МИНОБРНАУКИ РОССИИ

Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Юго-Западный государственный университет»
(ЮЗГУ)

Кафедра биомедицинской инженерии

Утверждаю
Проректор по учебной работе
О.Г. Локтионова
«25» 09 2023 г.

ЦИФРОВЫЕ ЭЛЕМЕНТЫ И МИКРОПРОЦЕССОРНЫЕ СИСТЕМЫ МЕДИЦИНСКОЙ ТЕХНИКИ

Методические рекомендации по выполнению самостоятельных работ для студентов направления подготовки 12.03.04 – «Биотехнические системы и технологии» (бакалавр)

Курск 2023

УДК 621.(076.1)

Составители: А.А.Кузьмин

Рецензент:
Кандидат технических наук, доцент *Т.Н. Конаныхина*

Цифровые элементы и микропроцессорные системы медицинской техники: методические рекомендации по выполнению самостоятельных работ для студентов направления подготовки 12.03.04 – «Биотехнические системы и технологии» (бакалавр) / Юго-Зап. гос. ун-т; сост.: А.А.Кузьмин. - Курск, 2023. – 65 с.

Содержат методические рекомендации к проведению самостоятельных работ по дисциплине «Цифровые элементы и микропроцессорные системы медицинской техники». Методические указания по структуре, содержанию и стилю изложения материала соответствуют методическим и научным требования, предъявляемым к учебным и методическим пособиям.

Предназначены для студентов направления подготовки 12.03.04 – «Биотехнические системы и технологии» (бакалавр)

Текст печатается в авторской редакции

Подписано в печать 25.09.23 Формат 60х84 1/16
Усо.печ.л. 3,8 . Уч.-изд.л. 3,4 .Тираж 30 экз. Заказ: 1078 .Бесплатно.
Юго-Западный государственный университет.
305040, г. Курск, ул. 50 лет Октября, 94

Самостоятельная работа №1

Система команд микроконтроллеров Microchip PIC16. Объявления переменных, пересылки, вычисления.

1 Краткие теоретические сведения

Микроконтроллеры семейства PIC16 имеют очень эффективную систему команд, состоящую всего из 35 инструкций. Все инструкции выполняются за один цикл, за исключением условных переходов и команд, изменяющих программный счетчик, которые выполняются за 2 цикла. Один цикл выполнения инструкции состоит из 4 периодов тактовой частоты. Таким образом, при частоте 4 МГц, время выполнения инструкции составляет 1 мксек. Каждая инструкция состоит из 14 бит, делящихся на код операции и операнд (возможна манипуляция с регистрами, ячейками памяти и непосредственными данными). Система команд микроконтроллеров PIC16 представлена в таблице 1.1.

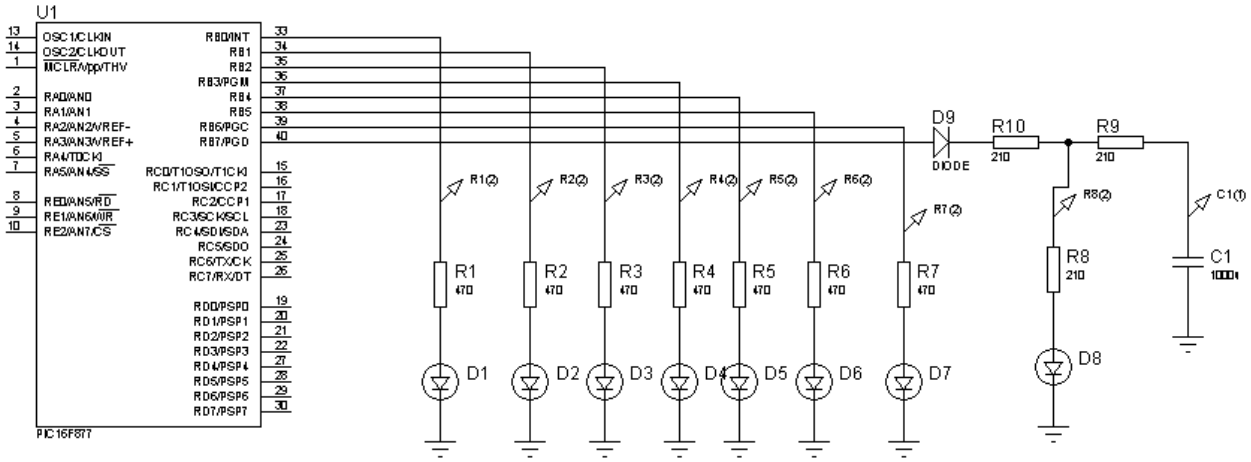

Рисунок 1.1 – Схема включения установки лабораторной работы №1 (осциллятор и питание микроконтроллера в схеме не показаны)

Принципиальная схема устройства индикации, которое мы будем использовать для демонстрации работы основных команд PIC16, приведена на рисунке 1.1. Устройство состоит из 8 светодиодов с токоограничивающими резисторами (диод D9 включен с параллельным светодиоду конденсатором для демонстрации эффекта «плавного» включения-выключения). Каждый вывод микроконтроллеров семейства PIC может непосредственно управлять светодиодом без дополнительных усилителей.

Начнем с описания базового кода, который будет использован в наших примерах. Весь код до строки с выражением ORG 0 условно называется секцией заголовка. В секции заголовка или определяются логические имена для всех используемых в проекте ресурсов - портов, битовых и байтовых переменных и регистров, или подключаются стандартные файлы определений. Например, можно непосредственно определить логические имена так:

```
; описание операционных регистров
TMR0          EQU     01h
PC            EQU     02h
STATUS        EQU     03h
FSR           EQU     04h
```

А можно включить стандартные описания командой

```
    #include p16f877.inc
```

Базовый код программы для микроконтроллера показан ниже:

```
;Назначение - ….
;Схема включения - ….
;Автор программы =…..

listp=16f877     ;Листинг для команд
                 ;микроконтроллера 16F877
#include p16f877.inc ;Определения для
                          ;микроконтроллера 16F877
;-----------------------------------------------
;Определение переменных
; Например так:
SCRATCH          EQU     0x20  ;Регистр
;-----------------------------------------------
org 0x00
nop      ;Первая операция - nop для
             ;внутрисхемных отладчиков
goto start   ;"Стандартное" начало

org 0x8
start        ;Метка начала программы
;-------Инициализация портов и т.д.--------------
bsf STATUS,RP0       ;Регистр TRISB не в
                    ;нулевой странице!
movlw 0x00       ;все выводы PORTB как выходы
movwf TRISB
bcf STATUS,RP0       ;Возвращаемся к
                        ;нулевой странице

;------Главный цикл-------------------------
;Здесь непосредственно программа
```

```
;-----------------------------------------
End       ;Конец программы
```

Все строки, начинающиеся со знака ";", воспринимаются ассемблером как комментарии. Разберем выражение «SCRATCH EQU 0x20». Ассемблер подставляет значение 0x20 всякий раз, когда встретится слово SCRATCH. Слово "EQU" означает эквивалентность. Таким образом, присваивается символам SCRATCH значение 0x20. Пользовательские регистры в микроконтроллере PIC16F877 начинаются с адреса 0x20. Использование символьных имен устраняет двусмысленность и позволяет облегчить чтение исходного текста.

Перед тем, как начать исполняемый код, необходимо задать выражение ORG 0. Это указатель для ассемблера, что код, следующий за этим выражением, начинается с нулевого адреса ЭППЗУ. Выражение "ORG" используется для размещения сегментов кода по различным адресам в пределах размеров ЭППЗУ. Еще одно выражение ORG находится перед меткой start, имеющей адрес 0x8, как задано выражением ORG 0x8. Исполняемый код должен заканчиваться директивой END, означающей, что за этой директивой отсутствуют исполняемые команды. При включении питания PIC16F877 переходит на адрес 000h. Первая инструкция, которая будет выполнена процессором, это команда NOP.

Таблица 1.1 – Список команд микроконтроллеров PIC16

Мнемоника команды		Описание	Циклов	14-разрядный код		Изм. флаги	Прим.
				Бит 13	Бит 0		
Байт ориентированные команды							
ADDWF	**f,d**	Сложение W и f	1	00 0111 dfff ffff		C,DC,Z	1,2
ANDWF	**f,d**	Побитное 'И' W и f	1	00 0101 dfff ffff		Z	1,2
CLRF	**f**	Очистить f	1	00 0001 1fff ffff		Z	2
CLRW	**-**	Очистить W	1	00 0001 0xxx xxxx		Z	
COMF	**f,d**	Инвертировать f	1	00 1001 dfff ffff		Z	1,2
DECF	**f,d**	Вычесть 1 из f	1	00 0011 dfff ffff		Z	1,2
DECFSZ	**f,d**	Вычесть 1 из f и пропустить если 0	1(2)	00 1011 dfff ffff			1,2,3
INCF	**f,d**	Прибавить 1 к f	1	00 1010 dfff ffff		Z	1,2
INCFSZ	**f,d**	Прибавить 1 к f и пропустить если 0	1(2)	00 1111 dfff ffff			1,2,3
IORWF	**f,d**	Побитное 'ИЛИ' W и f	1	00 0100 dfff ffff		Z	1,2
MOVF	**f,d**	Переслать f	1	00 1000 dfff ffff		Z	1,2
MOVWF	**f**	Переслать W в f	1	00 0000 1fff ffff			
NOP	**-**	Нет операции	1	00 0000 0xx0 0000			
RLF	**f,d**	Циклический сдвиг f влево через перенос	1	00 1101 dfff ffff		C	1,2
RRF	**f,d**	Циклический сдвиг f вправо через перенос	1	00 1100 dfff ffff		C	1,2
SUBWF	**f,d**	Вычесть W из f	1	00 0010 dfff ffff		C,DC,Z	1,2
SWAPF	**f,d**	Поменять местами полубайты в регистре f	1	00 1110 dfff ffff			1,2
XORWF	**f,d**	Побитное 'исключающее ИЛИ' W и f	1	00 0110 dfff ffff		Z	1,2
Бит ориентированные команды							
BCF	**f,b**	Очистить бит b в регистре f	1	01 00bb bfff ffff			1,2
BSF	**f,b**	Установить бит b в регистре f	1	01 01bb bfff ffff			1,2
BTFSC	**f,b**	Проверить бит b в регистре f, пропустить если 0	1(2)	01 10bb bfff ffff			3
BTFSS	**f,b**	Проверить бит b в регистре f, пропустить если 1	1(2)	01 11bb bfff ffff			3
Команды управления и операций с константами							
ADDLW	**k**	Сложить константу с W	1	11 111x kkkk kkkk		C,DC,Z	
ANDLW	**k**	Побитное 'И' константы и W	1	11 1001 kkkk kkkk		Z	
CALL	**k**	Вызов подпрограммы	2	10 0kkk kkkk kkkk			
CLRWDT	**-**	Очистить WDT	1	00 0000 0110 0100		-TO,-PD	
GOTO	**k**	Безусловный переход	2	10 1kkk kkkk kkkk			
IORLW	**k**	Побитное 'ИЛИ' константы и W	1	11 1000 kkkk kkkk		Z	
MOVLW	**k**	Переслать константу в W	1	11 00xx kkkk kkkk			
RETFIE	**-**	Возврат из подпрограммы с разрешением прерываний	2	00 0000 0000 1001			
RETLW	**k**	Возврат из подпрограммы с загрузкой константы в W	2	11 01xx kkkk kkkk			
RETURN	**-**	Возврат из подпрограммы	2	00 0000 0000 1000			
SLEEP	**-**	Перейти в режим SLEEP	1	00 0000 0110 0011		-TO,-PD	
SUBLW	**k**	Вычесть W из константы	1	11 110x kkkk kkkk		C,DC,Z	
XORLW	**k**	Побитное 'исключающее ИЛИ' константы и W	1	11 1010 kkkk kkkk		Z	

Примечания:

1. При выполнении операции "чтение - модификация - запись" с портом ввода/вывода исходные значения считываются с выводов порта, а не из выходных защелок. Например, если в выходной защелке было записана '1', а на соответствующем выходе низкий уровень сигнала, то обратно будет записано значение '0'.
2. При выполнении записи в TMR0 (и d=1) предделитель TMR0 сбрасывается, если он подключен к модулю TMR0.
3. Если условие истинно или изменяется значение счетчика команд PC, то инструкция выполняется за два цикла. Во втором цикле выполняется команда NOP.

Эта команда ничего не выполняет, и включение ее первой является требованием для использования внутрисхемных отладчиков. В конечном варианте программы эту команду можно исключить. Вторая команда - GOTO start, которая передает управление на адрес 0x8, и дальнейшая работа будет продолжаться с этого адреса. start - это выбираемое пользователем имя метки (метки всегда должны начинаться с первой позиции строки), которое ассемблер использует в качестве адресной ссылки. В процессе работы ассемблер определяет расположение метки start и запоминает, что если это имя будет встречено еще раз, вместо него будет подставлен адрес метки. Команды CALL и GOTO используют метки для ссылок в исходном тексте.

Следующая команда bsf STATUS,RP0 переключает страницы памяти, так как регистр TRISB, к которому мы хотим обратиться, находится в первой странице, а при включении микроконтроллера по умолчанию устанавливается нулевая страница памяти. Возврат к нулевой странице происходит после выполнения команды bcf STATUS,RP0. Команда movlw 0x00 загружает в рабочий регистр W значение, 0x00. Это значение может быть задано и в двоичном формате как B'00000000'.

Символы B' означают, что данные заданы в двоичном формате. Можно было бы написать в этом же месте 0h, 0x0 (шестнадцатеричный) и получить тот же самый результат. Десятичные значения должны начинаться с точки, т.е. число пятьдесят должно записываться как .50. По умолчанию, если число записывается без уточняющих формат символов, считается, что число записано в шестнадцатеричном формате! Эта особенность ассемблера является источником многочисленных ошибок!

Двоичное представление удобнее использовать в тех случаях, когда предполагается операция с битами в регистре. Следующая команда MOVWF TRISB загружает значение из рабочего регистра W в регистр управления конфигурацией порта B TRISB. Задание 0 в разряде этого регистра определяет, что соответствующий разряд порта B является выходом. В нашем случае все разряды порта B устанавливаются выходами. Если бы мы захотели, например, установить младший разряд порта B как вход, мы бы задали в регистре TRISB значение B'00000001'.

ПЕРВАЯ ПРОГРАММА

Для первой программы нам хватит всего трех команд:

```
MOVLW   k
MOVWF   f
GOTO    k
```

Мы уже использовали эти команды в заголовке нашего базового кода. Команда MOVLW загружает байтовый литерал или константу в рабочий регистр W. Следующая команда MOVWF пересылает байт из рабочего регистра W в заданный регистр f. Команда GOTO передает управление на адрес k. Следующая программа записывает в рабочий регистр W значение 01010101 и затем выдает его содержимое на порт B. После запуска этой программы Вы увидите свечение четырех светодиодов.

```
MOVLW   B'01010101'  ;загрузить 01010101
                      ;в регистр W
MOVWF   PORTB        ;записать W в порт B
GOTO    $            ;зациклиться навсегда
```

Директива ассемблера "$" означает текущее значение программного счетчика (PC). Поэтому команда GOTO $ означает переход туда, где мы в данный момент находимся. Такой цикл бесконечен, поскольку не существует способа (кроме прерывания) выйти из него. Команда GOTO $ часто применяется для остановки кода при отладке. В реальных программах для микроконтроллеров такая команда практически не применяется.

НАБОР КОМАНД PIC

Перейдем к описанию основного набора команд микроконтроллеров семейства PIC. Мы по-прежнему будем ориентироваться на PIC16F877, хотя почти все, о чем мы будем говорить, применимо и к другим микроконтроллерам семейства PIC. По ходу описания мы будем составлять короткие программы, чтобы лучше понять, как работают те или иные команды.

NOP

Начнем наше описание с команды NOP. Посмотреть результат выполнения этой команды трудно, поскольку она не делает ничего. Эта инструкция обычно используется в циклах временной задержки или для точной настройки времени выполнения определенного участка программы.

CLRW

Эта команда очищает рабочий регистр W. Добавим одну строчку в наш пример и увидим, что все светодиоды потухнут.

```
MOVLW   B'01010101' ;загрузить 01010101 в регистр W
CLRW                ;очистить регистр W
MOVWF   PORTB       ;записать W в порт B
GOTO    $           ;зациклиться навсегда
```

CLRF f

CLRF делает для любого регистра то же, что CLRW делает для рабочего регистра W. Следующая команда установит порт B в 0h.

```
CLRF    PORTB    ;очистить порт B (DATAPORT)
```

SUBWF f,d

Вычесть рабочий регистр W из любого регистра f. Эта команда также устанавливает признаки CARRY, DIGIT CARRY и ZERO в регистре STATUS. После выполнения команды можно проверить эти признаки и определить, является ли результат нулевым, положительным или отрицательным. Символ d после запятой означает адрес, куда будет помещен результат выполнения команды. Если d=0, то результат помещается в рабочий регистр W, а если d=1, то результат записывается в использованный в команде регистр f. Допускается также игнорировать написание символа d. В таком случае подразумевается, что d=1, т.е. результат поместится в регистр f. А вместо указания d=0 допускается писать символ W – так код выглядит намного нагляднее. Например, следующие две строчки эквивалентны:

```
SUBWF   SCRATCH,0    ;здесь надо помнить, что 0-это
                     ;признак рабочего регистра W
SUBWF   SCRATCH,W    ;Так намного нагляднее!
```

И эти две строчки тоже эквивалентны:

```
SUBWF   SCRATCH,1    ;здесь надо помнить, что 1-это
                     ;признак регистра-источника
SUBWF   SCRATCH      ;Так намного нагляднее!
```

В нашем примере в регистр SCRATCH загружается значение 0FFh, а в регистр W значение 01h. Затем выполняется команда SUBWF и результат отображается на светодиодах.

```
MOVLW   0FFh         ;загрузить 0FFh в регистр W
MOVWF   SCRATCH      ;загрузить содержимое W в
                     ;регистр SCRATCH
MOVLW   01h          ;загрузить 01h в регистр W
SUBWF   SCRATCH,W    ;выполнить вычитание
MOVWF   PORTB        ;записать W в порт B
GOTO    $            ;зациклиться навсегда
```

Светодиоды должны отобразить 11111110, где 0 соответствует потушенному светодиоду, а 1 - горящему.

Обратите внимание, что перед значением FFh в примере вычитания стоит "0". Символ "0" для ассемблера означает, что это число, а не метка. Если бы символа 0 не было, то ассемблер начал бы искать метку с именем FFh, которой в этой программе не существует и, соответственно, возникла бы ошибка. символ "h", сле-

дующий за значением 0FF, означает, что значение задано в шестнадцатеричном формате. Также шестнадцатеричный формат можно задавать как в языке Си в таком виде: 0xFF

ADDWF f,d

Команда ADDWF работает аналогично SUBWF f,d, прибавляя рабочий регистр W к любому регистру f и устанавливая те же признаки. Следующий пример демонстрирует работу команды ADDWF.

```
MOVLW   0h          ;загрузить 0 в регистр W
MOVWF   SCRATCH     ;загрузить содержимое W
                   ;в регистр SCRATCH
MOVLW   1h          ;загрузить 01h в регистр W
ADDWF   SCRATCH,W   ;выполнить сложение
MOVWF   PORTB       ;записать W в порт B
GOTO    $           ;зациклиться навсегда
```

Светодиоды должны отобразить 00000001

SUBLW k

ADDLW k

Эти две команды работают совершенно аналогично вышеописанным, за тем исключением, что операция производится между рабочим регистром W и байтовой константой, заданной в команде. Команда SUBLW вычитает рабочий регистр W из константы k, а команда ADDLW добавляет рабочий регистр W к константе k. Эти команды также устанавливают признаки CARRY, DIGIT CARRY и ZERO. Результат выполнения команды помещается в рабочий регистр W. Следующий пример уменьшит SCRATCH на 5.

```
MOVLW   0FFh        ;загрузить 0FFh в регистр W
MOVWF   SCRATCH     ;загрузить содержимое W
                   ; в регистр SCRATCH
SUBLW   05h         ;вычесть 5 из рабочего регистра
MOVWF   SCRATCH     ; записать W в регистр SCRATCH
MOVWF   PORTB       ;записать W в порт B
GOTO    $           ;зациклиться навсегда
```

Светодиоды должны отобразить 11111010.

DECF f,d

INCF f,d

Команда DECF уменьшает заданный регистр на 1, а INCF увеличивает заданный регистр на 1. Результат может быть помещен обратно в заданный регистр (при d=1) либо в рабочий регистр W (при d=0). В результате выполнения этих команд может установиться признак ZERO в регистре STATUS. Вот пример использования этих команд:

```
MOVLW   0FFh          ;загрузить 0FFh в регистр W
MOVWF   SCRATCH       ;загрузить содержимое W
                     ;в регистр SCRATCH
DECF    SCRATCH  ;уменьшить SCRATCH на 1
```

Этот пример увеличит SCRATCH с 0 до 1.

```
CLRF    SCRATCH          ;очистить SCRATCH
INCF    SCRATCH       ;увеличить SCRATCH на 1
```

IORWF f,d

ANDWF f,d

XORWF f,d

Эти три команды выполняют логические действия ИЛИ, И и ИСКЛЮЧАЮЩЕЕ ИЛИ. Операция логического сложения ИЛИ чаще всего используется для установки отдельных битов в регистрах. Сбрасываются эти биты затем операцией логического умножения И. Когда над одинаковыми битами выполняется операция ИСКЛЮЧАЮЩЕЕ ИЛИ, результат равен 0. Поэтому операция ИСКЛЮЧАЮЩЕЕ ИЛИ часто используется для проверки состояния (установлены или сброшены) определенных бит в регистре. Следующая процедура установит бит 1 в порте B при помощи команды IORWF:

```
MOVLW   B'00000000'  ;загрузить 0h в регистр W
MOVWF   PORTB        ;установить все биты в порте B
MOVLW   B'00000010'  ;установить маску в регистре W
IORWF   PORTB    ;установить биты в порте B
                    ; по маске W
GOTO    $               ;зациклиться навсегда
```

Светодиоды должны показать 00000010. А теперь сбросим 2 бита при помощи команды ANDWF:

```
MOVLW   B'11111111'  ;загрузить 0FFh в регистр W
MOVWF   PORTB        ;установить все биты в порте B
MOVLW   B'00000101'  ;установить маску в регистре W
ANDWF   PORTB  ,1    ;очистить биты в порте B
                    ; по маске W
GOTO    $               ;зациклиться навсегда
```

Светодиоды должны показать 00000101. Поэкспериментируйте с начальными значениями в порту B и со значениями маски.

Предположим, что мы использовали регистр SCRATCH и хотим знать, равен ли он значению 04h. Это удобный случай использовать команду XORWF:

```
MOVLW   04h         ;загрузить 04h в регистр W
MOVWF   SCRATCH     ;загрузить регистр W в SCRATCH
XORWF   SCRATCH,W   ;проверить равенство W и SCRATCH
MOVWF   PORTB       ;записать W в порт B
GOTO    $           ;зациклиться навсегда
```

Поскольку SCRATCH и W равны, результат выполнения операции XORWF равен нулю (все светодиоды потухли). В регистре STATUS установится бит ZERO, который реальная программа затем может проверить и обработать.

IORLW k

ANDLW k

XORLW k

Эти три команды выполняют те же действия, что и их вышеописанные аналоги, за тем исключением, что операция производится между рабочим регистром W и маской, заданной в команде. Результат выполнения команды помещается в рабочий регистр W. Например:

```
MOVLW   0FFh  ;загрузить 0FFh в регистр W
ANDLW   040h  ;оставить 6-й бит
MOVWF   PORTB      ;Светодиоды покажут 01000000.

MOVLW   10h     ;загрузить 10h в регистр W
IORLW   09h  ;установить 0-й и 3-й биты
MOVWF   PORTB      ;Светодиоды покажут 01000000.
MOVLW   B'00100000'     ;загрузить 40h в регистр W
XORLW   B'11111111'     ;проинвертировать W
MOVWF   PORTB      ;Светодиоды покажут 01000000.
```

MOVF f,d

Эта команда в основном используется для пересылки регистра в рабочий регистр W (d=0). Если же установить d=1, то эта команда загрузит регистр сам в себя, но при этом бит ZERO в регистре STATUS установится в соответствии с содержимым регистра. Например, мы хотим загрузить в регистр SCRATCH 0Fh, а потом загрузить регистр SCRATCH в рабочий регистр W.

```
MOVLW   0Fh      ;загрузить 0Fh в рабочий регистр W
MOVWF   SCRATCH  ;загрузить регистр W в SCRATCH
CLRW             ;очистить W
MOVF    SCRATCH,W ;загрузить SCRATCH в регистр W
```

Если в процессе выполнения программы мы хотим проверить регистр SCRATCH на ноль, мы можем выполнить следующую команду:

```
MOVF    SCRATCH
```

Бит ZERO регистра STATUS будет установлен, если условие будет выполнено (SCRATCH = 0h).

COMF f,d

Эта команда инвертирует любой заданный регистр. При d=0 результат заносится в рабочий регистр W, а при d=1 инвертируется

содержимое заданного регистра. В качестве примера проинвертируем значение 01010101:

```
MOVLW   B'01010101' ;загрузить 01010101 в регистр W
MOVWF   SCRATCH     ;загрузить регистр W в SCRATCH
COMF    SCRATCH,W   ;инвертировать SCRATCH
MOVWF   PORTB       ;записать W в порт B
GOTO    $           ;зациклиться навсегда
```

Светодиоды покажут 10101010.

2. Цель работы

Целью работы является приобретение навыков в написании простейших программ для микроконтроллеров PIC16.

3. Порядок выполнения работы

3.1 Написать программу, которая решает вычислительную задачу, согласно выданному варианту.

3.2 Промоделировать работу микроконтроллера с разработанной программой на эмуляторе микроконтроллера.

3.3 Составить контрольные примеры и вычислить при помощи ЭВМ контрольные результаты. По результатам проверить правильность выполнения разработанной программы.

4. Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя.

Цель работы.

Задание на лабораторную работу.

Контрольные примеры, результат расчета контрольных примеров на ЭВМ и на эмуляторе микроконтроллера.

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 Напишите подпрограмму, которая при частоте работы микроконтроллера 4МГц, выполнялась бы ровно 10 микросекунд.

5.2 Напишите программу, которая настраивает первые четыре вывода порта B на вывод информации, а остальные четыре вывода на ввод.

5.3 Напишите программу, которая вводит в рабочий регистр W число двадцать.

5.4 Напишите программу, которая складывает два числа.

5.5 Напишите программу, которая вычитает два числа.

5.6 Напишите программу, которая сбрасывает три младших бита числа в ноль.

5.7 Напишите программу, которая устанавливает три старших бита числа в единицу.

5.8 Напишите программу, которая инвертирует три старших бита числа.

Самостоятельная работа №2

Система команд микроконтроллеров Microchip. Подпрограммы, циклы, ветвления.

1 Краткие теоретические сведения

Продолжим разбирать основные команды микроконтроллеров Microchip.

DECFSZ f,d

INCFSZ f,d

При d=1 команда DECFSZ уменьшает на единицу, а INCFZ увеличивает на единицу заданный регистр и пропускает следующую команду, если регистр стал равен нулю. При d=0 результат записывается в регистр W и следующая команда пропускается, если рабочий регистр W стал равным нулю. Эти команды используются для формирования временных задержек, счетчиков, циклов и т.д.

Вот типичный пример использования цикла:

```
    START
    MOVLW   0FFh            ;загрузить FFh в регистр W
    MOVWF     SCRATCH             ;загрузить регистр W в
SCRATCH
    LOOP
    DECFSZ  SCRATCH,1       ;уменьшать SCRATCH на 1
    GOTO    LOOP            ;и переходить обратно, пока
                            ; не станет = 0
    MOVF    DIGIT,W         ;загрузить регистр DIGIT в W
    MOVWF   PORTB           ;вывести на светодиоды
    DECF    DIGIT,1         ;уменьшить регистр DIGIT на 1
    GOTO    START           ;перейти на начало
```

В результате светодиоды будут мигать с различной частотой. Светодиод младшего разряда будет мигать чаще всего, а светодиод старшего разряда реже всего. При тактовой частоте 4 МГц частота миганий светодиода старшего разряда будет примерно 8 Гц, а каждый следующий будет мигать вдвое чаще. Теперь разберемся, как это у нас получилось. Команда DECFSZ здесь работает в цикле задержки, состоящем из двух команд - DECFSZ и GOTO LOOP. Поскольку мы предварительно загрузили в регистр SCRATCH значение 0FFh, этот цикл выполнится 255 раз, пока SCRATCH не станет равным нулю. При тактовой частоте 4 МГц это дает задержку 1 мксек/команду * 2 команды * 255 = 510 мксек. В регистр DIGIT мы предварительно ничего не записывали, поэтому там могло находиться любое значение, которое и выво-

дится на светодиоды на первом проходе. Затем регистр DIGIT уменьшается на 1 и цикл повторяется сначала. В результате регистр DIGIT перебирает все значения за 256 циклов, т.е. за примерно за 130 мсек. Тот же код можно использовать и с командой INCFSZ, заменив загружаемое в регистр SCRATCH значение с FFh на 0h. Светодиоды будут мигать точно так же и если заменить команду DECF на команду INCF.

SWAPF f,d

Эта команда меняет местами полубайты в любом регистре. Как и для других команд, при d=0 результат записывается в рабочий регистр W, а при d=1 остается в регистре. Вот простой пример использования этой команды:

```
    MOVLW   B'00001111'   ;загрузить 0Fh в регистр W
    MOVWF     SCRATCH           ;загрузить регистр W в
SCRATCH
    SWAPF   SCRATCH,0     ;поменять полубайты
```

Светодиоды покажут 11110000.

RRF f,d

RLF f,d

В ассемблере PIC имеется две команды сдвига - сдвиг вправо через бит CARRY любого регистра RRF и сдвиг влево через бит CARRY любого регистра RLF. Как и для других команд, при d=0 результат сдвига записывается в регистр W, а при d=1 остается в регистре. Инструкции сдвига используются для выполнения операций умножения и деления, для последовательной передачи данных и для других целей. Во всех случаях бит, сдвигаемый из 8-битного регистра, записывается в бит CARRY в регистре STATUS, а бит CARRY записывается в другой конец регистра, в зависимости от направления сдвига. При сдвиге влево RLF CARRY записывается в младший бит регистра, а при сдвиге вправо RRF CARRY записывается в старший бит регистра.

```
    CLRF    STATUS        ;очистить регистр STATUS
    MOVLW   0FFh          ;загрузить 0FFh в регистр W
    MOVWF     SCRATCH           ;загрузить регистр W в
SCRATCH
    RRF     SCRATCH,0     ;сдвинуть вправо
```

Светодиоды должны показать 01111111, поскольку CARRY загрузился в старший бит. Теперь сдвинем влево:

```
    CLRF    STATUS       ;очистить регистр STATUS
    MOVLW   0FFH         ;загрузить 0FFh в регистр W
    MOVWF   SCRATCH      ;загрузить регистр W в SCRATCH
```

```
RLF     SCRATCH,1    ;сдвинуть влево
```

Светодиоды должны показать 11111110.

BCF f,b
BSF f,b

Команды очистки бита BCF и установки бита BSF используются для работы с отдельными битами в регистрах. Параметр b означает номер бита, с которым производится операция, и может принимать значения от 0 до 7. Попробуем включить светодиод, используя команду BCF:

```
MOVLW   0h            ;загрузить 0h в регистр W
MOVWF   PORTB         ;выключить светодиоды
BSF     PORTB ,7      ;установить бит 7 в порте B
GOTO    $             ;зациклиться навсегда
```

В результате загорится светодиод, соответствующий биту 7. Вспомните, мы делали аналогичные вещи при помощи использования маски и команды ANDWF. Разница в том, что команды ANDWF и IORWF требуют предварительного формирования маски и хранения ее в каком-либо регистре, но в то же время способны одновременно установить или очистить несколько бит. Команды же BCF и BSF оперируют только с одним битом. Кроме того, команды BCF и BSF не изменяют регистр состояния STATUS, поэтому они часто используются в тех случаях, когда не требуется последующая проверка регистра состояния.

BTFSC f,b
BTFSS f,b

Команды условных переходов BTFSC и BTFSS проверяют состояние заданного бита в любом регистре и в зависимости от результата пропускают или нет следующую команду. Команда BTFSC пропускает команду, если заданный бит сброшен, а команда BTFSS - если установлен. Вот простой пример:

```
MOVLW   0h            ;загрузить 0h в регистр W
MOVWF   DATAPORT      ;выключить светодиоды
LOOP
BTFSS   PORTA,0    ;проверить бит 0 в PORTA
GOTO    LOOP       ;ждать, пока бит 0 не установит-
ся
BSF     DATAPORT,7  ;включить светодиод
GOTO    $           ;зациклиться навсегда
```

В этом примере проверяется разряд 0 порта A (вывод 17 микросхемы) и, если этот вывод установлен в высокий уровень, включается светодиод. Попробуйте заменить BTFSS на BTFSC в этом примере. Светодиод будет включаться, когда разряд 0 порта A установится в низкий уровень. Ранее мы упоминали о возможности проверки битов состояния в регистре STATUS. Это также делается при помощи команд BTFSS и BTFSC:

```
                    ;Проверка бита CARRY
    BTFSS     STATUS,C  ;если  C  установлен,  пропустить
GOTO
    GOTO    WHERE_EVER
```

Аналогично проверяется бит ZERO:

```
    ;Проверка бита ZERO
    BTFSS     STATUS,Z  ;если  Z  установлен,  пропустить
GOTO
    GOTO    WHERE_EVER
```

Проверка битов Z и C в регистре STATUS позволяет определить факт равенства содержимого двух регистров (определяется по биту Z), а также того, что содержимое одного регистра больше или меньше содержимого другого (определяется по биту C).

CALL k
RETURN

Эти две команды предназначены для работы с подпрограммами. Команда CALL используется для перехода на подпрограмму по адресу, задаваемому в команде, а команда RETURN - для возврата из подпрограммы. Обе команды выполняются за 2 цикла. Адрес, на котором находилась команда CALL, запоминается в специально организованных регистрах, называемых стеком. Эти регистры недоступны для обращений и используются только при вызовах подпрограмм и возвратах. Глубина стека, т.е. число специальных регистров - 8. Поэтому из основной программы можно сделать не более 8 вложенных вызовов подпрограмм. После возврата из подпрограммы выполнение продолжается со следующей после CALL команды. Регистр W и регистр STATUS при вызове подпрограммы не сохраняются, поэтому, если необходимо, их можно сохранить в отдельных ячейках памяти. Вот простой пример использования подпрограммы:

```
START
BCF     DATAPORT,7      ;выключить светодиод
CALL    TURNON          ;вызвать подпрограмму
GOTO    START           ;перейти на начало
TURNON
BSF     DATAPORT,7      ;включить светодиод
RETURN                  ;вернуться из подпрограммы
```

В результате светодиод будет мигать с частотой около 150 кГц.

RETLW k

RETFIE

Существуют еще две команды, предназначенные для возврата из подпрограмм. Команда RETLW возвращает в рабочем регистре W константу, заданную в этой команде, а команда RETFIE разрешает прерывания. Команда RETLW часто используется для создания таблиц значений. Пусть в рабочем регистре W содержится смещение от начала таблицы. Тогда получить нужный элемент можно следующей процедурой:

```
    MOVLW   02h         ;задать смещение
    CALL    SHOWSYM     ;вызвать подпрограмму
    MOVWF    PORTB       ;вывести элемент таблицы в порт
B
    GOTO    $           ;зациклиться навсегда
    SHOWSYM
    ADDWF   PC        ;вычислить смещение в таблице
                      ;со счетчиком команд
    RETLW   0AAh      ;1-й элемент таблицы
    RETLW   0BBh      ;2-й элемент таблицы
    RETLW   0CCh      ;3-й элемент таблицы
```

Светодиоды должны отобразить 10111011 (0BBh).

СПЕЦИАЛЬНЫЕ КОМАНДЫ

Осталось упомянуть о двух специальных командах - **CLRWDT** и **SLEEP**. Команда CLRWDT предназначена для сброса сторожевого таймера, назначение которого мы уже обсуждали. Эта команда должна присутствовать в таких участках программы, чтобы время выполнения программы между двумя соседними командами CLRWDT не превышало времени срабатывания сторожевого таймера.

Команда SLEEP предназначена для перевода процессора в режим пониженного энергопотребления. После выполнения этой команды тактовый генератор процессора выключается и обратно в рабочий режим процессор можно перевести либо по входу сброса, либо по срабатыванию сторожевого таймера, либо по прерыванию.

2 Цель работы

Целью работы является усовершенствование навыков в написании программ для микроконтроллеров PIC16.

3 Порядок выполнения работы

3.1 Написать программу, которая решает задачу, согласно выданному варианту.

3.2 Промоделировать работу микроконтроллера с разработанной программой на эмуляторе микроконтроллера.

3.3 Составить контрольные примеры и вычислить при помощи ЭВМ контрольные результаты. По результатам проверить правильность выполнения разработанной программы.

4 Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя.

Цель работы.

Задание на лабораторную работу.

Контрольные примеры, результат расчета контрольных примеров на ЭВМ и на эмуляторе микроконтроллера.

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 Напишите подпрограмму умножения двух чисел, используя команды цикла DECFSZ f,d или INCFSZ f,d и команду сложения.

5.2 Напишите подпрограмму целочисленного деления числа на четыре, используя команды сдвига.

5.3 Как программно определить остаток от целочисленного деления числа на два?

5.4 Напишите подпрограммы очистки двух младших битов в регистре с использованием битовой маски и логических команд, и с помощью команды BCF. Какая из этих подпрограмм эффективнее?

5.5 Напишите подпрограмму проверки равенства содержимого двух регистров.

5.6 Напишите подпрограмму проверки того, что содержимое регистра reg1 больше содержимого регистра reg2.

5.7 Напишите подпрограмму проверки того, что содержимое регистра reg1 меньше содержимого регистра reg2.

Самостоятельная работа №3

Видеовывод в микроконтроллерных устройствах. Реализация знакогенератора.

1 Краткие теоретические сведения

Знакогенератор выполняет функцию замены кода символа его графическим изображением и выводом этого изображения на средство отображения информации (в нашем случае – ЖКИ). Графические изображения символов, как правило, хранятся в статических массивах. В микроконтроллерах Microchip (без использования дополнительных микросхем) статические массивы можно хранить в памяти программ и в EEPROM.

Для организации статического массива в EEPROM в микроконтроллере PIC16F877 достаточно объявить массив байт по адресу 0x2100 в листинге. Т.е. для сохранения в EEPROM массива из элементов [0,1,2,3,4,5] необходимо написать следующий кусок кода (как правило, в конце программы перед директивой END):

```
org 0x2100                  ; Инициализация EEPROM
de 0x0,0x1,0x2,0x3,0x4,0x5    ;массив
```

Заметим, что существуют процедуры записи в EEPROM во время выполнения программы.

Для чтения из EEPROM памяти необходимо только записать адрес в регистр EEADR и сбросить бит EEPGD в '0'. После установки в '1' бита RD данные будут доступны в регистре EEDATA на следующем машинном цикле. Данные в регистре EEDATA сохраняются до выполнения следующей операции чтения или записи в EEDATA.

Рекомендованная последовательность действий при чтении из EEPROM памяти данных:

1. Записать адрес в регистр EEADR. Проверьте, что записанный адрес корректен для данного типа микроконтроллера.

2. Сбросить в '0' бит EEPGD для обращения к EEPROM памяти данных.

3. Инициализировать операцию чтения установкой бита RD в '1'.

4. Прочитать данные из регистра EEDATA.

Следующая процедура читает данные из EEPROM:

```
;-------------------------------------------------
;Функция работы с EEPROM
;-------------------------------------------------
;Вход - регистр eepcount – адрес ячейки
```

```
;Выход - значение eeprom в W
;
geteeprom
    MOVF eepcount,W  ; Записать адрес
    BSF STATUS,RP1   ;
    BCF STATUS,RP0   ; Выбрать банк 2
    MOVWF EEADR      ; ячейки
    BSF STATUS,RP0   ; Выбрать банк 3
    BCF EECON1,EEPGD     ; Выбрать EEPROM память
    BSF EECON1,RD        ; Инициализировать чтение
    BCF STATUS,RP0   ; Выбрать банк 2
    MOVF EEDATA,W        ; W = EEDATA
    BCF STATUS,RP1   ;
 return
```

Для организации статического массива в памяти программ в микроконтроллере PIC16F877 существует несколько способов. Во-первых, для чтения и записи 14-битных значений памяти программ можно использовать процедуры, основанные на работе с регистрами EEADRH:EEADR и EEDATH:EEDATA (см. документацию).

А для организации статических 8-битных массивов можно воспользоваться процедурой, основанной на команде RETLW.

Для этого объявляют подпрограмму следующего вида:

```
str0              ;имя подпрограммы-массива
    retlw "P"         ;здесь могут быть любые
    retlw "A"         ;байтовые данные
    retlw "Б"
    retlw "O"
    retlw "T"
    retlw "A"
    retlw 0
```

А читают данные из такой подпрограммы с помощью процедуры:

```
;---------------------------------------
;Процедура доступа к данным в памяти программ
;Использует регистры tptrh и tptrl - указатель на
;объявленный статический массив
;вызов:
;movlw   str0 / 0x100     ;Настройка указателей
;movwf   tptrh            ;Старший байт
;movlw   str0 % 0x100
;movwf   tptrl            ;Младший байт
;call    getdata          ;Нулевой элемент - в W
;.........................
;incf    tptrl       ;Переходим к следующему элемен-
ту
```

```
;call    getdata              ;Второй элемент – в W

; код подпрограммы:
getdata
   movf    tptrh,W    ; Настройка  PCLATH для прыж-
ка
   movwf   PCLATH
   movf    tptrl,W    ; Младший адрес указателя
   movwf   PCL        ;Пересылаем в PCL, совершая
                  ;тем самым переход на инструкцию
                  ; retlw
```

Разработаем изображение символа «А» размером 5x8. Для этого составим следующую сетку:

	52	8	8	52

Где пустые значения соответствуют нулям, единицы – закрашенным точкам, в нижней строке вычислен (согласно правилам отображения данных в ЖКИ МТ-6116) десятичный код столбца, которому соответствуют данные нули и единицы. Т.е. если в память ЖКИ переслать последовательно цифры 0, 252, 18, 18, 252, то на экране нарисуется символ «А». Таким образом, изображение символа «А» можно записать в EEPROM:

```
org 0x2100              ; Инициализация EEPROM
de 0,.252,.18,.18,.252
```

Или изображение можно записать в память программ:

```
Znaki          ;имя подпрограммы-массива
    retlw 0
    retlw .252
    retlw .18
    retlw .18
    retlw .252
```

2 Цель работы: научиться хранить и использовать статические массивы информации (калибровочная информация, знакогенераторы и т.п.) в памяти EEPROM и в памяти программ; реализовывать на низком уровне вывод символьной информации на ЖКИ.

3 Порядок выполнения работы

3.1 Решить практическую задачу по программированию внешнего контроллера, обеспечивающего графический вывод на LCD дисплей, согласно своему варианту.

3.2 Проверить работоспособность разработанной программы в системе Proteus.

3.3 Проверить работоспособность разработанной программы на лабораторном стенде.

4 Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя

Цель работы

Задание на лабораторную работу

Контрольные примеры, результат выполнения микроконтроллером контрольных примеров.

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 Какие значения необходимо переслать в видеопамять, чтобы получить изображение буквы «Б»?

5.2 Напишите программу для хранения в EEPROM памяти изображения буквы «Б».

5.3 Напишите программу для хранения в памяти программ изображения буквы «Б».

5.4 Изображения букв в EEPROM памяти хранятся последовательно, причем каждая буква занимает 5 байт. Напишите подпрограмму, которая по коду буквы пересылает в видеопамять 5 байт изображения буквы.

5.5 Напишите алгоритм вывода изображения содержимого регистра в десятичном виде.

5.6 Напишите алгоритм вывода изображения содержимого регистра в двоичном виде.

5.7 Напишите алгоритм вывода изображения отрицательного содержимого регистра в десятичном виде.

5.8 Какова минимальная ширина букв в пикселах для реализации русского знакогенератора?

5.9 Сколько минимально необходимо байт памяти для реализации русского знакогенератора с постоянным количеством байт изображения на символ?

5.10 Как можно построить знакогенератор с переменным количеством байт изображения на символ?

5.11 Сколько минимально необходимо байт памяти для реализации русского знакогенератора с переменным количеством байт изображения на символ?

Самостоятельная работа №4
Программирование генератора аналоговых сигналов произвольной формы на основе таймера.

1 Краткие теоретические сведения

Программирование генератора аналоговых сигналов необходимо начинать с процедур обмена с ЦАП. У каждой микросхемы и схемы подключения будут свои процедуры обмена. Написание процедур обмена микроконтроллера с ЦАП, как правило, не сложнее, чем с ЖКИ и подобными устройствами.

При готовых процедурах обмена с ЦАП программирование вывода аналоговых сигналов не представляет собой сложной задачи – достаточно загружать цифровые значения, соответствующие отсчетам генерируемого сигнала, в регистр ЦАП. Одной из проблем, которую придется решать разработчику такого генератора, является точное задание частоты дискретизации сигнала.

Для задания временных интервалов дискретизации можно использовать программные задержки, как это было показано в предыдущих лабораторных работах, т.е. интервал дискретизации делился бы на две основные части – часть вывода информации в ЦАП и часть программной задержки. Понятно, что для обеспечения заданного интервала дискретизации величина программной задержки напрямую зависит от длительности процедуры вывода информации в ЦАП. При изменении микросхемы ЦАП или, в общем, при изменении процедуры вывода информации в ЦАП, величину программной задержки каждый раз необходимо было бы изменять. Поэтому более эффективный путь для формирования точной частоты дискретизации сигнала, который лишен этих недостатков – это использование таймеров.

В микроконтроллере PIC16F877 есть три таймера – TMR0, TMR1, TMR2. Для примера разберем механизмы работы с таймером TMR0.

TMR0 с точки зрения программиста – это просто 8-ми разрядный регистр, который может увеличивать свое значение по определенному фронту внешнего для микроконтроллера сигнала (и тогда он работает просто как счетчик внешних импульсов), или может увеличивать свое значение по сигналу от внутреннего тактового генератора (и тогда он работает как таймер или формирователь временных интервалов).

TMR0 – таймер/счетчик, имеет следующие особенности:

· 8-разрядный таймер/счетчик;

· Возможность чтения и записи текущего значения счетчика;
· 8-разрядный программируемый предделитель;
· Внутренний или внешний источник тактового сигнала;
· Выбор активного фронта внешнего тактового сигнала;
· Прерывания при переполнении (переход от FFh к 00h).

Когда бит T0CS сброшен в ‘0’ (OPTION_REG<5>), TMR0 работает от внутреннего тактового сигнала. Приращение счетчика TMR0 происходит в каждом машинном цикле, т.е. с частотой Fosc/4 (если предделитель отключен). После записи в регистр TMR0 приращение счетчика запрещено два следующих цикла. Пользователь должен скорректировать эту задержку перед записью нового значения в TMR0.

Если бит T0CS установлен в ‘1’ (OPTION_REG<5>), TMR0 работает от внешнего источника тактового сигнала с входа RA4/T0CKI. Активный фронт внешнего тактового сигнала выбирается битом T0SE в регистре OPTION_REG<4> (T0SE=0 – активным является передний фронт сигнала).

Предделитель может быть включен перед сторожевым таймером WDT или перед таймером TMR0, в зависимости от состояния бита PSA (OPTION_REG<3>). Использование предделителя пред TMR0 означает, что WDT работает без предделителя, и наоборот.

Коэффициент деления предделителя определяется битами PSA и PS2:PS0 в регистре OPTION_REG<3:0>. Если предделитель включен перед TMR0, любые команды записи в TMR0 (например, CLRF 1, MOVWF 1, BSF 1,x и т.д.) сбрасывают предделитель. Когда предделитель подключен к WDT, команда CLRWDT сбросит предделитель вместе с WDT. Предделитель также очищается при сбросе микроконтроллера. Предделитель недоступен для чтения/записи.

На рисунке 4.1 показаны основные конфигурационные биты регистра OPTION_REG, связанные с таймером TMR0.

Регистр OPTION_REG (адрес 81h или 181h)

R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1	R/W-1
-RBPU	**INTEDG**	**T0CS**	**T0SE**	**PSA**	**PS2**	**PS1**	**PS0**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **-RBPU**:

бит 6: **INTEDG**:

бит 5: **T0CS**: Выбор тактового сигнала для TMR0
1 = внешний тактовый сигнал с вывода RA4/T0CKI
0 = внутренний тактовый сигнал CLKOUT

бит 4: **T0SE**: Выбор фронта приращения TMR0 при внешнем тактовом сигнале
1 = приращение по заднему фронту сигнала (с высокого к низкому уровню) на выводе RA4/T0CKI
0 = приращение по переднему фронту сигнала (с низкого к высокому уровню) на выводе RA4/T0CKI

бит 3: **PSA**: Выбор включения предделителя
1 = предделитель включен перед WDT
0 = предделитель включен перед TMR0

биты 2-0: **PS2: PS0**: Установка коэффициента деления предделителя

Значение	Для TMR0	Для WDT
000	1:2	1:1
001	1:4	1:2
010	1:8	1:4
011	1:16	1:8
100	1:32	1:16
101	1:64	1:32
110	1:128	1:64
111	1:256	1:128

Рисунок 4.1 - Основные конфигурационные биты регистра OPTION_REG, связанные с таймером TMR0

При переполнении счетчика TMR0, т.е. при переходе его значения от FFh к 00h возникают прерывания. При возникновении прерывания устанавливается в ‘1’ бит T0IF(INTCON<2>). Само прерывание может быть разрешено/запрещено установкой/сбросом бита T0IE (INTCON<5>). Флаг прерывания от TMR0 T0IF (INTCON<2>) должен быть сброшен в подпрограмме обработки прерываний. В SLEEP режиме микроконтроллера модуль TMR0 выключен и не может генерировать прерывания.

Даже при запрещенных прерываниях от таймера TMR0 при переполнении счетчика устанавливается в ‘1’ бит T0IF(INTCON<2>). Поэтому факт переполнения счетчика можно установить путем опроса этого бита.

Суммируя все вышесказанное, напишем процедуру формирования относительно больших временных задержек с использованием таймера TMR0.

В начале программы необходимо проинициализировать конфигурацию таймера:

```
;-----------------------------------------------
;Процедура инициализации таймера0
;Версия для работы без прерываний!
;прерывания - запрещены, срабатывание определяем
;по биту T0IF в INTCON
;установим предделитель к таймеру 0
;Настроим предделитель на деление 1:256
InitTmr0
    bcf INTCON,T0IE  ;Запретим прерывания
                     ;от таймера 0
    bsf STATUS,RP0   ;OPTION_REG - в первой страни-
це
    bcf OPTION_REG,5 ;0 = внутренний тактовый
                     ; сигнал CLKOUT
    bcf OPTION_REG,3 ;0 = предделитель включен
                     ;перед TMR0
    movlw 0x7        ;установим три младших бита -
iorwf OPTION_REG     ; Настроим предделитель
                     ;на деление 1:256
    bcf STATUS, RP0
    clrf TMR0            ;очистим TMR0
    return
```

Теперь временной интервал между двумя срабатываниями таймера можно определить, например, по опросу бита T0IF(INTCON<2>). Ниже приведен фрагмент программы, который формирует задержку в 3.6 сек при частоте кварца у микроконтроллера в 20 МГц. Для этого используется дополнительный регистр tmr0count, который дополнительно делит формируемую таймером частоту в 76.3 Гц еще на 256. Если необходима частота больше, чем 0.3 Гц, но меньше чем 76.3 Гц, то необходимо инициализировать регистр tmr0count ненулевым значением. Если необходима частота больше, чем 76.3 Гц, то можно уменьшить значение коэффициента деления предделителя, или после переполнения таймера записать в него (в регистр TMR0) ненулевое значение.

```
mainloop         ;Основной цикл программы

;Ждем срабатывания таймера 0
;таймер настроен на 20МГц/4/256(предделитель)/256
;(регистр таймера)=76.3 Гц
;Вывод информации будет осуществляться с частотой
; 76.3Гц/tmr0count=76.3/256=0.3 Гц
; (или раз за 3,6 сек)
;В протеусе частота процентов на 20
;оказывается меньшей! -
```

```
;проверено на эксперименте!

    clrw tmr0count        ;Максимальное число в
                          ;tmr0count (256)
    ;……….
    ;тут можно выводить информацию в ЦАП и т.д.
;……….

waitmr0 btfss INTCON,T0IF
    goto waitmr0      ;Опрос бита T0IF
    bcf INTCON,T0IF   ;программный сброс бита-
                      ;признака переполнения
    decfsz tmr0count ;Уменьшаем наш регистр
    goto waitmr0

    ; Дождались!

    goto mainloop
```

2 Цель работы: научиться пользоваться таймером и формировать аналоговые сигналы с помощью ЦАП.

3 Порядок выполнения работы

3.1 Решить практическую задачу по программированию генератора аналоговых сигналов произвольной формы, согласно своему варианту.

3.2 Проверить работоспособность разработанной программы в системе Proteus.

3.3 Проверить работоспособность разработанной программы на лабораторном стенде.

4 Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя

Цель работы

Задание на лабораторную работу

Осциллограммы выводимых сигналов.

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 Разработайте алгоритм формирования треугольного сигнала.

5.2 Разработайте алгоритм формирования пилообразного сигнала.

5.3 Разработайте алгоритм формирования синусоидального сигнала.

5.4 Какую максимальную и минимальную задержку можно организовать на определенной частоте работы микроконтроллера при помощи таймера TMR0 без использования дополнительных регистров?

5.5 Как с помощью микроконтроллера можно сформировать сигнал ЭКГ?

5.6 Рассчитайте максимальную частоту дискретизации сигнала, который можно сформировать на основе определенной конкретной микросхемы ЦАП.

5.7 Какие существуют возможные пути повышения максимальной частоты дискретизации сигнала?

Самостоятельная работа №5
Программирование прерываний.

1 Краткие теоретические сведения

Микроконтроллеры PICmicro среднего семейства могут иметь несколько источников прерываний. Для каждого периферийного модуля назначен отдельный источник прерываний. Некоторые периферийные модули содержат несколько источников прерываний (например, модуль USART).

Основные источники прерываний в микроконтроллерах PICmicro среднего семейства:

• Внешний источник прерываний INT;
• Переполнение таймера TMR0;
• Изменение уровня сигнала на входах PORTB (выводы RB7:RB4);
• Прерывания от USART;
• Завершение преобразования АЦП;
• Переполнение таймера TMR1;
• Переполнение таймера TMR2.

В микроконтроллерах среднего семейства присутствует как минимум один регистр, управляющий прерываниями. Это регистр INTCON. Если в микроконтроллере есть дополнительные периферийные модули, то в нем будут реализованы регистры для управления прерываниями от периферийных модулей (регистр маски, чтобы разрешить/запретить прерывания; регистр флагов прерываний, указывающий на возникшее прерывание). В зависимости от типа микроконтроллера в нем могут быть реализованы регистры: PIE1, PIR1, PIE2, PIR2.

Регистр управления прерываниями INTCON содержит индивидуальные биты флагов прерываний для ядра микроконтроллера, биты маски разрешения прерываний, а также бит глобального разрешения прерываний. Если бит глобального разрешения прерываний GIE (INTCON<7>) установлен в '1', то разрешены все немаскированные прерывания. Все прерывания запрещены, если GIE (INTCON<7>) сброшен в '0'. Прерывания индивидуально запрещены сбросом соответствующего бита в регистре INTCON. При сбросе микроконтроллера бит GIE сбрасывается в '0'.

Возврат из обработки прерываний выполняется по команде RETFIE, при этом происходит установка бита GIE в '1', что позволяет обработать любое отложенное прерывание.

Регистр INTCON содержит биты управления следующими прерываниями:

- внешнее прерывание INT;
- изменение сигнала на входах RB7:RB4;
- переполнение TMR0.

В регистре INTCON также расположен бит разрешения прерываний от периферийных модулей PEIE. Если PEIE=1, то разрешен переход по вектору прерываний при возникновении периферийного прерывания. Структура регистра INTCON показана на рисунке 5.1.

При обработке прерываний бит GIE=0, чтобы предотвратить повторную загрузку счетчика команд PC в стек и запись в PC адреса вектора прерываний 0004h. В обработчике прерываний источник прерываний может быть идентифицирован проверкой флагов прерываний. Как правило, флаги прерываний должны быть сброшены в обработчике прерываний перед разрешением прерываний в системе, чтобы предотвратить повторный переход на обработку прерываний.

Индивидуальные флаги прерываний устанавливаются независимо от состояния бита общего разрешения прерываний GIE и соответствующих битов маски. Индивидуальные флаги прерываний устанавливаются **независимо** от состояния бита общего разрешения прерываний GIE и соответствующих битов маски. Это позволяет выполнять программный контроль возникновения условия прерываний.

Регистр INTCON (адрес 0Bh, 8Bh, 10Bh или 18Bh)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-x
GIE	**PEIE**	**T0IE**	**INTE**	**RBIE**	**T0IF**	**INTF**	**RBIF**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **GIE**: Глобальное разрешение прерываний
1 = разрешены все немаскированные прерывания
0 = все прерывания запрещены

бит 6: **PEIE**: Разрешение прерываний от периферийных модулей
1 = разрешены все немаскированные прерывания периферийных модулей
0 = прерывания от периферийных модулей запрещены

бит 5: **T0IE**: Разрешение прерывания по переполнению TMR0
1 = прерывание разрешено
0 = прерывание запрещено

бит 4: **INTE**: Разрешение внешнего прерывания INT
1 = прерывание разрешено
0 = прерывание запрещено

бит 3: **RBIE**: Разрешение прерывания по изменению сигнала на входах RB7:RB4 PORTB
1 = прерывание разрешено
0 = прерывание запрещено

бит 2: **T0IF**: Флаг прерывания по переполнению TMR0
1 = произошло переполнение TMR0 (сбрасывается программно)
0 = переполнения TMR0 не было

бит 1: **INTF**: Флаг внешнего прерывания INT
1 = выполнено условие внешнего прерывания на выводе RB0/INT (сбрасывается программно)
0 = внешнего прерывания не было

бит 0: **RBIF**: Флаг прерывания по изменению уровня сигнала на входах RB7:RB4 PORTB
1 = зафиксировано изменение уровня сигнала на одном из входов RB7:RB4 (сбрасывается программно)
0 = не было изменения уровня сигнала ни на одном из входов RB7:RB4

Рисунок 5.1 – Структура регистра INTCON

Регистр PIE1 доступен для чтения и записи, содержит биты разрешения периферийных прерываний. Структура регистра PIE1 микроконтроллера PIC16F877 показана на рисунке 5.2.

Регистр PIR1 доступен для чтения и записи, содержит флаги прерываний периферийных модулей. Программное обеспечение пользователя должно сбрасывать соответствующие флаги при обработке прерываний от периферийных модулей. Структура регистра PIR1 микроконтроллера PIC16F877 показана на рисунке 5.3.

В микроконтроллере PIC16F877 еще также имеются регистры PIE2 и PIR2. Регистр PIE1 доступен для чтения и записи, содержит биты разрешения прерываний от модуля CCP2, возникновения коллизий на шине I2C и окончания записи в EEPROM память данных. Регистр PIR1 доступен для чтения и записи, содержит флаги прерываний от модуля CCP2, возникновения коллизий на шине I2C и окончания записи в EEPROM память данных.

Регистр PIE1 (адрес 8Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIE(1)	**ADIE**	**RCIE**	**TXIE**	**SSPIE**	**CCP1IE**	**TMR2IE**	**TMR1IE**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **PSPIE(1)**: Разрешение прерывания записи/чтения ведомого параллельного порта
1 = прерывание разрешено
0 = прерывание запрещено

бит 6: **ADIE**: Разрешение прерывания по окончанию преобразования АЦП
1 = прерывание разрешено
0 = прерывание запрещено

бит 5: **RCIE**: Разрешение прерывания от приемника USART
1 = прерывание разрешено
0 = прерывание запрещено

бит 4: **TXIE**: Разрешение прерывания от передатчика USART
1 = прерывание разрешено
0 = прерывание запрещено

бит 3: **SSPIE**: Разрешение прерывания от модуля синхронного последовательного порта
1 = прерывание разрешено
0 = прерывание запрещено

бит 2: **CCP1IE**: Разрешение прерывания от модуля CCP1
1 = прерывание разрешено
0 = прерывание запрещено

бит 1: **TMR2IE**: Разрешение прерывания по переполнению TMR2
1 = прерывание разрешено
0 = прерывание запрещено

бит 0: **TMR1IE**: Разрешение прерывания по переполнению TMR1
1 = прерывание разрешено
0 = прерывание запрещено

Рисунок 5.2 – Структура регистра PIE1 микроконтроллера PIC16F877

Регистр PIR1 (адрес 0Ch)

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
PSPIF(1)	ADIF	RCIF	TXIF	SSPIF	CCP1IF	TMR2IF	TMR1IF
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **PSPIF**(1): Флаг прерывания от ведомого параллельного порта
1 = произошла операция чтения или записи (сбрасывается программно)
0 = операции чтения или записи не происходило

бит 6: **ADIF**: Флаг прерывания от модуля АЦП
1 = преобразование АЦП завершено
0 = преобразование АЦП не завершено

бит 5: **RCIF**: Флаг прерывания от приемника USART
1 = буфер приемника USART полон
0 = буфер приемника USART пуст

бит 4: **TXIF**: Флаг прерывания от передатчика USART
1 = буфер передатчика USART пуст
0 = буфер передатчика USART полон

бит 3: **SSPIF**: Флаг прерываний от модуля MSSP
1 = выполнено условие возникновения прерывания от модуля MSSP (сбрасывается программно).
Условия возникновения прерывания:
- SPI
  - Выполнен прием/передача данных.
- Ведомый I2C
  - Выполнен прием/передача данных.
- Ведущий I2C
  - Выполнен прием/передача данных.
  - Завершено формирование на шине бита START.
  - Завершено формирование на шине бита STOP.
  - Завершено формирование на шине бита повторный START.
  - Завершено формирование на шине бита подтверждения.
  - Обнаружено на шине формирование бита START (для режима с несколькими ведущими).
  - Обнаружено на шине формирование бита STOP (для режима с несколькими ведущими).

0 = условие возникновения прерывания от модуля MSSP не выполнено

бит 2: **CCP1IF**: Флаг прерывания от модуля CCP1
Режим захвата
1 = выполнен захват значения TMR1 (сбрасывается программно)
0 = захвата значения TMR1 не происходило
Режим сравнения
1 = значение TMR1 достигло указанного в регистрах CCPR1H:CCPR1L(сбрасывается программно)
0 = значение TMR1 не достигло указанного в регистрах CCPR1H:CCPR1L
ШИМ режим
Не используется

бит 1: **TMR2IF**: Флаг прерывания по переполнению TMR2
1 = произошло переполнение TMR2 (сбрасывается программно)
0 = переполнения TMR2 не было

бит 0: **TMR1IF**: Флаг прерывания по переполнению TMR1
1 = произошло переполнение TMR1 (сбрасывается программно)
0 = переполнения TMR1 не было

Рисунок 5.3 – Структура регистра PIR1 микроконтроллера PIC16F877

При переходе на подпрограмму обработки прерываний в стеке сохраняется только адрес возврата. Как правило, необходимо сохранять значения ключевых регистров при обработке прерываний (например, регистр W и STATUS), что выполняется программным способом. Т.к. старшие 16 байт каждого банка микроконтроллеров

PIC16F876/877 доступны во всех банках, то регистры STATUS_TEMP, PCLATH_TEMP и W_TEMP могут быть размещены в этой области, например, так:

```
    ;Блок переменных для сохранения регистров в прерывании
    CBLOCK 0x70
    STATUS_TEMP
    PCLATH_TEMP
    W_TEMP
    ENDC
```

В следующем примере показан текст программы сохранения контекста:

```
    ;Пример  - Сохранение и восстановление
    ;регистров STATUS, W и PCLATH
    MOVWF W_TEMP      ;Сохранить W в регистре
                      ;текущего банка
    SWAPF STATUS,W    ;Обменять местами полубайты
                      ; и сохранить в W
    CLRF STATUS           ;Выбрать банк 0
    MOVWF STATUS_TEMP     ;Сохранить регистр STATUS
    MOVF PCLATH,W
    MOVWF PCLATH_TEMP     ;Сохранить регистр PCLATH
    :
    : ; Код программы обработки прерываний
    :
    MOVF PCLATH_TEMP,W
    MOVWF PCLATH          ;Восстановить регистр PCLATH
    SWAPF STATUS_TEMP,W   ;Прочитать регистр STATUS_TEMP
                          ;в W, восстанавливая банк
                          ;памяти программ
    MOVWF STATUS          ;Переписать W в регистр STATUS
    SWAPF W_TEMP,F        ;Обменять местами полубайты
                          ;в W_TEMP
    SWAPF W_TEMP,W        ;Обменять местами полубайты
                          ;в W_TEMP и записать в W
    retfie
```

Для построения портативных систем очень эффективно использовать режим энергосбережения SLEEP. Переход в режим энергосбережения происходит по команде SLEEP. При переходе в режим SLEEP сторожевой таймер WDT сбрасывается, но продолжает работать. В регистре STATUS бит -PD сбрасывается в '0', бит – TO устанавливается в '1', тактовый генератор микроконтроллера выключен. Порты ввода/вывода остаются в том же состоянии, что и до выполнения команды SLEEP (высокий уровень, низкий уровень, третье состояние).

Микроконтроллер выйдет из режима SLEEP по одному из следующих событий:

1. Внешний сброс по сигналу на входе -MCLR;

2. Переполнение сторожевого таймера WDT (если он разрешен);

3. Периферийное прерывание (по сигналу INT, изменение уровня сигнала на входах RB7:RB4 и др.).

Внешний сброс по сигналу -MCLR вызывает сброс микроконтроллера. Два других события вызывают продолжение выполнения программы.

Список прерываний от периферийных модулей, которые могут вывести микроконтроллер из режима SLEEP:

1. Чтение/запись PSP (только для PIC16F874/877);

2. Переполнение TMR1 в режиме асинхронного счетчика;

3. Прерывание от модуля CCP;

4. Триггер специального события (TMR1 должен работать в режиме асинхронного счетчика);

5. Обнаружение START/STOP на шине I2C модулем MSSP;

6. Прием/передача байта в режиме ведомого SPI/I2C;

7. Прием/передача USART в ведомом синхронном режиме;

8. Завершение преобразования АЦП (когда используется внутренний RC генератор для АЦП);

9. Завершение записи в EEPROM.

Другие прерывания от периферийных модулей не могут вывести микроконтроллер из режима SLEEP.

При выполнении команды SLEEP происходит предвыборка следующей инструкции (PC+1). Если прерывание должно выводить микроконтроллер из режима SLEEP, то соответствующий бит разрешения прерывания устанавливается в '1'. Микроконтроллер выходит из режима SLEEP независимо от состояния бита GIE. Если GIE=0, выполняется следующая инструкция после SLEEP без перехода по вектору прерываний. Если GIE=1, исполняется следующая инструкция после SLEEP и происходит переход на подпрограмму обработки прерываний (адрес 0004h). Когда выполнение какой-либо команды при выходе из режима SLEEP нежелательно, необходимо после команды SLEEP использовать инструкцию NOP.

Для примера разберем программирование SLEEP-режима микроконтроллера и прерываний в приложении обработки событий клавиатуры [Инструкция по применению AN552 фирмы Microchip].

Соберем схему, приведенную на рисунке 5.4. Соответствующая программа должна до перевода микроконтроллера в sleep-

режим разрешить прерывания по изменению состояния линий порта B. Нажатие любой клавиши вызовет прерывание, которое выведет микроконтроллер из sleep-режима.

Все это реализует программа, приведенная в листинге 1. Индикация активизированной клавиши осуществляется светодиодами.

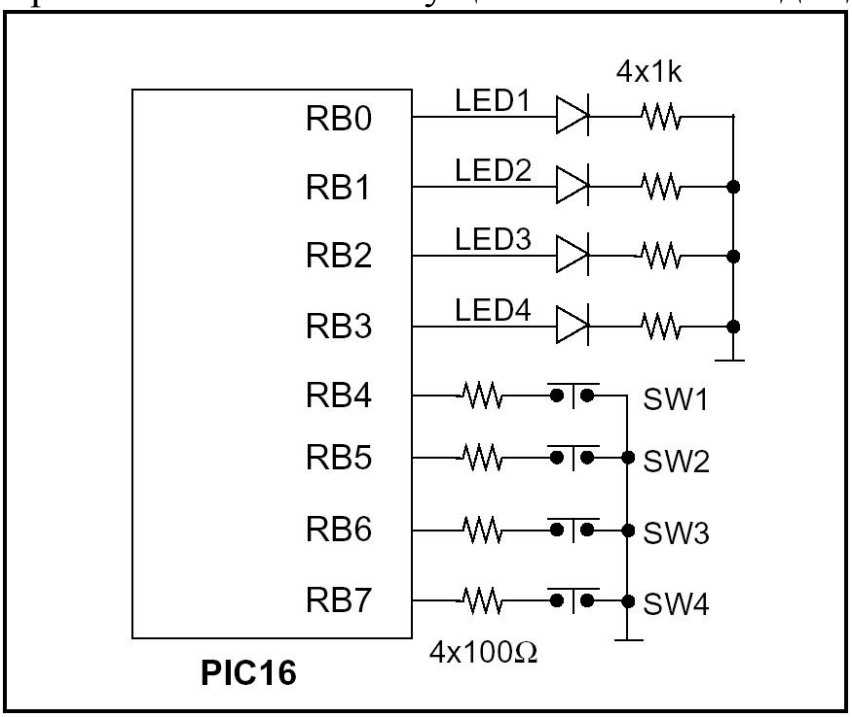

Рисунок 5.4 – Подключение клавиатуры для реализации низкопотребляющего режима работы микроконтроллера (sleep – режим) [Инструкция по применению AN552 - Microchip]

Листинг 1 – программа реакции на нажатие клавиатуры

```
    ; Эта программа иллюстрирует принцип "пробуждения"
    ;PIC-микроконтроллера
    ; в Результате воздействия на подключенную
    ;к нему клавиатуру.
    ; Выход из sleep-режима осуществляется
    ;за счет прерывания,
    ;возникающего при изменении состояния линий порта
B.
    ; В этом примере был использован
    ; микроконтроллер PIC16F877.
    ; Программа основана на инструкции по
    ;применению AN552 фирмы Microchip.
    ;
        listp=16f877
        #include p16f877.inc
    temp equ 20h
```

```
    org 0
    goto start
    org 4
    goto ServiceInterrupt
    start
    call InitPortB ; Инициализация порта B.
    loop
    sleep        ; Вход в Sleep-режим,
                 ;пока не нажмется клавиша
    nop
    goto loop
    ServiceInterrupt
    btfsc INTCON,RBIF    ; Проверка флага RBIF,
                         ; который индицирует
                         ;изменения состояния порта B.
    goto ServiceWakup    ; Да, тогда обработка,
    bcf INTCON,T0IE      ; очистка маски TMR0
    bcf INTCON,T0IF      ; очистка флага
    return

    ; Эта подпрограмма определяет, какая клавиша нажа-
та,
    ; и зажигает соответствующий светодиод.
    ; Затем ожидает, когда все клавиши будут отпущены.

    ServiceWakup
    bcf INTCON,RBIE  ; Запрет прерываний от порта B.
    comf PORTB,W     ; Считывание порта B с инверсией.
    bcf INTCON,RBIF  ;Сброс флага прерываний от порта
B.
    call delay16     ;Задержка на 16 мс.
                     ;для предотвращения дребезга
    comf PORTB,W     ; Чтение порта B с инверсией.

    andlw B'11110000' ; Маскировка выходов.
    movwf temp       ;сохраняем во временный регистр
    swapf temp,W     ; Перемена мест полубайтов.
    movwf PORTB  ; Включение индикации.
    call KeyRelease  ; Проверка отпускания клавиши
    retfie

    ; Эта подпрограмма ожидает, чтобы все
    ; клавиши были отпущены
    ;В дополнение сохраняет энергию, пока
    ; клавиши не отпущены
    KeyRelease
    call delay16     ; Задержка 16 мс для
                     ;предотвращения дребезга
```

```
comf PORTB,W     ; Считывание порта B с инверсией
bcf INTCON,RBIF  ; Сброс флага RBIF
bsf INTCON,RBIE  ; Разрешение прерываний от порта B
andlw B'11110000' ; очистка выходов
btfsc STATUS,Z   ; Клавиша остается нажатой?
return           ; Нет, тогда возврат.
    sleep        ;иначе засыпаем – экономим энергию
bcf INTCON,RBIE  ; Запрет прерываний от порта B.
comf PORTB,W     ;Считывание порта B с инверсией
bcf INTCON,RBIF  ; Сброс флага RBIF
goto KeyRelease  ; Повтор проверки.

; Эта подпрограмма инициализирует порт B.

InitPortB
    bsf STATUS,RP0       ;select bank1
    movlw B'11110000'    ; RBO - RB3 - выходы,
    movwf TRISB          ; RB4 - RB7 - входы,

    bcf OPTION_REG,7     ;RBPU - Разрешить
                    ;подключения "подтягивающих"
                    ;резисторов к линиям порта B.

    bcf STATUS,RP0   ;select page 0
    clrf PORTB       ;обнулить порт B
    bcf INTCON,RBIE  ;запрет прерываний от порта B
    movf PORTB,W     ;читаем порт
    bcf INTCON,RBIF  ;стираем флаг
    bsf INTCON,RBIE  ;разрешение прерываний от
                     ;порта B
retfie           ;возврат и разрешение прерываний


;delay16 ждет около 16.4 mSecs использует
;прерывания TMR0
;Частота кварца – 4 МГц
delay16
bsf STATUS,RP0       ;выбор банка 1
movlw B'00000111'    ;fosc/256 --> TMR0
movwf OPTION_REG     ; /
bcf STATUS,RP0       ; выбор банка 0
clrf TMR0
bcf INTCON,T0IF      ;очистка флага
bsf INTCON,T0IE      ;разрешение прерываний
                     ;от таймера 0
CheckAgain
btfss INTCON,T0IF    ;таймер переполнился?
goto CheckAgain      ;нет, тогда зациклимся
```

```
bcf INTCON,T0IE      ;иначе запрещаем прерывания
                     ;от таймера 0
bcf INTCON,T0IF      ;очищаем флаг
return
;
end
```

2 Цель работы: уметь программировать спящий режим и режим прерываний.

3 Порядок выполнения работы

3.1 Решить практическую задачу согласно своему варианту с использованием возможностей спящего режима микроконтроллера и обработки прерываний.

3.2 Проверить работоспособность разработанной программы в системе Proteus или на лабораторном стенде.

4 Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя

Цель работы

Задание на лабораторную работу

Схема спроектированной системы

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 Зачем необходим режим Sleep микроконтроллера?

5.2 Какие применения могут быть у сторожевого таймера WDT?

5.3 Как при сброшенном бите глобального разрешения прерывания GIE=0 узнать о наступлении того или иного события, которое могло бы возбудить прерывание?

5.4 Чем отличается команда микроконтроллера RETFIE от команды RETURN и RETLW?

5.5 Какие последствия могут наступить, если в подпрограмме обработки прерывания не сохранять значения ключевых регистров (контекста)?

5.6 Нарисуйте алгоритм работы программы листинга 1 (программы инструкции по применению AN552 фирмы Microchip).

Самостоятельная работа №6

Программирование связи в операционных системах Win32 с микроконтроллерными устройствами по последовательному интерфейсу RS232C.

1 Краткие теоретические сведения

С последовательными и параллельными портами в Win32 работают как с файлами. Следовательно, начинать надо с открытия порта как файла - необходимо воспользоваться функцией CreateFile. Ее прототип выглядит так:

```
HANDLE CreateFile(
   LPCTSTR                lpFileName,
   DWORD                  dwDesiredAccess,
   DWORD                  dwShareMode,
   LPSECURITY_ATTRIBUTES  lpSecurityAttributes,
   DWORD                  dwCreationDistribution,
   DWORD                  dwFlagsAndAttributes,
   HANDLE                 hTemplateFile
);
```

Функция имеет много параметров, большинство из которых можно не использовать. Краткое описание параметров:

lpFileName - Указатель на строку с именем открываемого или создаваемого файла. Формат этой строки может быть очень хитрым. В частности, можно указывать сетевые имена для доступа к файлам на других компьютерах. Можно открывать логические разделы или физические диски и работать в обход файловой системы. Последовательные порты имеют имена "COM1", "COM2", "COM3", "COM4" и так далее. Параллельные порты называются "LPT1", "LPT2" и так далее.

dwDesiredAccess - Задает тип доступа к файлу. Возможно использование следующих значений: 0 - Опрос атрибутов устройства без получения доступа к нему.

GENERIC_READ - Файл будет считываться.

GENERIC_WRITE - Файл будет записываться.

GENERIC_READ|GENERIC_WRITE - Файл будет и считываться и записываться.

dwShareMode - Задает параметры совместного доступа к файлу. Коммуникационные порты нельзя делать разделяемыми, поэтому данный параметр должен быть равен 0.

lpSecurityAttributes - Задает атрибуты защиты файла. Поддерживается только в Windows NT. Однако при работе с портами должен в любом случае равняться NULL.

dwCreationDistribution - Управляет режимами автосоздания, автоусечения файла и им подобными. Для коммуникационных портов всегда должно задаваться OPEN_EXISTING.

dwFlagsAndAttributes - Задает атрибуты создаваемого файла. Так же управляет различными режимами обработки. Для наших целей этот параметр должен быть или равным 0, или FILE_FLAG_OVERLAPPED. Нулевое значение используется при синхронной работе с портом, а FILE_FLAG_OVERLAPPED при асинхронной, или другими словами, при фоновой обработке ввода/вывода.

hTemplateFile - Задает описатель файла-шаблона. При работе с портами всегда должен быть равен NULL.

При успешном открытии файла, в нашем случае порта, функция возвращает описатель (HANDLE) файла. При ошибке INVALID_HANDLE_VALUE. Код ошибки можно получить путем вызова функции GetLastError.

Открытый порт должен быть закрыт перед завершением работы программы. В Win32 закрытие объекта по его описателю выполняет функция CloseHandle:

```
BOOL CloseHandle(HANDLE hObject);
```

Функция имеет единственный параметр - описатель закрываемого объекта. При успешном завершении функция возвращает не нулевое значение, при ошибке нуль.

Приведем пример открытия порта COM2:

```
#include <windows.h>
. . .
HANDLE port;
. . .
port=CreateFile("COM2",GENERIC_READ|GENERIC_WRITE,0
,NULL,OPEN_EXISTING,0,NULL);
if(port==INVALID_HANDLE_VALUE) {
   MsgBox(NULL,"Не возможно открыть последова-
тельный порт","Error",MB_OK);
   ExitProcess(1);
}
. . .
CloseHandle(port);
. . .
```

В данном примере открывается порт COM2 для чтения и записи, используется синхронный режим обмена. Проверяется успешность открытия порта, при ошибке выводится сообщение и программа завершается. Если порт открыт успешно, то он закрывается.

Открыв порт, мы получили его в свое распоряжение. Теперь с портом может работать только наша программа. Однако, прежде чем мы займемся вводом/выводом, мы должны настроить порт. Это касается только последовательных портов, для которых мы должны задать скорость обмена, параметры четности, формат данных и прочее. Кроме того, существует несколько специфичных для Windows параметров. Речь идет о тайм-аутах, которые позволяют контролировать как интервал между принимаемыми байтами, так и общее время приема сообщения. Есть возможность управлять состоянием сигналов управления модемом.

Основные параметры последовательного порта описываются структурой DCB. Временные параметры структурой COMMTIMEOUTS. Настройка порта заключается в заполнении управляющих структур и последующем вызове функций настройки.

Основную информацию содержит структура DCB:

```
typedef struct _DCB {
   DWORD DCBlength; // sizeof(DCB)
   DWORD BaudRate;  // current baud rate
   DWORD fBinary:1; // binary mode, no EOF check
   DWORD fParity:1;    // enable parity checking
   DWORD fOutxCtsFlow:1;  // CTS output flow con-
trol
   DWORD fOutxDsrFlow:1;  // DSR output flow con-
trol
   DWORD fDtrControl:2;   // DTR flow control type
   DWORD fDsrSensitivity:1;  // DSR sensitivity
   DWORD fTXContinueOnXoff:1; // XOFF continues Tx
   DWORD fOutX:1;       // XON/XOFF out flow con-
trol
   DWORD fInX:1;         // XON/XOFF in flow con-
trol
   DWORD fErrorChar:1;   // enable error replace-
ment
   DWORD fNull:1;        // enable null stripping
   DWORD fRtsControl:2;  // RTS flow control
   DWORD fAbortOnError:1;// abort reads/writes
                         // on error
   DWORD fDummy2:17;   // reserved
   WORD  wReserved;    // not currently used
   WORD  XonLim;       // transmit XON threshold
   WORD  XoffLim;      // transmit XOFF threshold
   BYTE  ByteSize;     // number of bits/byte, 4-8
   BYTE  Parity;       // 0-
4=no,odd,even,mark,space
   BYTE  StopBits;     // 0,1,2 = 1, 1.5, 2
```

```
        char  XonChar;       // Tx and Rx XON character
        char  XoffChar;      // Tx and Rx XOFF character
        char  ErrorChar;    // error replacement charac-
ter
        char  EofChar;       // end of input character
        char  EvtChar;       // received event character
        WORD  wReserved1;    // reserved; do not use
    } DCB;
```

Эта структура содержит почти всю управляющую информацию, которая в реальности располагается в различных регистрах последовательного порта. Описание полей структуры:

DCBlength - Задает длину, в байтах, структуры DCB. Используется для контроля корректности структуры при передаче ее адреса в функции настройки порта.

BaudRate - Скорость передачи данных. Возможно указание следующих констант: CBR_110, CBR_300, CBR_600, CBR_1200, CBR_2400, CBR_4800, CBR_9600, CBR_14400, CBR_19200, CBR_38400, CBR_56000, CBR_57600, CBR_115200, CBR_128000, CBR_256000. Как видно, эти константы соответствуют всем стандартным скоростям обмена. На самом деле, это поле содержит числовое значение скорости передачи, а константы просто являются символическими именами. Поэтому можно указывать, например, и CBR_9600, и просто 9600. Однако рекомендуется указывать символические константы.

fBinary - Включает двоичный режим обмена. Win32 не поддерживает недвоичный режим, поэтому данное поле всегда должно быть равно 1, или логической константе TRUE (что предпочтительней). В Windows 3.1, если это поле было равно FALSE, включался текстовый режим обмена. В этом режиме поступивший на вход порта символ заданный полем EofChar свидетельствовал о конце принимаемых данных.

fParity - Включает режим контроля четности. Если это поле равно TRUE, то выполняется проверка четности, при ошибке, в вызывающую программу, выдается соответствующий код завершения.

fOutxCtsFlow - Включает режим слежения за сигналом CTS. Если это поле равно TRUE и сигнал CTS сброшен, передача данных приостанавливается до установки сигнала CTS. Это позволяет подключенному к компьютеру прибору приостановить поток передаваемой в него информации, если он не успевает ее обрабатывать.

fOutxDsrFlow - Включает режим слежения за сигналом DSR. Если это поле равно TRUE и сигнал DSR сброшен, передача данных прекращается до установки сигнала DSR.

fDtrControl - Задает режим управления обменом для сигнала DTR. Это поле может принимать следующие значения:

DTR_CONTROL_DISABLE - Запрещает использование линии DTR

DTR_CONTROL_ENABLE - Разрешает использование линии DTR

DTR_CONTROL_HANDSHAKE - Разрешает использование рукопожатия для выхода из ошибочных ситуаций. Этот режим используется, в частности, модемами при восстановлении в ситуации потери связи.

fDsrSensitivity - Задает чувствительность коммуникационного драйвера к состоянию линии DSR. Если это поле равно TRUE, то все принимаемые данные игнорируются драйвером (коммуникационный драйвер расположен в операционной системе), за исключением тех, которые принимаются при установленном сигнале DSR.

fTXContinueOnXoff - Задает, прекращается ли передача при переполнении приемного буфера и передаче драйвером символа XoffChar. Если это поле равно TRUE, то передача продолжается, несмотря на то, что приемный буфер содержит более XoffLim символов и близок к переполнению, а драйвер передал символ XoffChar для приостановления потока принимаемых данных. Если поле равно FALSE, то передача не будет продолжена до тех пор, пока в приемном буфере не останется меньше XonLim символов и драйвер не передаст символ XonChar для возобновления потока принимаемых данных. Таким образом, это поле вводит некую зависимость между управлением входным и выходным потоками информации.

fOutX - Задает использование XON/XOFF управления потоком при передаче. Если это поле равно TRUE, то передача останавливается при приеме символа XoffChar, и возобновляется при приеме символа XonChar.

fInX - Задает использование XON/XOFF управления потоком при приеме. Если это поле равно TRUE, то драйвер передает символ XoffChar, когда в приемном буфере находится более XoffLim, и XonChar, когда в приемном буфере остается менее XonLim символов.

fErrorChar - Указывает на необходимость замены символов с ошибкой четности на символ задаваемый полем ErrorChar. Если это поле равно TRUE, и поле fParity равно TRUE, то выполняется замена.

fNull - Определяет действие, выполняемое при приеме нулевого байта. Если это поле TRUE, то нулевые байты отбрасываются при передаче.

fRtsControl - задает режим управления потоком для сигнала RTS. Если это поле равно 0, то по умолчанию подразумевается RTS_CONTROL_HANDSHAKE. Поле может принимать одно из следующих значений:

RTS_CONTROL_DISABLE - Запрещает использование линии RTS

RTS_CONTROL_ENABLE - Разрешает использование линии RTS

RTS_CONTROL_HANDSHAKE - Разрешает использование RTS рукопожатия. Драйвер устанавливает сигнал RTS когда приемный буфер заполнен менее, чем на половину, и сбрасывает, когда буфер заполняется более чем на три четверти.

RTS_CONTROL_TOGGLE - Задает, что сигнал RTS установлен, когда есть данные для передачи. Когда все символы из передающего буфера переданы, сигнал сбрасывается.

fAbortOnError - Задает игнорирование всех операций чтения/записи при возникновении ошибки. Если это поле равно TRUE, драйвер прекращает все операции чтения/записи для порта при возникновении ошибки. Продолжать работать с портом можно будет только после устранения причины ошибки и вызова функции ClearCommError.

fDummy2 - Зарезервировано и не используется.

wReserved - Не используется, должно быть установлено в 0.

XonLim - Задает минимальное число символов в приемном буфере перед посылкой символа XON.

XoffLim - Определяет максимальное количество байт в приемном буфере перед посылкой символа XOFF. Максимально допустимое количество байт в буфере вычисляется вычитанием данного значения из размера приемного буфера в байтах.

ByteSize - Определяет число информационных бит в передаваемых и принимаемых байтах.

Parity - Определяет выбор схемы контроля четности. Данное поле должно содержать одно из следующих значений:

EVENPARITY - Дополнение до четности

MARKPARITY - Бит четности всегда 1

NOPARITY - Бит четности отсутствует

ODDPARITY - Дополнение до нечетности

SPACEPARITY - Бит четности всегда 0

StopBits - Задает количество стоповых бит. Поле может принимать следующие значения:

ONESTOPBIT - Один стоповый бит

ONE5STOPBIT - Полтора стоповых бита

TWOSTOPBIT - Два стоповых бита

XonChar - Задает символ XON используемый как для приема, так и для передачи.

XoffChar - Задает символ XOFF используемый как для приема, так и для передачи.

ErrorChar - Задает символ, использующийся для замены символов с ошибочной четностью.

EofChar - Задает символ, использующийся для сигнализации о конце данных.

EvtChar - Задает символ, использующийся для сигнализации о событии.

wReserved1 - Зарезервировано и не используется.

Так как поля структуры DCB используются для конфигурирования микросхем портов, на них накладываются некоторые ограничения. Размер байта должен быть 5, 6, 7 или 8 бит. Комбинация из пяти битного байта и двух стоповых бит является недопустимой. Так же, как и комбинация из шести, семи или восьми битного байта и полутора стоповых бит.

Структура DCB самая большая из всех, использующихся для настройки последовательных портов. Заполнение всех полей этой структуры может вызвать затруднения. Возможно воспользоваться функцией BuildCommDCB, которая позволяет заполнить поля структуры DCB на основе строки, по синтаксису аналогичной строке команды mode. Вот как выглядит прототип этой функции:

```
BOOL BuildCommDCB(LPCTSTR lpDef, LPDCB lpDCB);
```

Как видно, функция очень проста и имеет всего два параметра:

lpDef - Указатель на строку с конфигурационной информацией в формате команды mode. Например, следующая строка задает скорость 1200, без четности, 8 бит данных и 1 стоповый бит.

baud=1200 parity=N data=8 stop=1

lpDCB - Указатель на заполняемую структуру DCB. При этом структура должна быть уже создана и заполнена нулями, кроме поля DCBlength, которое должно содержать корректное значение. Возможно так же использование уже заполненной структуры DCB, например полученной вызовом одной из функций чтения параметров порта.

В случае успешного завершения, функция BuildCommDCB возвращает ненулевое значение. В случае ошибки возвращается 0.

Обычно функция BuldCommDCB изменяет только явно перечисленные в строке lpDef поля. Однако существуют два исключения из этого правила:

- При задании скорости обмена 110 бит в секунду автоматически устанавливается формат обмена с двумя стоповыми битами. Это сделано для совместимости с командой mode из MS-DOS или Windows NT.
- По умолчанию запрещается программное (XON/XOFF) и аппаратное управление потоком. Вы должны вручную заполнить требуемые поля DCB, если требуется управление потоком.

Функция BuildCommDCB поддерживает как новый, так и старый форматы командной строки mode. Однако Вы не можете смешивать эти форматы в одной строке.

Следует заметить, что функция BuildCommDCB только заполняет поля DCB указанными значениями. Это подготовительный шаг к конфигурированию порта, но не само конфигурирование, которое выполняется рассматриваемыми далее функциями. Поэтому Вы можете вызвать BuildCommDCB для общего заполнения структуры DCB, затем изменить значения не устраивающих Вас полей, и после этого вызывать функцию конфигурирования порта.

Заполнить DCB можно еще одним способом. Вызовом функции GetCommState. Эта функция заполняет DCB информацией о текущем состоянии устройства, точнее о его настройках. Вот как она выглядит:

```
BOOL GetCommState(
   HANDLE hFile,
   LPDCB  lpDCB );
```

Функция очень проста и имеет всего два параметра:

hFile - Описатель открытого файла коммуникационного порта. Этот описатель возвращается функцией CreateFile. Следовательно, прежде чем получить параметры порта, Вы должны его открыть. Для функции BuildCommDCB это не требовалось.

lpDCB - Указатель на DCB. Для DCB должен быть выделен блок памяти.

При успешном завершении функция возвращает ненулевое значение. При ошибке нуль. Получить параметры порта можно в любой момент, а не только при начальной настройке.

Заполнив DCB можно приступать к собственно конфигурированию порта. Это делается с помощью функции SetCommState:

```
BOOL SetCommState(
   HANDLE hFile,
   LPDCB  lpDCB   );
```

Эта функция имеет точно такие же параметры, как GetCommState. Различается только направление передачи информации. GetCommState считывает информацию из внутренних управляющих структур и регистров порта, а SetCommState наоборот, заносит ее. Следует быть осторожным при вызове функции SetCommState, поскольку она изменит параметры даже в том случае, если очереди приема/передачи не пусты, что может вызвать искажение потока передаваемых или принимаемых данных.

Еще одна тонкость этой функции заключается в том, что она завершится с ошибкой, если поля XonChar и XoffChar в DCB содержат одинаковые значения.

В случае успешного завершения возвращается отличное от нуля значение, а в случае ошибки - нуль.

Приведем пример инициализации COM-порта:

```
//-------------------------------------
DCB dcb;         //Структуры состояния ком-порта
HANDLE hCom;             //Windows дескриптор
DWORD dwError;           //Переменная для ошибок
BOOL fSuccess;           //Флаг удачной операции

hCom = CreateFile("COM2",
    GENERIC_READ | GENERIC_WRITE,
    0,   /* без разделения ресурса-монопольно */
    NULL, /* без секьюрити атрибутов  */
    OPEN_EXISTING,  /*  порт  обязан  использовать
OPEN_EXISTING */
    0,     /* не многопотоковый (overlapped) ввод-
вывод */
    NULL  /* hTemplate должен быть  NULL для портов
*/
    );

if (hCom == INVALID_HANDLE_VALUE) {
    dwError = GetLastError();

    /* ошибка открытия порта */
   throw "Не могу открыть порт!"; }

/*
 * Получаем текущую конфигурацию порта
```

```
     */

    fSuccess = GetCommState(hCom, &dcb);

    if (!fSuccess) {
        /* ошибка. */
      throw "Не могу определить состояние порта!";
    }

    /* Введем параметры: скорость=9600, 8 бит данных,
нет четности, 1 стоповый бит */
    dcb.BaudRate = 9600;
    dcb.ByteSize = 8;
    dcb.Parity = NOPARITY;
    dcb.StopBits = ONESTOPBIT;
    dcb.fRtsControl=0;
    dcb.fDtrControl=1;
    fSuccess = SetCommState(hCom, &dcb);
    if (!fSuccess) {
        /* ошибка. */
      throw  "Не  могу  установить  требуемые  настройки
порта!";
    }
    //-----------------------------------------------
```

Прием и передача данных выполняется функциями ReadFile и WriteFile, то есть теми же самыми, которые используются для работы с дисковыми файлами. Вот как выглядят прототипы этих функций:

```
BOOL ReadFile(
   HANDLE       hFile,
   LPVOID       lpBuffer,
   DWORD        nNumOfBytesToRead,
   LPDWORD      lpNumOfBytesRead,
   LPOVERLAPPED lpOverlapped
);

BOOL WriteFile(
   HANDLE       hFile,
   LPVOID       lpBuffer,
   DWORD        nNumOfBytesToWrite,
   LPDWORD      lpNumOfBytesWritten,
   LPOVERLAPPED lpOverlapped
);
```

Описание параметров:

hFile - Описатель открытого файла коммуникационного порта.

lpBuffer - Адрес буфера. Для операции записи данные из этого буфера будут передаваться в порт. Для операции чтения в этот буфер будут помещаться принятые из линии данные.

nNumOfBytesToRead, nNumOfBytesToWrite - Число ожидаемых к приему или предназначенных к передаче байт.

nNumOfBytesRead, nNumOfBytesWritten - Число фактически принятых или переданных байт. Если принято или передано меньше данных, чем запрошено, то для дискового файла это свидетельствует об ошибке, а для коммуникационного порта совсем не обязательно. Причина в тайм-аутах.

lpOverlapped - Адрес структуры OVERLAPPED, используемой для асинхронных (многопоточных) операций. Для синхронных операций данный параметр должен быть равным NULL.

Коммуникационный порт не совсем обычный файл. Например, для него нельзя выполнить операцию позиционирования файлового указателя. С другой стороны, порт позволяет управлять потоком, что нельзя делать с обычным файлом.

Как правило, первой операцией, после открытия порта, является его сброс, который реализуется следующей функцией:

```
BOOL PurgeComm(
   HANDLE hFile,
   DWORD  dwFlags );
```

Вызов этой функции позволяет решить две задачи: очистить очереди приема/передачи в драйвере и завершить все находящиеся в ожидании запросы ввода/вывода. Какие именно действия выполнять задается вторым параметром (значения можно комбинировать с помощью побитовой операции OR:

PURGE_TXABORT - Немедленно прекращает все операции записи, даже если они не завершены

PURGE_RXABORT - Немедленно прекращает все операции чтения, даже если они не завершены

PURGE_TXCLEAR - Очищает очередь передачи в драйвере

PURGE_RXCLEAR - Очищает очередь приема в драйвере

Вызов этой функции нужен для отбрасывания мусора, который может находиться в приемном буфере на момент запуска программы, или как результат ошибки в работе устройства. Очистка буфера передачи и завершение операций ввода/вывода так же потребуются при ошибке, как процедура восстановления, и при завершении программы, для красивого выхода.

Следует помнить, что очистка буфера передачи, как и экстренное завершение операции записи, не выполняют передачу данных находящихся в этом буфере. Данные просто отбрасываются. Если

же передача остатка данных необходима, то перед вызовом PurgeComm следует вызвать функцию:

```
BOOL FlushFileBuffers( HANDLE hFile );
```

Последовательный канал передачи данных можно перевести в специальное состояние, называемое разрывом связи. При этом передача данных прекращается, а выходная линия переводится в состояние "0". Приемник, обнаружив, что за время необходимое для передачи стартового бита, битов данных, бита четности и стоповых битов, приемная линия ни разу не перешла в состояние "1", так же фиксирует у себя состояние разрыва.

```
BOOL SetCommBreak( HANDLE hFile  );
BOOL ClearCommBreak(HANDLE hFile );
```

Следует заметить, что состояние разрыва линии устанавливается аппаратно. Поэтому нет другого способа возобновить прерванную, с помощью SetCommBreak, передачу данных, кроме вызова ClearCommBreak.

Более тонкое управление потоком данным позволяет осуществить функция:

```
BOOL EscapeCommFunction(  HANDLE hFile,
   DWORD  dwFunc   );
```

Выполняемое действие определяется вторым параметром, который может принимать одно из следующих значений:

CLRDTR – Сбрасывает сигнал DTR

CLRRTS - Сбрасывает сигнал RTS

SETDTR - Устанавливает сигнал DTR

SETRTS - Устанавливает сигнал RTS

SETXOFF - Симулирует прием символа XOFF

SETXON - Симулирует прием символа XON

SETBREAK - Переводит выходную линию передатчика в состояние разрыва.

CLRBREAK - Снимает состояние разрыва для выходной линии передатчика.

2. Цель работы: уметь программировать в операционных системах Win32 связь с микроконтроллерными устройствами по последовательному интерфейсу RS232C.

3. Порядок выполнения работы

3.1 Решить практическую задачу согласно своему варианту с использованием связи по последовательному интерфейсу RS232C.

3.2 Проверить работоспособность разработанной программы в системе Proteus или на лабораторном стенде.

4. Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя

Цель работы

Задание на лабораторную работу

Схема спроектированной системы

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 Как по COM-порту передавать значения отсчетов 10 битного АЦП микроконтроллера PIC16F877?

5.2 С какой целью необходимо закрывать дескриптор (описатель) открытого порта?

5.3 Какими командами можно изменить скорость обмена COM порта?

5.4 Какими командами можно зажечь светодиод, подключенный к линии CTS?

5.5 Будет ли правильно воспринимать информацию приемник, если скорость COM-порта приемника отличается от скорости COM-порта передатчика на 10%?

5.6 Какие механизмы контроля правильности передаваемых данных есть в COM порту?

5.7 С какой целью используется функция сброса порта PurgeComm ?

Самостоятельная работа №7
Программирование модуля АЦП в микроконтроллерах Microchip

1 Краткие теоретические сведения

Модуль аналого-цифрового преобразования (АЦП) в микроконтроллере PIC16F877 имеет восемь входных каналов. Входной аналоговый сигнал через коммутатор каналов заряжает внутренний конденсатор АЦП C_{HOLD}. Модуль АЦП преобразует напряжение, удерживаемое на конденсаторе $C_{HOLD,}$ в соответствующий 10-разрядный цифровой код методом последовательного приближения. Источник верхнего и нижнего опорного напряжения может быть программно - выбран с выводов VDD, VSS, AN3/VREF+ или AN2/VREF-.

Допускается работа модуля АЦП в SLEEP режиме микроконтроллера, при этом в качестве источника тактовых импульсов для АЦП должен быть выбран RC генератор.

Для управления АЦП в микроконтроллере используется 4 регистра:

· Регистр результата ADRESH (старший байт);
· Регистр результата ADRESL (младший байт);
· Регистр управления ADCON0;
· Регистр управления ADCON1.

Регистр ADCON0 используется для настройки работы модуля АЦП, а с помощью регистра ADCON1 устанавливается, какие входы микроконтроллера будут использоваться модулем АЦП и в каком режиме (аналоговый вход или цифровой порт ввода/вывода). При сбросе микроконтроллера все выводы, мультиплицированные с модулем АЦП (ANx), настраиваются как аналоговые входы.

Структурная схема модуля АЦП показана на рисунке 7.1. Структура регистра ADCON0 приведена на рисунке 7.2. Структура регистра ADCON1 приведена на рисунке 7.3.

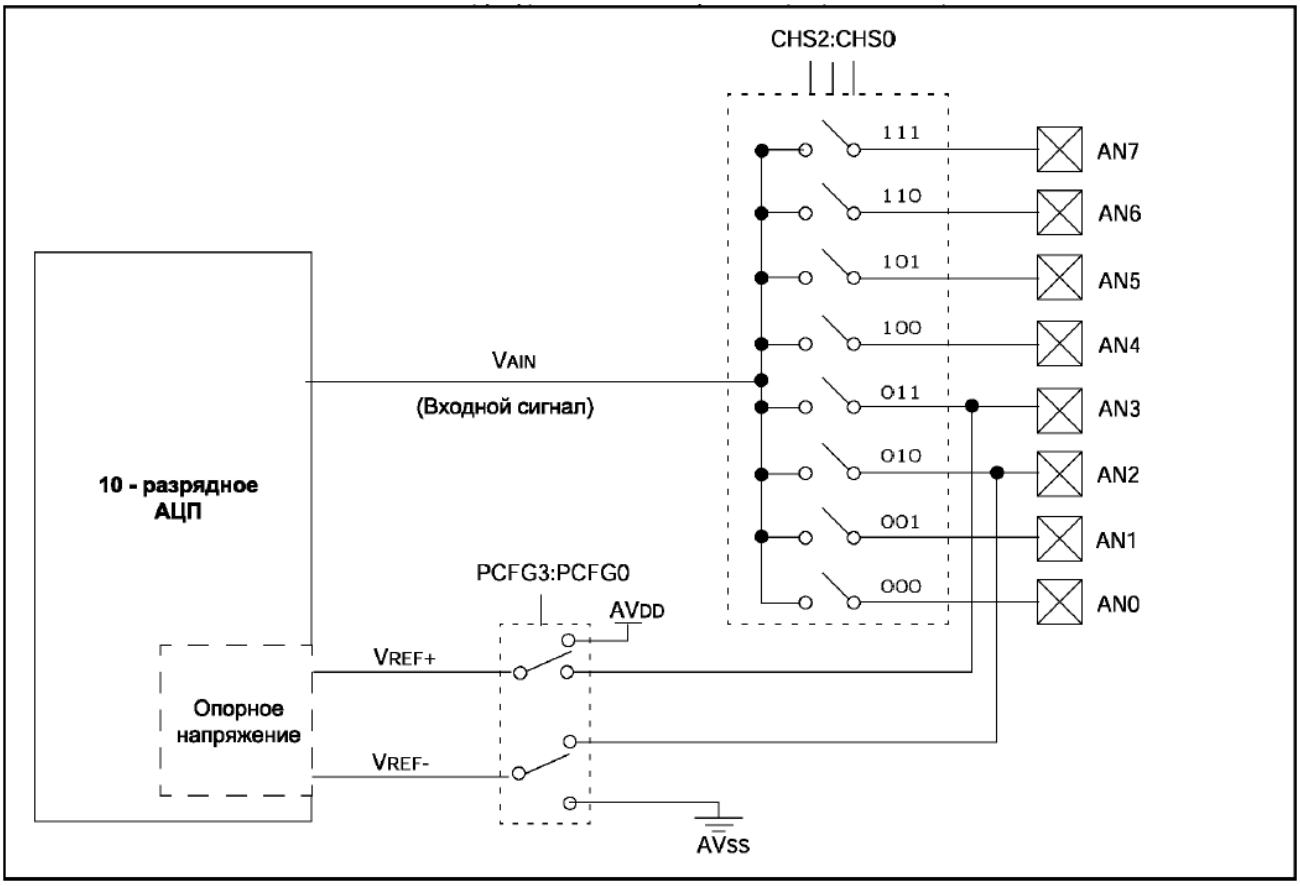

Рисунок 7.1 - Структурная схема модуля 10 - разрядного АЦП

В регистрах ADRESH:ADRESL сохраняется 10 - разрядный результат аналого-цифрового преобразования. Когда преобразование завершено, результат преобразования записывается в регистры ADRESH:ADRESL, после чего сбрасывается бит GO/-DONE (ADCON0<2>) и устанавливается флаг прерывания ADIF.

После включения и настройки АЦП необходимо выбрать рабочий аналоговый канал. Соответствующие биты TRIS аналоговых каналов должны настраивать канал порта ввода/вывода на вход. Перед началом преобразования необходимо выдержать временную паузу для обеспечения необходимой точности преобразования (конденсатор C_{HOLD} должен успевать полностью заряжаться до уровня входного напряжения). Точные формулы расчета временной паузы приведены в документации. Отметим, что при сопротивлении источника сигнала 10 кОм эта пауза составляет примерно 20 мкс, а при сопротивлении источника сигнала 50 Ом эта пауза составляет примерно 11 мкс.

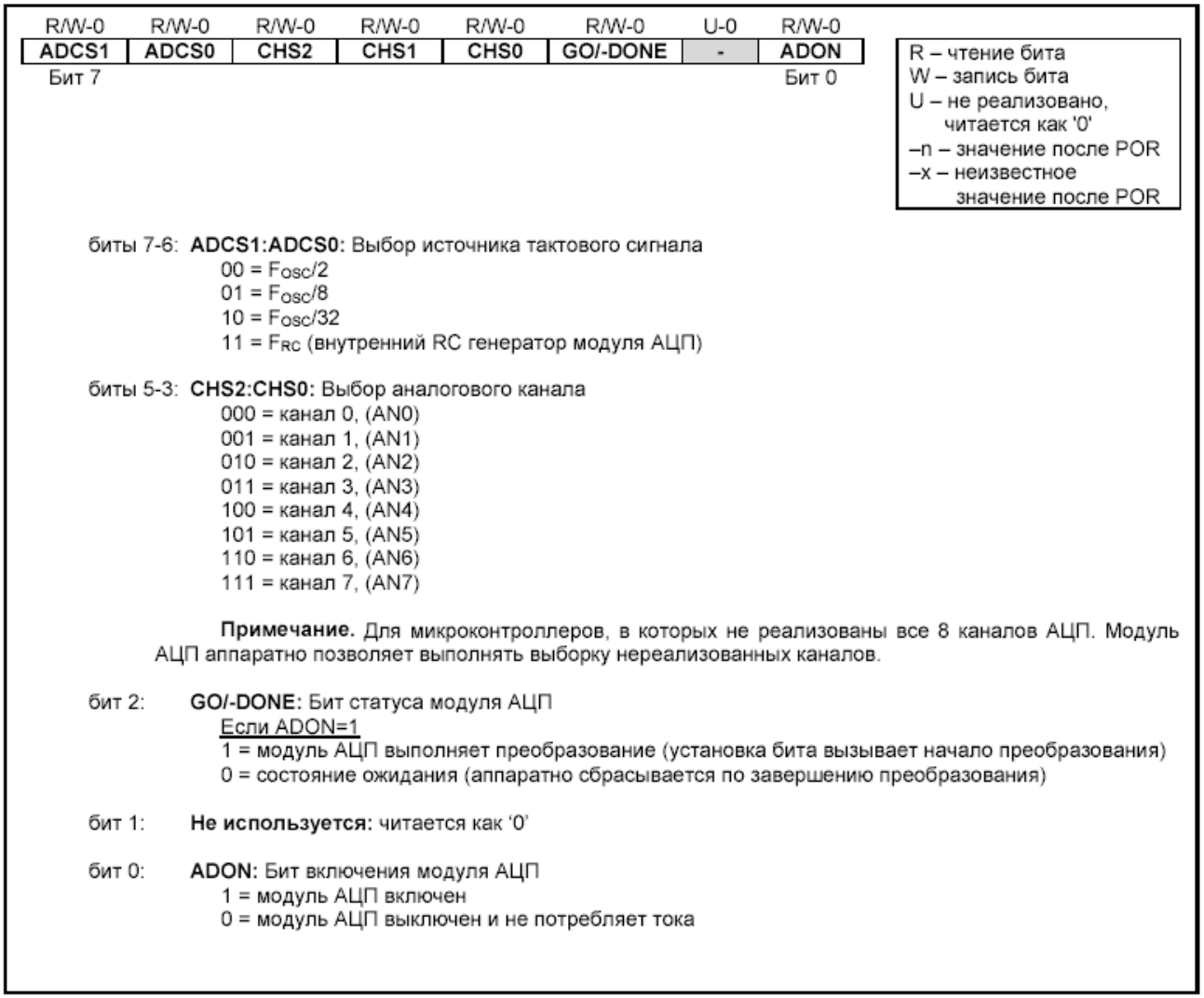

Рисунок 7.2 – Структура регистра ADCON0

Рекомендованная последовательность действий для работы с АЦП:

1. Настроить модуль АЦП:

· Настроить выводы как аналоговые входы, входы VREF или цифровые каналы ввода/вывода (ADCON1);

· Выбрать входной канал АЦП (ADCON0);

· Выбрать источник тактовых импульсов для АЦП (ADCON0);

· Включить модуль АЦП (ADCON0).

2. Настроить прерывание от модуля АЦП (если необходимо):

· Сбросить бит ADIF в '0';

· Установить бит ADIE в '1';

· Установить бит PEIE в '1';

· Установить бит GIE в '1'.

3. Выдержать паузу, необходимую для зарядки конденсатора C_{HOLD}.

R/W-0	U-0	U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0
ADFM	-	-	-	**PCFG3**	**PCFG2**	**PCFG1**	**PCFG0**
Бит 7							Бит 0

R – чтение бита
W – запись бита
U – не реализовано, читается как 0
–n – значение после POR
–x – неизвестное значение после POR

бит 7: **ADFM:** Формат сохранения 10-разрядного результата (см. рисунок 23-6)
1 = правое выравнивание, 6 старших бит ADRESH читаются как '0'
0 = левое выравнивание, 6 младших бит ADRESL читаются как '0'

биты 6-4: **Не используются:** читаются как '0'

биты 3-0: **PCFG3:PCFG0:** Управляющие биты настройки каналов АЦП

PCFG3: PCFG0	AN7	AN6	AN5	AN4	AN3	AN2	AN1	AN0	$V_{REF}+$	$V_{REF}-$	Кан./ V_{REF} (1)
0000	A	A	A	A	A	A	A	A	AV_{DD}	AV_{SS}	8/0
0001	A	A	A	A	$V_{REF}+$	A	A	A	AN3	AV_{SS}	7/1
0010	D	D	D	A	A	A	A	A	AV_{DD}	AV_{SS}	5/0
0011	D	D	D	A	$V_{REF}+$	A	A	A	AN3	AV_{SS}	4/1
0100	D	D	D	D	A	D	A	A	AV_{DD}	AV_{SS}	3/0
0101	D	D	D	D	$V_{REF}+$	D	A	A	AN3	AV_{SS}	2/1
011x	D	D	D	D	D	D	D	D	AV_{DD}	AV_{SS}	0/0
1000	A	A	A	A	$V_{REF}+$	$V_{REF}-$	A	A	AN3	AN2	6/2
1001	D	D	A	A	A	A	A	A	AV_{DD}	AV_{SS}	6/0
1010	D	D	A	A	$V_{REF}+$	A	A	A	AN3	AV_{SS}	5/1
1011	D	D	A	A	$V_{REF}+$	$V_{REF}-$	A	A	AN3	AN2	4/2
1100	D	D	D	A	$V_{REF}+$	$V_{REF}-$	A	A	AN3	AN2	3/2
1101	D	D	D	D	$V_{REF}+$	$V_{REF}-$	A	A	AN3	AN2	2/2
1110	D	D	D	D	D	D	D	A	AV_{DD}	AV_{SS}	1/0
1111	D	D	D	D	$V_{REF}+$	$V_{REF}-$	D	A	AN3	AN2	1/2

A = аналоговый вход D = цифровой канал ввода/вывода

Примечание 1. В этом столбце указывается число аналоговых каналов, доступных для выполнения преобразования, и число входов источника опорного напряжения.

Рисунок 7.3 – Структура регистра ADCON1

4. Начать аналого-цифровое преобразование:

· Установить GO/-DONE бит в '1' (ADCON0).

5. Ожидать, окончания преобразования:

· Ждать, пока бит GO/-DONE не будет сброшен в '0'; ИЛИ

· Ожидать прерывание по окончанию преобразования.

6. Считать результат преобразования из регистров ADRESH:ADRESL, сбросить бит ADIF в '0', если это необходимо.

7. Для следующего преобразования необходимо выполнить шаги, начиная с пункта 1 или 2. Время преобразования одного бита определяется как время T_{AD}. Минимальное время ожидания перед следующим преобразованием должно составлять $2T_{AD}$.

Время получения одного бита результата равно T_{AD}. Для 10-разрядного результата требуется как минимум 11.5 T_{AD}. Параметры тактового сигнала для АЦП определяются программно, T_{AD} может принимать следующие значения:

· $2T_{OSC}$;

· $8T_{OSC}$;

· $32T_{OSC}$;

· Внутренний RC генератор модуля АЦП (2-6 мкс).

Для получения корректного результата преобразования необходимо выбрать источник тактового сигнала АЦП, обеспечивающий время TAD не менее 1.6 мкс. Таблицы выбора источника тактового сигнала приведены в документации. Отметим, что для частоты 20 МГц необходимо выбирать источником TAD или RC-генератор, или 32TOSC=1.6 мкс (значения конфигурационных битов ADCS1:ADCS0 = 10). Когда тактовая частота микроконтроллера больше 1МГц, рекомендуется использовать RC генератор АЦП только для работы в SLEEP режиме.

В следующем примере показана последовательность действий для работы с АЦП. Выводы настроены как аналоговые входы. Источник опорного напряжения – AV_{DD}, AV_{SS}. Разрешены прерывания от модуля АЦП. Источником импульсов преобразования является RC генератор АЦП. Аналоговое цифровое преобразование выполняется с вывода AN0:

```
BSF STATUS, RP0  ; Выбрать банк 1
CLRF ADCON1      ; Настроить входы АЦП
BSF PIE1, ADIE   ; Разрешить прерывания от АЦП
BCF STATUS, RP0 ; Выбрать банк 0
MOVLW 0xC1       ; Тактовые импульсы от RC
                 ; генератора АЦП,
MOVWF ADCON0     ; включить АЦП, выбрать канал 0
BCF PIR1, ADIF   ; Сбросить флаг прерываний от АЦП
BSF INTCON, PEIE ;Разрешить периферийные прерывания
BSF INTCON, GIE  ; Разрешить прерывания в системе
;
; Выдержать паузу, необходимую для
;заряда внутреннего конденсатора CHOLD.
; Затем начинать преобразование АЦП.
;
BSF ADCON0, GO   ; Старт преобразования
; Ожидать установку флага ADIF или сброс
; бита GO/-DONE по завершению преобразования
```

10-разрядный результат преобразования сохраняется в спаренном 16-разрядном регистре ADRESH:ADRESL. Запись результата преобразования может выполняться с правым или левым выравниванием, в зависимости от значения бита ADFM (см. рисунок 7.4). Не задействованные биты регистра ADRESH:ADRESL читаются как ‘0’. Если модуль АЦП выключен, то 8-разрядные регистры

ADRESH и ADREL могут использоваться как регистры общего назначения.

Модуль АЦП может работать в SLEEP режиме микроконтроллера при условии, что источником импульсов преобразования АЦП будет внутренний RC генератор (ADCS1:ADCS0=11). При выборе RC генератора импульсов модуль АЦП сделает задержку в один машинный цикл перед началом преобразования. Это позволяет программе пользователя выполнить команду SLEEP, тем самым уменьшить “цифровой шум” во время преобразования. После завершения преобразования аппаратно сбрасывается бит GO/-DONE в '0', результат преобразования сохраняется в регистрах ADRESH:ADRESL. Если разрешено прерывание от АЦП, то микроконтроллер выйдет из режима SLEEP. Если же прерывание было запрещено, то после преобразования модуль АЦП будет выключен, хотя бит ADON останется установленным.

Если был выбран другой источник тактовых импульсов АЦП (не внутренний RC генератор), то выполнение программой инструкции SLEEP прервет процесс преобразования и выключит модуль АЦП, оставив установленным бит ADON. Выключение модуля АЦП уменьшит ток потребления микроконтроллера. Инструкция SLEEP должна быть выполнена сразу после команды, устанавливающей бит GO/-DONE в '1'.

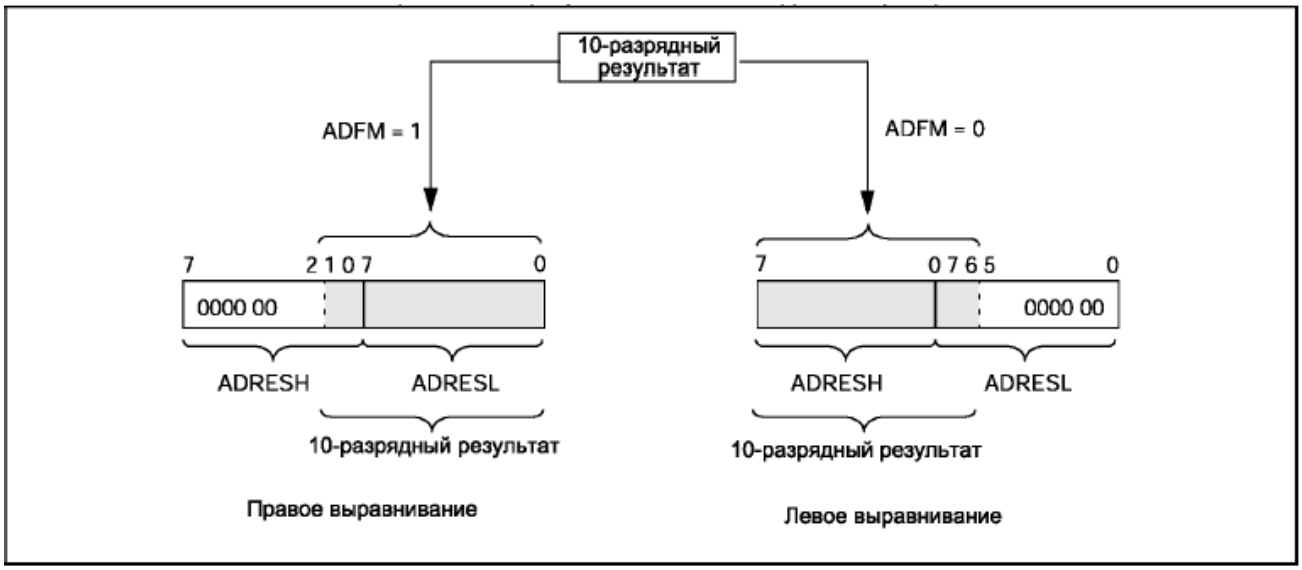

Рисунок 7.4 - Выравнивание результата аналого-цифрового преобразования

Абсолютная точность АЦП определяется суммарной ошибкой, исходя из ошибки дискретизации, интегральной ошибки, ошибки шкалы, ошибки смещения и монотонности. Суммарная ошибка определяется как максимальный разброс между текущим и идеаль-

ным результатом для любого значения. Абсолютная ошибка АЦП меньше ± 1 значащего бита при VDD=VREF, но она возрастает при отклонении VREF от VDD.

В некотором диапазоне напряжений на аналоговом входе цифровой результат будет один и тот же. Это возникает из-за дискретизации, которая неизбежна при преобразовании аналоговой величины в цифровую форму. Ошибка дискретизации составляет ± от значащего бита, и единственный способ уменьшить ее - увеличить разрядность АЦП.

Ошибку смещения составляет разность между результатом первого преобразования и идеальным значением. Эта ошибка сдвигает всю передаточную функцию, и может быть учтена при помощи калибровки. Ошибка вносится в результате наложения токов утечки и выходного сопротивления источника сигнала. Ошибка усиления измеряется как максимальное отклонение результата, скорректированного с учетом ошибки смещения. Эта ошибка проявляется в виде изменения наклона передаточной функции. Ошибка усиления может быть откалибрована и учтена. Ошибка линейности определяется как разница в приращении входного напряжения для получения одинакового приращения выходного кода и не поддается калибровке. Интегральная ошибка вычисляется как отклонение результата, скорректированного с учетом ошибки усиления. Дифференциальная ошибка вычисляется как отклонение максимальной длины кода результата от идеальной длины кода без учета других ошибок.

Таблица 7.1 – Показатели точности АЦП микроконтроллера PIC16F877

№ пар.	Обоз.	Описание		Мин.	Тип**	Макс.	Ед.	Примечание
A01	N_R	Разрядность		-	-	10	бит	$V_{REF} = V_{DD} = 5.12В$, $V_{SS} \le V_{AIN} \le V_{REF}$
A03	E_{IL}	Интегральная погрешность		-	-	< ± 1	LSb	$V_{REF} = V_{DD} = 5.12В$, $V_{SS} \le V_{AIN} \le V_{REF}$
A04	E_{DL}	Дифференциальная погрешность		-	-	< ± 1	LSb	$V_{REF} = V_{DD} = 5.12В$, $V_{SS} \le V_{AIN} \le V_{REF}$
A06	E_{OFF}	Ошибка смещения		-	-	< ± 2	LSb	$V_{REF} = V_{DD} = 5.12В$, $V_{SS} \le V_{AIN} \le V_{REF}$
A07	E_{GN}	Ошибка усиления		-	-	< ± 1	LSb	$V_{REF} = V_{DD} = 5.12В$, $V_{SS} \le V_{AIN} \le V_{REF}$
A10	-	Монотонность[3]		Гарантируется			-	$V_{SS} \le V_{AIN} \le V_{REF}$
A20	V_{REF}	Опорное напряжение (V_{REF+}-V_{REF-})		2.0	-	V_{DD} + 0.3	В	Минимальное значение для 10-разрядного АЦП
A21	V_{REF+}	Положительное опорное напр.		AV_{DD} – 2.5		AV_{DD} + 0.3	В	
A22	V_{REF-}	Отрицательное опорное напр.		AV_{SS} – 0.3		V_{REF+} - 2.0	В	
A25	V_{AIN}	Аналоговый вход		V_{SS} – 0.3	-	V_{REF} + 0.3	В	
A30	Z_{AIN}	Сопротивление источника сигн.		-	-	10.0	кОм	
A40	I_{AD}	Потребляемый ток АЦП	F	-	220	-	мкА	Среднее потребление при включенном АЦП[1]
			LF	-	90	-	мкА	
A50	I_{REF}	Потребляемый ток от источника опорного напряжения[2]		10 -	- -	1000 10	мкА мкА	Во время выборки V_{AIN}. Основано на дифференц. значении заряда C_{HOLD} до V_{AIN}. Во время преобразования.

** - В столбце "Тип." приведены параметры при VDD=5.0В, 25ºС, если не указано иное. Эти параметры являются ориентировочными, используются при разработке устройств и не измеряются.

Примечания:

1. Выключенный модуль АЦП не потребляет тока, кроме токов утечки.

2. Ток со входа RA3 или VDD в зависимости от выбранного источника опорного напряжения.

3. Результат АЦП никогда не уменьшается с увеличением напряжения на входе и не имеет кодов отсутствия напряжения.

2 Цель работы: уметь использовать модуль АЦП в микроконтроллерах Microchip.

3 Порядок выполнения работы

3.1 Решить практическую задачу согласно своему варианту с использованием модуля АЦП.

3.2 Проверить работоспособность разработанной программы в системе Proteus.

4 Содержание отчета:

Титульный лист с названием и номером работы, а также с фамилией исполнителя

Цель работы

Задание на лабораторную работу

Схема спроектированной системы

Листинг программы.

Выводы.

5 Контрольные вопросы

5.1 С какой максимальной частотой дискретизации может оцифровывать сигнал микроконтроллер PIC16F877?

5.2 Можно ли использовать микроконтроллер PIC16F877 для оцифровки 6-ти каналов ЭКГ?

5.3 Можно ли использовать микроконтроллер PIC16F877 для оцифровки 6-ти каналов ФКГ?

5.4 С какой целью в микроконтроллере используются выводы VREF+ и VREF- ?

5.5 Какие плюсы и какие минусы дает RC генератор в качестве источника тактовых импульсов для АЦП?

5.6 Из-за каких настроек микроконтроллера модуль АЦП может выдавать неверный результат?

5.7 Рассчитайте, какую погрешность, обусловленную разрядностью АЦП, вносит микроконтроллер при регистрации сигнала с максимальной амплитудой 0.1 В?