Министерство образования Республики Беларусь
Учреждение образования
«Белорусский государственный университет
информатики и радиоэлектроники»

Факультет компьютерного проектирования

Кафедра электронной техники и технологии

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА PIC. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

Рекомендовано УМО по образованию в области информатики и радиоэлектроники в качестве пособия для специальностей 1-36 04 01 «Программно-управляемые электронно-оптические системы», 1-39 02 02 «Проектирование и производство программно-управляемых электронных средств», 1-39 02 03 «Медицинская электроника»

Минск БГУИР 2020

УДК 004.312.46(076.5)
ББК 32.973.26-04я73
П78

Авторы:

П. В. Камлач, М. В. Давыдов,
И. И. Ревинская, Д. П. Куничников

Рецензенты:

кафедра интеллектуальных и мехатронных систем
Белорусского национального технического университета
(протокол №12 от 24.04.2019);

заведующий кафедрой автоматизированных систем управления производством учреждения образования «Белорусский государственный аграрный технический университет» кандидат технических наук, доцент А. Г. Сеньков

Программирование микроконтроллеров семейства PIC. Лабораторный практикум : пособие / П. В. Камлач [и др.]. – Минск : БГУИР, 2020. – 75 с. : ил.

П78

ISBN 978-985-543-540-3.

Лабораторный практикум составлен в соответствии с программой дисциплины «Программно-управляемые микроконтроллерные устройства» и состоит из пяти лабораторных работ по изучению и разработке программного обеспечения для микроконтроллеров серии PIC.

УДК 004.312.46(076.5)
ББК 32.973.26-04я73

ISBN 978-985-543-540-3

Содержание

Лабораторная работа №1
СОЗДАНИЕ ПРОЕКТА В MPLAB X IDE И РАБОТА С PICSIMLAB

Цель работы: сформировать общее представление о разработке программного обеспечения для микроконтроллеров серии PIC; создать проект и научиться управлять портами ввода/вывода микроконтроллера.

1.1 Установка необходимого программного обеспечения

1.1.1 Установка MPLAB X IDE

Для программирования микроконтроллеров требуется какая-либо среда программирования, а также компилятор. У компании Microchip есть бесплатный IDE – это MPLAB X IDE, которую можно скачать с официального сайта Microchip. Надо пройти по пунктам меню: DESIGN SUPPORT → Development Tools Software Tools For PIC® MCUs And DsPIC® DSCs → MPLAB® X IDE (или установить из учебного комплекса MPLAB X-v4.15-windows-installer.exe) (рисунок 1.1).

Рисунок 1.1 – Рабочее окно сайта Microchip: выбор MPLAB X IDE

На следующей странице необходимо выбрать закладку Downloads, в которую скачаем последнюю версию среды разработки. После скачивания необходимо установить MPLAB X IDE.

1.1.2 Установка MPLAB® XC Compilers

Для написания кода на языке С потребуется компилятор. Компиляторы для контроллеров разной битности (8, 16 и 32) отдельные. На официальном сайте Microchip проследуем по пунктам меню: DESIGN SUPPORT → Development Tools → Software Tools For PIC® MCUs And DsPIC® DSCs → MPLAB® XC Compilers (или установить из учебного комплекса xc8-v1.45-full-install-windows-installer.exe) (рисунок 1.2).

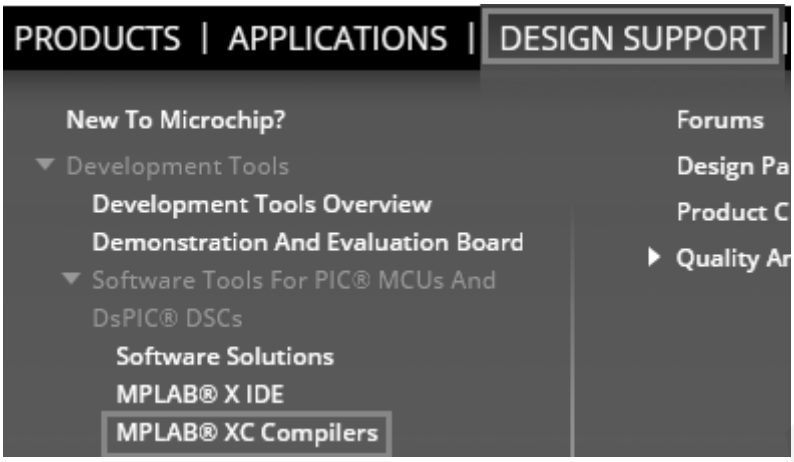


Рисунок 1.2 – Рабочее окно сайта Microchip: выбор MPLAB® XC Compilers

На открывшейся странице нужно перейти по закладке Downloads и скачать последнюю версию 8-битного компилятора (XC 8). После скачивания установить MPLAB® XC Compilers.

1.1.3 Установка PicsimLab

Для работы с микроконтроллером необходимы сам микроконтроллер и программатор для записи бинарного кода программы в ПЗУ. В учебных целях можно использовать свободно распространяемый программный эмулятор отладочных плат PicsimLab. Данный эмулятор можно скачать по ссылке https://sourceforge.net/projects/picsim/ (или установить из учебного комплекса picsimlab_0_7_win_setup.exe) (рисунок 1.3). Во вкладке Files выберите необходимую версию программы.

Рисунок 1.3 – Рабочее окно сайта SOURCEFORGE

1.2 Создание проекта в MPLAB X IDE

Запустите среду программирования MPLAB X IDE и перейдите на вкладку Projects (рисунок 1.4).

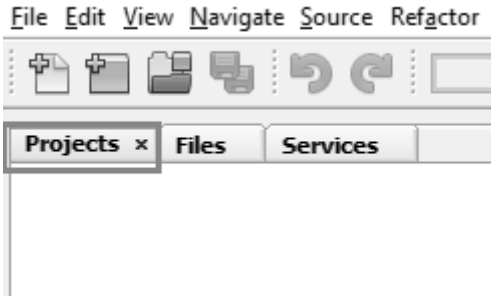


Рисунок 1.4 – Рабочее окно MPLAB X IDE: выбор Projects

Создайте новый проект. Для этого выберите пункт меню File → New Project. Выбираем Standalone Project и нажимаем Next> (рисунок 1.5).

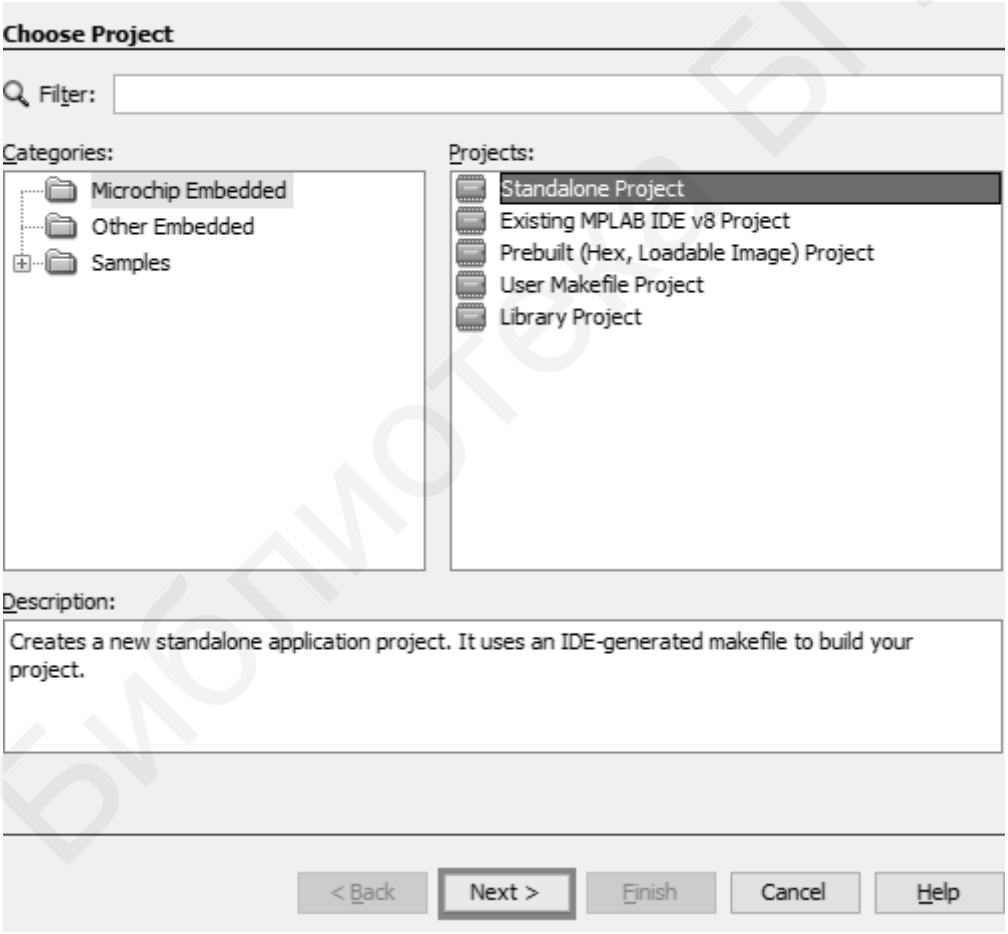


Рисунок 1.5 – Рабочее окно MPLAB X IDE: выбор Standalone Project

Выберите из выпадающего списка контроллер, для удобства можно отфильтровать список по семейству. Первый проект создайте для контроллера PIC16F84A (рисунок 1.6).

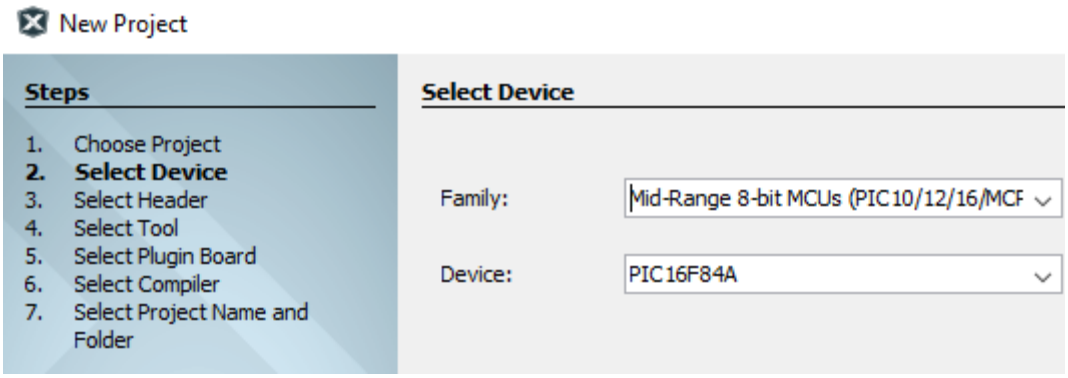


Рисунок 1.6 – Рабочее окно MPLAB X IDE: выбор контроллера PIC16F84A

Выберите Simulator, так как именно здесь необходимо отлаживать проект (рисунок 1.7).

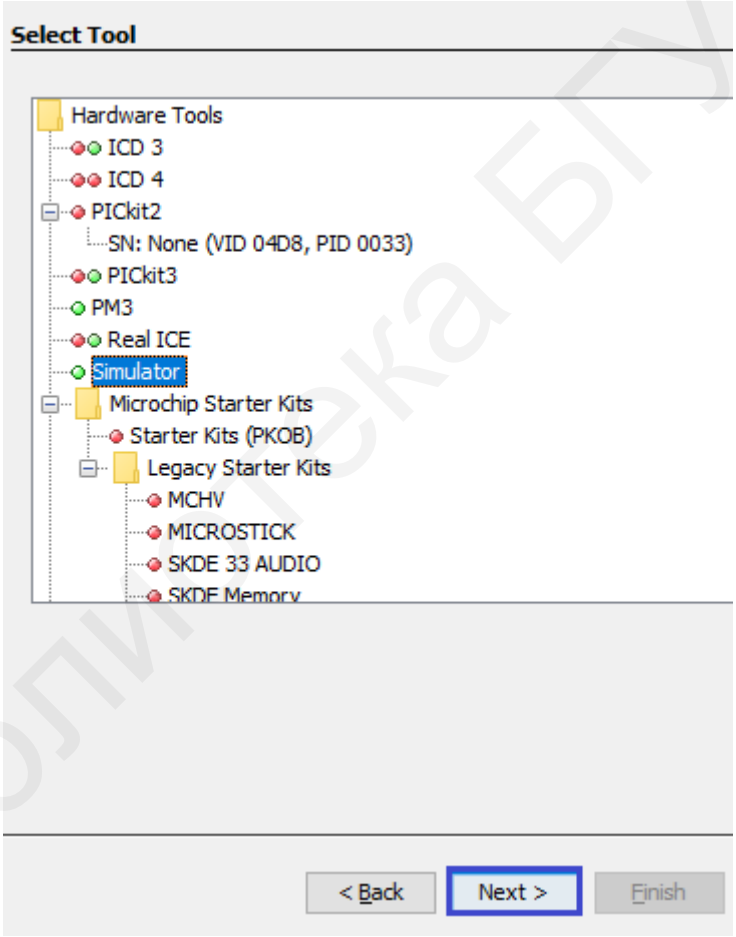


Рисунок 1.7 – Рабочее окно MPLAB X IDE: выбор Simulator

В следующем окне выберите установленный ранее компилятор, а еще в следующем – назовите ваш первый проект (предлагается BLINK01), выберите папку для его хранения и галочкой отметьте, что данный проект стал главным, затем нажимаете Finish.

Проект появится в дереве проектов. Создайте в нем файл main.c, выбрав соответствующий пункт контекстного меню в папке Source Files (рисунок 1.8).

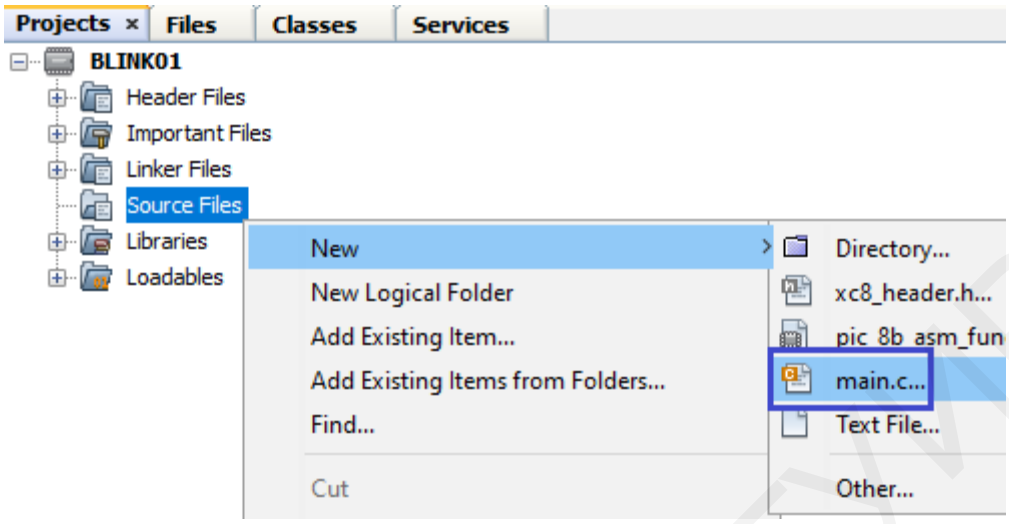


Рисунок 1.8 – Рабочее окно MPLAB X IDE: создание файла main.c

Файл откроется самостоятельно. В нем будет уже некоторый код:

```
#include <xc.h>
void main(void)
{
}
return;
```

То есть у нас уже подключена стандартная библиотека, которой нам в первое время будет достаточно, так как в ней уже находится подключение многих необходимых библиотек. И также у нас есть точка входа в программу – главная функция main.

Контроллеру необходимо постоянно находиться в работе. Для этого необходимо в каждый проект в главную функцию добавлять бесконечный цикл. Добавьте такой цикл и в этот проект:

```
void main(void) {
 while (1)
 {
 }
 return;
```

Соберите проект, нажав соответствующую кнопку на панели инструментов (рисунок 1.9).

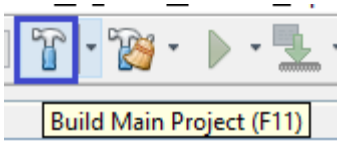


Рисунок 1.9 – Рабочее окно MPLAB X IDE: сбор проекта

В окне вывода сообщений, в самом конце информационного сообщения, находится путь к файлу. Этот файл прошивки будет записываться в контроллер.

1.3 Ознакомительное руководство с PICSimLab

PICSimLab означает PIC Simulator Laboratory.

PICSimLab – это эмулятор разработки в реальном времени с интегрированным отладчиком MPLAB X/avr-gdb. PICSimLab поддерживает микроконтроллеры PIC (PIC16F84, PIC16F628, PIC16F648, PIC16F777, PIC16F877A, PIC18F452, PIC18F4520, PIC18F4550 и PIC18F4620) и микроконтроллер simavr (ATMEGA328). PICSimLab имеет интеграцию с MPLAB X/Arduino IDE для программирования микроконтроллеров плат.

1.3.1 Интерфейс

Главное окно состоит из меню, строки состояния, выпадающего списка выбора частоты, кнопки включения/выключения для запуска отладки, некоторых элементов управления для конкретной платы и части интерфейса платы (рисунок 1.10).

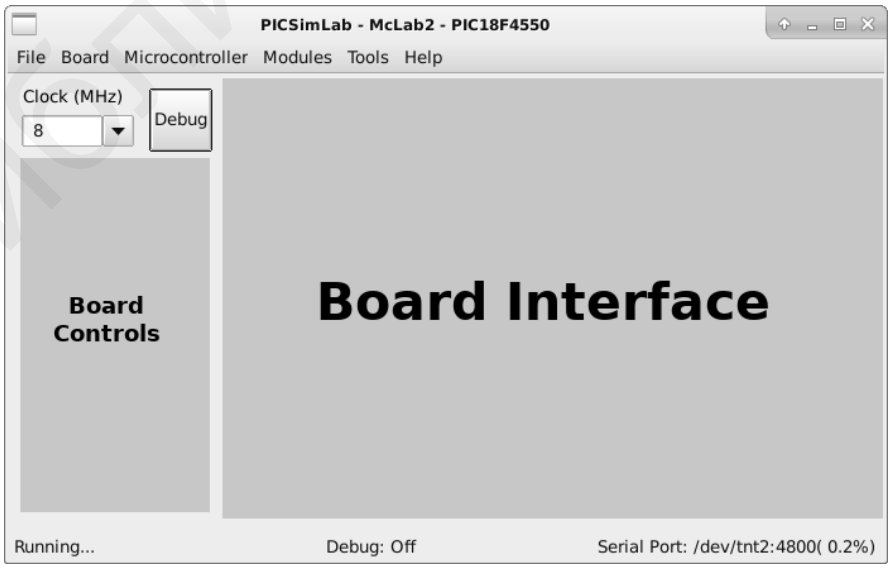


Рисунок 1.10 – Рабочее окно PICSimLab: главное окно

В названии окна отображаются имя симулятора PICSimLab, а затем плата и используемый микроконтроллер.

Комбобокс выбора частоты напрямую изменяет рабочую скорость микроконтроллера, когда метка *Clock* (*MHz*) красного цвета означает, что компьютер не может запустить программу в реальном времени для выбранных часов. В этом случае симуляция может отличаться от ожидаемой, и загрузка ЦП будет увеличена.

1.3.2 Меню PICSimLab

Меню и их функции следующие:

- File (файл);
- Load Hex – загрузка файла *.hex;
- *Reload Last* – перезагрузка последнего использованного файла *.hex;
- *Configure* – открыть окно конфигурации;
- Save Workspace – сохранить все текущие настройки рабочего пространства в файле *.pzw;
- *Load Workspace* – загрузить сохраненные настройки из файла *.pzw;
- Exit;
- *Board* (плата):
- Mclab1;
- K16f;
- Mclab2;
- Picgenios;
- Arduino;
- Microcontroller (микроконтроллер):

xxxxx – выбор используемого микроконтроллера (зависит от выбранной платы);

- *Modules* (модули):
- Oscilloscope – открыть окно осциллографа;
- Spare parts – открыть окно запасных частей;
- *Tools* (инструменты):
- Serial term – открыть последовательный терминал Cutecom;
- *Help* (Помощь):
- Contents – открыть окно справки;
- Examples – загрузить примеры;
- About – показать сообщение об авторе и версии.

Первая часть строки состояния показывает состояние моделирования, в средней части – статус поддержки отладки, а в последней части – имя используемого последовательного порта, его скорость по умолчанию и ошибку относительно реальной скорости, настроенной в микроконтроллере.

1.3.3 Команды

На области интерфейса платы можно взаимодействовать следующим образом:

- нажмите на разъем ICSP, чтобы загрузить файл .hex;
- нажмите кнопку PWR для включения/выключения эмулятора;
- кнопки можно активировать с помощью мыши или клавиш 1, 2, 3, 4.

1.4 Порты ввода/вывода микроконтроллера

Порты ввода/вывода (ПВВ) предназначены для общения микроконтроллера с внешними устройствами. С их помощью передается информация другим устройствам и принимается от них. В зависимости от типа микроконтроллер может иметь на своем борту от один и более ПВВ. Каждому порту ввода/вывода присвоено буквенное обозначение: A, B, C, D, E, F, G. Все порты в микроконтроллере (они же выводы, они же разряды, они же биты) N-разрядные (содержат N линий) и двунаправленные – могут как передавать, так и принимать информацию. ПВВ в микроконтроллере обслуживают все его устройства, в том числе и периферийные. Поэтому в зависимости от того, какое устройство будет работать с ним, порт может принимать и передавать или цифровую информацию, или аналоговую.

Порты классифицируются по типу сигнала:

- цифровые порты – порты, которые работают с цифровыми сигналами – логическими нулями и логическими единицами;
- аналоговые порты, порты, которые работают с аналоговыми сигналами – использующими плавно весь диапазон входных напряжений от нуля вольт до напряжения питания МК;
- смешанные порты – порты, которые могут оперативно переключаться с режима «цифровой порт» в режим «аналоговый порт», и обратно.

В технической литературе и схемах ПВВ обозначаются следующим образом:

- P – первая буква, означающая слово «порт»;
- A (B, C, D, E, F, G) – вторая буква, обозначающая конкретный порт;
- 0 (1, 2, 3, 4, 5, 6, 7) – третий символ – цифра, обозначающая конкретный вывод (регистр, бит) порта.

Например, порт A – PA, пятый разряд порта A – PA5.

Если в МК есть несколько портов, то необязательно их имена могут идти по порядку: A, B, C. Может быть и так: B, C, D.

Кроме того, хотя порты N-разрядные, выводов у порта не обязательно должно быть N – может быть и меньше, например, три: PA0, PA1, PA2. В таком случае порт называют неполным, или урезанным.

Для управления портами в их электрической схеме имеется два переключателя, которыми мы можем «щелкать» программно, используя специальные регистры ввода/вывода. Такие переключатели имеются для каждого вывода, что означает возможность управлять любым выводом порта. К примеру, один вывод порта можно настроить на ввод информации, три разряда этого же порта – на вывод, а оставшиеся – вообще не настраивать, т. е. оставить их в Z-состоянии.

1.4.1 Цифровые входы

Входную цепь МК в цифровом режиме проще всего представить в виде логического КМОП-элемента (рисунок 1.11), который по параметрам примерно соответствует стандартным сериям микросхем 74HC, 74AC, К561. Примерно, потому что есть разница в электрических характеристиках продукции разных фирм, изготавливающих МК.

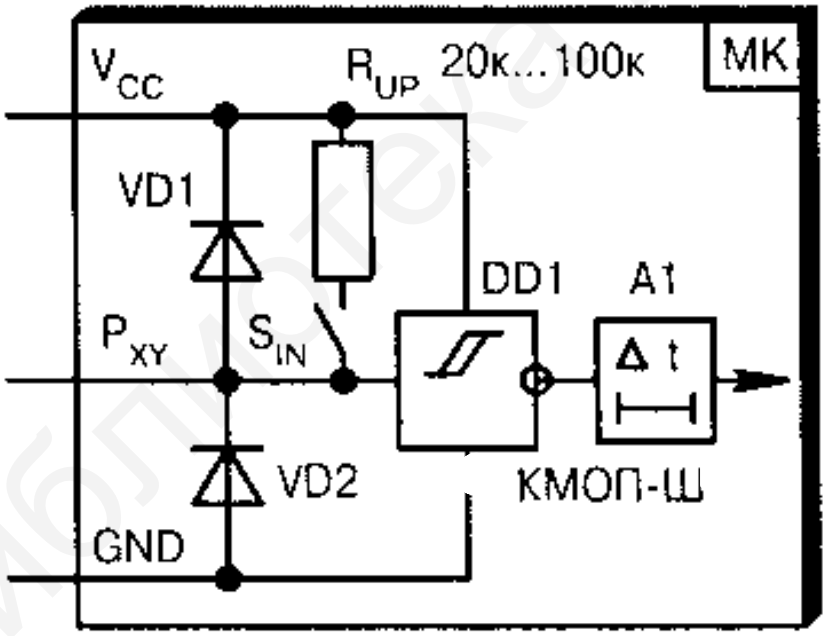
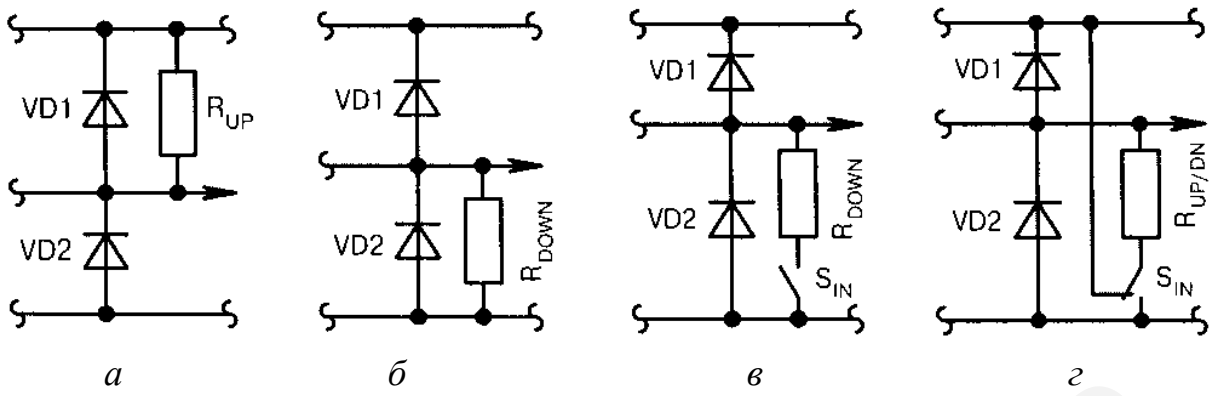


Рисунок 1.11– Схема цифрового входа МК с pull-up резистором R_{UP}

В отличие от микросхем обычной логики, на входе МК находится подтягивающий резистор R_{UP} и программно управляемый ключ S_{IN} (рисунок 2.25). Если ключ разомкнут, то входное сопротивление МК очень велико. В некоторых семействах резистор присутствует постоянно или же он «притягивается» к общему проводу через отдельный ключ (рисунок 1.12).



а *б* *в* *г*

а – постоянное соединение резистора с цепью питания;

б – постоянное соединение резистора с общим проводом;

в – программное подключение резистора к общему проводу;

г – программный выбор резистора pull-up/pull-down

Рисунок 1.12 – Варианты подключения внутренних резисторов в МК

Логический элемент DD1 (см. рисунок 1.11) выполняется по технологии КМОП и имеет характеристику триггера Шмитта (рисунок 1.13). Это повышает помехоустойчивость. Той же цели служит линия задержки A1, которая отсекает короткие импульсные помехи.

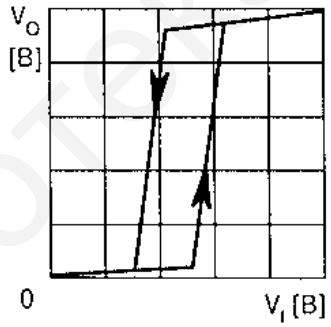


Рисунок 1.13 – Передаточная характеристика элемента «триггер Шмитта»

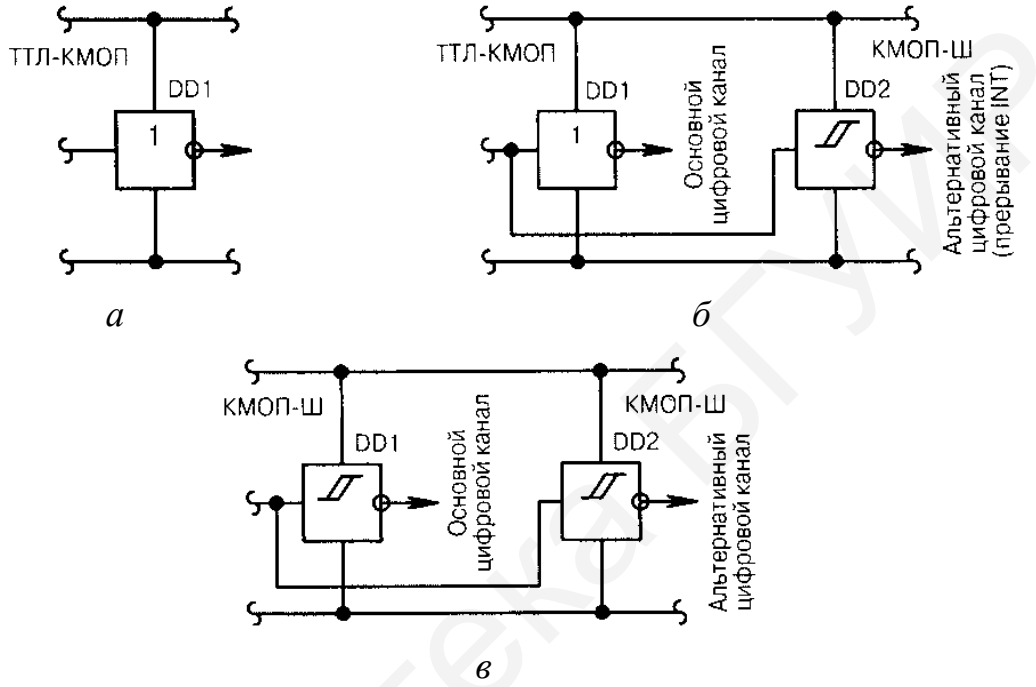
Резисторы R_{UP}, R_{DOWN} изготавливаются на основе интегральных полевых транзисторов. Большой технологический разброс определяет широкие допуски, вплоть до ±50 % от номинала, например, 30…100 кОм.

Редко, но встречаются МК с индивидуальной настройкой сопротивления входных резисторов.

Резисторы имеют англоязычные названия: pull-up (тянуть вверх), pull-down (тянуть вниз). Если для коммутации резисторов применяется ключ (switch), то используются похожие термины: switched pull-up, switched pulldown.

Диоды VD1, VD2 защитные, антистатические. Они присутствуют в МК так же, как и в обычных микросхемах КМОП-логики. При входном напряжении выше уровня V_{CC} открывается диод VD1. При уменьшении входного напряжения ниже уровня GND открывается диод VD2.

Схемные разновидности цифровых входов показаны на рисунке 1.14. Сокращение ТТЛ-КМОП означает логический элемент, выполненный по технологии КМОП, но имеющий входные уровни, совместимые с ТТЛ (+0.4/+2.4 В).



а – вход ТТЛ, выход КМОП;
б – параллельное соединение элементов ТТЛ-КМОП и КМОП-Ш;
в – параллельное соединение двух одинаковых элементов КМОП-Ш
Рисунок 1.14 – Варианты включения логического элемента DD1

Важный практический нюанс, общий для всех входных цифровых схем: если линии портов настроены как входы без внутренних pull-up резисторов и к ним ничего извне не подключается, то потребляемый МК ток увеличивается чуть ли не на четверть. Это заставляет хаотично работать логические элементы внутри МК, что приводит к увеличению динамических потерь и повышению энергопотребления. Рекомендуется все неиспользуемые в схеме входы надо снаружи или изнутри нагружать на резисторы сопротивлением 10…100 кОм или же переводить линии в режим цифровых выходов.

1.4.2 Аналоговые входы

Аналоговый режим отличается от цифрового уровнями подаваемых на вход сигналов. Для цифрового режима это стандартные двоичные перепады напряжения НИЗКИЙ/ВЫСОКИЙ. Для аналогового режима допускаются сигналы любой формы в диапазоне 0…V_{CC}.

Изолированные аналоговые входы в МК встречаются редко. В целях экономии места их совмещают с цифровыми линиями и используют как альтернативную функцию порта. В условном изображении МК такие выводы легко идентифицируются по аббревиатуре, начинающейся с первой буквы латинского алфавита: AN, ADC4, AIN и т. д. Примером служат линии RA0/AN0 в микросхеме Microchip PIC16F628A или PC1/ADC1/PCINT9 в микросхеме Atmel ATmegal68.

Аналоговые входы МК отождествляются с аналоговым компаратором и АЦП (усилитель с программируемым коэффициентом усиления).

Принцип работы аналогового компаратора (рисунок 1.15) заключается в сравнении сигналов на положительном («+», AIN0) и отрицательном («–», AIN1) входах. Если на «положительном» входе напряжение станет больше, чем на «отрицательном», то выходной сигнал компаратора DA1 станет ВЫСОКИМ, и наоборот. Результат сравнения помещается в один из регистров из области SFR. В некоторых МК сигнал AIN физически выводится наружу, что показано на схеме пунктиром. Программно управляемый переключатель S1 служит для подачи на вход компаратора сигнала от внутреннего источника опорного напряжения (ИОН) или от внешнего вывода AIN0. В некоторых семействах МК переключатель S1 многопозиционный и обеспечивает не два, а несколько дополнительных вариантов коммутации.

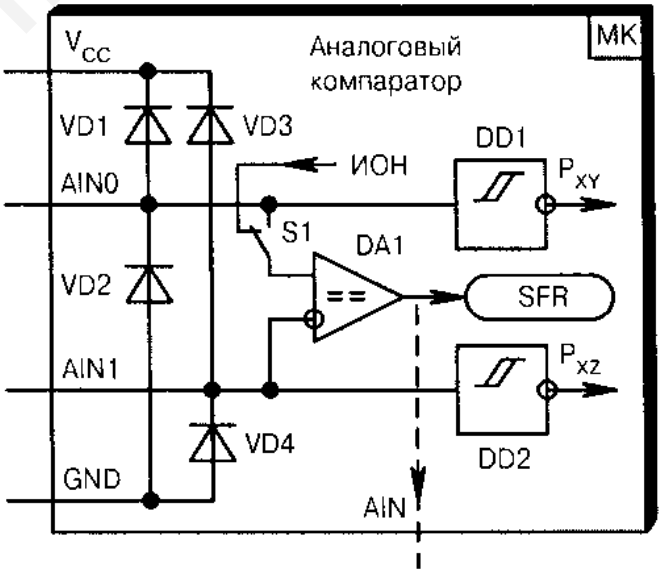


Рисунок 1.15 – Структурная схема узла аналогового компаратора

Диоды VD1…VD4 и логические элементы DD1, DD2 относятся к цифровому порту, но они имеют очень высокое сопротивление и на точность работы компаратора не влияют.

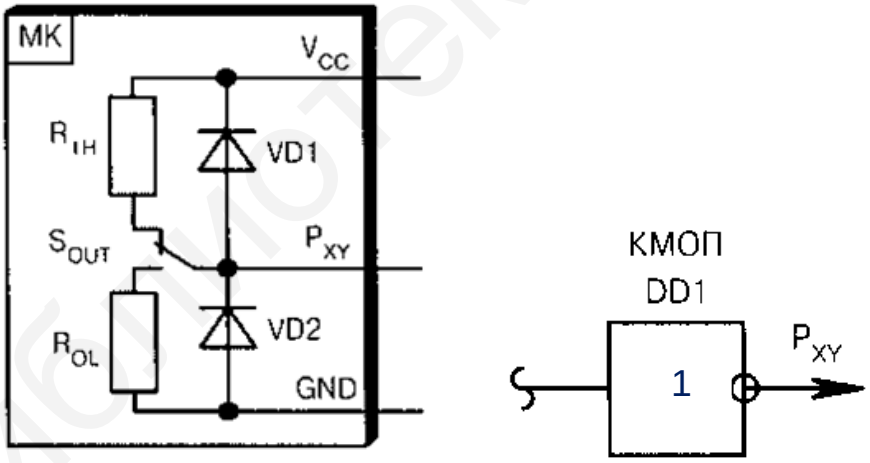
Принцип работы многоканального АЦП в МК заключается в измерении абсолютного уровня входного сигнала методом последовательных приближений (чаще) или дельта-сигма-преобразованием (реже). Основным параметром АЦП считается разрядность.

1.4.3 Цифровые выходы

Основная функция цифровых выходов в МК заключается в формировании НИЗКИХ и ВЫСОКИХ логических уровней. Выходную цепь в цифровом режиме можно представить КМОП-элементом, который имеет «закрытый» или «открытый» выход со стандартной или повышенной нагрузочной способностью. Прототипы подобных элементов широко представлены в сериях микросхем 74HC, 74AC, КР1561, ICP1554.

Различают четыре основных варианта организации выходов в МК:

1. Двухтактный каскад с выходными уровнями, близкими к Vcc и GND (рисунок 1.16). Сокращенное название – push-pull. Встречается практически во всех микроконтроллерных семействах.

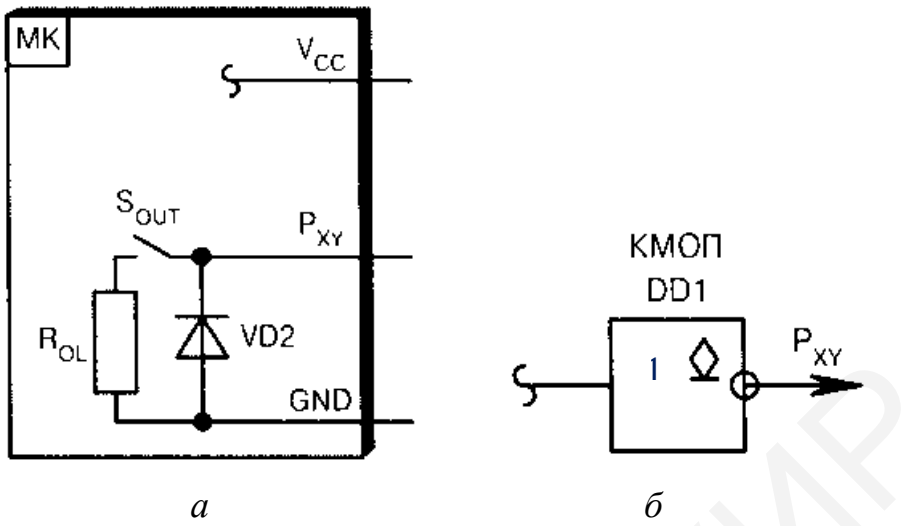


а *б*

а – внутреннее устройство; *б* – эквивалентный логический элемент

Рисунок 1.16 – Двухтактный цифровой выход push-pull

2. Логический элемент с истинно открытым стоком, или «с открытым коллектором» (разг.) (рисунок 1.17). Сокращенное название – open drain, или true open drain. Встречается в семействах Microchip PIC12/16, STMicroelectronics STR71xF, в МК с ядром MCS-51.

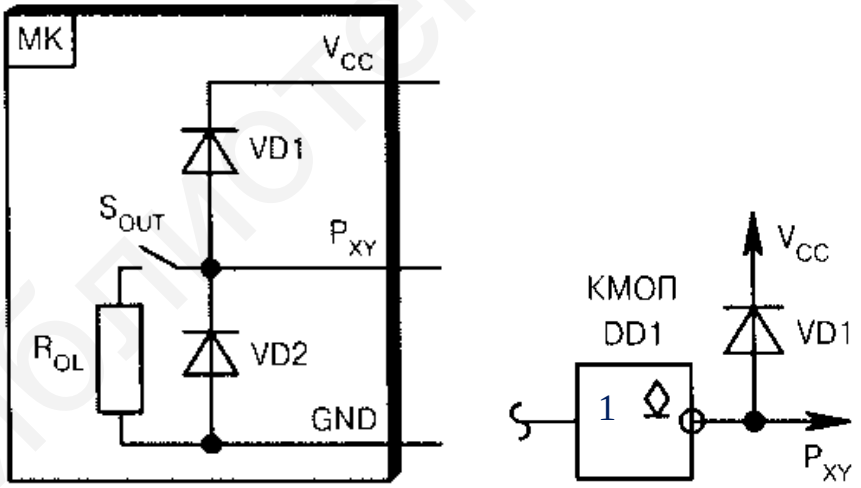


а *б*

а – внутреннее устройство; *б* – эквивалентный логический элемент

Рисунок 1.17 – Выход с открытым стоком open drain

3. Логический элемент с квазиоткрытым стоком (рисунок 1.18). Сокращенное название – false open drain (из-за присутствия ограничительного диода VD1). Встречается в МК с двунаправленными портами.

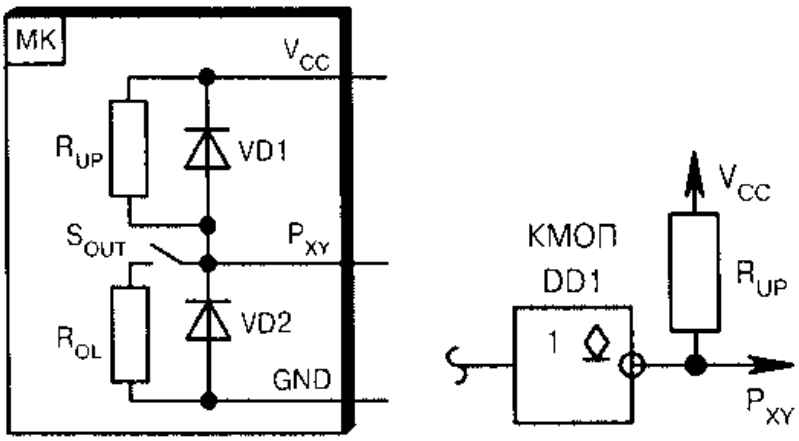


а *б*

а – внутреннее устройство; *б* – эквивалентный логический элемент

Рисунок 1.18 – Выход с квазиоткрытым стоком

4. Логический элемент с открытым стоком, дополненный внутренним нагрузочным резистором (рисунок 1.19). Сокращенное название – квазидвунаправленный. Встречается в МК с ядром MCS-51.



а *б*

а – внутреннее устройство; *б* – эквивалентный логический элемент

Рисунок 1.19 – Квазидвунаправленный выход

Для облегчения понимания физических процессов в функциональные схемы вводятся эквивалентные переключатели. Они замещают реальные электронные ключи, выполненные на полевых транзисторах.

1.5 Программа «Бегущие огни»

Ниже приведен код, который будет управлять свечением светодиодов, подключенных к ножкам портов микроконтроллера. То есть за счет поочередного поступления положительного потенциала от ножек портов на ножки светодиодов, а также нулевого номинала на другие ножки создается поочередное свечение и потухание светодиодов – эффект «бегущих огней».

Сначала давайте разберем схему с контроллером PIC16F84A и подключенными через токоограничивающие резисторы светодиодами в программе-стимуляторе PICSimLab (рисунок 1.20).

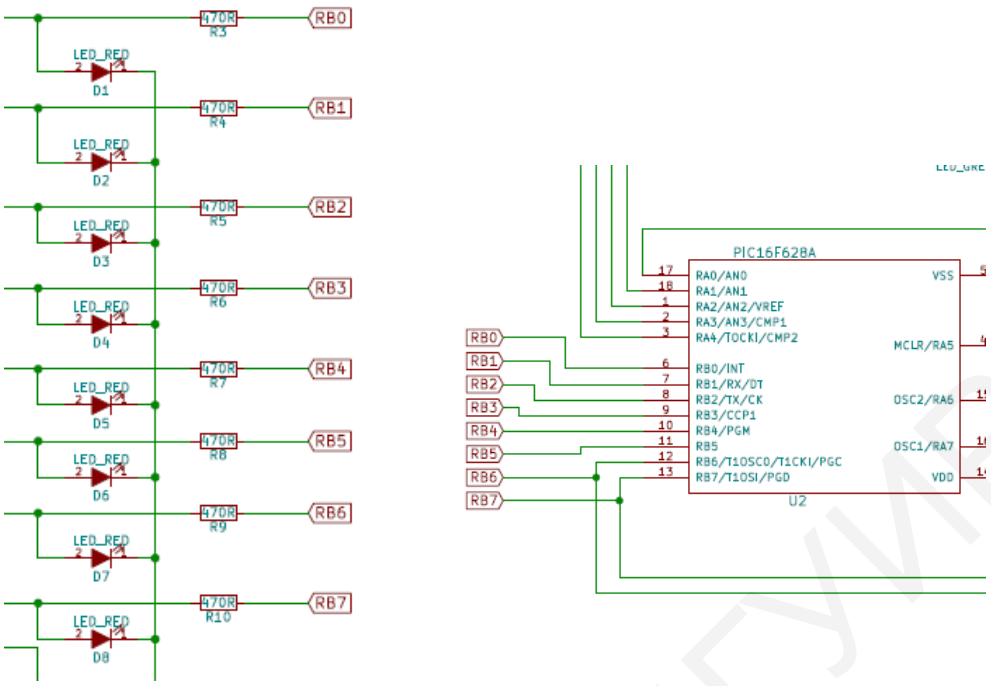


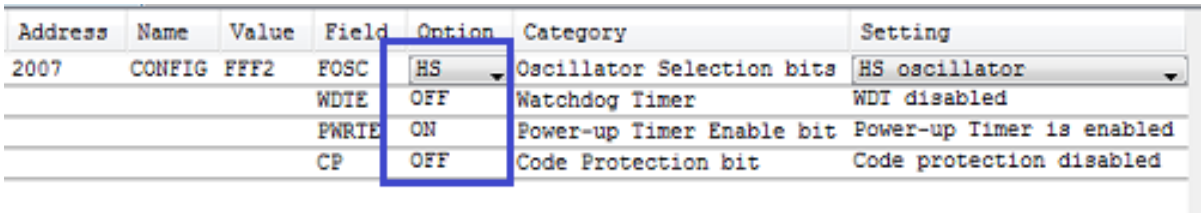
Рисунок 1.20 – Часть схемы платы McLab1 Board1

Из схемы видно, что светодиоды подключены к порту B на выводы с 0 по 7. Если подать логическую единицу на RB0, то зажжется светодиод D1, если вернуть логический нуль, то он потухнет.

Далее откроем проект и настроим ножки портов, к которым подключены светодиоды, на выход. Также установим низкий уровень на этих ножках:

```
void main(void) {
 TRISB = 0x00;
 PORTB = 0x00;
 TRISA &= ~0x03;
 PORTA &= ~0x03;
 while(1)
```

Настроим конфигурационные биты. С помощью них мы настраиваем некоторые свойства нашего контроллера. Для удобства их настраивания в среде программирования MPLAB X IDE есть специальный инструмент. Проследуем по пунктам меню: Window → PIC Memory Views → Configuration Bits. И внизу появятся настройки этих самых битов конфигурации. Настроим там следующие значения (рисунок 1.21).



Address	Name	Value	Field	Option	Category	Setting
2007	CONFIG	FFF2	FOSC	HS	Oscillator Selection bits	HS oscillator
			WDTE	OFF	Watchdog Timer	WDT disabled
			PWRTE	ON	Power-up Timer Enable bit	Power-up Timer is enabled
			CP	OFF	Code Protection bit	Code protection disabled

Рисунок 1.21 – Рабочее окно MPLAB X IDE: настройки битов конфигурации

Выбрав нужный кварцевый резонатор – высокоскоростной резонатор 4 МГц – сохраняем конфигурационные биты с помощью нажатия кнопки Generate Source Code to Output и получаем код в окне вывода информации (рисунок 1.22).

```
// CONFIG
#pragma config FOOSC = HS          //oscillator selection bits (HS oscillator)
#pragma config WDTE = OFF //Watchdogod timer
#pragma config PWRTE = ON // Power-up Timer Enable bit (Power-upp Timer is enebled)
#pragma config CP = OFF// Code Protection bit
```

Рисунок 1.22 – Окно вывода информации MPLAB X IDE

Скопируем его и добавим в файл main.c сразу после подключения библиотеки. Заодно сразу объявим частоту резонатора, иначе задержки не будут работать корректно:

```
#include <xc.h>
#define _XTAL_FREQ 4000000
#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF // Watchdog Timer (WDT disabled)
#pragma config PWRTE = ON // Power-up Timer Enable bit (Power-up Timer is enabled)
#pragma config CP = OFF // Code Protection bit (Code protection disabled)
```

Теперь по инструкциям, написанным с помощью макросов в стандартной библиотеке PIC, добавим код в бесконечный цикл:

```
while(1)
{
PORTAbits.RA1 = 0;
PORTBbits.RB0 = 1;
__delay_ms(100);
```

```
PORTBbits.RB0 = 0;
PORTBbits.RB1 = 1;
__delay_ms(100);
PORTBbits.RB1 = 0;
PORTBbits.RB2 = 1;
__delay_ms(100);
PORTBbits.RB2 = 0;
PORTBbits.RB3 = 1;
__delay_ms(100);
PORTBbits.RB3 = 0;
PORTBbits.RB4 = 1;
__delay_ms(100);
PORTBbits.RB4 = 0;
PORTBbits.RB5 = 1;
__delay_ms(100);
PORTBbits.RB5 = 0;
PORTBbits.RB6 = 1;
__delay_ms(100);
PORTBbits.RB6 = 0;
PORTBbits.RB7 = 1;
__delay_ms(100);
PORTBbits.RB7 = 0;
PORTAbits.RA0 = 1;
__delay_ms(100);
PORTAbits.RA0 = 0;
PORTAbits.RA1 = 1;
__delay_ms(100);
}
```

Соберите проект, нажав соответствующую кнопку в панели инструментов. Запустите программу PICSimLab. Выберите плату McLab1 и контроллер PIC16F84A. Запрограммируйте контроллер: File → Load Hex.

1.6 Программа «Бегущие огни при нажатии кнопки»

Изменим данный проект так, чтобы при запуске светодиоды зажигались не сразу, а лишь при нажатии кнопки. Для этого надо настроить ножку порта RA2 на вход соответствующим образом в функции main():

```
PORTA &= ~0x03;
TRISA |= 0x04; //Включим ножку RA2 на вход
```

Теперь над функцией main() напишется функция обнаружения нажатой кнопки, чтобы не загромождать функцию main:

```
unsigned char CheckButton(void)
{
unsigned char result=0;
unsigned int butcount=0;
while(!RA2)
{
if(butcount < 10000)
{
butcount++;
}
else
{
result = 1;
break;
}
}
return result;
}
```

Добавляются две локальные беззнаковые целочисленные переменные: одна разрядностью в 8 бит, а другая – в 16.

Затем проверяется наличие единицы в бите RA2. Если бит установлен, то программа переходит в тело цикла, в котором будем наращивать счетчик butcount с каждым прохождением по циклу при условии, что бит будет оставаться в единице, т. е. кнопка будет все еще нажата. Если мы в таком состоянии достигнем отметки в 10 000, что будет соответствовать определенному количеству миллисекунд, программа не попадет в тело условия, когда счетчик меньше 10 000, а попадет в тело обратного условия, где установлена переменная результата в единицу. Выход происходит из цикла по команде break, а затем и из функции также с положительным результатом. А если программа (можно подобрать и другую) так и не досчитает до 10 000, то тогда выйдет из цикла while, и выйдет из функции, но уже с результатом нуль, что будет свидетельствовать о

том, что кнопка не нажата. Таким образом, не только будет определено состояние кнопки, но еще учтен дребезг контактов.

Ниже приведен код бесконечного цикла:

```
while(1)
{
 if(CheckButton())
 {
  PORTBbits.RB0 = 1;
  __delay_ms(100);
  PORTBbits.RB0 = 0;
  PORTBbits.RB1 = 1;
  __delay_ms(100);
  PORTBbits.RB1 = 0;
  PORTBbits.RB2 = 1;
  __delay_ms(100);
  PORTBbits.RB2 = 0;
  PORTBbits.RB3 = 1;
  __delay_ms(100);
  PORTBbits.RB3 = 0;
  PORTBbits.RB4 = 1;
  __delay_ms(100);
  PORTBbits.RB4 = 0;
  PORTBbits.RB5 = 1;
  __delay_ms(100);
  PORTBbits.RB5 = 0;
  PORTBbits.RB6 = 1;
  __delay_ms(100);
  PORTBbits.RB6 = 0;
  PORTBbits.RB7 = 1;
 }
}
return;
```

1.7 Индивидуальные задания

1. Мигать попеременно светодиодами D1 и D3. Время горения светодиода в два раза больше, чем время простоя. При нажатии кнопки загораются D2 и D4.

2. Мигать попеременно светодиодами D2 и D4. Время горения светодиода в три раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8.

3. Мигать попеременно светодиодами D1 и D5. Время горения светодиода в четыре раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8.

4. Мигать попеременно светодиодами D2 и D6. Время горения светодиода в пять раз больше, чем время простоя. При нажатии кнопки загораются D7 и D9.

5. Мигать попеременно светодиодами D1, D2 и D3. Время горения светодиода в два раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8.

6. Мигать попеременно светодиодами D3, D4 и D8. Время горения светодиода в два раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8.

7. Мигать попеременно светодиодами D3, D4 и D8. Время горения светодиода в два раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8.

8. Мигать попеременно светодиодами D1, D5 и D7. Время горения светодиода в три раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8.

9. Мигать попеременно светодиодами D2, D6 и D8. Время горения светодиода в три раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8.

10. Мигать попеременно светодиодами D2, D6 и D8. Время горения светодиода в четыре раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8.

1.8 Требования к отчету

В отчете необходимо привести полный код программы с комментариями к каждой строке. Сделать выводы.

1.9 Контрольные вопросы

1. Что такое микроконтроллер?
2. Расскажите о назначении IDE.
3. Что такое PICSimLab?
4. Что такое порты ввода/вывода?
5. Какое бывает обозначение портов ввода/вывода?

Лабораторная работа №2
РАБОТА С ТАЙМЕРАМИ-СЧЕТЧИКАМИ

Цель работы: сформировать общее представление о разработке программного обеспечения для микроконтроллеров серии PIC; создать проект и научиться управлять таймерами-счетчиками.

2.1 Теоретические сведения

Таймеры-счетчики предназначаются для формирования временных интервалов и подсчета событий, что позволяет при использовании соответствующего программного обеспечения реализовывать на их основе любые функции времени, в том числе управление в реальном времени (т. е. во временном масштабе объекта).

Обобщенная структура таймера-счетчика показана на рисунке 2.1.

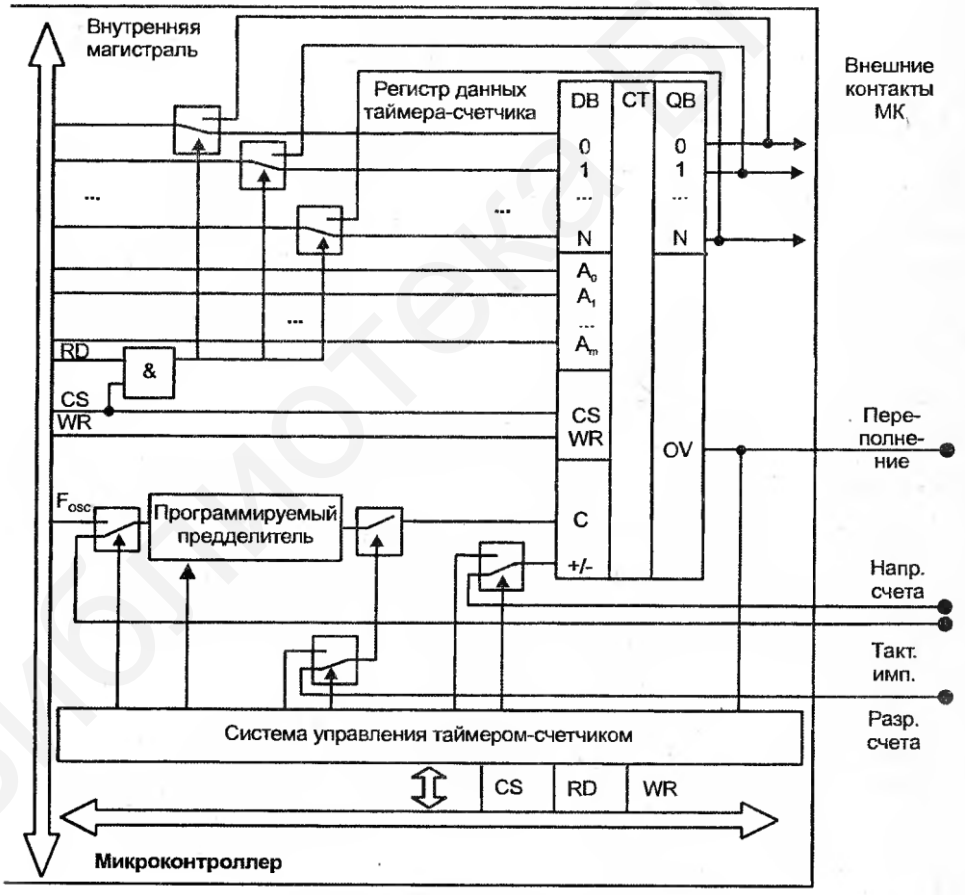


Рисунок. 2.1 – Структура таймера-счетчика

Таймер-счетчик базируется на синхронном двоичном реверсивном счетчике с возможностью параллельной загрузки и чтения информации. Так как разрядность счетчика МК, как правило, превышает разрядность процессора,

используется адресный обмен информацией между ядром МК и частью регистра данных счетчика.

В указанной структуре при одновременной активизации сигналов RD и CS на внутреннюю шину МК коммутируется выходная шина таймера-счетчика, в противном случае внутренняя шина МК коммутируется к входам параллельной загрузки. Собственно загрузка данных в таймер-счетчик произойдет при одновременной активизации сигналов WR и CS.

Система управления позволяет определить источник задания направления счета (программно-доступный бит регистра управления или внешний сигнал), а также источник тактирования (опорная частота ядра или внешние импульсы) и коэффициент деления тактовых импульсов. Полный диапазон счета – от 0 до $2^{(N+1)(m+1)} - 1$. Для задания временных интервалов, отличных от номинального (являющегося максимальным), используется предварительная программная загрузка счетчика (величиной $N = T/T_{кванта}$ при счете на убывание или $N_{max} - N$ при счете на возрастание).

При достижении кода 0 (при счете на убывание) или N_{max} (при счете на возрастание) генерируется сигнал переполнения OV (Overflow), доступный как для программных средств (при чтении регистра состояния системы управления таймером-счетчиком), так и для внешней аппаратуры.

Следует отметить, что решение задачи выдержки временных интервалов возможно применением чисто программных средств: зная время выполнения некоторой инструкции t, несложно организовать ее N-кратное выполнение в цикле, тем самым обеспечив задержку длительностью $T = Nt$, однако в этом случае невозможно выполнение иных программных действий (например, сбора информации, обмена данными с оператором и пр.); кроме того, значение кванта t может оказаться неудовлетворительно большим.

Таймеры-счетчики реализованы практически во всех моделях МК.

Таймер 0 или TMR0 микроконтроллера PIC, как он обозначен в технической документации, – это 8-разрядный таймер-счетчик, который считает только от 0 до 255, и как только он достигает данной величины, происходит прерывание, если оно задействовано. В ходе программы в любой момент можно узнать значение счетчика. Также можно устанавливать скорость счета посредством использования предделителя, выбирать внутренний или внешний источник тактирования таймера, активный фронт (инкрементирование значения счетчика по положительному или отрицательному фронту), но только при условии, что источник тактового сигнала внешний.

Блок-схема данного таймера представлена на рисунке 2.2.

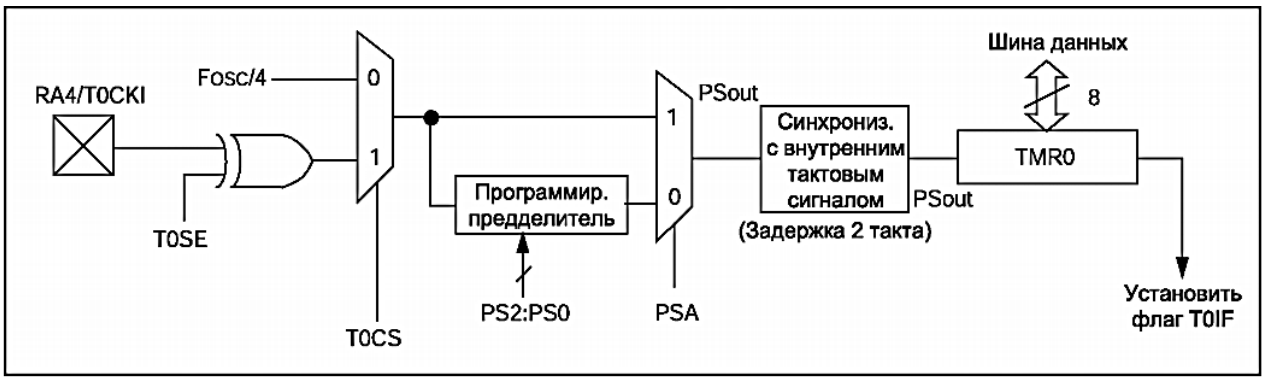


Рисунок 2.2 – Блок-схема таймера

В схеме (см. рисунок 2.2) показаны все блоки таймера, а также указаны биты, которые управляют данными блоками. Биты T0CS, T0SE, PSA, PS2:PS0 расположены в регистре OPTION_REG (рисунок 2.3).

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

Рисунок 2.3 – Регистр OPTION_REG

Данный регистр служит для управления различными параметрами и предназначен не только для таймера. Биты, предназначенные именно для таймера, следующие:

1. TOCS (TMR0 Clock Source Select) – выбор сингала для таймера: 0 – внутренний тактовый сигнал, 1 – внешний.

2. T0SE (TMR0 Source Edge Select) – выбор фронта приращения при внешнем тактовом сигнале: 0 – по переднему фронту, 1 – по заднему.

3. PSA (Prescaler Assignment) – выбор способа включения предделителя: 0 – предделитель включен через TMR0, 1 – через WDT.

4. PS2:PS0 (Prescaler Rate Select) – коэффициент деления предделителя (рисунок 2.4).

Bit Value	TMR0 Rate	WDT Rate
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 2.4 – Значения предделителя
в зависимости от значения PS2:PS0

Особенность работы таймера – это возможность свободно изменять значение счета в регистре, но любое изменение в нем вызовет запрещение наращивания счета в течение двух машинных циклов.

На рисунке 2.5 представлены биты регистра INTCON, предназначенные для управления прерываниями.

INTCON REGISTER (ADDRESS 0Bh, 8Bh)

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	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
bit 7							bit 0

Рисунок 2.5 – Регистр INTCON

В данном регистре потребуются три бита для работы таймера:

1. GIE (Global Interrupt Enable) – разрешение глобальных прерываний: 0 – все прерывания запрещены, 1 – все немаскированные прерывания разрешены.

2. T0IE (TMR0 Overflow Interrupt Enable) – разрешение прерывания по переполнению таймера 0: 0 – прерывание запрещено, 1 – прерывание разрешено.

3. T0IF (TMR0 Overflow Interrupt Flag) – флаг прерывания по переполнению таймера 0: 0 – внешнего прерывания нет, 1 – произошло переполнение счетчика таймера 0 (сбрасывается программно).

На рисунке 2.6 представлены биты регистра значения счета таймера.

Addr	Name	Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0	Value on Power-on RESET
01h	TMR0	8-bit Real-Time Clock/Counter								xxxx xxxx

Рисунок 2.6 – Регистр значения счета таймера

2.2 Пример программы «Бегущие огни по таймеру»

В файле main.c (возьмите его из предыдущей лабораторной работы) в функции main сначала включите нужные нам биты в регистре OPTION_REG:

```
TRISA |= 0x04;
OPTION_REG=0x07;
```

Эта операция включает биты 0, 1 и 2 (это биты предделителя), использует максимальное деление частоты – на 256. То есть частота приращения счета будет 1 000 000/256, или приблизительно 3,9 кГц. Таймер будет считать от 0 до 255, еще раз делим на 256 – и получаем приблизительно 15,3 Гц. Прерывание происходит по окончании счета или по переполнении счетчика. Исходя из

этого и получается, что обработка прерывания будет вызываться именно с такой частотой. То есть светодиоды будут «бежать» с периодом около 65 мс.

Далее включаются глобальные прерывания и прерывания от таймера установкой в один бит седьмого и пятого регистров INTCON:

```
OPTION_REG = 0x07;
INTCON = 0xA0;
```

И затем заносится нуль в регистр счета таймера – запустится таймер:

```
INTCON = 0xA0;
TMR0 = 0;
```

Необходимо добавить глобальную переменную для счета вхождений в обработчик прерывания:

```
#pragma config CP = OFF // Code Protection bit (Code protection disabled)
//-----------------------------------------
unsigned int TIM0_count = 0;
//-----------------------------------------
```

Также необходимо добавить специальную функцию – обработчик прерывания от таймера 0 выше функции main():

```
//-----------------------------------------
void interrupt timer0()
{
}
//-----------------------------------------
```

В данную функцию программа будет попадать, когда происходит переполнение счета, т. е. когда счет таймера достигнет числа 255.

Глобальную переменную необходимо инкрементировать каждый раз, когда программа будет попадать в данную функцию.

В данном случае восемь светодиодов будут по очереди зажигаться, а предыдущий гаснуть – и так каждое вхождение. Чтобы это было проще реализовать, необходимо поделить значение глобальной переменной на восемь – это даст нам возможность получать значение от нуля до семи каждом вхождении в

функцию. И в зависимости от цифры будет зажигаться соответствующий светодиод. Для этого воспользуемся оператором switch:

```
void interrupt timer0()
{
 switch(TIM0_count % 10)
 {
 case 0:
 PORTAbits.RB7 = 0;
 PORTBbits.RB0 = 1;
 break;
 case 1:
  PORTBbits.RB0 = 0;
  PORTBbits.RB1 = 1;
  break;
 case 2:
  PORTBbits.RB1 = 0;
  PORTBbits.RB2 = 1;
  break;
 case 3:
  PORTBbits.RB2 = 0;
  PORTBbits.RB3 = 1;
  break;
 case 4:
  PORTBbits.RB3 = 0;
  PORTBbits.RB4 = 1;
  break;
 case 5:
  PORTBbits.RB4 = 0;
  PORTBbits.RB5 = 1;
  break;
 case 6:
  PORTBbits.RB5 = 0;
  PORTBbits.RB6 = 1;
  break;
 case 7;
  PORTBbits.RB7 = 1;
  break;
```

```
 }
}
```

Далее необходимо инкрементировать переменную счета:

```
 break;
 }
 TIM0_count++;
}
```

Так как 16-битная переменная для счета беззнаковая, то она будет считать до 65 535. И получится, что в самом конце она досчитает только до пяти в состоянии, разделенном по модулю на десять. Поэтому обнулим ее раньше, чтобы все циклы доходили до девяти при разделении на десять по модулю:

```
TIM0_count++;
if(TIM0_count > 3999)
{
 TIM0_count = 0;
}
```

2.3 Индивидуальные задания

1. Мигать попеременно светодиодами D1 и D3. Время горения светодиода в два раза больше, чем время простоя. При нажатии кнопки загораются D2 и D4. Организовать мигание с помощью прерываний от таймера.
2. Мигать попеременно светодиодами D2 и D4. Время горения светодиода в три раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8. Организовать мигание с помощью прерываний от таймера.
3. Мигать попеременно светодиодами D1 и D5. Время горения светодиода в четыре раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8. Организовать мигание с помощью прерываний от таймера.
4. Мигать попеременно светодиодами D2 и D6. Время горения светодиода в пять раз больше, чем время простоя. При нажатии кнопки загораются D7 и D9. Организовать мигание с помощью прерываний от таймера.
5. Мигать попеременно светодиодами D1, D2 и D3. Время горения светодиода в два раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8. Организовать мигание с помощью прерываний от таймера.

6. Мигать попеременно светодиодами D3, D4 и D8. Время горения светодиода в два раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8. Организовать мигание с помощью прерываний от таймера.

7. Мигать попеременно светодиодами D3, D4 и D8. Время горения светодиода в два раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8. Организовать мигание с помощью прерываний от таймера.

8. Мигать попеременно светодиодами D1, D5 и D7. Время горения светодиода в три раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8. Организовать мигание с помощью прерываний от таймера.

9. Мигать попеременно светодиодами D2, D6 и D8. Время горения светодиода в три раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8. Организовать мигание с помощью прерываний от таймера.

10. Мигать попеременно светодиодами D2, D6 и D8. Время горения светодиода в четыре раза больше, чем время простоя. При нажатии кнопки загораются D5 и D8. Организовать мигание с помощью прерываний от таймера.

2.4 Требования к отчету

В отчете необходимо привести полный код программы с комментариями на каждую строку. Сделать выводы.

2.5 Контрольные вопросы

1. Что такое таймер-счетчик?
2. На каком устройстве базируется типовой таймер-счетчик?
3. Поясните работу блок-схемы работы таймера (см. рисунок 2.2).
4. В чем особенность работы таймера?
5. Какое назначение бита TOCS (TMR0 Clock Source Select)?
6. Какое назначение бита T0SE (TMR0 Source Edge Select)?
7. Какое назначение бита PSA (Prescaler Assignment)?
8. Какое назначение бита PS2:PS0 (Prescaler Rate Select)?
9. Какое назначение бита GIE (Global Interrupt Enable)?
10. Какое назначение бита T0IE (TMR0 Overflow Interrupt Enable)?
11. Какое назначение бита T0IF (TMR0 Overflow Interrupt Flag)?

Лабораторная работа №3

СЕМИСЕГМЕНТНЫЙ ИНДИКАТОР

Цель работы: сформировать общее представление о разработке программного обеспечения для микроконтроллеров серии PIC; создать проект и научиться управлять семисегментными индикаторами.

3.1 Теоретические сведения

На рисунке 3.1 представлено назначение ножек контроллера PIC17F876A.

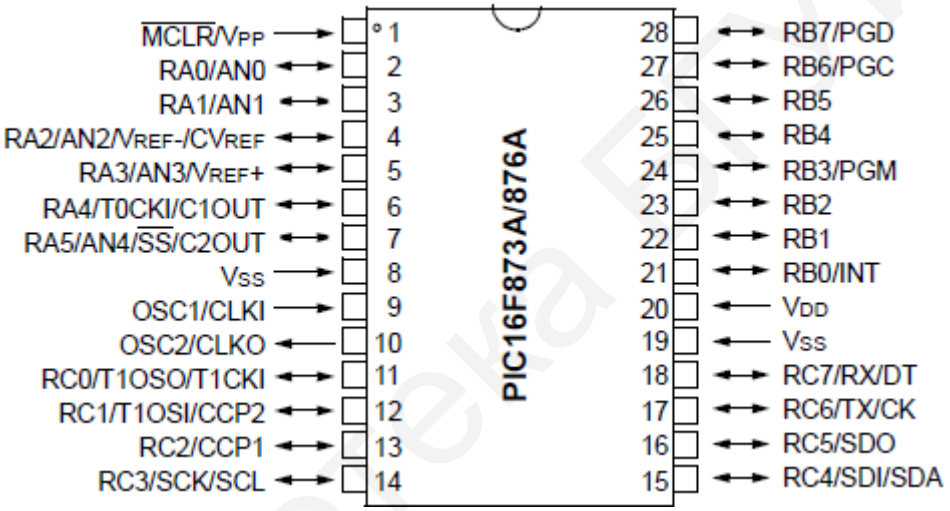


Рисунок 3.1 – Назначение ножек контроллера PIC17F876A

Назначение ножек практически такое же, как и у рассмотренного в лабораторных работах №1–2 контроллера 64A, но только этих ножек уже больше, и есть уже несколько новых обозначений, касающихся периферии, которая отсутствует у предыдущего контроллера. С назначением данных ножек вы будете знакомиться в процессе изучения данных видов периферии. Программирование статической индикации – это не что иное, как управление светодиодами сегментов с помощью ножек портов, а этим вы уже занимались на предыдущих занятиях.

Семисегментный индикатор (рисунок 3.2) представляет собой совокупность планарных светодиодов, определенным образом расположенных в корпусе с целью отображения в основном цифр, но также может отображать и некоторые другие символы. Каждый светодиод является сегментом индикатора, и обозначать его принято определенной буквой.

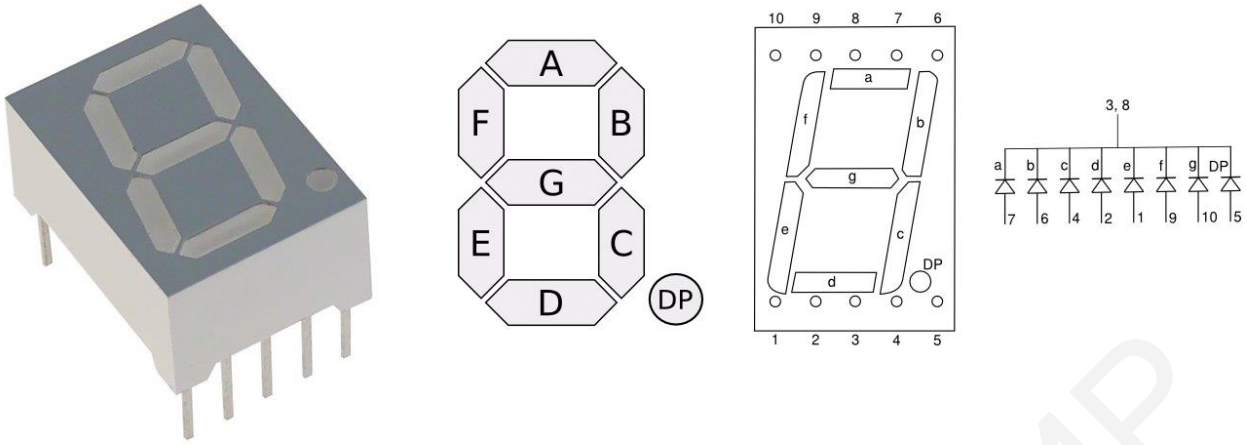


Рисунок 3.2 – Семисегментный индикатор

Один контакт каждого сегмента светодиода выведен на определенную ножку, а другие контакты соединены и выведены на одну общую ножку так же, как и на схеме «бегущих огней». Только существуют две разновидности подключения. Индикатор, у которого объединены аноды, а катоды выведены отдельно, называется светодиодным семисегментным индикатором с общим анодом, а если наоборот, то индикатором с общим катодом.

Простейший вид индикации – статический. При ее использовании каждый сегмент индикатора постоянно находится в одном из двух состояний – включен или выключен. Ее основное преимущество в том, что после записи информации, например, в регистр сдвига, состояние индикатора не изменится, пока не будут изменены данные в регистрах. Поскольку напряжение присутствует постоянно, яркость индикатора будет максимальной.

Динамическая индикация – это метод отображения целостной картины из-за быстрого последовательного отображения отдельных элементов этой картины. Причем «целостность» восприятия создается благодаря инерционности человеческого зрения. Преимущества динамической индикации перед статической заключаются в более простом подключении, меньшем количестве соединений. Недостаток в том, что вывод занимает много времени: частота смены сегментов выбирается обычно не ниже 50 Гц.

С целью уменьшения необходимых для подключения выводов каждый разряд индикатора может иметь общий катод для всех семи сегментов или общий анод. Это позволяет уменьшить количество выводов до восьми на разряд (девяти, если используется точка). Принципиальные схемы для обоих типов на примере серии FYQ-5641 приведены на рисунках 3.3 и 3.4.

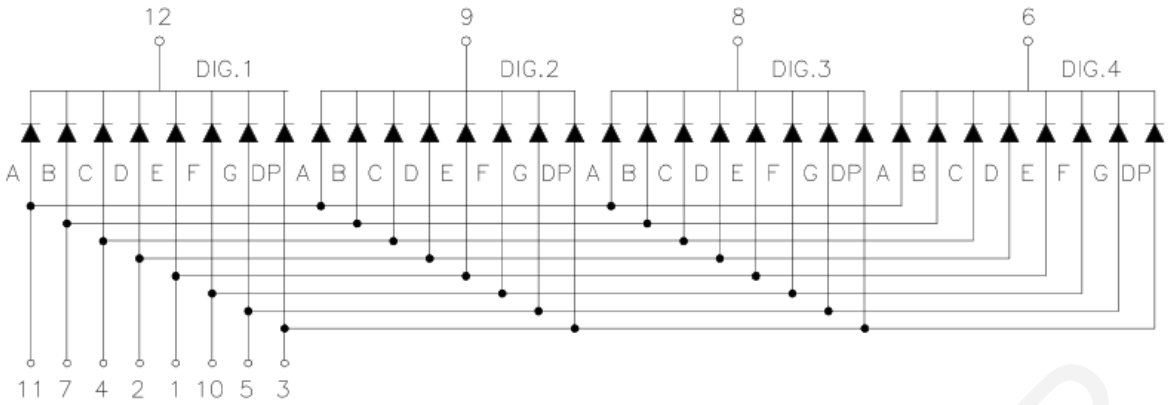


Рисунок 3.3 – Схема с общим катодом серии FYQ-5641

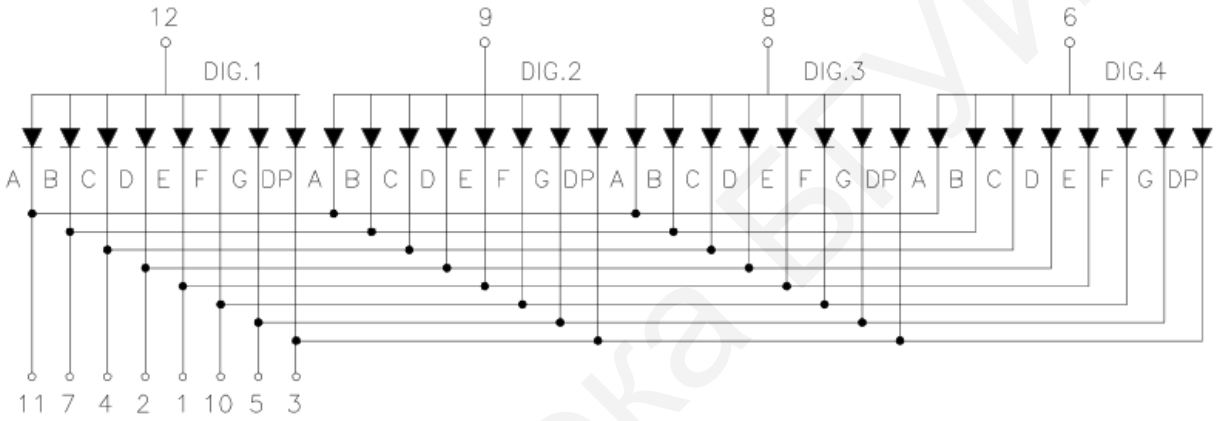


Рисунок 3.4 – Схема с общим анодом серии FYQ-5641

В данном примере будет индикатор с общим анодом, поэтому данный общий анод мы подсоединим к проводу питания, а катодами уже будете управлять с помощью ножек портов, к которым мы их и подсоединим, не забывая о токоограничивающих резисторах. Вы должны знать характеристики конкретного индикатора и подобрать для него резисторы нужного номинала, чтобы ток не превышал максимально допустимой отметки, а также чтобы не превысить максимальный ток, протекающий через ножку порта. В примере установлены такие же резисторы, как и на «бегущие огни» – по 220 Ом (рисунок 3.5).

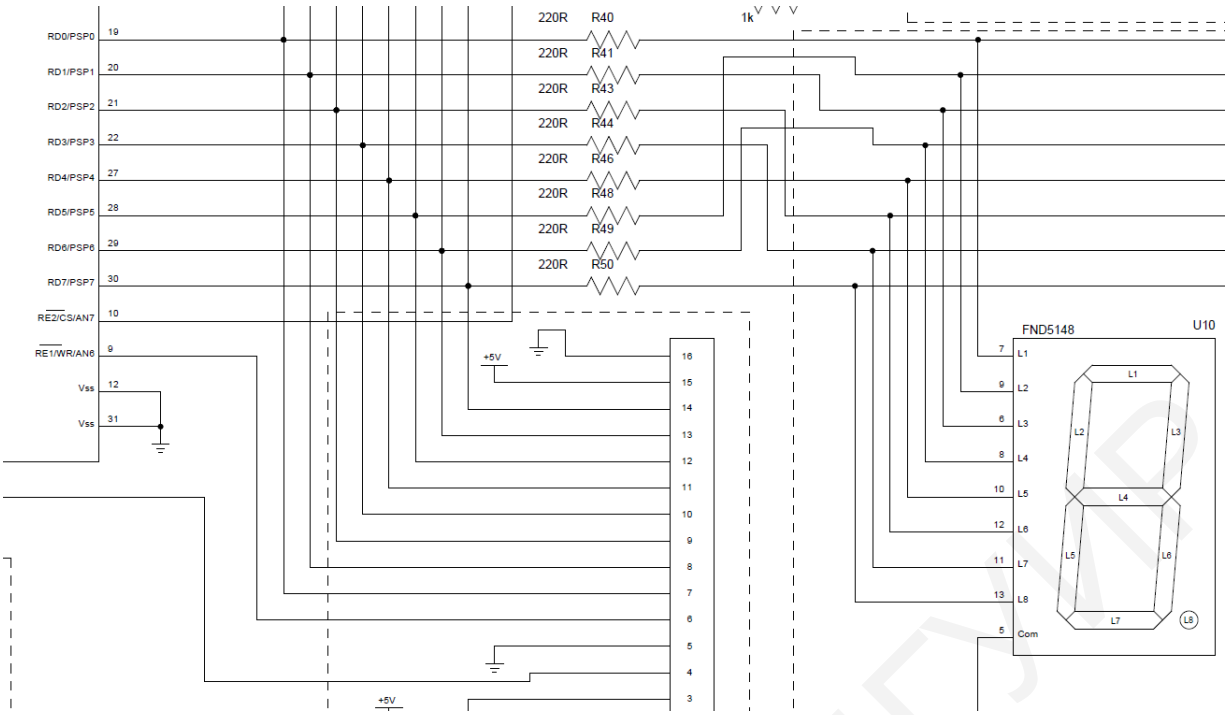


Рисунок 3.5 – Схема включения семисегментного индикатора в Picsimlab board 3

Для того чтобы было легче управлять данными сегментами и выводить с помощью их свечения определенные цифры или символы, соединим их по порядку с ножками одного порта: a – с ножкой RB0, b – с RB1 и так далее по порядку.

3.2 Пример программы счета от нуля до девяти

Необходимо создать новый проект в среде программирования MPLABX с именем LED_STAT и выбрать другой контроллер (рисунок 3.6).

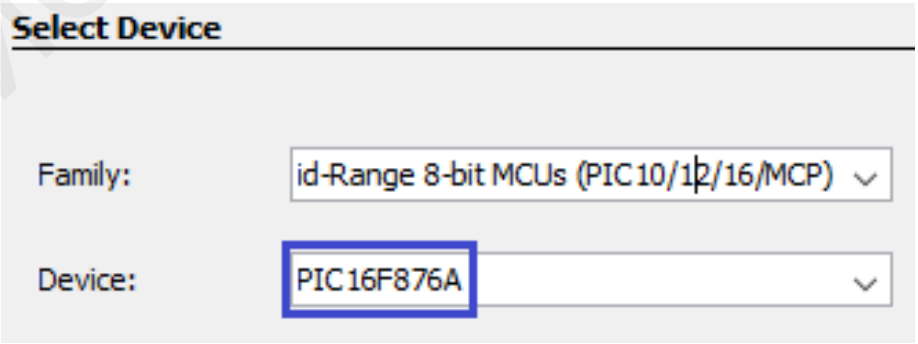


Рисунок 3.6 – Выбор контроллера в MPLAB X

Создадим в проекте файл main.c со следующим стандартным содержимым:

main.c:

```
#include <xc.h>
#define _XTAL_FREQ 4000000

void main(void) {
 while(1)
 {
 }
 return;
}
```

Конфигурационные биты необходимо сконфигурировать следующим образом (назначение каждого бита разъяснено в комментариях).

```
#define _XTAL_FREQ 4000000
// CONFIG
#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)
#pragma config PWRTE = ON // Power-up Timer Enable bit (PWRT enabled)
#pragma config BOREN = OFF // Brown-out Reset Enable bit (BOR disabled)
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)
#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)
//------------------------------------------------------------------
```

Необходимо добавить переменную для счетчика, а также инициализируйте ножки порта:

```
void main(void) {
 unsigned char i;
 TRISB = 0x00;
 PORTB = 0xFF;
```

Настроив ножки на выход и установив на всех ножках порта высокий уровень, так как индикатор у нас с общим анодом, и чтобы сегмент не светился, необходимо подавать на него высокий уровень, тогда разность потенциалов между анодом и катодом будет равна нулю, ведь анод также подключен к высокому уровню (шине питания), и сегмент светиться не будет.

Выше функции main() необходимо добавить функцию отображения определенной цифры на индикаторе:

```
//-------------------------------------------------------------
void segchar (unsigned int seg)
{
 switch (seg)
 {
  case 1:
   PORTB = 0b11111001;
   break;
  case 2:
   PORTB = 0b10100100;
   break;
  case 3:
   PORTB = 0b10110000;
   break;
  case 4:
   PORTB = 0b10011001;
   break;
  case 5:
   PORTB = 0b10010010;
   break;
  case 6:
   PORTB = 0b10000010;
   break;
  case 7:
   PORTB = 0b11111000;
```

```
  break;
 case 8:
  PORTB = 0b10000000;
  break;
 case 9:
  PORTB = 0b10010000;
  break;
 case 0:
  PORTB = 0b11000000;
  break;
 }
}
//-------------------------------------------------------------
```

В зависимости от пришедшего в функцию целого положительного однозначного числа программа будет попадать в соответствующий ему кейс и включать необходимые уровни на ножке порта B: соответственно единица устанавливается на ножке, управляющей сегментом, который не должен светиться, а нуль – на ножку, управляющую сегментом, который будет светиться.

Нужно написать код для отправки цифр по порядку. Для этого мы воспользуемся циклом for, который поможет считать от 0 до 9:

```
while(1)
{
 for(i = 0; i < 10;i++)
 {
  segchar(i);
  __delay_ms(500);
 }
}
```

Счетчик инкрементируется раз в полсекунды, а как только он дойдет до девяти, то процесс опять повторится с нуля.

3.3 Индивидуальные задания

1. Вывести на индикатор попеременно текущую дату (месяц и число) и время окончания текущего занятия. Частота смены – 1 Гц.

2. Вывести на индикатор попеременно «месяц и число» и «год». Частота смены – 2 Гц.

3. Вывести на индикатор попеременно время начала текущего занятия и время его окончания. Частота смены – 3 Гц.

4. Выполнить отсчет времени от 22 до 5 (с частотой 1 Гц), по окончании вывести слово End.

5. Выполнить отсчет времени от 2500 до 1500 с шагом 50. Частота смены – 1 Гц.

6. Вывести на индикатор попеременно текущую дату (месяц и число) и время окончания текущего занятия. Частота смены – 0,3 Гц.

7. Вывести на индикатор попеременно «месяц и число» и «год». Частота смены – 0,2 Гц.

8. Вывести на индикатор попеременно время начала текущего занятия и время его окончания. Частота смены – 0,3 Гц.

9. Выполнить отсчет времени от 20 до 1 (с частотой 1 Гц), по окончании вывести слово End.

10. Выполнить отсчет времени от 10 до 1 (с частотой 1 Гц), по окончании вывести слово Ett.

3.4 Требования к отчету

В отчете необходимо привести полный код программы с комментариями на каждую строку. Сделать выводы.

3.5 Контрольные вопросы

1. Что такое семисегментный индикатор?
2. Чем отличаются индикаторы с общим анодом и общим катодом?
3. Как уменьшить количество выводов МК для подключения семисегментного индикатора?
4. Каково назначение резисторов, подключаемых параллельно индикатору?

Лабораторная работа №4

СИМВОЛЬНЫЙ ЖИДКОКРИСТАЛЛИЧЕСКИЙ ДИСПЛЕЙ ПОД УПРАВЛЕНИЕМ КОНТРОЛЛЕРА HD44780

Цель работы: сформировать общее представление о разработке программного обеспечения для микроконтроллеров серии PIC; создать проект и научиться управлять жидкокристаллическим дисплеем.

4.1 Теоретические сведения

Жидкокристаллический дисплей (ЖК-дисплей, ЖКД, от англ. Liquid Crystal Display, LCD) – плоский дисплей на основе жидких кристаллов.

Жидкокристаллический монитор предназначен для отображения графической информации с компьютера, телевизора, цифрового фотоаппарата, электронного переводчика, калькулятора и пр.

Изображение формируется с помощью отдельных элементов, как правило, через систему развертки. Простые приборы (электронные часы, телефоны, плееры, термометры и пр.) могут иметь монохромный или 2…5 цветный дисплей. Многоцветное изображение формируется с помощью RGB-триад.

Большой популярностью у специалистов пользуются алфавитно-цифровые ЖКИ-модули на базе контроллера HD44780 фирмы Hitachi (рисунок 4.1) или его аналогов производства других фирм: Epson, Philips, Samsung, Sanyo, Toshiba.

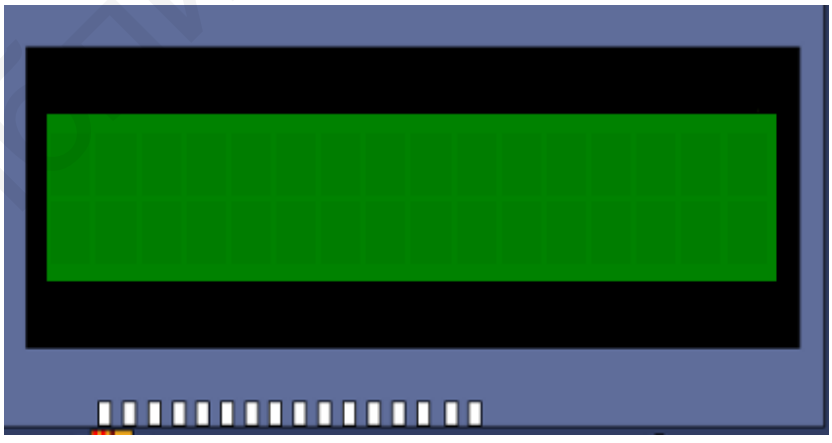


Рисунок 4.1 – ЖКИ-модуль на базе контроллера HD44780 в программе PICSimLab

В таблице 4.1 представлено назначение выводов символьного ЖКИ на базе HD44780.

Таблица 4.1 – Функциональное назначение выводов ЖКИ

Номер вывода	Название выводов	Функциональное назначение
1	GND	Общий вывод
2	VCC	Напряжение питания
3	V0	Напряжение управления контрастностью
4	RS	Выбор записи команды/данные
5	R/W	Выбор направления передачи данных запись/чтение
6	E	Вход тактовых импульсов
7…14	DB7...DB0	Шина данных
15	A	Анод светодиодной подсветки
16	K	Катод светодиодной подсветки

Информацию можно записывать и считывать из индикатора. Как правило, на практике не используется режим чтения, поэтому вывод 5 (R/W) подключают к общей линии, таким образом, индикатор всегда работает в режиме записи. В таблице 4.2. представлены команды записи в индикатор.

Таблица 4.2 – Команды записи в HD44780

N	Состояние линий при R/W = 0									Команды
	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
1	0	0	0	0	0	0	0	0	1	Полная отчистка дисплея и установка курсора в нулевую позицию
2	0	0	0	0	0	0	0	1	–	Установка курсора в нулевую позицию. Установка дисплея в начальное положение
3	0	0	0	0	0	0	1	I/D	S	I/D(Increment/Decrement) – направление сдвига курсора после записи (I/D = 1 – сдвиг вправо, I/D = 0 – сдвиг влево); S(Shift) – разрешение сдвига дисплея вместе с курсором (S = 1 – сдвиг разрешен, S = 0 – сдвиг запрещен)

Продолжение таблицы 4.2

N	Состояние линий при R/W = 0									Команды
	RS	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0	
4	0	0	0	0	0	1	D	C	B	D(Display) – включение дисплея (D = 1 – дисплей включен, D = 0 – дисплей отключен). C(Cursor) – видимость курсора (C = 1 – видимый курсор, C = 0 – погашенный курсор); B(Blink) – мигание курсора (B = 1 – курсор мигает, B = 0 – курсор не мигает)
5	0	0	0	0	1	S/C	R/L	–	–	S/C(Screen/Cursor) – перемещение дисплея/курсора (S/C=1 – перемещается дисплей, S/C = 0 – перемещается курсор); R/L(Right/Left) – направление перемещения дисплея/курсора (R/L = 1 – перемещение вправо, R/L = 0 – перемещение влево)
6	0	0	0	1	DL	N	F	–	–	DL(Data Length) – разрядность шины данных (DL = 1 – 8 бит, DL = 0 – 4 бита). N(Number) – число строк дисплея (N = 1 – две строки, N = 0 – одна строка); F(Font) – размер шрифта (F = 1 – шрифт 5×10 точек, F = 0 – шрифт 5×7 точек)
7	0	0	1	ADRESS						Установка адреса CGRAM (Character Generator RAM). После команды должны следовать данные для записи/чтения в/из CGRAM
8	0	1	ADRESS							Установка адреса DDRAM (Display Data RAM). После команды должны следовать данные для записи/чтения

Контроллер HD44780 допускает подключение к микропроцессорам по 4-битному или 8-битному интерфейсу. Выбор интерфейса производится пользователем исходя из ограничений времени, затрачиваемого на обмен данными с котроллером, или количества используемых линий подключения к микро-ЭВМ. В случае 4-битного подключения шину команд/данных формируют линии DB7…DB4 (линии DB3…DB0 остаются незадействованными). Скорость записи снижается в два раза, но это обычно не вызывает никаких проблем во время работы.

В PICSimLab подключение выполнено по 8-битному интерфейсу (рисунок 4.2).

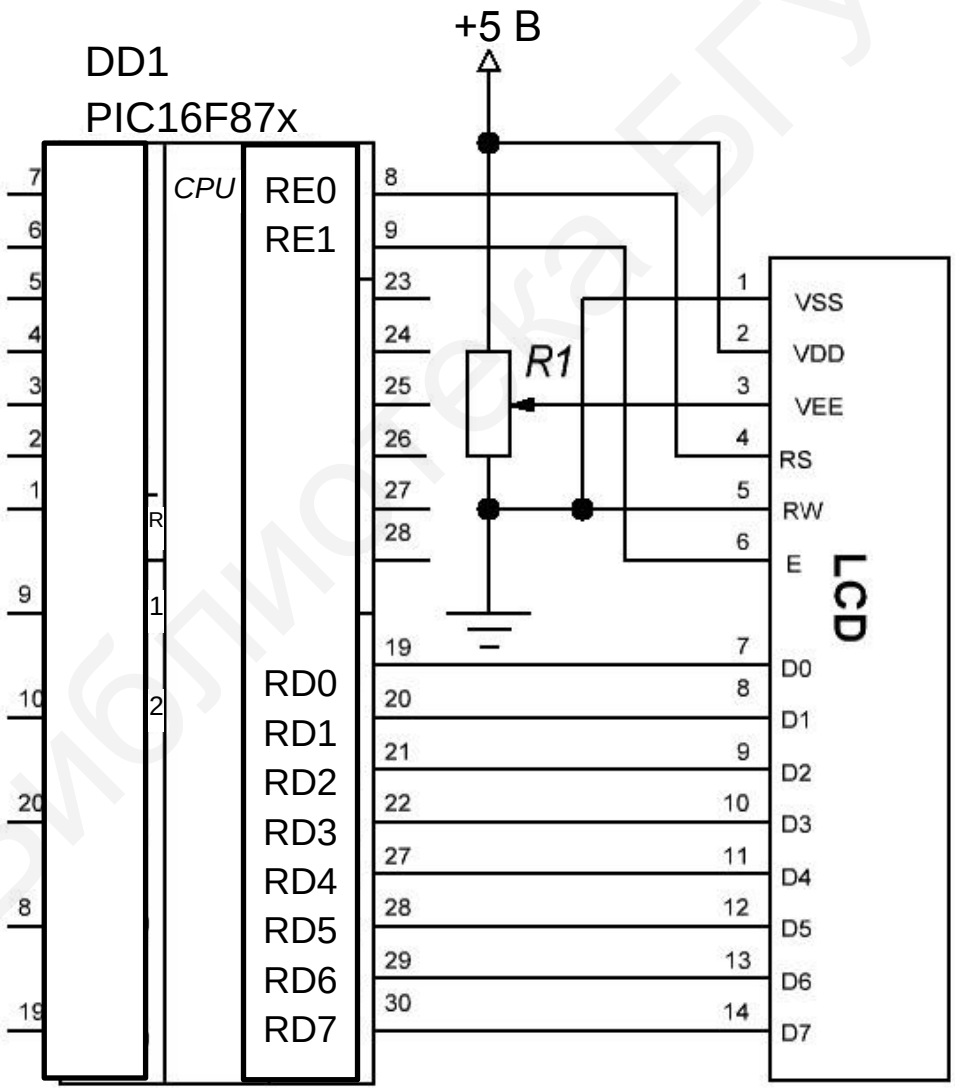


Рисунок 4.2 – Схема подключения дисплея к контроллеру в PICSimLab

У жидкокристаллических индикаторов на базе контроллера HD44780 есть существенный недостаток. Он заключается в том, что вышеуказанные индикаторы являются знакосинтезирующими, размерами 12×2, 16×2, 20×2 (наиболее часто встречающиеся). Поэтому очевидно, что на данные экраны можно выводить только текстовую информацию, причем в очень ограниченом количестве.

К достоинствам можно отнести то, что при условии вывода небольших объемов текстовой информации не нужно хранить в памяти управляющего устройства, которая, как правило, ограничена, битовую матрицу всех необходимых для отображения символов. Также есть возможность задавать свои собственные символы путем занесения битовой матрицы в память управляющего контроллера, что значительно расширяет область применения данных экранов. И еще несколько актуальных достоинств: легкодоступность на территории Беларуси, их низкая стоимость.

4.2 Пример программы вывода информации на дисплей

Создадим также файл главного модуля main.c со стандартным содержимым:

```
void main()
{
 while(1)
 {
 }
}
```

Создадим также файл main.h тоже со стандартным содержимым:

```
 #ifndef MAIN_H
#define MAIN_H
//-------------------------------------------------------------
#define _XTAL_FREQ 4000000
#include <xc.h>
//-------------------------------------------------------------
// CONFIG
#pragma config FOSC = HS // Oscillator Selection bits (HS oscillator)
#pragma config WDTE = OFF // Watchdog Timer Enable bit (WDT disabled)
```

```
#pragma config PWRTE = ON // Power-up Timer Enable bit (PWRT enabled)
#pragma config BOREN = OFF // Brown-out Reset Enable bit (BOR disabled)
#pragma config LVP = OFF // Low-Voltage (Single-Supply) In-Circuit Serial Programming Enable bit (RB3 is digital I/O, HV on MCLR must be used for programming)
#pragma config CPD = OFF // Data EEPROM Memory Code Protection bit (Data EEPROM code protection off)
#pragma config WRT = OFF // Flash Program Memory Write Enable bits (Write protection off; all program memory may be written to by EECON control)
#pragma config CP = OFF // Flash Program Memory Code Protection bit (Code protection off)
//------------------------------------------------------------------
//------------------------------------------------------------------
#endif /* MAIN_H */
```

Подключим данный файл к файлу main.c:

```
#include "main.h"
//-------------------------------------------------
void main()
```

Создадим также заголовочный файл lcd.h для библиотеки дисплея.

```
#ifndef LCD_H
#define LCD_H
//------------------------------------------------------------------
#include <xc.h>
//------------------------------------------------------------------
//------------------------------------------------------------------
#endif /* LCD_H */
```

И создадим файл lcd.c пока почти пустой:

```
#include "lcd.h"
//------------------------------------------------------------------
```

Подключим также нашу библиотеку в файле main.h:

```
#include <xc.h>
#include "lcd.h"
```

Перейдем в файл lcd.c и начнем «писать» библиотеку. Добавим макросы для ножек управления дисплеем (см. рисунок 4.2):

```
#include "lcd.h"
//------------------------------------------------------------
#define rs RE0 //выбираем режим команда/данные
#define e RE1 //выбираем дисплей
//------------------------------------------------------------
```

Вывод RS – это ножка для переключения режима данных (данные/команда). То есть если передаются данные, то на данной ножке мы предварительно устанавливаем высокий уровень, а если команда, то низкий.

Вывод E – это ножка выбора дисплея. Перед передачей или приемом данных мы на нее подаем низкий уровень, а после передачи – высокий.

Время между установкой логической единицы и логического нуля данного вывода регламентировано. Также регламентировано и время минимального нахождения ножки в высоком состоянии перед тем, как записать данные, и оно должно быть не менее 300 нс. Данное время никак не будет программно выдерживаться, так как вывод будет переводиться в другое состояние сразу после отправки данных. Также известно общее минимальное время для отправки данных в контроллер дисплея, т. е. время всего цикла. Оно должно быть не менее 500 нс.

Так как нам потребуется небольшая задержка, то добавим для нее свою функцию. Данная функция есть в примере для платы, которую предоставляет производитель:

```
#define e RC6

//-------------------------------------------------------------
void LCD_delay()
{
 int i;
 for(i=0;i<19;i++);
}
//-------------------------------------------------------------
```

Добавим функцию инициализации выводов портов:

```
//-------------------------------------------------------------
void LCD_PORT_init()
{
 TRISE=0X00;
 TRISD=0X00;
}
//-------------------------------------------------------------
```

Добавим функцию отправки байта в контроллер дисплея:

```
//-------------------------------------------------------------
void sendbyte(unsigned char c, unsigned char mode) // функция передачи байта переменной
{
 PORTD=c;
 if(mode==0) rs=0;
 else rs=1;
 e=0;
 LCD_delay();
 e=1;
}
//-------------------------------------------------------------
```

В данной функции будет два параметра. Первый – непосредственно значение отправляемого байта, а второй – тип отправки (данные или команда).

Сначала отправляем наш байт в регистр порта D, чтобы на его выводах выставились уровни, соответствующие значению нашего байта, так как именно ножки данного порта подключены к ножкам данных дисплея. Затем выставляем уровень на ножке RS в зависимости от того, команду или данные мы будем отправлять в дисплей. Далее мы опускаем ножку E, применяем задержку и затем поднимаем данную ножку. Данные должны будут уйти в дисплей.

Добавим функцию инициализации:

```
//-----------------------------------------------
void LCD_Init()
{
  LCD_delay();
 sendbyte(0X30,0);//on
 LCD_delay();
 sendbyte(0X30,0);//on
 LCD_delay();
 sendbyte(0X30,0);//on
 sendbyte(0X01,0);//Clear Display
 sendbyte(0X38,0);//Function set: 8-bit bus mode,
 sendbyte(0X0c,0);//Display ON, Cursor OFF, blink OFF
 sendbyte(0X06,0);//direction left to right
 sendbyte(0X80,0);//SET POS LINE 0
}
//-----------------------------------------------
```

Память DDRAM контроллера дисплея – это память, которая хранит коды символов, отображаемых в данный момент на дисплее. Вот мы и ставим указатель на нуль адрес данной памяти.

Добавим функцию вывода строки символов на дисплей в текущую позицию:

```
//-----------------------------------------------

void LCD_String(char* st)
{
 unsigned char i=0;
 while(st[i]!=0)
 {
```

```
 sendbyte(st[i],1);
 i++;
 }
}
//-----------------------------------------------ё
```

Байты с кодами символов в контроллер дисплея посылаются до тех пор, пока байт не встретится с нулем. Это и есть окончание строки. Следует обратить внимание, что режим используется уже для передачи данных, а не команд, так как пишем уже напрямую в DDRAM. Поэтому во втором аргументе функции передачи байта у нас единица, а не нуль.

Выше только что написанной нами функции LCD_String добавим функцию установки указателя в требуемую позицию:

```
//-----------------------------------------------
void LCD_SetPos(unsigned char x, unsigned char y)
{
 switch(y)
 {
  case 0:
   sendbyte((unsigned char)(x|0x80),0);
   break;
  case 1:
   sendbyte((unsigned char)((0x40+x)|0x80),0);
   break;
  case 2:
   sendbyte((unsigned char)((0x14+x)|0x80),0);
   break;
  case 3:
   sendbyte((unsigned char)((0x54+x)|0x80),0);
   break;
 }
}

//-----------------------------------------------
```

Данная функция во входных аргументах имеет значения координат по горизонтали и вертикали.

Далее с помощью оператора вариантов отслеживается координата по вертикали и затем посылается команда для установки указателя в определенное место DDRAM контроллера дисплея. В этой команде первый аргумент привязывается к началу строки, используя вышеописанную таблицу адресов DDRAM, прибавляет к началу строки координату по горизонтали и далее устанавливает самый старший бит, который и заставит выполнить именно команду установки указателя.

В заголовочный файл lcd.h необходимо добавить прототипы на функции, которые нам потребуется вызывать из внешних модулей:

```
#include <xc.h>
//-------------------------------------------------------------
void LCD_PORT_init();
void LCD_Init();
void LCD_String(char* st);
void LCD_SetPos(unsigned char x, unsigned char y);
//-------------------------------------------------------------
```

В файл main.c необходимо добавить код в функцию main():

```
void main()
{
LCD_PORT_init();
 LCD_Init();
 LCD_SetPos(0,0);
 LCD_String((char*)"Kaf. ETT");
 LCD_SetPos(0,1);
 LCD_String((char*)"PUMU");
 while(1)
```

Сначала готовим порты, затем инициализируем наш дисплей. Самую первую строку выводим в нулевую позицию первой строки, а далее уже все остальные строки выводим на другие строки дисплея в определенные позиции, используя при этом самостоятельно написанную функцию позиционирования. Также помним, что строки и колонки считаются с нуля, а не с единицы.

В процессе прошивания контроллера могут выдаваться некоторые предупреждения (рисунок 4.3).

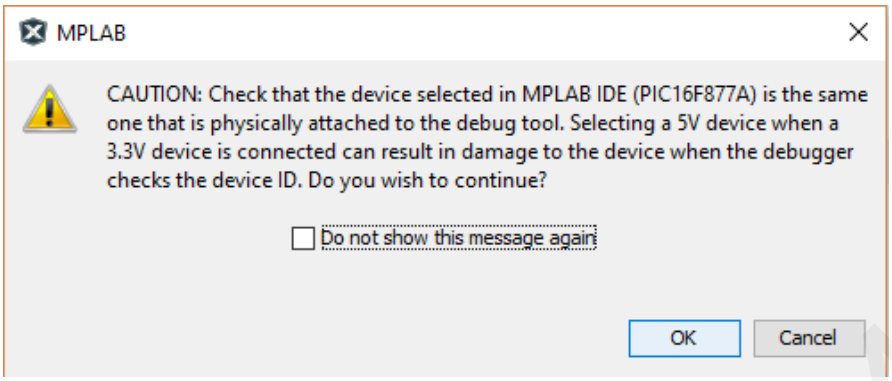


Рисунок 4.3 – Предупреждение при прошивке контроллера

Такое предупреждение можно проигнорировать. Здесь IDE предупреждает о том, что иногда контроллеры питаются напряжением 3,3 В, и в этом случае мы рискуем его вывести из строя.

После прошивки мы должны увидеть картину, представленную на рисунке 4.4.

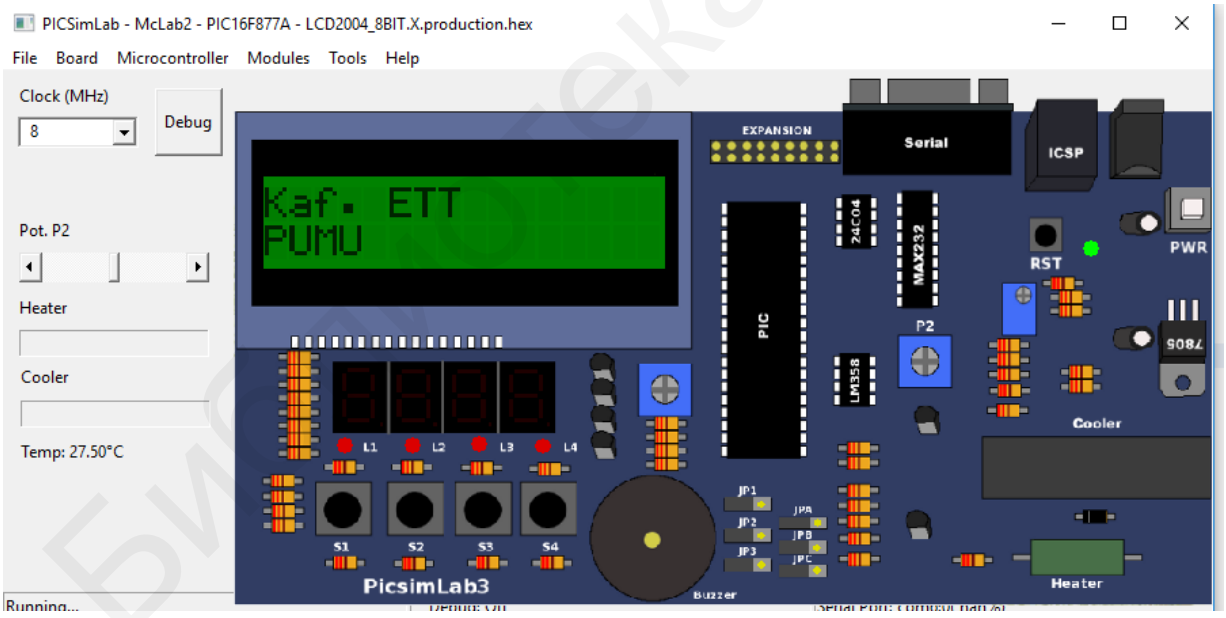


Рисунок 4.4 – Пример работы дисплея

4.3 Индивидуальные задания

1. Вывести на индикатор попеременно текущую дату (месяц и число) и время окончания текущего занятия. Частота смены – 1 Гц.

2. Вывести на индикатор попеременно «месяц и число» и «год». Частота смены – 2 Гц.

3. Вывести на индикатор попеременно время начала текущего занятия и время окончания текущего занятия. Частота смены – 3 Гц.

4. Выполнить отсчет времени от 22 до 5 (с частотой 1 Гц), по окончании вывести слово End.

5. Выполнить отсчет времени от 2500 до 1500 с шагом 50. Частота смены – 1 Гц.

6. Вывести на индикатор попеременно текущую дату (месяц и число) и время окончания текущего занятия. Частота смены – 0,3 Гц.

7. Вывести на индикатор попеременно «месяц и число» и «год». Частота смены – 0,2 Гц.

8. Вывести на индикатор попеременно время начала текущего занятия и время окончания текущего занятия. Частота смены – 0,3 Гц.

9. Выполнить отсчет времени от 20 до 1 (с частотой 1 Гц), по окончании вывести слово End.

10. Выполнить отсчет времени от 10 до 1 (с частотой 1 Гц), по окончании вывести слово Ett.

4.4 Требования к отчету

В отчете необходимо привести полный код программы с комментариями на каждую строку. Сделать выводы.

4.5 Контрольные вопросы

1. Расскажите о назначении ЖК-дисплея.
2. Какие особенности подключения дисплея по 4-битной шине данных?
3. Каковы особенности подключения дисплея по 8-битной шине данных?
4. Какое назначение вывода E ЖКИ-модуля на базе контроллера HD4478?
5. Какое назначение вывода R/W ЖКИ-модуля на базе контроллера HD4478?
6. Какое назначение вывода RS ЖКИ-модуля на базе контроллера HD4478?
7. Какое назначение вывода VEE ЖКИ-модуля на базе контроллера HD4478?

Лабораторная работа №5
МОДУЛЬ ADC (АЦП)

Цель работы: сформировать общее представление о разработке программного обеспечения для микроконтроллеров серии PIC; создать проект и научиться работать с модулем ADC (АЦП).

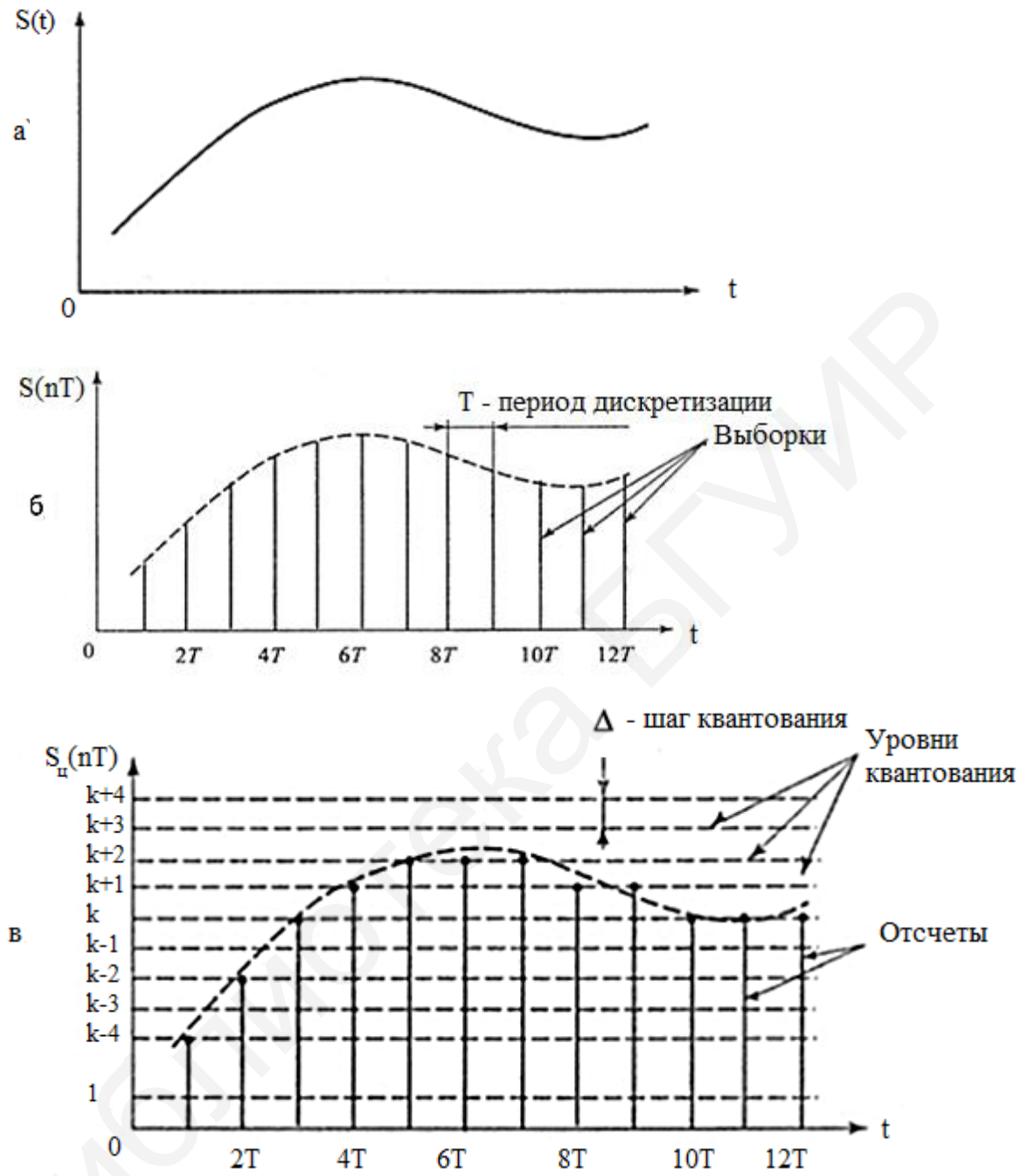
5.1 Теоретические сведения

Аналого-цифровой преобразователь (АЦП, от англ. Analog to digital converter – ADC) – устройство, преобразующее входной аналоговый сигнал в дискретный код (цифровой сигнал). Обратное преобразование осуществляется при помощи ЦАП (цифроаналогового преобразователя, от англ. DAC). Как правило, АЦП – электронное устройство, преобразующее напряжение в двоичный цифровой код. Тем не менее некоторые неэлектронные устройства с цифровым выходом следует также относить к АЦП, например, некоторые типы преобразователей угол – код. Простейшим одноразрядным двоичным АЦП является компаратор.

5.2 Дискретизация, квантование, кодирование

Аналоговый сигнал является непрерывной функцией времени, в АЦП он преобразуется в последовательность цифровых значений. Сам процесс преобразования включает в себя три основные операции: дискретизацию, квантование и кодирование (рисунок 5.1).

Операция дискретизации состоит в том, что по заданному аналоговому сигналу S(t) (рисунок 5.1, *а*) строится дискретный сигнал S(nT), причем S(nT) = S(t). Физически такая операция эквивалентна мгновенной фиксации выборки из непрерывного сигнала S(t) в моменты времени t = nT, после чего образуется последовательность выборочных значений {(S(nT)}. Конечно, такую дискретизацию на практике осуществить невозможно. Реальные устройства, запоминающие значения аналогового сигнала (они называются устройствами выборки и хранения – УВХ), не в состоянии сделать это мгновенно – время подключения их к источнику сигнала всегда учитывается. Кроме того, из-за неидеальности ключей и цепей заряда запоминающей емкости УВХ значение взятой выборки S(nT) в той или иной степени отличается от величины исходного сигнала S(t). Тем не менее в абстрактных рассуждениях равенство S(t) = S(nT) считается справедливым.



а – исходный аналоговый сигнал; *б* – дискретизация; *в* – квантование

Рисунок 5.1 – Аналого-цифровое преобразование

Поскольку дискретный сигнал S(nT) в моменты времени t = nT сохраняет информацию об аналоговом сигнале S(t) и в спектре сигнала S(nT) содержится спектр сигнала S(t) (см. рисунок 5.1), то последний, очевидно, может быть восстановлен. Для этого дискретный сигнал достаточно пропустить через фильтр низких частот, полоса которого соответствует полосе частот исходного сигнала. Условие, при котором восстановление исходного сигнала S(t) по его дискрет-

ным значениям S(nT) будет возможным, сформулировано в известной теореме Котельникова (теореме отсчетов): «Если наивысшая частота в спектре функции S(t) меньше f_{max}, то функция S(t) полностью определяется последовательностью своих значений в моменты, отстающие друг от друга не более чем на $1/f_{max}$ секунд.

Другими словами, чтобы восстановление было точным, частота дискретизации F должна по меньшей мере в два раза превышать максимальную частоту f_{max} в спектре преобразуемого аналогового сигнала S(t). Эта предельно допустимая максимальная частота f_{max} в спектре сигнала называется частотой Найквиста f_H.

Нередко частоту Найквиста f_H путают со скоростью Найквиста F_H, которая характеризует минимально возможную для данной частоты Найквиста скорость дискретизации аналогового сигнала и вдвое выше максимальной частоты в его спектре (частоты Найквиста).

На практике при дискретизации широкополосных сигналов приходится жестко ограничивать их спектры с помощью высокодобротных фильтров низких частот, которые называются антиэлайсинг-фильтрами. Спад характеристики у таких фильтров (как, впрочем, и у любых других фильтров) не бывает строго вертикальным. Поэтому реально частота f_{max} должна быть несколько ниже частоты Найквиста f_H. Тем не менее при анализе теоретических моделей аналого-цифровых преобразователей часто пользуются понятиями частоты и скорости Найквиста, полагая, что скорость Найквиста F_H – это удвоенная частота Найквиста f_H, т. е. $F_H = 2f_H$. В подавляющем большинстве случаев используется равномерная (с постоянным периодом) дискретизация – как по причине того, что к ней легче применить математический аппарат, так и по причине, что устройства для ее осуществления гораздо проще реализовать физически.

После того как сигнал дискретизирован, производятся его квантование и кодирование, что, собственно, и является основной операцией при аналого-цифровом преобразовании. На этом этапе по заданному дискретному сигналу S(nT) строится цифровой кодированный сигнал $S_ц(nT)$. Так же как и дискретный, цифровой сигнал описывается решетчатой функцией, но в данном случае эта решетчатая функция является еще и квантованной, т. е. способной принимать лишь ряд дискретных значений, которые называются уровнями квантования (рисунок 5.1, *в*). Уровни квантования образуются путем разбиения всего диапазона, в котором изменяется аналоговый сигнал, на ряд участков, каждому из которых присваивается определенный номер. Эти номера кодируются заранее выбранным кодом, чаще всего двоичным, а их число N выбирается равным 2^m, где m – разрядность кода.

Если сигнал однополярный, то все 2^m уровней будут выражать положительные значения аналогового сигнала. Для двухполярного сигнала одна половина ($2m/2 = 2^{m-1}$) уровней будет выражать отрицательные значения сигнала, другая (также 2^{m-1}) – положительные.

Квантование может осуществляться двумя способами. При первом способе расстояние между любыми двумя соседними уровнями, которое называется шагом квантования, будет одинаковым (так называемое линейное квантование). Второй способ, когда шаг квантования изменяется, – это нелинейное квантование. В дальнейшем будут рассмотрены линейные АЦП.

Дискретные сигналы, как и аналоговые, образуют линейное пространство относительно операций сложения, вычитания, умножения, если выполняется условие теоремы Котельникова. Цифровые же сигналы, полученные путем квантования, линейного пространства относительно операций сложения и умножения не образуют. Во-первых, процедура квантования почти всегда сопровождается появлением неустранимой погрешности. Во-вторых, линейная комбинация цифровых сигналов, выражаемых m-разрядными кодами, может иметь разрядность, большую чем m (особенно при операциях умножения). Чтобы получить m-разрядный код результата, приходится выполнять операцию округления и усечения. Поэтому устройства цифровой обработки сигналов, реализующие преобразование одной цифровой последовательности $S_{ц1}(nT)$ в другую $S_{ц2}(nT)$ путем выполнения обычных арифметических операций сложения и умножения (в САУ обычно расчет регуляторов), являются, в принципе, нелинейными.

Часто при проектировании систем, включающих в себя устройства аналого-цифрового и цифроаналогового преобразований сигналов, полученных в результате ограничения спектра широкополосных сигналов с помощью фильтров низких частот, разработчики переносят утверждение теоремы Котельникова о возможности точного восстановления исходного аналогового сигнала по отсчетам дискретного на результат аналого-цифрового и цифроаналогового преобразований, что является, в принципе, ошибочным. Поэтому в том виде, в котором теорема Котельникова сформулирована для дискретных сигналов, к системам, включающим в себя АЦ- и ЦА-преобразования, не применима: она может служить только теоретической моделью для очень приблизительных расчетов.

Поскольку реальные АЦП не могут произвести аналого-цифровое преобразование мгновенно, входное аналоговое значение должно удерживаться постоянным, по крайней мере, от начала до конца процесса преобразования (этот интервал времени называют время преобразования).

В настоящее время выпускается большее число интегральных АЦП, которые отличаются конструктивной и функциональной закономерностью, но в основу работы заложены некоторые стандартные, фундаментальные принципы. При этом в структуре некоторых АЦП присутствует устройство УВХ, а в других – отсутствует.

5.3 Разрядность АЦП

Разрядность АЦП характеризует количество дискретных значений, которые преобразователь может выдать на выходе. В двоичных АЦП разрядность измеряется в битах. Разрядностью АЦП определяется и его разрешение – минимальное изменение величины входного аналогового сигнала, которое может быть зафиксировано данным АЦП. АЦП преобразовывает сигнал (напряжение), находящийся в диапазоне измеряемых сигналов. Нижняя и верхняя границы этого диапазона определяются напряжениями, поданными на соответствующие выводы. Для микроконтроллера (МК) со встроенным АЦП нижняя граница – это уровень GND (0 В), а верхняя подается на отдельный вывод (AREF – Analog Reference) или используются внутренние источники опорных напряжений.

При диапазоне входных напряжений от 0 до 5 В и использовании 10-битного АЦП мы имеем разрешение АЦП, представленное на рисунке 5.2. То есть АЦП в состоянии различить сигналы, которые отличаются на 4,9 мВ. При увеличении сигнала на 4,9 мВ результат преобразования увеличится на единицу. Если для такого же диапазона входных сигналов использовать АЦП с большей разрядностью, то можно зафиксировать меньшие значения, т. е. получить более точное значение сигнала (на рисунке 5.3 представлены значения при использовании 24-битного АЦП). При отсутствии различного рода ошибок разрядность АЦП определяет теоретически возможную точность АЦП. На практике разрешение АЦП ограничено отношением сигнал/шум входного сигнала. При большой интенсивности шумов на входе АЦП различение соседних уровней входного сигнала становится невозможным, т. е. ухудшается разрешение. При этом реально достижимое разрешение описывается эффективной разрядностью (Effective Number Of Bits – ENOB), которая меньше, чем реальная разрядность АЦП. При преобразовании сильно зашумленного сигнала младшие разряды выходного кода практически бесполезны, так как содержат шум.

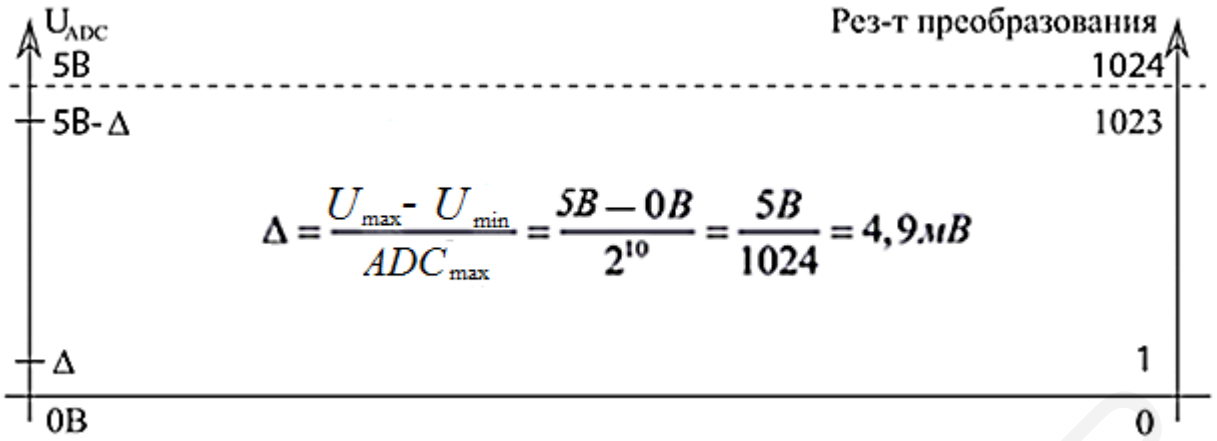


Рисунок 5.2 – Разрешение 10-битного АЦП

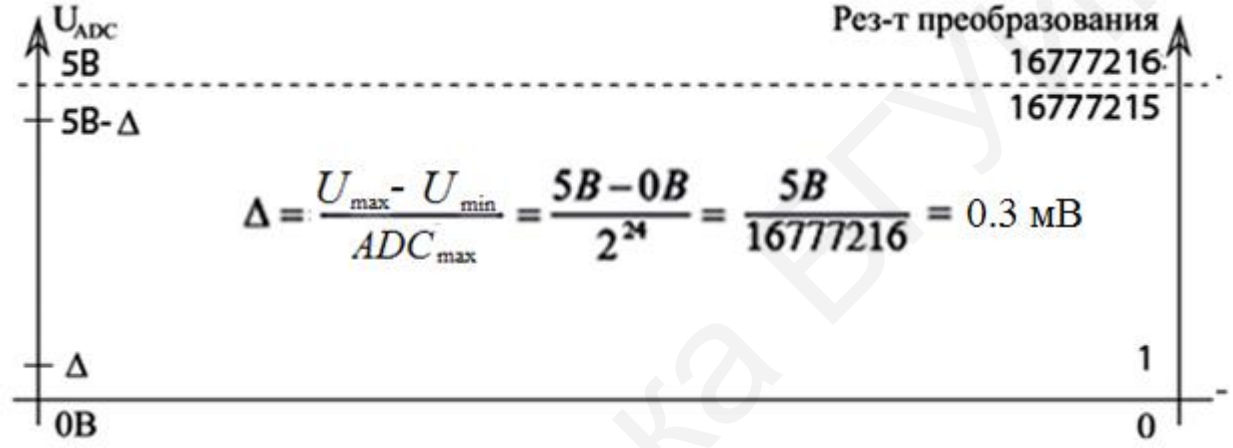


Рисунок 5.3 – Разрешение 24-битного АЦП

5.4 АЦП последовательного приближения

Преобразователь этого типа, называемый в литературе также АЦП с поразрядным уравновешиванием, является наиболее распространенным вариантом последовательных АЦП.

В основе работы этого класса преобразователей лежит принцип дихотомии, т. е. последовательного сравнения измеряемой величины с 1/2, 1/4, 1/8 и т. д. от ее возможного максимального значения. Это позволяет для N-разрядного АЦП последовательного приближения выполнить весь процесс преобразования за N последовательных шагов (итераций) вместо $2^N - 1$ при использовании последовательного счета и получить существенный выигрыш в быстродействии. Так, уже при N = 10 этот выигрыш достигает выполнения 100 раз и позволяет получить с помощью таких АЦП до $10^5...10^6$ преобразований в секунду. В то же время статическая погрешность этого типа преобразователей, определяемая в основном используемым в нем ЦАП, может быть очень малой, что позволяет

реализовать разрешающую способность до 18 двоичных разрядов при частоте выборок до 200 кГц (например, DSP101 фирмы Burr-Brown).

Рассмотрим принципы построения и работы АЦП последовательного приближения на примере классической структуры 4-разрядного преобразователя, состоящего из трех основных узлов: компаратора, регистра последовательного приближения (РПП) и ЦАП (рисунок 8.4).

После подачи команды «Пуск» с приходом первого тактового импульса РПП принудительно задает на вход ЦАП код, равный половине его шкалы (для 4-разрядного ЦАП это $1000_2 = 8_{10}$). Благодаря этому напряжение $U_{ос}$ на выходе ЦАП будет равно

$$U_{ос} = 2^3 h,$$

где h – квант выходного напряжения ЦАП, соответствующий единице младшего разряда (ЕМР).

Эта величина составляет половину возможного диапазона преобразуемых сигналов. Если входное напряжение больше, чем эта величина, то на выходе компаратора устанавливается единица, если меньше, то нуль. В этом последнем случае схема управления должна переключить старший разряд d_3 обратно в состояние нуля. Непосредственно вслед за этим остаток будет равен

$$U_{вх} – d_3 2^3 h.$$

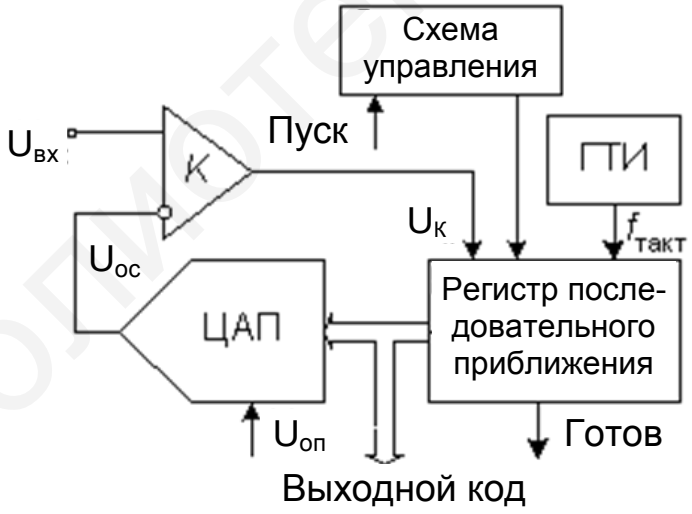


Рисунок 5.4 – Структурная схема и временные диаграммы АЦП

После четырех подобных выравнивающих шагов в регистре последовательного приближения оказывается двоичное число, из которого после цифро-аналогового преобразования получается напряжение, соответствующее $U_{вх}$ с точностью до 1 ЕМР. Выходное число может быть считано с РПП в виде параллельного двоичного кода по N-линиям. Кроме того, в процессе преобразования

на выходе компаратора формируется выходное число в виде последовательного кода старшими разрядами вперед.

Быстродействие АЦП данного типа определяется суммой времени установления $t_{уст}$ ЦАП до установившегося значения с погрешностью, не превышающей 0,5 ЕМР времени переключения компаратора $t_к$ и задержки распространения сигнала в регистре последовательного приближения $t_з$. Сумма $t_к + t_з$ является величиной постоянной, а $t_{уст}$ уменьшается с уменьшением веса разряда. Следовательно, для определения младших разрядов может быть использована более высокая тактовая частота. При поразрядной вариации $f_{такт}$ возможно уменьшение времени преобразования $t_{пр}$ на 40 %. Для этого в состав АЦП может быть включен контроллер.

При работе без устройства выборки/хранения апертурное время равно времени между началом и фактическим окончанием преобразования, которое так же, как и у АЦП последовательного счета, по сути зависит от входного сигнала, т. е. является переменным. Возникающие при этом апертурные погрешности носят также нелинейный характер. Поэтому для эффективного использования АЦП последовательного приближения между его входом и источником преобразуемого сигнала следует включать УВХ. Большинство выпускаемых в настоящее время ИМС АЦП последовательного приближения (например, 12-разрядный MAX191, 16-разрядный AD7882 и др.), имеют встроенные устройства выборки/хранения или, чаще, устройства слежения/хранения (track/hold), управляемые сигналом запуска АЦП. Устройство слежения/хранения отличается тем, что постоянно находится в режиме выборки, переходя в режим хранения только на время преобразования сигнала.

Данный класс АЦП занимает промежуточное положение по быстродействию, стоимости и разрешающей способности между последовательно-параллельными и интегрирующими АЦП и находит широкое применение в системах управления, контроля и цифровой обработки сигналов.

5.5 Модуль10-разрядного АЦП PIC16F877A

Модуль аналого-цифрового преобразования (АЦП) имеет пять каналов у 28-выводных микросхем и восемь каналов у 40/44-выводных микросхем.

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, RA2 или RA3.

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

Для управления АЦП в микроконтроллере используется четыре регистра:

• регистр результата ADRESH (старший байт);

• регистр результата ADRESL (младший байт);

• регистр управления ADCON0 (рисунок 5.5);

• регистр управления ADCON1 (рисунок 5.6).

РегистрADCON0 используется для настройки работы модуля АЦП, а с помощью регистраADCON1 устанавливается, какие входы микроконтроллера будут использоваться модулем АЦП и в каком режиме (аналоговый вход или цифровой порт ввода/вывода).

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	U-0	R/W-0
ADCS1	**ADCS0**	**CHS2**	**CHS1**	**CHS0**	**GO/-DONE**	**-**	**ADON**
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

биты 7-6: **ADCS1:ADCS0:** Выбор источника тактового сигнала
00 = $F_{OSC}/2$
01 = $F_{OSC}/8$
10 = $F_{OSC}/32$
11 = F_{RC} (внутренний RC генератор модуля АЦП)

биты 5-3: **CHS2:CHS0:** Выбор аналогового канала
000 = канал 0, (RA0/AN0)
001 = канал 1, (RA1/AN1)
010 = канал 2, (RA2/AN2)
011 = канал 3, (RA3/AN3)
100 = канал 4, (RA5/AN4)
101 = канал 5, (RE0/AN5)(1)
110 = канал 6, (RE1/AN6)(1)
111 = канал 7, (RE2/AN7)(1)

бит 2: **GO/-DONE:** Бит статуса модуля АЦП
Если ADON=1
1 = модуль АЦП выполняет преобразование (установка бита вызывает начало преобразования)
0 = состояние ожидания (аппаратно сбрасывается по завершении преобразования)

бит 1: **Не используется:** читается как ‘0’

бит 0: **ADON:** Бит включения модуля АЦП
1 = модуль АЦП включен
0 = модуль АЦП выключен и не потребляет тока

Рисунок 5.5 – Регистр управления ADCON0

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
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

PCGF3: PCGF0	AN7[1] RE2	AN6[1] RE1	AN5[1] RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	V_{REF}+	V_{REF}-	Кан./ V_{REF}[2]
0000	A	A	A	A	A	A	A	A	V_{DD}	V_{SS}	8/0
0001	A	A	A	A	V_{REF}+	A	A	A	RA3	V_{SS}	7/1
0010	D	D	D	A	A	A	A	A	V_{DD}	V_{SS}	5/0
0011	D	D	D	A	V_{REF}+	A	A	A	RA3	V_{SS}	4/1
0100	D	D	D	D	A	D	A	A	V_{DD}	V_{SS}	3/0
0101	D	D	D	D	V_{REF}+	D	A	A	RA3	V_{SS}	2/1
011x	D	D	D	D	D	D	D	D	V_{DD}	V_{SS}	0/0
1000	A	A	A	A	V_{REF}+	V_{REF}-	A	A	RA3	RA2	6/2
1001	D	D	A	A	A	A	A	A	V_{DD}	V_{SS}	6/0
1010	D	D	A	A	V_{REF}+	A	A	A	RA3	V_{SS}	5/1
1011	D	D	A	A	V_{REF}+	V_{REF}-	A	A	RA3	RA2	4/2
1100	D	D	D	A	V_{REF}+	V_{REF}-	A	A	RA3	RA2	3/2
1101	D	D	D	D	V_{REF}+	V_{REF}-	A	A	RA3	RA2	2/2
1110	D	D	D	D	D	D	D	A	V_{DD}	V_{SS}	1/0
1111	D	D	D	D	V_{REF}+	V_{REF}-	D	A	RA3	RA2	1/2

A = аналоговый вход D = цифровой канал ввода/вывода

Рисунок 5.6 – Регистр управления ADCON1

В регистре ADRESH:ADRESL сохраняется 10-разрядный результат аналого-цифрового преобразования. Когда преобразование завершено, результат преобразования записывается в регистр ADRESH:ADRESL, после чего сбрасывается флаг GO/-DONE (ADCON0<2>) и устанавливается флаг прерывания ADIF. Структурная схема модуля АЦП показана на рисунке 5.7.

После включения и конфигурации АЦП выбирается рабочий аналоговый канал. Соответствующие биты TRIS аналоговых каналов должны настраивать порт ввода/вывода на вход. Перед началом преобразования необходимо выдержать временную паузу.

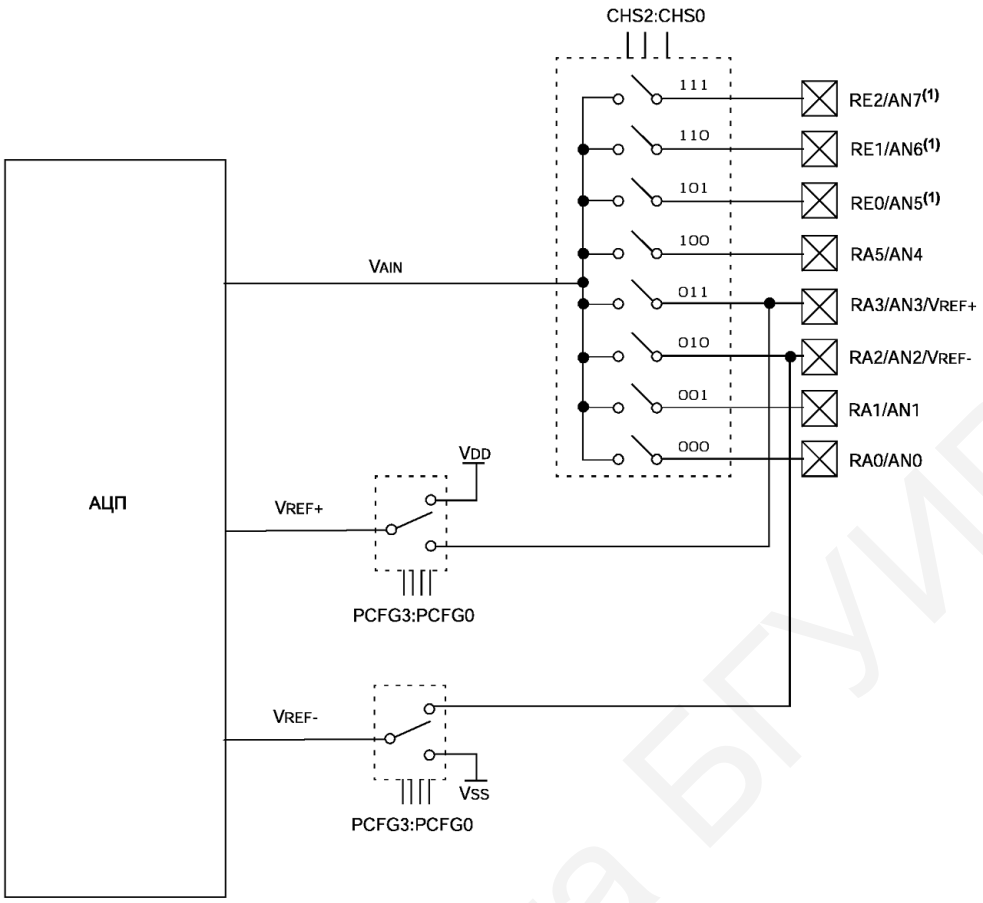


Рисунок 5.7 – Структурная схема модуля АЦП

Рекомендованная последовательность действий для работы с АЦП:

1. Настроить модуль АЦП:

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

б) выбрать входной канал АЦП (ADCON0);

в) выбрать источник тактовых импульсов для АЦП (ADCON0);

г) включить модуль АЦП(ADCON0).

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

а) сбросить бит ADIF в '0';

б) установить бит ADIE в '1';

в) установить бит PEIE в '1';

г) установить бит GIE в '1'.

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

4. Начать аналого-цифровое преобразование: установить бит GO/-DONE в '1' (ADCON0).

5. Ожидать окончания преобразования:

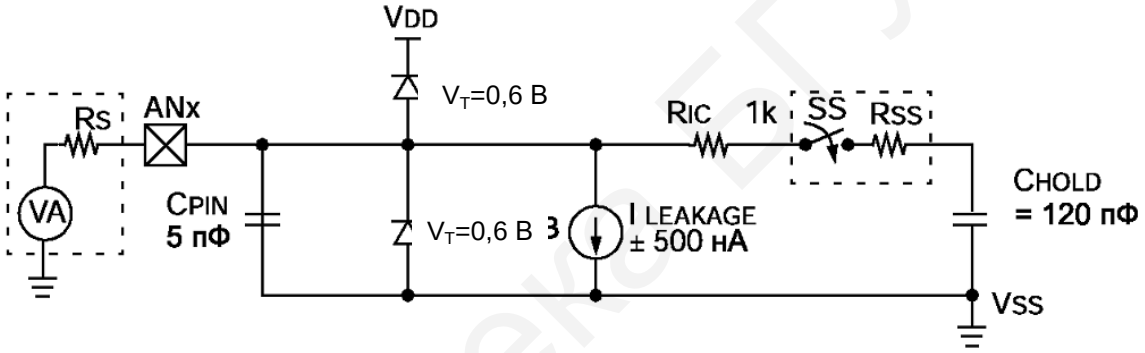
а) ожидать пока бит GO/-DONE не будет сброшен в '0';

б) ожидать прерывание по окончании преобразования.

6. Посчитать 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 не менее $2T_{AD}$.

Для обеспечения необходимой точности преобразования конденсатор C_{HOLD} должен успевать полностью заряжаться до уровня входного напряжения. Схема аналогового входа АЦП показана на рисунке 5.8. Сопротивления R_S и R_{SS} непосредственно влияют на время зарядки конденсатора C_{HOLD}. Величина сопротивления ключа выборки (R_{SS}) зависит от напряжения питания V_{DD} (см. рисунок 5.8).



C_{PIN} – входная емкость; V_T – пороговое напряжение;
$I_{LEAKAGE}$ – ток утечки вывода; R_{IC} – сопротивление соединения;
SS – переключатель защелки; C_{HOLD} – конденсатор защелки

Рисунок 5.8 – Схема аналогового входа АЦП

Максимально рекомендуемое значение внутреннего сопротивления источника аналогового сигнала – 10 кОм. При меньших значениях сопротивления источника сигнала меньше суммарное время 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 (рисунок 5.9). Незадействованные биты регистра ADRESH:ADRESL читаются как нуль. Если модуль АЦП выключен, то 8-разрядные регистры ADRESH и ADREL могут использоваться как регистры общего назначения.

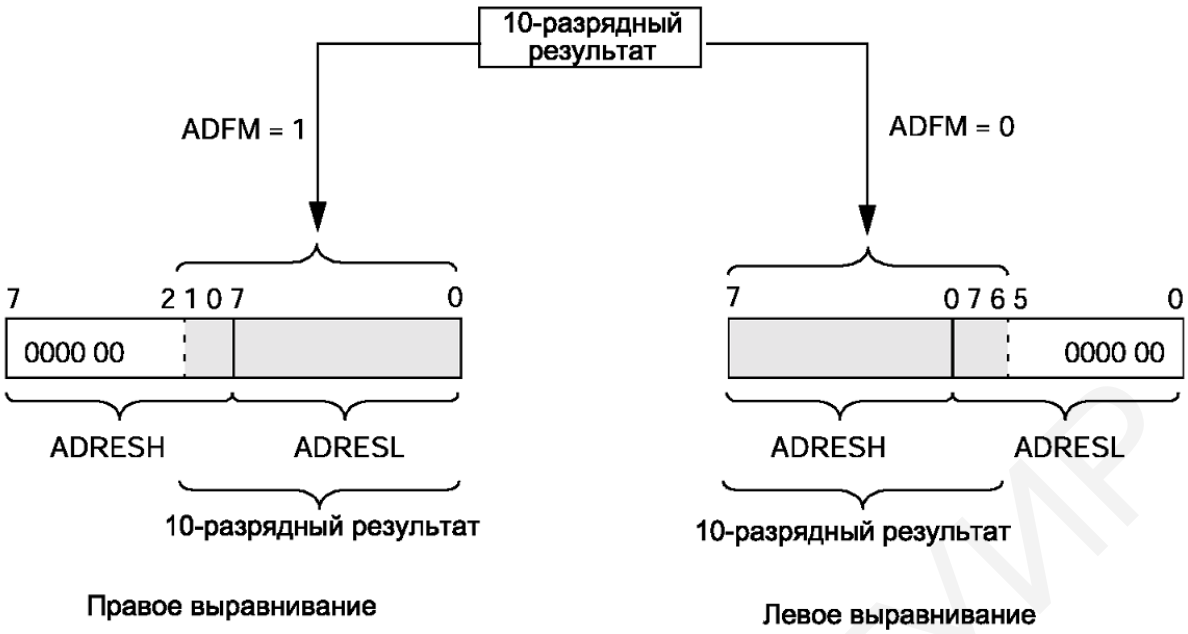


Рисунок 5.9 – Схема выравнивания результата АЦП

Модуль АЦП может работать в режиме микроконтроллера SLEEP при условии, что источником импульсов преобразования АЦП будет внутренний генератор RC (ADCS1:ADCS0 = 11). При выборе генератора импульсов RC 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, результат преобразования сохраняется в регистре 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.

5.6 Пример работы с АЦП

Данная программа отслеживает изменение напряжения на выводе AN1, к которому подключен потенциометр R10, а результат отображается на жидкокристаллическом дисплее.

Сигнал с потенциометра через усилитель U3 поступает на вывод микроконтроллера RA1/ AN1 (рисунок 5.10).

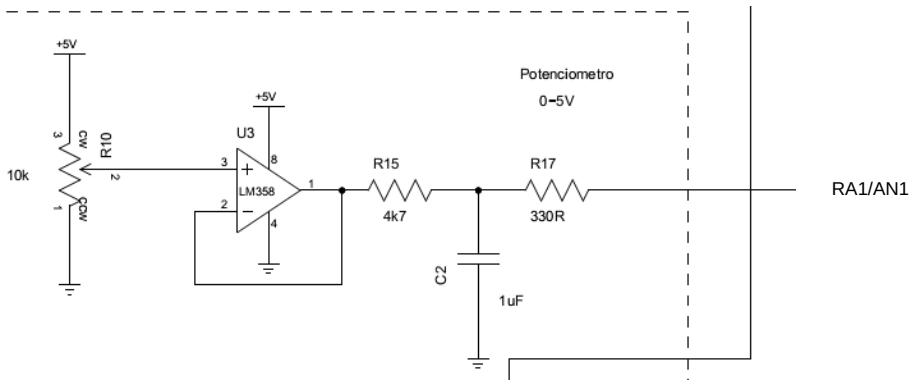


Рисунок 5.10 – Схема подключения потенциометра

Создадим файл lcd.c для работы с дисплеем:

```
#include "lcd.h"
//-------------------------------------------------------------
#define rs RE0 // выбираем режим команда/данные
#define e RE1 // выбираем дисплей
//-------------------------------------------------------------
void LCD_delay() // функция задержки 19 пустых циклов
{
  int i;
  for(i=0;i<19;i++);
}
void sendbyte(unsigned char c, unsigned char mode) // функция передачи байта переменной с присваиванием байтов
{
  PORTD=c; // порту д присваиваем с
  if(mode==0) rs=0;
  else rs=1;
  e=0;
  LCD_delay();
  e=1;
}
//-------------------------------------------------------------
void LCD_Init() // функция инициализации дисплея
{
    LCD_delay();
  sendbyte(0X30,0);// on
  LCD_delay();
```

```
  sendbyte(0X30,0);//on
  LCD_delay();
  sendbyte(0X30,0);//on
  sendbyte(0X01,0);//Clear Display
  sendbyte(0X38,0);//Function set: 8-bit bus mode,
  //2-line display mode is set (2004 – 4-line), 5x8 dots format
  sendbyte(0X0c,0);//Display ON, Cursor OFF, blink OFF
  sendbyte(0X06,0);//direction left to right
  sendbyte(0X80,0);//SET POS LINE 0
 }
 //-------------------------------------------------------------
 void LCD_SetPos(unsigned char x, unsigned char y) // функция выбора строки и позиции
 {
  switch(y) // строка кейс 0 – это 1 строка и 1 – это 2
  {
    case 0:
      sendbyte((unsigned char)(x|0x80),0);// позиция в строке
      break;
    case 1:
      sendbyte((unsigned char)((0x40+x)|0x80),0);
      break;
  }
 }
 //-------------------------------------------------------------
 void LCD_String(char* st)// функция вывода на дисплей
 {
  unsigned char i=0;
  while(st[i]!=0) // почленно из массива строки в дисплей
  {
   sendbyte(st[i],1); // начиная с 0
    i++;
  }
 }
 //-------------------------------------------------------------
```

Файл MAIN.C имеет следующий вид:

```
#include "main.h"
#include "lcd.h"
#include "string.h"
//---------------------------------------------
char str01[30]={'\0'};
//---------------------------------------------
void delay() //функция задержки
 {
  int i;
  for(i=0;i<19;i++);
 }
//---------------------------------------------
void main() // основная функция
{
  int adc_data_raw = 0; //переменная для хранения аналогового сигнала
  float adc_data = .0f; // для хранения цифрового
  unsigned char ch = 0; // переменная для выбора канала, откуда читаем
  TRISA = 0x03; // включаем порты
  PORTA = 0x00;
  TRISE=0X00;
  TRISD=0X00;
  ADFM = 1; //right justified

  PCFG3 = 0; // при такой комбинации порты работают как нам нужно, есть картинка в документации
  PCFG2 = 1;//0100 AN7:AN1 цифра, AN0 аналог, питание = Vdd, земля = Vss
  PCFG1 = 0;
  PCFG0 = 0;
  ADCS0 = 1;
  ADON = 1;//ADC On
  LCD_Init();
  delay();
  while(1)
  {
    GO = 1; // переменная, которая отвечает за АЦП
```

```
        while(GO);
        __delay_us(10);
        if(ch==0) // так как изначально равно 0, то 1 канал читаем
        {
        adc_data_raw = (unsigned int)ADRESH << 8; // сдвигаем биты старшие
        adc_data_raw |= ADRESL; // берем значение из входа и записываем временное
        adc_data = (float)adc_data_raw * 5 / 1024;// переводим сигнал в цифру
           LCD_SetPos(0,0);// первая строка
           CHS0 = 0; // переходим на другой порт
           ch =1; // то же самое
        }
        else
        {
        adc_data_raw = (unsigned int)ADRESH << 8;
        adc_data_raw |= ADRESL;
        adc_data = (float)adc_data_raw * 5 / 1024;
           CHS0 = 1;
           ch =0;
        }

        sprintf(str01,"%.2f V",adc_data); // переводим до двух знаков после точки и v на конце
        LCD_String(str01); // выводим на экран
        __delay_ms(100);
      }
    }
```

При изменении положения движка потенциометра (рисунок 5.11) на дисплее будет отображаться соответствующее напряжение.

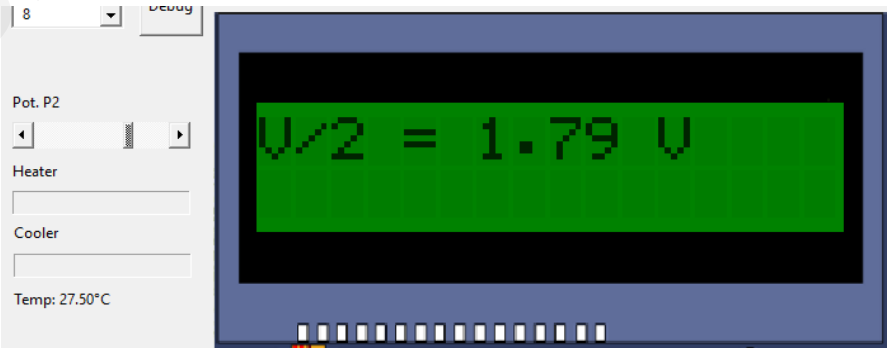


Рисунок 5.11 – Процесс измерения напряжения на потенциометре

5.7 Индивидуальные задания

1. Значение напряжения на потенциометре, деленное на 2, вывести на индикатор.
2. Значение напряжения на потенциометре, умноженное на 2, вывести на индикатор.
3. Значение напряжения на потенциометре – до 2 В в первой строке, более 2 В во второй строке – вывести на индикатор.
4. Значение напряжения на потенциометре – до 1 В в первой строке, более 1 В во второй строке – вывести на индикатор.
5. Значение напряжения на потенциометре – до 3 В в первой строке, более 3 В во второй строке – вывести на индикатор.
6. Значение напряжения на потенциометре – до 4 В в первой строке, более 4 В во второй строке – вывести на индикатор.
7. Значение напряжения на потенциометре, деленное на 10, вывести на индикатор.
8. Значение напряжения на потенциометре, умноженное на 10, вывести на индикатор.
9. Значение напряжения на потенциометре – более 3 В в первой строке, до 3 В во второй строке – вывести на индикатор.
10. Значение напряжения на потенциометре – более 3 В в первой строке, до 3 В во второй строке – вывести на индикатор.

5.8 Требования к отчету

В отчете необходимо привести полный код программы с комментариями на каждую строку. Сделать выводы.

5.9 Контрольные вопросы

1. Что такое АЦП?
2. Каковы параметры аналого-цифрового преобразования?
3. Что такое разрядность АЦП?
4. Принцип работы последовательного АЦП.
5. Каковы особенности АЦП PIC16?
6. Какие регистры управления АЦП PIC16?
7. Какие особенности выравнивания данных в АЦП PIC16?

Список сокращений

АЦ – аналого-цифровой
АЦП – аналого-цифровой преобразователь
ЕМР – единица младшего разряда
ЖК – жидкокристаллический
ЖКИ – жидкокристаллический индикатор
ИМС – интегральная микросхема
МК – микроконтроллер
ПВВ – порты ввода/вывода
ПЗУ – постоянное запоминающее устройство
РПП – регистр последовательного приближения
ЦА – цифроаналоговый
ЦАП – цифроаналоговый преобразователь

Список использованных источников

1. Однокристальные 8-разрядные FLASH CMOS микроконтроллеры компании Microchip Technology Incorporated ООО «Микро-Чип» [Электронный ресурс]. URL : http://www.microchip.ru/files/d-sheets-rus/pic16f87x.pdf.

2. Васильев, А. Е. Микроконтроллеры. Разработка встраиваемых приложений : учеб. пособие / А. Е. Васильев. – СПб. : СПбГПУ, 2003. – 210 с.

3. Рюмик, С. М. 1000 и одна микронтроллерная схема. Вып. 1 / С. М. Рюмик. – М. : Додэка-XXI, 2010. – 356 с.

4. Рюмик, С. М. 1000 и одна микронтроллерная схема. Вып. 2 / С. М. Рюмик. – М. : Додэка-XXI, 2011. – 400 с.

Св. план 2019, поз. 6

Учебное издание

Камлач Павел Викторович
Давыдов Максим Викторович
Ревинская Инна Ивановна
Куничников Дмитрий Петрович

ПРОГРАММИРОВАНИЕ МИКРОКОНТРОЛЛЕРОВ СЕМЕЙСТВА PIC. ЛАБОРАТОРНЫЙ ПРАКТИКУМ

ПОСОБИЕ

Редактор *Е. В. Иванюшина*
Корректор *Е. Н. Батурчик*
Компьютерная правка, оригинал-макет *Е. Г. Бабичева*

Подписано в печать 03.03.2020. Формат 60×84 1/16. Бумага офсетная. Гарнитура «Таймс».
Отпечатано на ризографе. Усл. печ. л. 4,53. Уч.-изд. л. 4,0. Тираж 30 экз. Заказ 339.

Издатель и полиграфическое исполнение: учреждение образования
«Белорусский государственный университет информатики и радиоэлектроники».
Свидетельство о государственной регистрации издателя, изготовителя,
распространителя печатных изданий №1/238 от 24.03.2014,
№2/113 от 07.04.2014, №3/615 от 07.04.2014.
Ул. П. Бровки, 6, 220013, г. Минск