. Розглянемо приклад:

Так, на 8051 ”`flags |= 0x80;`” виглядає безпроблемним. Проте доступ до змінної не є атомарним, якщо `flags` зберігається в `xdata`. Встановлення ”`flags |= 0x40;`” з середини функції переривання може бути втрачено, якщо переривання відбувається у невдалий момент часу.

”`counter += 8;`” не є атомарною операцією на 8051, навіть, якщо змінна `counter` розміщена в пам’яті даних.

Схожі помилки важко розпізнати, і вони можуть викликати великі проблеми.

A.2.1.3 Загальна пастка переривань: переповнення стеку

Адреса повернення та регістри, які використовуються в підпрограмі обробки переривання, зберігаються у стеку. Тому обсяг стеку має бути достатнім, інакше змінні, регістри (або навіть безпосередньо адреса повернення) будуть ушкоджені. Це переповнення стеку, найбільш вірогідно, виходить, якщо переривання відбувається під час "найбільш глибокої" підпрограми, коли стек вже використовується для багатьох адрес повернення.

A.2.1.4 Загальна пастка переривань: використання неінтерантних функцій

Спеціальне зауваження: операції цілочисельного ділення, множення та взяття модуля для змінних типів `int` (16 біт) та `long` (32 біт), а також операції з плаваючою точкою можна реалізувати, використовуючи зовнішні підпрограмі підтримки. Якщо процедура обробки переривання має зробити будь-яку з цих операцій, тоді підпрограми підтримки треба перекомпілювати, використовуючи опцію `--stack-auto`, а вхідний файл треба відкомпілювати з опцією компілятора - `int-long-reent`.

Зауважимо, що підтримка типу, яку потребує ANSI C, може зумовити появу 16-бітних підпрограм без відому програміста. Див., наприклад, приведення `(unsigned char)(tail-1)` в межах виразу `if` в пункті 3.13.2

[9].

Виклик інших функцій з процедури обробки переривання не рекомендується, уникайте цього, якщо можливо Зауважимо, що, коли деяка не повторновикористовна (not reentrant) функція викликається з процедури обробки переривання, перед нею має стояти `#pragma nooverlay`. Крім того, неповторновикористовні функції не мають викликатися з основної програми в той час, коли б могла активізуватися процедура обробки переривання. Їх також не можна викликати з низькопріоритетних процедур обробки переривання, в той час, коли б могла активізуватися процедура обробки переривання з вищим пріоритетом. Можна використати семафори або зробити функцію критичною (`__critical`), якщо всі параметри передаються в регістрах.

Також див. підрозділ 3.8 [9] про Overlaying та підрозділ 3.11 про функції, які використовують приватні регістрові банки.

А.2.2 Підпрограми обробки переривань для MCS51

Номери переривань, відповідні адреси та описи для стандарту 8051/8052 показані нижче. SDCC автоматично корегує адреси до максимального визначеного номеру переривання.

Таблиця А.1 – Номери переривань та відповідні адреси для MCS-51

Переривання #	**Опис**	**Адреса вектора**
0	External 0	0x0003
1	Timer 0	0x000b
2	External 1	0x0013
3	Timer 1	0x001b
4	Serial	0x0023
5	Timer 2 (8052)	0x002b
...		...
n		0x0003 + 8*n

Якщо процедуру обробки переривання буде визначено без використання `__using` для банку регістрів або з банком регістрів 0 (`__using 0`), то компілятор збереже використані регістри у стеку після входу та поверне їх при виході. Однак, якщо така процедура обробки переривання викличе іншу функцію, тоді весь регістровий банк буде збережений у стеку. Ця схема може бути вигідною для маленьких процедур обробки переривання, у яких регістри використовуються мало.

Якщо процедуру обробки переривання буде визначено з використанням певного банку регістрів, тоді зберігаються та повертаються тільки `a`, `b`, `dptr` та `psw`. Якщо така процедура обробки переривання викличе іншу функцію, тоді весь регістровий банк буде збережений у стеку. Ця схема рекомендується для великих процедур обробки переривання.

А.3 Дозвіл та заборона переривань

А.3.1 Критичні функції та критичні оператори

З блоком або функцією може бути асоційовано спеціальне ключове слово за допомогою об'явлення її як `__critical`. SDCC згенерує код, який заборонить усі переривання після входу в критичну функцію та верне дозволи у попередній стан перед поверненням. Вкладення критичних функцій буде потребувати один додатковий байт у стеку для кожного виклику.

```
int foo () __critical
{
...
...
}
```

Критичний атрибут можна використовувати з іншими атрибутами, наприклад, `reentrant`.

Ключове слово `__critical` також можна використати, щоб заборонити переривання більш локально:

```
__critical{ i++; }
```

В блок можна включити більше, ніж один оператор.

А.3.2 Прямий дозвіл та заборона переривань

Переривання можуть також бути заборонені або дозволені безпосередньо (8051):

```
EA = 0; або: EA_SAVE = EA;
...
EA = 1; ...
EA = EA_SAVE;
```

Зауважимо: іноді достатньо заборонити тільки конкретне джерело переривання, наприклад таймер або послідовний інтерфейс, маніпулюючи регістром маски переривань.

Зазвичай час, коли переривання заблоковані, має бути щонайкоротшим. Це мінімізує як час очікування переривання (інтервал між виникненням переривання та виконанням першої команди в підпрограмі переривання), так і дрижання фази переривання (*interrupt jitter* – різниця між самим коротким та самим довгим часом очікування переривання). Наприклад, переривання від послідовного інтерфейсу має бути обслугованим перед переповненням буферу.

Ви можете знову дозволяти переривання в межах підпрограми обслуговування, а в деяких архітектурах можна зробити два (або більше) рівня пріоритетів переривань. У деяких архітектурах, де не підтримуються пріоритети переривань, можна маніпулювати маскою переривань та знову дозволяти переривання в межах підпрограми обслуговування. Перевіряйте, чи достатній обсяг стеку, а також без потреби не ускладнюйте програму.

А.3.3 Семафорний захват (mcs51)

Деякі архітектури (mcs51) підтримують команди атомарної перевірки та скидання бітів. Подібні команди зазвичай використовують в витискаючих багатозадачних системах, де підпрограма, наприклад, потребує використання структури даних («отримує блокування на ній»), робить деякі модифікації, а надалі, коли структура даних знову стає цілісною, відпускає блокування. Подібна команда також може використовуватися, якщо переривання та безперервний код мають конкурувати за ресурс. З командами атомарної перевірки та скидання бітів переривання не повинні бути заборонені для операцій блокування.

SDCC генерує таку команду, якщо вхідний текст відповідає наступному шаблону:

```
volatile bit resource_is_free;
if (resource_is_free)
{
resource_is_free=0;
...
resource_is_free=1;
}
```

Зауважимо, що mcs51 підтримує тільки атомарну перевірку та очистку бітів (на противагу атомарній перевірці та встановленню).