МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ
РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное автономное образовательное
учреждение высшего образования
«СЕВАСТОПОЛЬСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

ИНСТИТУТ РАДИОЭЛЕКТРОНИКИ И ИНФОРМАЦИОННОЙ БЕЗОПАСНОСТИ

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ДИСЦИПЛИНЕ «ПЕРИФЕРИЙНЫЕ ИНТЕГРИРОВАННЫЕ КОНТРОЛЛЕРЫ В РАДИОЭЛЕКТРОННЫХ СРЕДСТВАХ»

Учебно-методическое пособие
для очной и заочной форм обучения
укрупненной группы специальностей
и направлений 11.00.00

Севастополь
СевГУ
2020

УДК 621.396.6
Л12

Р е ц е н з е н т ы:
Д.Г. Мурзин – доцент кафедры «Электронная техника»,
кандидат технических наук
Ю.Н. Тыщук – ст. преподаватель кафедры
«Радиоэлектроника и телекоммуникации»

Ответственный за выпуск:
Ю.П. Михайлюк – канд. техн. наук, доцент,
заведующий кафедрой «Электронная техника»

Составители: И.В. Скорик, И.Б. Широков

Л12 Лабораторный практикум по дисциплине «Периферийные интегрированные контроллеры в радиоэлектронных средствах» : учебно-методическое пособие для очной и заочной форм обучения укрупненной группы специальностей и направлений 11.00.00 / Сост. И.В. Скорик, И.Б. Широков ; Министерство науки и высшего образования РФ, ФГАОУ ВО «Севастопольский государственный университет», Институт радиоэлектроники и информационной безопасности. – Севастополь : СевГУ, 2020. – 95 с. – Текст : электронный.

Содержит теоретическую и практическую информацию необходимую для подготовки к лабораторным занятиям, а также получения дополнительных знаний по изучаемой дисциплине.

УДК 621.396.6

Учебно-методическое пособие рассмотрено и утверждено на заседании кафедры «Электронная техника», протокол № 6 от 30 января 2020 г.

Учебно-методическое пособие рассмотрено и рекомендовано к изданию на заседании ученого совета Института радиоэлектроники и информационной безопасности, протокол № 6 от 31 января 2020 г.

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Цикл лабораторных работ, представленный в настоящих методических указаниях, позволяет изучить основы программирования микроконтроллеров семейства PIC16Fxxx фирмы «Microchip» на языке Ассемблер. Разработанные лабораторные работы предоставляют полноценную возможность для практического освоения принципов функционирования и эксплуатации порядка 90 % имеющейся периферии микроконтроллера PIC16F877. Данный микроконтроллер заложен в основу лабораторного макета «Evolution board PIC16F877 based», который спроектирован и реализован специально для выполнения данного цикла лабораторных работ. Лабораторный макет соединяется с персональным компьютером посредством «USB» программатора «PICkit2», который позволяет программировать, отлаживать и питать микроконтроллер и лабораторный макет.

Выполнение лабораторных работ начинается с обязательного освоения среды интегрированной разработки «MPLAB», предоставленной компанией «Microchip», которая поставляется с компилятором для языка ассемблер и является бесплатным программным продуктом. Суть выполнения каждой лабораторной работы заключается в реализации программного обеспечения «firmware — неизменяемое программное обеспечение, загружаемое в ROM память», которое должно выполнять поставленные задачи в зависимости от выполняемой лабораторной работы. Разработку и симуляцию программного обеспечения необходимо выполнить в предоставленной среде «MPLAB» с последующей отладкой и программированием разработанного программного обеспечения для микроконтроллера PIC16F877 лабораторного макета. Каждая лабораторная работа должна быть выполнена с реальной демонстрацией работоспособности написанной программы на лабораторном макете и представлением отчета, который должен включать фрагмент электрической принципиальной схемы задействованной части макета, алгоритм и код разработанной программы и подробные выводы.

Защита отчета лабораторной работы состоит из двух этапов: в написанный код программы, преподаватель вносит изменения, которые нарушают работу программы, после чего, обучающемуся необходимо доработать программу для устранения дефектов в ее работе. Далее обучающийся отвечает на контрольные вопросы, после чего защита лабораторной работы считается выполненной.

Лабораторный практикум составлен таким образом, что разработка программы для каждой последующей лабораторной работы основывается на наработках, выполненных в предыдущих лабораторных работах. Таким образом, все лабораторные работы взаимосвязаны и поэтому необходимо сохранять написанный код программ на съемных носителях информации во избежание потери кода.

ЛАБОРАТОРНАЯ РАБОТА № 1.
ИНТЕГРИРОВАННАЯ СРЕДА РАЗРАБОТКИ MPLAB

1.1. Цель работы

Изучение электрической принципиальной схемы и внешнего вида лабораторного макета, построенного на базе микроконтроллера PIC16F877 [1]. Знакомство со средой интегрированной разработки «MPLAB». Приобретение практических навыков написания программы на языке Ассемблер с использованием системы команд для «RISC» микроконтроллера.

Изучение с использованием симулятора законов изменения флагов «Z» и «C» регистра «STATUS» при выполнении арифметических и логических операций.

Освоение программирования микроконтроллера и внутрисхемной отладки программы с использованием внешнего программатора.

1.2. Теоретические сведения

1.2.1. Описание устройства лабораторного макета на базе PIC16F877

Лабораторный макет на базе PIC16F877 представляет собой минимальный набор внешних компонентов, позволяющих задействовать до 90 % периферии микроконтроллера.

Электрическая принципиальная схема лабораторного макета, используемого для выполнения цикла лабораторных работ, показана на рис. 1.1. Ключевые компоненты схемы, входящие во внешнюю периферию микроконтроллера PIC16F877, перечислены ниже:

- микроконтроллер PIC16F877 (DD1);
- четырехразрядный семисегментный динамический индикатор (HL1);
- транзисторы для управления разрядами светодиодного индикатора (VT3 — VT6);
- цифровой датчик температуры DS18D20 (DD2);
- часы реального времени DS1307 (DD3);
- кнопочные контакты (SW1—SW5);
- светодиоды (HL2—HL7);
- переменные резисторы для исследования модуля АЦП и модуля сравнения (R4, R1, R27);
- двойной операционный усилитель LM358 (DA1);
- контактная группа для подключения USB программатора PICkit2 (XS1).

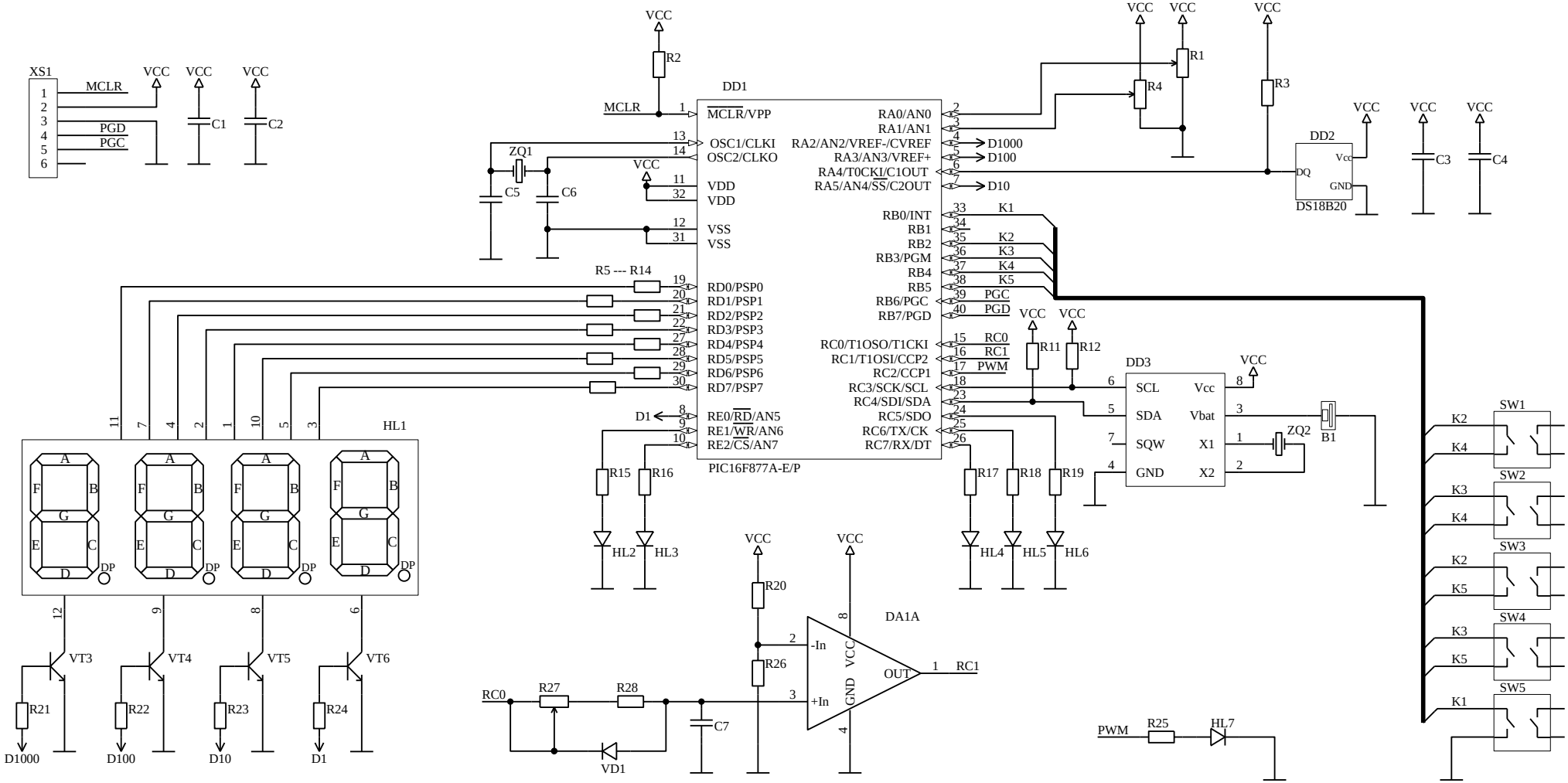

Рис 1.1 — Электрическая принципиальная схема макета на базе микроконтроллера PIC16F877

Внешний вид лабораторного макета показан на рис. 1.2.

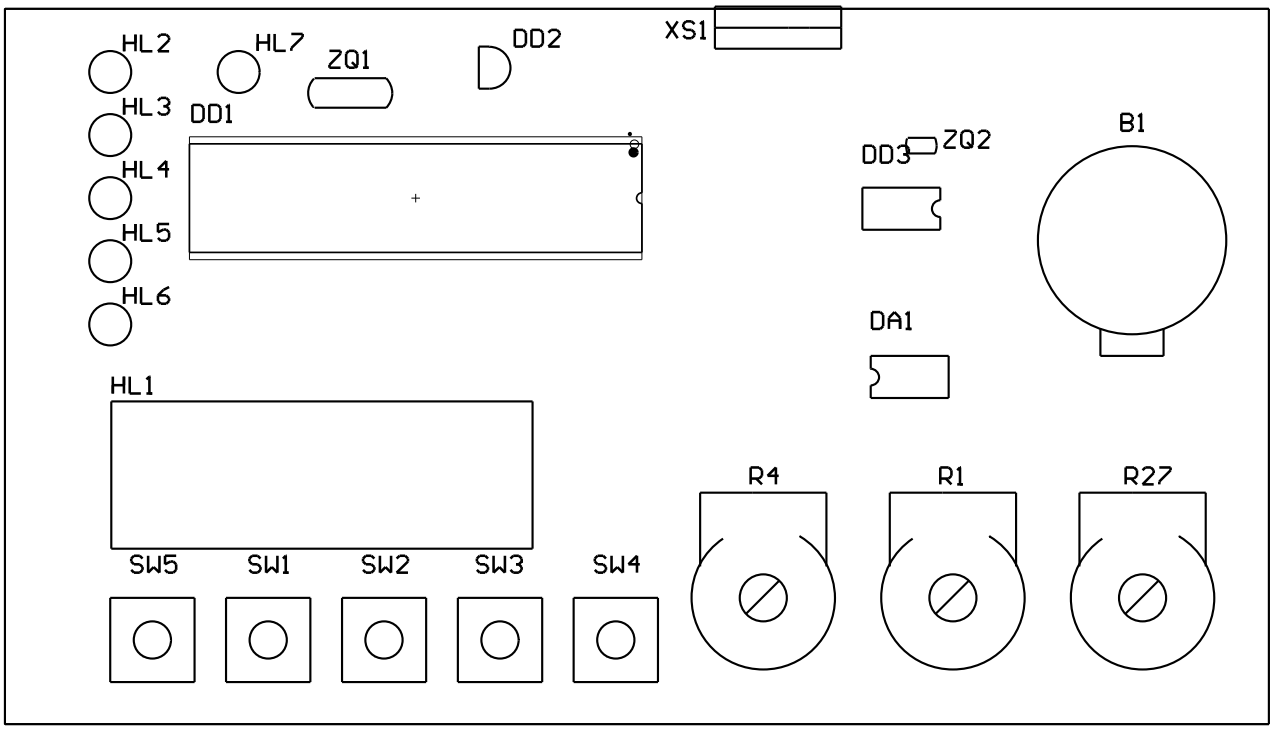

Рис. 1.2 — Внешний вид лабораторного макета

1.2.2. Описание интерфейса среды разработки MPLAB

Среда интегрированной разработки «MPLAB» является полнофункциональной системой разработки программного обеспечения для микроконтроллеров и других цифровых устройств фирмы «Microchip Technology». Программное обеспечения «MPLAB» позволяет осуществить полную разработку программного обеспечения, его отладку, а также программирование выбранного типа микроконтроллера с использованием внешнего программатора. Следует отметить, что абсолютно любая версия «MPLAB» (в дальнейшем «интегрированная среда») является бесплатной и находится в свободном распространении.

Интегрированная среда позволяет использовать различные компиляторы для разработки программного обеспечения, его компиляции, отладки написанной программы и внутрисхемной отладки в режиме реального времени. По умолчанию интегрированная среда снабжена компилятором языка Ассемблера, который позволяет работать с «RISC» микроконтроллерами компании «Microchip». Также возможно использование С-компиляторов, которые дополнительно устанавливаются и встраиваются в среду интегрированной разработки. Для каждого семейства микроконтроллеров «PIC» используется соответствующий компилятор, но бесплатный С-компилятор доступен только с ограничениями по размеру кода.

Чтобы начать разработку программного обеспечения для «PIC» микроконтроллера, требуется создать проект. Наиболее удобным инструментом для создания проекта является «Project Wizard», который настраивает проект в соответствии с параметрами используемого микроконтроллера. Далее написание

кода на языке Ассемблер осуществляется в текстовом редакторе, который распознает мнемоники «MPLAB» Ассемблера.

С помощью модуля «Debugger» осуществляется отладка программы в режиме симуляции встроенными средствами среды либо в режиме реального времени с использованием внешнего программатора.

Работа с памятью программ микроконтроллера осуществляется через модуль «Programmer». Модуль «Configure» позволяет задать функциональную конфигурацию микроконтроллера.

1.2.3. Описание модулей интегрированной среды разработки

Основные возможности модуля «View» (выпадающее меню «View»). Данный модуль позволяет просматривать состояния и содержимое каждого из следующих типов памяти микроконтроллера:

— регистры специального назначения «Special Function Registers»;
— регистры общего назначения «File Registers»;
— аппаратный стек «Hardware Stack»;
— постоянная память данных «EEPROM»;
— код дисассемблера «Disassembly Listing».

Модуль «Memory Usage Gauge» позволяет контролировать использование памяти программ и памяти данных.

Другие возможности модуля «View» не рассмотрены, так как не используются для изучаемого типа микроконтроллера.

Основные возможности модуля «Project» (выпадающее меню «Project»). Модуль «Project» позволяет создавать, управлять и компилировать файлы, включенные в проект. Для разработки программного обеспечения требуется создать новый проект или открыть существующий проект. Для создания нового проекта необходимо воспользоваться помощником создания проекта «Project Wizard» (Project>Project Wizard) или при использовании существующего проекта требуется воспользоваться опцией «Open» (Project> Open). Главные органы управления проектом расположены на панели инструментов (рис. 1.3).

Рис. 1.3 — Главные органы управления проектом

Описание символьных обозначений на панели управления проектом слева на право:

— «New project» — создание нового проекта;
— «Open project» — открыть существующий проект;
— «Save workspace» — сохранить расположение окон текущего проекта;
— «Build options» — настройки компиляции проекта;

— «Package project» — архивация всех файлов проекта (опция удобна для пересылки файлов проекта);
— «Toolsuit info» — инструмент отображения информации о текущем проекте;
— «Make» — компиляция исходного файла;
— «Build all» — компиляция всех фалов, включенных в проект;
— «Export makefile» — экспорт исходного файла.

Основные возможности модуля «Debugger» (выпадающее меню «Debugger»). Данный модуль позволяет производить отладку и тестирование разработанного программного обеспечения для микроконтроллера. Список отладочных инструментов находится во вкладке «Select Tools» (Debugger>Select Tools). Для отладки написанного кода в режиме симуляции, без программирования реального устройства, следует выбирать инструмент «MPLAB SIM». Если требуется производить отладку микроконтроллера в режиме реального времени, тогда необходимо выбрать используемый внешний программатор (при выполнении лабораторной работы может использоваться PICkit2 или PICkit3).

С помощью модуля отладки «Debugger» можно также выборочно или одновременно очистить все типы памяти и конфигурационные биты, используя вкладку «Clear Memory» (Debugger>Clear Memory). Данная вкладка имеет следующие варианты очистки памяти:

— Debugger>Clear Memory>Program memory — очистить память программ;
— Debugger>Clear Memory>GPRs — очистить регистры общего назначения;
— Debugger>Clear Memory>EEPROM — очистить постоянную память данных;
— Debugger>Clear Memory>Configuration bits — сбросить все конфигурационные биты.

При включении режима отладки на панели инструментов высветится дополнительная панель инструментов с органами управления для отладочной процедуры (рис. 1.4).

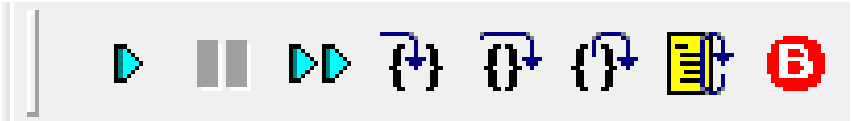

Рис. 1.4 — Органы управления отладочной процедурой

Описание символьных обозначений на панели управления отладочной процедурой слева на право:

— «RUN» — запуск автоматического отладочного режима без анимации (без указателя положения программного счетчика);
— «Halt» — остановка автоматического режима отладки;

— «Animate» — режим отладки с анимацией (отображение положения указателя программного счетчика);
— «Step into» — режим пошаговой отладки с заходом в подпрограммы;
— «Step over» — режим пошаговой отладки с пропуском подпрограмм;
— «Step out» — выход из подпрограммы в основную программу в режиме пошаговой отладки;
— «Reset» — сброс программного счетчика, переход по вектору сброса;
— «Breakpoints» — использование точек останова.

Последняя процедура позволяет приостановить выполнение программы в автоматическом режиме отладки, который далее может быть продолжен в пошаговом или автоматическом режимах. Точки останова не влияет на пошаговую отладку.

Начальная настройка микроконтроллера в модуле «Configure» (выпадающее меню «Configure»). Для задания требуемой конфигурации микроконтроллера требуется определить конфигурационные биты. Определение конфигурационных битов следует задавать перед загрузкой программы в память программ микроконтроллера. Установка битов осуществляется при помощи опции «Configuration bits» (Configure > Configuration bits), которая открывается в отдельном окне (рис. 1.5). Для работы с микроконтроллером данного лабораторного макета следует определить конфигурационные биты, как показано на рис. 1.5, сняв галочку с «Configuration bits set in code» (важно галочку обратно не устанавливать).

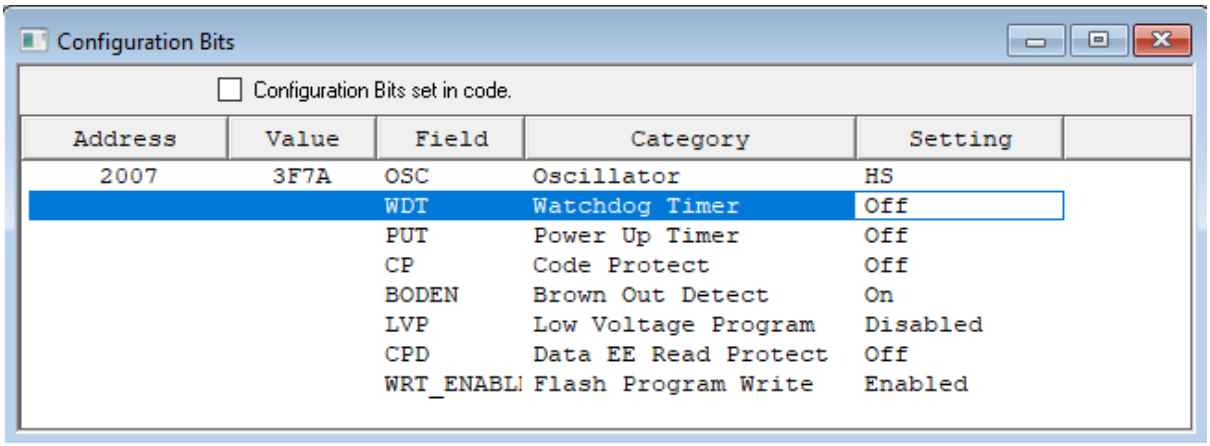

Рис. 1.5 — Окно установки конфигурационных битов

Для корректной работы микроконтроллера лабораторного макета, конфигурационные биты 1 (OSC), 2(WDT) и 6(LVP) в поле «Settings» должны быть идентичны примеру, настройки остальных битов можно оставить неизменными (см. рис. 1.5). После завершения установки битов необходимо закрыть данное окно и выполнить компиляцию проекта, все настройки будут автоматически сохранены в проекте.

Основные возможности модуля «Programmer» (выпадающее меню «Programmer»). Данный модуль позволяет загрузить программу в память про-

грамм микроконтроллера с использованием внешнего программатора. Программирование микроконтроллера возможно только после компиляции программы со всеми файлами, включенными в проект. Когда компиляция завершена, необходимо подключить программатор к персональному компьютеру, затем в меню «Programmer» выбрать подключенный программатор (Programmer > Select tools) или, если программатор был выбран заранее, требуется выполнить соединение с программатором (Programmer > Connect).

После установки соединения с программатором для программирования микроконтроллера следует воспользоваться панелью инструментов, показанной на рис. 1.6.

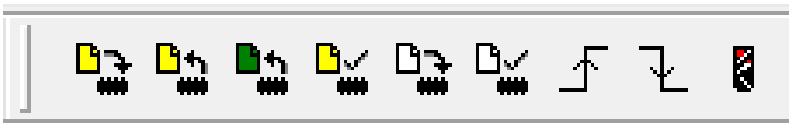

Рис. 1.6 — Органы управления процессом программирования микроконтроллера

Описание символьных обозначений на панели управления процессом программирования микроконтроллера слева на право:

— загрузить последний компилированный код в память микроконтроллера;
— прочесть память программ микроконтроллера;
— прочесть «EEPROM» память;
— проверить записанные данные;
— очистить всю память;
— проверить очистку всей памяти;
— подать высокий уровень на вывод «MCLR» для снятия режима сброса;
— подать низкий уровень на вывод «MCLR» для активации режима сброса;
— переподключить и проверить готовность внешнего программатора.

1.2.4. Работа в интегрированной среде

Создание проекта. Для составления программы в среде «MPLAB» следует создать новый проект, при использовании встроенной утилиты «Project Wizard», далее семь этапов для создания нового проекта.

Этап первый. Запустите интегрированную среду «MPLAB». Запустите «Project Wizard» который располагается по пути Project > Project Wizard.

Этап второй. После запуска «Project Wizard» появится окно приветствия «Welcome», после чего нажмите кнопку «далее».

Этап третий. В этом диалоговом окне необходимо осуществить выбор устройства из выпадающего списка «Device». Выберите PIC16F877 или

PIC16F877A (наличие буквы А необходимо определять по надписи на **корпусе** МК имеющегося на печатной плате макета), и нажмите кнопку «далее».

Этап четвертый. В следующем окне, в строке «Active toolsuite», из выпадающего списка следует выбрать компилятор «Microchip MPASM Toolsuite» (обычно этот компилятор выбран по умолчанию). В остальных строках все остается без изменений, нажмите кнопку «далее».

Этап пятый. В следующем диалоговом окне необходимо выбрать директорию сохранения проекта. Для этого нажмите кнопку «Browse», откроется новое диалоговое окно, в нем выберите папку в которой необходимо разместить файлы проекта, далее введите имя проекта в строку «Имя файла», нажмите кнопку «Сохранить» (после чего дополнительное окно закроется) и нажмите кнопку «далее».

Этап шестой. В следующем диалоговом окне можно добавлять вспомогательные файлы в проект на этапе создания проекта. В данном случае файлы добавлять не следует. Нажмите кнопку «далее».

Этап седьмой. В последнем диалоговом окне приводится резюме проекта, который будет создан, проверьте эти данные, если все верно, тогда нажмите кнопку «Готово», в противном случае вернитесь на требуемый предыдущий этап и внесите соответствующие корректировки.

После этого этапа процедура создания проекта завершена, и оболочка проекта подготовлена для взаимодействия с микроконтроллером.

Включение файлов поддержки в проект. После создания проекта на рабочем столе интегрированной среды будут находиться два окна: «Project» и «Output». Окно «Project» (в случае отсутствия этого окна, его можно открыть View > Project) отображает папки проекта, в которые следует поместить файлы поддержки, необходимые для компиляции проекта. В окне «Output» выводятся сообщения, информирующие разработчика о наличии ошибок или их отсутствии.

Для упрощения процедуры написания программы следует включить заголовочные файлы в папку проекта «Header files», которая находится в окне «Project». Заголовочные файлы обычно имеют следующие имена: «PIC_MAC.inc» и «P16F877.inc».

Файл с адресами регистров специального назначения «P16F877.inc» необходимо найти в директории C:\Program Files(x86)\Microchip\MPASM Suite и скопировать в папку настоящего проекта. Данный файл может быть использован как для МК PIC16F877 с буквой «А» так и без буквы «А».

Файл макросов «PIC_MAC.inc» создается в мануальном режиме. Для этого следует создать текстовый документ в папке настоящего проекта и заменить название и расширение файла «Новый текстовый документ.txt» на «PIC_MAC.inc».

Этот файл необходимо наполнить следующей информацией:

```
b0 macro
bcf    STATUS, RP0
bcf    STATUS, RP1
endm
b1 macro
bsf    STATUS, RP0
bcf    STATUS, RP1
endm
```

В результате выполнения вышеуказанных действий, в папке проекта должны быть размещены два файла «PIC_MAC.inc» и «P16F877.inc». После этого щелчком правой кнопки мыши по папке с названием «Header files» следует выбрать опцию «Add files», и в открывшемся диалоговом окне следует выбрать два вышеупомянутых файла из папки проекта и нажать кнопку «Открыть». Два файла будут отображены в виде подфайлов папки «Header files».

Для написания исходного кода, который и является основной программой, необходимо создать новый файл. Новый файл можно создать, выбрав опцию «File» (File > Add new file to Project). В открывшемся диалоговом окне в строке «Имя файла» необходимо записать следующее «main.asm» (без кавычек) и нажать кнопку «Сохранить». Если все прошло успешно, созданный файл отобразится как подфайл в папке «Source files» с именем «main.asm».

На этом этапе все подготовительные действия выполнены и проект полностью готов к составлению программного кода.

Составление программы на языке ассемблера. Написание программного кода на языке ассемблера должно осуществляется в файле «main.asm». Шаблон начальной программы для микроконтроллера, используемого в лабораторном макете, показан на рис. 1.7.

Программный код начальной программы включает в себя:

— подключение заголовочных файлов с использованием директивы «include»;

— размещение по адресам векторов сброса и прерывания, команд безусловного перехода «goto» на метки «Main» и «IRS» с использованием директивы «org»;

— директиву «CBLOCK», которая обозначает начало блока объявления адресных констант, начиная с адреса 0x20, что соответствует адресу первого регистра общего назначения в нулевом банке памяти данных. В данном блоке пишутся имена, которые будут использоваться для привязки адресов регистров общего назначения. Закрытие блока должно сопровождаться директивой «ENDC»;

— метку «Main», которая обозначает начало основной программы. Весь программный код, который помещен между метками «Main» и «Loop», будет выполняться один раз при каждом включении микро-

контроллера или в случае выполнения условия сброса. Данная часть программы используется для начальной инициализации и конфигурации периферийных модулей микроконтроллера;

— метку «Loop», которая обозначает начало основного или бесконечного цикла программы. Программный код, помещенный между меткой «Loop» и инструкцией «goto Loop», будет выполняться бесконечное количество раз;

— метку «ISR» (Interrupt Service Routine), которая обозначает начало подпрограммы обработки прерываний. Программный код, заключенный между меткой «IRS» и командой «retfie», будет выполняться каждый раз при возникновении какого-либо прерывания;

— код всей программы, который обязательно должен завершаться директивой «end».

```
#include "P16F877.inc"
#include "PIC_MAC.inc"

 org 0x00    ;вектор сброса
     goto MAIN

 org 0x04    ;вектор прерывания
     goto ISR

 CBLOCK
     ;блок объявления констант
     ;здесь пишутся имена регистров
     ;общего назначения
 ENDC
 MAIN
     ;секция предварительной
     ;настройки периферийных модулей
 LOOP
     ;основной цикл программы
     goto LOOP

 ISR
     ;обработчик прерываний
     retfie
 end ;конец программы
```

Рис. 1.7 — Шаблон начальной программы на языке ассемблера

Чтобы убедиться в отсутствии в программном коде синтаксических ошибок, следует выполнить компиляцию написанной программы. **ВАЖНО,** перед компиляцией проекта, установить конфигурационные биты как показано на рисунке 1.5. Компиляция проекта выполняется через панель, показанную на рисунке 1.3.

1.2.5. Симуляция программы

По усмотрению разработчика проверка корректности выполнения программного кода может быть выполнена как при помощи симулятора интегрированной среды, так и без использования симуляции сразу же после компиляции.

В режиме симуляции требуется использовать модуль «Debugger» (см. рис. 1.4), так как в данной интегрированной среде симуляция выполняется при помощи отладочных средств самой среды. В данном случае для симуляции требуется использовать инструмент под названием «MPASM SIM» (Debugger>MPLAB SIM).

Для просмотра состояния и содержания регистров специального и общего назначения следует использовать модуль «View» (описание в пункте 1.2.3) непосредственно в режиме симуляции с пошаговой отладкой. Это позволит контролировать изменение всех битов регистров по мере выполнения инструкций программы.

Фрагмент программы, показанный на рис. 1.8, демонстрирует загрузку десятичного числа в регистр общего назначения и выполнение арифметической операции (сложения) значения аккумулятора и значения регистра общего назначения «V1» [2]. Полный список команд приведен в Приложении А.

```
    CBLOCK 0x20
    V1  ;назанчение РОН
    ENDC

MAIN
    MOVLW   .120    ;загрузка величины 120 в аккумулятор
    MOVWF   V1      ;загрузка значения аккумулятора в регистр V1
LOOP            ;метка начала основной программы
    MOVLW   .10     ;загрузка величины 10 в аккумулятор
    ADDWF   V1,1    ;команда сложения аккумулятора с регистром V1
    goto LOOP
ISR
    retfie
end
```

Рис. 1.8 — Фрагмент программы, выполняющий арифметическую операцию сложения

Приведенный фрагмент программы показывает, что для выполнения операции сложения требуется загрузить первое слагаемое в аккумулятор, затем загрузить его из аккумулятора в регистр. Потом необходимо загрузить второе слагаемое только в аккумулятор, а затем использовать команду сложения.

При симуляции фрагмента программы циклическое сложение значений регистра «V1» и аккумулятора будут изменять арифметические флаги регистра «STATUS». Флаги, которые отображают результат арифметической операции, находятся по адресам 0 и 2. Полное описание битов регистра «STATUS» приведено в приложении Б. Для просмотра состояния битов регистра «STATUS» следует открыть окно просмотра регистров View> File registers. Для контроля

нескольких регистров, одновременно может быть открыто несколько окон просмотра регистров данных (при открытии окна «File registers», в его нижней части следует выбрать вкладку «Symbolic»).

При пошаговом выполнении инструкций все изменения мгновенно будут отображаться в окне «File registers». Обучающемуся следует изучить, как изменяются упомянутые флаги арифметических состояний АЛУ при циклическом выполнении команды сложения.

1.2.6. Загрузка программы в память программ микроконтроллера и использование внутрисхемной отладки

Фрагмент программы, показанный на рис. 1.9, демонстрирует использование выводов порта «E» для зажигания светодиодов, которые зажигаются при формировании высокого логического уровня на выводах «RE1» и «RE2». Для использования фрагмента программы (рис. 1.9) требуется также использовать шаблон программы (рис. 1.8).

В разделе инициализации (рис. 1.9) осуществляется конфигурация выводов порта «E» таким образом, чтобы выводы работали на выход в цифровом режиме, которая осуществляется посредством очистки всех битов регистра «TRISE» и загрузкой конфигурационного числа в регистр «ADCON1». Далее в бесконечном цикле осуществляется установка и сброс высоких и низких логических уровней на выводах «RE1» и «RE2», что позволяет зажечь и погасить светодиоды.

```
MAIN    ;начало основной программы
    b1  ;выбор первого банка памяти данных
    CLRF    TRISE   ;порт E на выход
    MOVLW   b'00000110' ;загрузка константы в аккумулятор
    MOVWF   ADCON1      ; загрузка константы в регистр
    b0 ;выбор нулевого банка памяти данных
    CLRF    PORTE   ;очистка выводов прота
LOOP            ;метка начала бесконечного цикла
    bsf     PORTE,1 ;установка 1 на выводе RE1
    nop     ;пустой цикл
    bsf     PORTE,2 ;установка 1 на выводе  RE2
    nop
    bcf     PORTE,1 ;установка 0 на выводе RE1
    nop     ;пустой цикл
    bcf     PORTE,2 ;установка 0 на выводе  RE2
    goto LOOP
```

Рис. 1.9 — Фрагмент программы, выполняющей зажигание светодиодов на выводах «RE1» и «RE2»

Фрагмент программы (см. рис. 1.9) следует повторить в исходном файле и скомпилирован для дальнейшей загрузки в память программ микроконтроллера. Перед загрузкой необходимо установить конфигурационные биты (см. рис. 1.5). После установки конфигурационных битов требуется скомпилировать программу и затем ее можно загрузить в память микроконтроллера, используя модуль «Programmer» (Programmer -> PicKit2).

После выбора программатора, используйте панель программирования (см. рис. 1.6) для программирования. После программирования микроконтроллера, для того чтобы программа в памяти микроконтроллера начала выполняться, требуется подать высокий логический уровень на вывод «MCLR» микроконтроллера, чтоб вывести микроконтроллер из режима сброса, используйте панель инструментов для программирования (см. рис. 1.6), после чего можно будет увидеть свечение светодиодов. Но визуальное мерцание светодиодов наблюдаться не будет, так как программа выполняется очень быстро, поэтому, для того чтобы убедиться, что программа выполняется корректно, требуется перейти в режим отладки и пошагово выполнить каждую инструкцию программы, наблюдая за тем, как каждый светодиод будет зажигаться или гаснуть в соответствии с выполняемой инструкцией. Отладка программы выполняется аналогично симуляции, приведенной в предыдущем пункте (см. рис. 1.4), только инструментом для отладки следует выбирать внешний программатор (Debugger -> PicKit2), подключенный к персональному компьютеру.

Во время выполнения пошаговой внутрисхемной отладки необходимо использовать модуль «View» (см. пункт 1.2.3) для просмотра состояния битов регистров в памяти данных. Следует обратить **ВНИМАНИЕ**, на то что в окне «File registers» можно принудительно изменять значения регистров в режиме реального времени. В качестве примера можно изменить значения битов с адресами 1 и 2 в регистре «PORTE» и наблюдать процесс включения или выключения соответствующих светодиодов (для удобного просмотра состояния битов регистра удобно использовать формат «bin», данный формат можно выбрать, выполнив щелчок правой кнопки мыши по верхней панели окна «File registers»).

1.3. Порядок выполнения лабораторной работы

1.3.1. Создайте новый проект начиная с пункта 1.2.4 и повторите все остальные пункты в соответствии с порядком изложения в теоретической части (действия, выполненные в данном пункте не должны отображаться в отчете).

1.3.2. Используя примеры программ (см. рис. 1.7 и рис. 1.8), следует разработать новую программу, в которой регистр общего назначения складывается со значением, загруженным в аккумулятор (начальные значения для регистра общего назначения и аккумулятора выбираются по номеру в списке группы). Далее необходимо зажигать или гасить светодиоды в произвольном порядке, в зависимости от состояний флагов «Z» и «C». Для этого необходимо воспользоваться списком команд, приведенных в приложении А.

1.3.3. После написания программы следует выполнить компиляцию и произвести симуляцию. Используя модуль «View», убедиться, что значения битов изменяются в соответствии с заданием. Сделать скриншоты симуляции и привести их в отчете.

1.3.4. После успешной симуляции программы, выполните программирование микроконтроллера лабораторного стенда и произведите внутрисхемную

пошаговую отладку программы, чтобы убедится в корректности выполнения программы.

Все этапы выполнения работы следует описать в отчете с соответствующими рисунками окон интегрированной среды.

1.4. Содержание отчета

Цель работы.

Электрическая принципиальная схема задействованной части макета.

Теоретическая часть.

Привести ассемблерный код программы (по пункту 1.3.2).

Привести подробное описание процесса симуляции и внутрисхемной отладки с сопутствующими снимками окон проекта.

Подробные выводы.

1.5. Контрольные вопросы

1.5.1. Последовательность действий для создания нового проекта.

1.5.2. Последовательность действий для симуляции программы без программирования микроконтроллера.

1.5.3. Последовательность действий для программирования микроконтроллера при помощи внешнего программатора.

1.5.4. Средства интегрированной среды, позволяющие отслеживать состояния битов в регистрах.

1.5.5. Команды проверки состояния битов в регистре. Приведите примеры.

1.5.6. Арифметические флаги «Z» и «C» регистра «STATUS». Пример использования упомянутых арифметических флагов.

ЛАБОРАТОРНАЯ РАБОТА № 2. ДИНАМИЧЕСКАЯ ИНДИКАЦИЯ С ИСПОЛЬЗОВАНИЕМ ПРОГРАММНОЙ И АППАРАТНОЙ ЗАДЕРЖЕК

2.1. Цель работы

Изучение принципа динамической индикации на базе семисегментного светодиодного индикатора с использованием программной и аппаратной задержек;

2.2. Теоретические сведения

В лабораторной работе используется лабораторный макет, принципиальная схема которого приведенный на рис. 1.1. В нем будут задействованы следующие устройства:

— микроконтроллер PIC16F877 (DD1);

— четырехразрядный семисегментный динамический индикатор (HL1).

2.2.1. Устройство четырехразрядного семисегментного индикатора

Семисегментный индикатор состоит из семи элементов индикации (сегментов), включающихся и выключающихся по отдельности. Сегменты обозначаются буквами: A, B, C, D, E, F, G и DP. Обозначения сегментов одного разряда индикатора показаны на рис. 2.1.

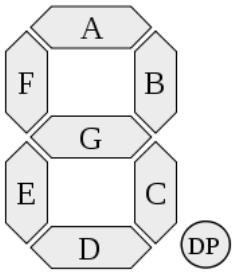

Рис. 2.1 — Обозначения сегментов семисегментного индикатора

Используемый лабораторный макет оснащен четырехразрядным индикатором с общим катодом, электрическая принципиальная схема которого показана на рис. 2.2.

Согласно приведенной электрической принципиальной схемы макета (см. рис. 1.1), восемь выводов порта D подключены к анодным выводам каждого сегмента, а общие выводы катодов каждого разряда подключены через транзисторы управления к выводам: RE0, RA2, RA3 и RA5.

Для включения определенных сегментов одного из четырех разрядов необходимо подать логическую «1» на базу транзистора соответствующего разряда и затем установить необходимые логические уровни на выводах порта D. В таком случае будут включены те сегменты разряда, на которые поступают высокие логические уровни с выводов порта D.

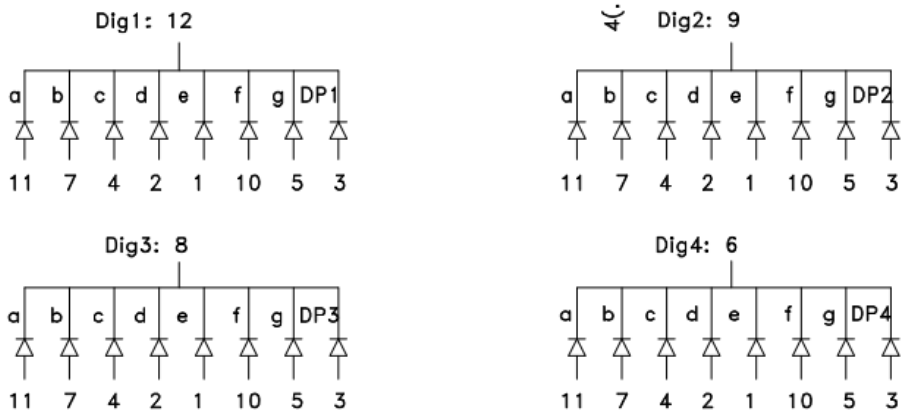

Рис. 2.2 — Схема четырехразрядного семисегментного индикатора

2.2.2. Принцип работы динамической индикации

Динамическая индикация — это тип индикации, при котором вывод информации на разряды индикатора осуществляется поочередно. При этом частота вывода сигналов на индикатор выбирается такой, чтобы переключение разрядов индикатора не было заметным глазу человека.

Для одновременного изображения десятичных цифр на 4 разрядах индикатора, необходимо относительно быстро выводить определенные комбинации логических «0» и «1» на выводы порта D и синхронно переключать управляющие транзисторы таким образом, чтобы единовременно был «открыт» только один транзистор, что обеспечит включение только одного разряда индикатора в определенный момент времени.

Фрагмент алгоритма для реализации динамической индикации и вывода 1, 2, 3 и 4 на разряды индикатора показан на рис.2.3.

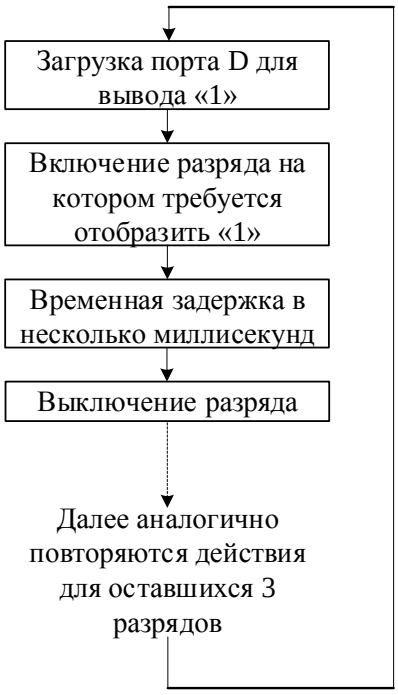

Рис. 2.3 — Фрагмент алгоритма для реализации динамической индикации

2.2.3. Реализация программной задержки

Идея программной задержки заключается в том, чтобы временно приостановить выполнение основной программы путем выполнения арифметической операций, такой как цикличное инкрементирование или декрементирование значение регистра. Время, которое будет затрачено на выполнение арифметической операции, и будет являться временем задержки [3].

Программная задержка может быть реализована в виде подпрограммы «delay». Пример подпрограммы фиксированной задержки показан на рис. 2.4.

```
delay               ;название подпрограммы

    MOVLW   .255    ;число определяющее длительность
    MOVWF   CNT1

    DECFSZ  CNT1,1  ;декремент регистра CNT1, при обнулении
                    ;пропустить след. инструкцию
    goto    $-1     ;уменьшить программный счетчик на 1

    return
```

Рис. 2.4 — Подпрограмма реализации задержки

Для управления длительностью задержки загружаемое в регистр CNT1 значение необходимо помещать в аккумулятор перед каждым вызовом подпрограммы задержки. Чем большее значение загружено в регистр CNT1, тем дольше будет выполняться декрементирование, что также увеличит задержку.

Регистр общего назначения CNT1 необходимо объявить в CBLOCK программы, что позволит назначить слово CNT1 адресу регистра общего назначения в памяти данных.

Для увеличения задержки необходимо использовать дополнительный регистр для последовательного декрементирования, что позволит увеличить задержку.

2.2.4. Реализация аппаратной задержки

Логика прерываний микроконтроллера PIC16F877. Прерывание — это сигнал, сообщающий процессору о наступлении какого-либо события, не предписанного для выполнения текущей инструкции. При этом выполнение текущей последовательности инструкций программы приостанавливается, и управление передаётся обработчику прерываний, который идентифицирует источник прерывания и обслуживает его, после чего возвращает управление в прерванный код.

Прерывания важны для своевременного реагирования процессора на незапланированные события. Такими событиями могут быть изменение уровня сигнала на одном из выводов микроконтроллера, переполнение регистра одного из модулей таймеров, завершение конвертирования модуля АЦП и др.

Изучаемый тип микроконтроллера насчитывает 14 источников прерываний. Фрагмент схемы логики прерываний микроконтроллера PIC16F877 показан на рис. 2.5.

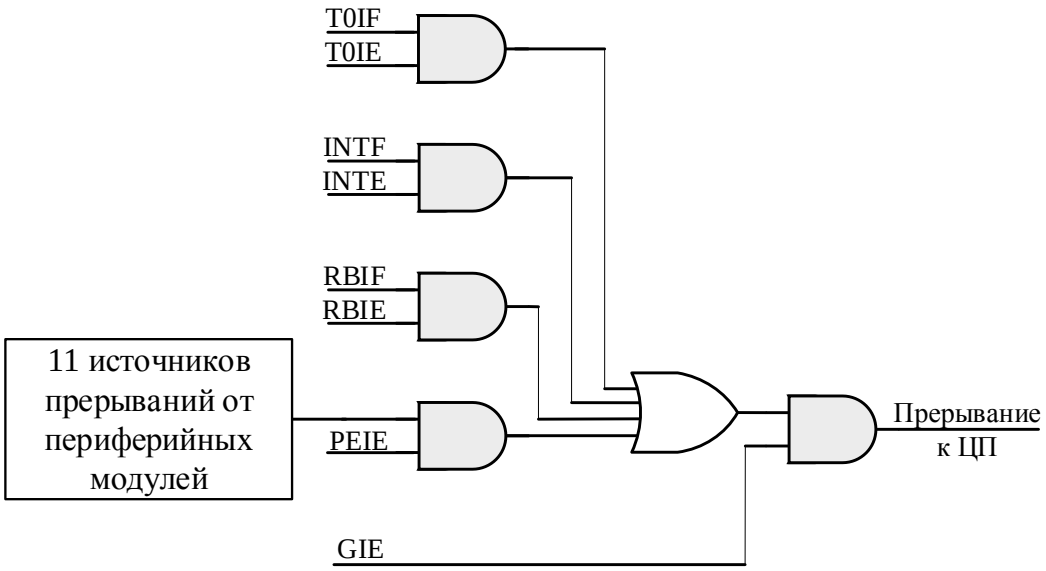

Рис. 2.5 — Фрагмент схемы логики прерываний PIC16F877

Согласно схеме логики прерываний, все устройства, которые могут генерировать прерывания, связаны с определенными флагами. Например, флаг INTF при выполнении условия прерывания примет состояние логической «1». Состояние каждого флага прерывания передается на соответствующие входы логических элементов 2И, а на вторые входы (тех же логических элементов 2И) поступают состояния битов, разрешающих прерывания, например, INTE. В случае, если прерывание от определенного устройства разрешено и соответствующий флаг прерывания примет высокое логическое состояние, то на выходе данного логического элемента 2И сформируется логическая «1». Далее высокий логический уровень проходит через объединяющее ИЛИ и поступает на один из входов конечного логического элемента И. Если бит разрешения всех прерываний GIE установлен в «1», то сигнал прерывания с выхода конечного логического элемента «И» поступает к ЦП.

На рис. 2.5 показаны только три немаскированных источника прерываний, а остальные одиннадцать источников прерывания замаскированы битом разрешения прерываний PEIE от периферийных модулей. В данной лабораторной работе маскированные прерывания рассматриваться не будут.

Адреса векторов прерывания и сброса. При возникновении прерывания выполняется аппаратная функция сохранения адреса текущей инструкции в аппаратном стеке, после чего происходит загрузка адреса вектора прерывания 04h. В связи с этим предполагается, что подпрограмма обработки прерываний должна начинаться именно по адресу вектора прерывания. Однако на практике для удобства написания программы используется запись, показанная на рис. 2.6.

```
org 0x00          ;адрес вектора сброса
    goto MAIN     ;безусловный переход по адресу
                  ;метки Main

org 0x04          ;адрес вектора прерывания
    goto ISR      ;безусловный переход по адресу
                  ;метки ISR
```

Рис. 2.6 — Запись векторов сброса и прерывания для перехода по меткам Main и ISR

Согласно рис. 2.6, команды безусловного перехода по меткам Main и ISR расположены по адресам векторов сброса и прерывания. Такая запись в программе упрощает написание и чтения программного кода.

Конфигурация модуля таймера TMR0 для аппаратной задержки. Отличие аппаратной задержки от программной заключается в том, что аппаратная задержка не использует вычислительных ресурсов процессора. Ниже приводится пример реализации аппаратной задержки модуля таймера TMR0. Структурная схема модуля таймера TMR0 показана на рис. 2.7.

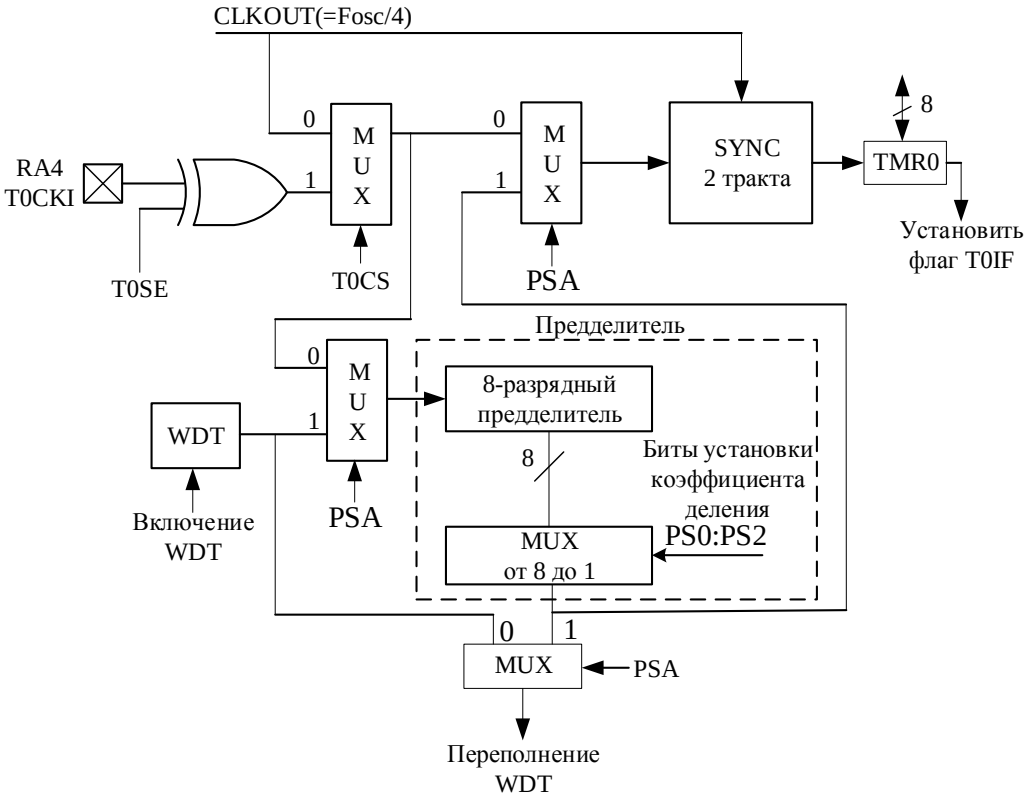

Рис. 2.7 — Структурная схема модуля таймера TMR0 с битами управления

Конфигурационные биты для настройки модуля TMR0 находятся в регистре OPTION_REG. Биты разрешения прерываний от модуля TMR0 содержатся в регистре INTCON.

Если прерывания от модуля TMR0 разрешены, то при каждом переполнении 8-разрядного регистра TMR0 будет происходить установка флага T0IF в «1» и, следовательно, будет генерироваться сигнал прерывания.

Без использования предварительного делителя приращение регистра TMR0 составляет $F_{osc}/4$, где F_{osc} — тактовая частота микроконтроллера.

При необходимости можно замедлить приращение регистра TMR0, тем самым увеличив интервалы времени между переполнениями регистра TMR0. Для этого необходимо задействовать биты PS0, PS1 и PS2, находящиеся в регистре OPTION_REG, которые управляют коэффициентом предварительного делителя для модуля TMR0 (см. рис. 2.7).

Примеры программ с настройкой модуля TMR0 и разрешением прерываний показаны на рис. 2.8 и рис. 2.9.

```
#include "P16F877.inc"
#include "PIC_MAC.inc"

org 0x00
    goto MAIN
org 0x04
    goto ISR

CBLOCK 0x20
ENDC
```

Рис. 2.8 — Фрагмент программы с векторами начала памяти программ и прерывания

```
MAIN

    b1
    CLRF    INTCON
    b0
    bcf     T0IF    ;очистка флага прерывания
    bsf     T0IE    ;разрешение прерываний от таймер 0
    bsf     GIE     ;разрешение всех прерываний
;если требуется замедлить приращение регистра TMR0
;следует установить биты PS0 - PS2 регистра OPTION_REG
```

Рис. 2.9 — Фрагмент программы разрешения прерываний от модуля таймера TMR0

Все действия, которые необходимо производить при каждом переполнении регистра TMR0, необходимо поместить в подпрограмму обработки прерываний.

Подпрограмма обработки прерываний по прерыванию от модуля таймера TMR0 показана на рис. 2.10.

```
ISR
;-----------------------------------------
        MOVWF   W_TEMP      ;сохранение контекста программы
        SWAPF   STATUS,W    ;при входе в попрограмму прерываний
        CLRF    STATUS
        MOVWF   S_TEMP
        MOVF    PCLATH,W
        MOVWF   PC_TEMP
        CLRF    PCLATH
;-----------------------------------------

        BTFSS   T0IF        ;проверка флага переполнения TMR0
        goto    OTHER_INT   ;переход к другим прерываниям

        ;здесь необходимо пометить программу которая будет
        ;выполняться при каждом прерывании при переполнении
        ;регистра TMR0

        bcf     T0IF

OTHER_INT

;-----------------------------------------
        MOVF    PC_TEMP,W   ;загрузка сохраненного контекста прогаммы
        MOVWF   PCLATH      ;по выходу из подпрограммы обработки прерываний

        SWAPF   S_TEMP,W
        MOVWF   STATUS

        SWAPF   W_TEMP,F
        SWAPF   W_TEMP,W
;-----------------------------------------
    retfie
end
```

Рис. 2.10 — Подпрограмма обработки прерываний от модуля TMR0

По завершению выполнения всех действий в подпрограмме обработки прерываний необходимо обязательно очистить флаг T0IF модуля таймера TMR0. Очистка флага необходима для того, чтобы разрешить следующее прерывание по переполнению регистра TMR0.

При использовании подпрограммы обработки прерываний от модуля TMR0 (рис. 2.10) следует объявить все регистры общего назначения (PC_TEMP, W_TEMP, S_TEMP) в С-блоке, в противном случае компиляция исходного файла не осуществится.

Для использования битов без названия регистра следует использовать директиву «#define» в следующем виде «#define T0IF INTCON,2» в заголовочном файле «P16F877.inc».

2.2.5. Организация чтения транскодировочной таблицы на языке Ассемблера

Для удобства вывода десятичных цифр на разряды индикатора необходимо использовать подпрограмму транскодирования или таблицу, содержащую заранее подготовленные комбинации логических уровней для изображения всех десятичных цифр. Таблица транкодировки показана на рис. 2.11.

```
TRANSCODING
    addwf   PCL,f
    retlw   b'00111111' ;0
    retlw   b'00000110' ;1
    retlw   b'01011011' ;2
    retlw   b'01001111' ;3
    retlw   b'01100110' ;4
    retlw   b'01101101' ;5
    retlw   b'01111101' ;6
    retlw   b'00000111' ;7
    retlw   b'01111111' ;8
    retlw   b'01101111' ;9
```

Рис. 2.11 — Подпрограмма транскодирования для десятичных чисел

Для отображения десятичной цифры на выбранном разряде индикатора из транскодировочной таблицы считывается соответствующая комбинация битов, которая затем передается на порт D.

В бесконечном цикле программы для вызова подпрограммы транскодирования следует использовать метку TRANSCODING.

Для чтения комбинации битов из транскодировочной таблицы необходимо выполнить следующие этапы:

1) загрузить адрес смещения в аккумулятор (адрес смещения равен цифре, которую необходимо отобразить на разряде);

2) вызвать таблицу, используя метку TRANSCODING;

3) в таблице первой инструкцией следует к содержимому регистра PCL прибавить адрес смещения, ранее загруженный в аккумулятор;

4) в таблице использовать команду возврата с сохранением константы в аккумуляторе;

5) после возврата в основную программу содержимое аккумулятора необходимо переместить в регистр порта D.

Все этапы чтения таблицы необходимо повторять для каждого сегмента. Таблица транскодирования должна быть расположена сразу после основного цикла программы или перед директивой окончания программы end.

2.2.6. Программа для реализации динамической индикации с использованием программной задержки

Для работы с семисегментным индикатором задействованы порты A, D и E. Поскольку все выводы портов должны работать на выход, биты регистров

TRISA, TRISD, TRISE, отвечающих за направленность выводов портов, должны быть сброшены в «0».

Для использования выводов портов A и E в цифровом режиме необходимо загрузить двоичную комбинацию 00000110 в регистр ADCON0. Для вывода логических уровней следует использовать регистры портов PORTA, PORTD и PORTE. При этом необходимо указывать банки памяти данных, в которых находятся эти регистры.

Пример программы настройки выводов портов показан на рис. 2.12.

```
MAIN
    b1
    CLRF    TRISA   ;сброс всех битов
    CLRF    TRISD   ;для настройки выводов
    CLRF    TRISE   ;портов на выход
    MOVLW   b'00000110' ;загрузка константы в аккумулятор
    MOVWF   ADCON1      ; загрузка константы в регистр
    b0
    CLRF    PORTD   ;очистка выводов прота
    CLRF    PORTA   ;очистка выводов прота
    CLRF    PORTE   ;очистка выводов прота
LOOP            ;метка начала основной программы
    goto LOOP
ISR
    retfie
    end
```

Рис. 2.12 — Код программы настройки выводов портов

Фрагмент программы реализации динамической индикации без использования транскодировочной таблицы показан на рис. 2.13. В приведенном фрагменте программы реализована динамическая индикация, при которой задействованы только два разряда из четырех. Для реализации полноценной динамической индикации в дальнейшем необходимо реализовать программную задержку (delay).

```
LOOP            ;метка начала основной программы

    MOVLW   b'00111111' ;загрузка аккумулятора для вывода 0
    MOVWF   PORTD       ;на первый разряд индикатора
    bsf     PORTE,0     ;включение первого разряда
    CALL    delay       ;вызов задержки
    bcf     PORTE,0     ;выключение первого разряда
    MOVLW   b'00000110' ;загрузка аккумулятора для вывода 1
    MOVWF   PORTD       ;на второй разряд индикатора
    bsf     PORTA,2     ;включение второго разряда
    CALL    delay       ;вызов задержки
    bcf     PORTA,2     ;включение второго разряда

    goto LOOP
```

Рис. 2.13 — Код программы реализации динамической индикации с использованием двух разрядов

Показанный пример программы не содержит управляемой задержки и чтения транскодировочной таблицы. Эти элементы программы следует внедрить самостоятельно, используя фрагменты кода, рассмотренные ранее. Далее, запустите режим внутрисхемной отладки и последовательно выполните каждую инструкции кода наблюдая связь выполнения инструкций с зажиганием разрядов.

2.2.7. Структура программы для реализации динамической индикации с применением аппаратной задержки

Для реализации аппаратной задержки необходимо воспользоваться кодом программ (см. рис. 2.9), который содержит настройки прерываний от модуля TMR0 и содержит полную подпрограмму обработки прерываний. При этом при реализации динамической индикации с аппаратной задержкой необходимо произвести изменения в подпрограмме обработки прерываний, как показано на рис. 2.14.

```
BTFSS   T0IF        ;проверка флага переполнения TMR0
goto    OTHER_INT   ;переход к другим прерываниям

;здесь необходимо применить программное решение
;которое позовлит выводить цифру только на один разряд
;при каждом прерывании от модуля TMR0

;если этот блок инструкций выполняется,
;то следующий блок должен быть пропущен
CLRF    PORTA       ;выключение всех разрядов
CLRF    PORTE
MOVLW   b'00111111' ;загрузка аккумулятора для вывода 0
MOVWF   PORTD       ;на первый разряд индикатора
bsf     PORTE,0     ;включение первого разряда

;если этот блок инсрукций выполняется,
;то предыдущий должен быть пропущен
CLRF    PORTA       ;выключение всех разрядов
CLRF    PORTE
MOVLW   b'00000110' ;загрузка аккумулятора для вывода 1
MOVWF   PORTD       ;на второй разряд индикатора
bsf     PORTA,3     ;включение второго разряда


THER_INT
```

Рис. 2.14 — Фрагмент подпрограммы обработки прерываний для реализации динамической индикации с аппаратной задержкой

2.3. Порядок выполнения лабораторной работы

2.3.1. Реализация динамической индикации с использованием программной задержки

1) Используйте примеры программ, приведенных в теоретической части, в среде MPLAB написать программу для включения всех сегментов одного из разрядов (статическая индикация), затем запрограммировать микроконтроллер лабораторного макета и убедиться в правильности работы программы.

2) Изучите написанную программу и внести в нее изменения, которые позволят задействовать другой разряд индикатора, засветив все сегменты и выключив предыдущий разряд (статическая индикация). Запрограммировать микроконтроллер и убедиться в правильности выполнения программы.

3) Выведите на один из разрядов индикатора произвольную цифру от 0 до 9, загрузив необходимую комбинацию «1» и «0» в регистр порта D (статическая индикация).

4) Выведите произвольную цифру на одни из разрядов индикатора, используя транскодировочную таблицу (статическая индикация). Затем запрограммировать микроконтроллер и убедиться в правильности выполнения программы.

5) Используя примеры программ, приведенных в теоретической части (п. 2.2.6.), реализовать динамическую индикацию, в которой задействованы все разряды индикатора для вывода 1, 2, 3 и 4. Для этого необходимо реализовать программную задержку длительностью порядка (0,5…1) секунды (для возможности визуального наблюдения переключение разрядов). Написанную программу требуется загрузить в микроконтроллер и убедиться в правильности выполнения программы.

6) Заключительный этап выполнения п. 2.5.1 состоит в доработке подпрограммы задержки, с целью получения возможности задавать требуемую временную задержку. Реализуйте три отдельных подпрограммы задержки: delay_us, delay_ms, delay_s для генерации задержек в микросекундном, миллисекундном и секундном диапазонах соответственно. После чего необходимо уменьшить временную задержку между переключениями разрядов так, чтобы мерцания индикатора не были заметны.

Данный пункт может быть выполнен, только при успешном выполнении всех предыдущих пунктов.

7) Окончательный вариант программы следует сохранен отдельным файлом и прокомментирован в ключевых этапах программы.

2.3.2. Реализация динамической индикации с использованием аппаратной задержки

1) Используйте программный код, разработанный ранее, реализуйте динамическую индикацию с помощью аппаратной задержки, используя прерывания от модуля TMR0, для вывода цифр 6, 7, 8 и 9 на разряды индикатора. Для

реализации аппаратной задержки используйте фрагменты программ, приведенные в теоретической части

2) Окончательный вариант программы следует сохранить отдельным файлом и с необходимыми комментариями.

2.4. Содержание отчета

Цель работы.

Электрическая принципиальная схема задействованной части макета.

Теоретическая часть.

Ассемблерные коды программ для динамической индикации с программной и аппаратной задержками.

Подробные выводы.

2.5. Контрольные вопросы

2.5.1. Динамический и статический методы индикации. Сравнительный анализ.

2.5.2. Порты микроконтроллера использующиеся для управления сегментами и разрядами индикатора.

2.5.3. Регистры управления направленностью выводов портов микроконтроллера.

2.5.4. Составление и использование таблицы транскодирования.

2.5.5. Принцип реализации аппаратной и программной задержек. Сравнительный анализ двух типов задержек.

2.5.6. Расчет длительности программной и аппаратной задержек.

2.5.7. Схемотехнические решения, позволяющие уменьшить количество используемых выводов микроконтроллера при реализации динамической индикации с сохранением полной функциональности.

ЛАБОРАТОРНАЯ РАБОТА № 3. ФУНЦИОНИРОВАНИЕ ПРЕРЫВАНИЙ ДЛЯ РЕГИСТРАЦИИ НАЖАТИЯ КНОПОЧНЫХ КОНТАКТОВ

3.1. Цель работы

Ознакомиться с логикой прерываний микроконтроллера PIC16F877 и изучить прерывания, связанные с портом B.

Изучить устройство и принцип опроса клавиатуры, имеющей матричное соединение кнопочных контактов.

3.2. Теоретические сведения

Для выполнения данной лабораторной работы потребуется задействовать следующие части лабораторного макета:

— семисегментный четырехразрядный индикатор;
— 5 кнопочных контактов.

Пять пар кнопочных контактов лабораторного макета подключены к выводам порта B. При работе порта B микроконтроллера PIC16F877 с кнопочными контактами аппаратная часть позволяет генерировать два типа прерываний по изменению логических уровней на некоторых выводах.

3.2.1. Использование выводов порта B в качестве источников прерываний

Выводы порта B изучаемого микроконтроллера могут служить источниками прерываний двух типов:

— внешнее прерывание от вывода RB0, которое обеспечивает аппаратную регистрацию и генерацию прерывания по положительному или отрицательному изменению логического уровня;
— прерывания по изменению логического уровня на одном из выводов RB7—RB4.

Для инициализации внешнего прерывания от вывода RB0 требуется выполнить следующие шаги:

— настроить вывод порта на вход, используя регистр TRISB;
— если требуется, задействовать подтягивающий резистор, сбросив бит RBPU;
— выбрать активный фронт для прерывания, установив или сбросив бит INTEDG;
— сбросить флаг прерывания INTF;
— разрешить прерывания от вывода RB0, установив бит INTE в «1»;
— разрешить все прерывания установкой бита GIE в «1».

После этого поместить желаемые действия в подпрограмму обработки прерываний, которое будет выполняться при каждом внешнем прерывании. Но

важно пред завершением обработки прерывания произвести программный сброс флага INTF, в противном случае внешнее прерывание более не произойдет.

Для инициализации прерываний по изменению логического уровня на одном из выводов RB7—RB4 следует выполнить следующие шаги:

— настроить выводы порта на вход, используя регистр TRISB;
— если требуется, задействовать подтягивающие резисторы, сбросив бит RBPU;
— произвести чтение или запись порта B;
— сбросить флаг прерывания RBIF;
— разрешить прерывания по изменению логического уровня на одном из выводов, установив бит RBIE в «1»;
— разрешить все прерывания установкой бита GIE в «1».

Так же, как и для обработки внешнего прерывания, все действия следует поместить в подпрограмму обработки прерываний. После завершения обработки прерывания требуется сбросить флаг RBIF. Необходимо помнить, что очищать флаг RBIF можно только после процедуры чтения или записи порта B.

При необходимости использования двух источников прерывания общие шаги настроек не нуждаются в повторении.

Пример подпрограммы совместной обработки внешнего прерывания от вывода RB0 и прерывания по изменению уровня на одном из выводов RB7—RB4 показан на рис. 3.3.

```
ISR ;Interrupt Service Routine
    ;необходимые действия
    ;для сохранения контекста
    ;основной программы
    BTFSS   INTF    ;проверка внешнего прерывания
    GOTO    NEXT_INT;переход к следующему флагу
    ;здесь помещается программа обработки прерывания
    bcf     INTF    ;очистка флага
NEXT_INT
    BTFSS   RBIF ;проверка прерывания от одного из выводов RB7:RB4
    goto    NEXT_INT1   ;переход к следующему флагу
    ;здесь помещается программа обработки прерывания
    MOVF    PORTB,1
    bcf     RBIF    ;очистка флага
NEXT_INT1
    ;восстановление контекста
    ;основнйо программы
retfie
```

Рис. 3.1 — Подпрограмма обработки прерываний от вывода RB0 и по изменению логического уровня на одном из выводов RB7—RB4

В подпрограмме обработки прерываний (см. рис. 3.1) не показана часть кода, которая позволяет сохранить контекст основной программы. Это необходимо выполнить аналогично примеру обработчика прерываний в лабораторной работе № 2 (см. рис. 2.10).

3.2.2. Конфигурация и опрос матричного включения кнопочных контактов

Соединение кнопочных контактов в виде матрицы позволяет обеспечить использование максимально возможного количества кнопочных контактов при использовании наименьшего количества выводов микроконтроллера. Таким образом, имея всего 8 выводов, можно соединить 16 пар кнопочных контактов, как показано на рис. 3.2.

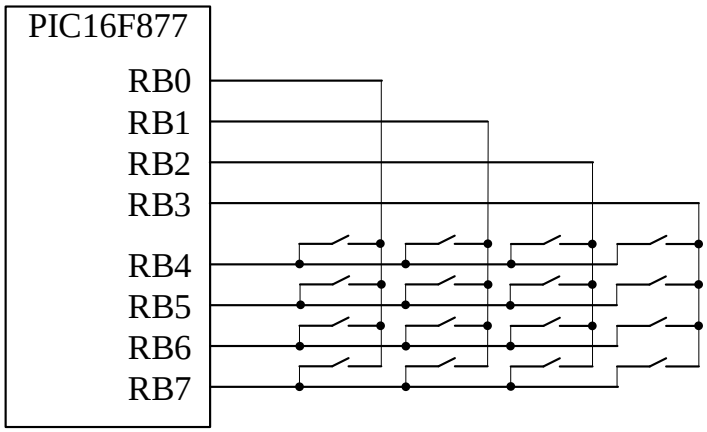

Рис. 3.2 — Матричное включение кнопочных контактов

Количество используемых выводов микроконтроллера и кнопочных контактов может быть изменено по усмотрению проектировщика.

Особенностью работы матричной клавиатуры является необходимость проверки наличия замыкания или размыкания кнопочных контактов на пересечении двух линий.

Поскольку используемый микроконтроллер может генерировать прерывания по изменению логического уровня на выводах RB7—RB4, то следует выполнить следующие шаги для настройки выводов и детектирования замыкания одной из 16 пар кнопочных контактов, включенных матрицей 4 × 4.

Для настройки выводов RB7—RB4 на прерывания по изменению логического уровня необходимо:

— настроить четыре младших вывода RB3—RB0 на выход и вывести логические нули (этот этап можно не выполнять, если для опроса кнопочных контактов не используются младшие выводы порта B);
— настроить четыре старших вывода порта B на вход, задействовав внутренние подтягивающие резисторы;
— разрешить прерывание по изменению логического уровня на одном из выводов RB7—RB4.

Метод детектирования замыкания и размыкания кнопочных контактов основан на генерации прерывания. Поскольку младшие выводы находятся в низком логическом состоянии и настроены на выход («сильный» логический ноль), а старшие выводы порта находятся в высоком логическом состоянии (за счет внутренних подтягивающих резисторов образуется «слабая» логическая единица) и настроены на вход, то замыкание какой-либо пары кнопочных кон-

тактов приведет к изменению состояния входа порта B и, следовательно, к возникновению условия прерывания. Это произойдет за счет того, что уровень «сильного» логического нуля поступит на один из входов, имеющих «слабую» логическую единицу (изменение логического уровня), что приведет к установлению нуля на соответствующем входе и, следовательно, выполнится условие прерывания.

В подпрограмме обработки прерываний необходимо поочередно проверить все старшие выводы с целью обнаружения низкого логического уровня, так как они настроены на вход. Присутствие низкого логического уровня на одном из выводов RB7—RB4 свидетельствует о замкнутых кнопочных контактах на одной из линий. Далее требуется выводить логическую «1» на каждый из четырех младших выводов порта B, пока логический уровень вывода, на котором был обнаружен низкий логический уровень, не изменится на противоположный. Когда высокий уровень зафиксирован, то найдена вторая линия, следовательно, на пересечении этих линий и были замкнуты кнопочные контакты.

Очень важным моментом при написании подпрограммы обработки является строка MOVF PORTB,f. Данная инструкция осуществляет считывание текущих логических уровней из входных защелок порта B и запись считанных логических уровней обратно в регистр PORTB. Другими словами, регистр PORTB записывается теми же значениями, которые в нем и были. Такое действие позволяет снять условие несоответствия, в противном случае флаг прерывания RBIF вновь примет значение логической единицы сразу же после его очистки.

Одиночное «нажатие» и «отпускание» кнопки будет сопровождаться двумя прерываниями. Однако проверка по размыканию не может быть осуществлена, так как все четыре старших вывода принимают высокий логический уровень и в подпрограмме обработки прерываний все условия проверки логических уровней старших выводов будут пропущены.

Таким образом, настройка выводов, описанная выше, позволяет детектировать только замкнутые кнопочные контакты, в то время как разомкнутые контакты при таком включении не могут быть проверены. Однако при необходимости детектирование размыкания кнопочных контактов может быть организовано в программе.

3.2.3. Подпрограмма обработки прерываний от порта B по изменению логического уровня на выводах RB7—RB4

Фрагмент программного кода (рис. 3.3) демонстрирует детектирование замкнутых кнопочных контактов для схемы лабораторного макета. Согласно электрической принципиальной схемы (см. рис. 1.1), микроконтроллер лабораторного макета имеет пять пар кнопочных контактов.

```
ISR
    BTFSS   RBIF
    goto    OTHER_INT
    ; проверка, нажата ли одна из кнопок по линии RB4
    BTFSC   PORTB,4 ; проверка низкого состояния на линии RB4
    GOTO    N1      ; если высокое, то переходим по метке
    ; проврека кнопки на линиях RB2 и RB4
    bsf     PORTB,2 ;устанавливаем линию RB2 в единицу
    BTFSC   PORTB,4 ;проверка единицы на линии RB4
    ;здесь пишется действие для обработки по нажатию кнопки RB2 и RB4
    bcf     PORTB,2
    ; проврека кнопки на линиях RB3 и RB4
    bsf     PORTB,3 ;устанавливаем линию RB3 в единицу
    BTFSC   PORTB,4 ;проверка единицы на линии RB4
    ;здесь пишется действие для обработки по нажатию кнопки RB3 и RB4
    bcf     PORTB,3
    ; проверка, нажата ли одна из кнопок по линии RB5
N1  BTFSC   PORTB,5 ; проверка низкого состояния на линии RB5
    GOTO    N2      ;если высокое, то переходим по метке
    bsf     PORTB,2 ;устанавливаем линию RB2 в единицу
    BTFSC   PORTB,5 ;проверка единицы на линии RB5
    ;здесь пишется действие для обработки по нажатию кнопки RB2 и RB5
    bcf     PORTB,2
    bsf     PORTB,3 ;устанавливаем линию RB3 в единицу
    BTFSC   PORTB,5 ;проверка единицы на линии RB5
    ;здесь пишется действие для обработки по нажатию кнопки RB3 и RB5
    bcf     PORTB,3
     ;проверка всех кнопок завершена
N2  MOVF    PORTB,1 ;перезапись регистра порта
    bcf     RBIF    ;сброс флага прерывания
OTHER_INT
    retfie
```

Рис. 3.3 — Фрагмент подпрограммы обработки прерываний по изменению логического уровня на одном из входов RB7—RB4

Одна из пар контактов индивидуально подключена к выводу RB0, а оставшиеся четыре пары кнопочных контактов подключены матрицей к выводам RB2, RB3 и RB4, RB5.

Показанный на рис. 3.3 фрагмент программы следует дополнить, поскольку он позволяет идентифицировать только замыкание контактов (т.е. реагирование только на нажатие), но не позволяет идентифицировать размыкание кнопочных контактов. При этом важно помнить о том, что при замыкании и размыкании контактов возникает «дребезг», который порождает кратковременные помехи с частотой приблизительно равной звуковым колебаниям в виде щелчка, который образовывается при воздействии на контакты кнопки. Для устройства цифрового входа, «дребезг» контактов распознается как многократное замыкание и размыкание кнопочных контактов, хотя по факту было осуществлено единичное замыкание или размыкание. Исключить «дребезг» контактов кнопки практически невозможно, но можно исключить или в значительной степени снизить его влияние.

Можно рассмотреть два пути снижения негативного эффекта от «дребезга» контактов: программный и схемотехнический в виде RC фильтра нижних частот. Смысл программного метода состоит в том, что после фиксации изменения первого логического уровня от кнопочных контактов, следует отключить вход или выключить прерывания на некоторое время в течение которого длится «дребезг», а затем возобновить нормальную работу вывода или включить прерывания.

3.3. Порядок выполнения лабораторной работы

1) Для освоения обработки прерываний по замыканию и размыканию кнопочных контактов требуется написать программу, которая осуществляет вывод чисел 1, 2, 3 и 4 на разряды индикатора в динамическом режиме. При замыкании соответствующих кнопочных контактов, включенных матрицей, разряды должны выключаться и включаться при повторном замыкании кнопочных контактов. А пара кнопочных контактов, подключенная к входу внешнего прерывания, должна выключать все разряды по размыканию контактов. Написанную программу следует загрузить в микроконтроллер и убедиться, что программа выполняется правильно.

Дополнительно, задействуйте индикаторные светодиоды для индикации нажатия кнопок.

2) После успешного выполнения предыдущего задания в соответствии с индивидуальным заданием требуется реализовать программу счетчика нажатий кнопочных контактов. Диапазон счета, используемые кнопочные контакты и разряд выбираются по усмотрению обучающегося. Для реализации динамической индикации следует воспользоваться программой из предыдущей лабораторной работы. Далее, требуется запрограммировать микроконтроллер лабораторного стенда и убедиться, что программа выполняется верно.

3.4. Содержание отчета

Цель работы.

Электрическая принципиальная схема задействованной части макета.

Часть программы, где реализована обработка прерываний от кнопочных контактов с программой индивидуального задания.

Подробные выводы.

3.5. Контрольные вопросы

3.5.1. Логика прерываний PIC16F877.

3.5.2. Биты разрешения/запрещения и флаги прерываний.

3.5.3. Прерывания по порту B.

3.5.4. Особенность матричного соединения кнопочных контактов. Способы защиты от короткого замыкания при одновременном нажатии нескольких пар кнопочных контактов.

3.5.5. Детектирование нажатия пар кнопочных контактов матричной клавиатуры.

3.5.6. Организация подпрограммы обработки прерываний по изменению логического уровня на выводах RB7—RB4.

ЛАБОРАТОРНАЯ РАБОТА № 4.
МОДУЛЬ АНАЛОГО-ЦИФРОВОГО ПРЕОБРАЗОВАНИЯ

4.1. Цель работы

Изучить функционирование модуля АЦП.

Изучить способы аналого-цифрового преобразования с разрешением и без разрешения прерывания от модуля АЦП.

Изучить, как использовать выводы портов в качестве аналоговых каналов.

Освоить принцип двоично-десятичного преобразования для вывода результатов аналого-цифрового преобразования в десятичном формате на индикатор.

4.2. Теоретическая часть

Для выполнения данной лабораторной работы потребуется использовать следующие части лабораторного макета:

— переменные резисторы, подключенные к выводам RA0 и RA1;

— семисегментный индикатор;

— одна пара кнопочных контактов, подключенная к выводу RB0.

4.2.1. Устройство модуля АЦП

Данный микроконтроллер имеет 10 разрядный модуль АЦП, который мультиплицируется с 8 аналоговыми каналами. Некоторые выводы портов A и E используются в качестве аналоговых каналов. Управление модулем АЦП осуществляется через следующие регистры ADCON0 и ADCON1.

Регистр ADCON0 содержит биты конфигурации для модуля АЦП, а регистр ADCON1 содержит биты, переключающие выводы портов на аналоговые каналы. Подробное описание регистров, связанных с модулем АЦП, изложено в Приложении Б.

Структурная схема модуля АЦП показана на рис. 4.1.

Выбор аналоговых каналов

В качестве аналоговых каналов используются некоторые выводы портов A и E (см. рис. 4.1). Для использования соответствующих выводов портов в режиме аналоговых входов, требуется настроить эти выводы на входы, используя регистры TRISA и TRISE. А также необходимо определить выводы как аналоговые входы установкой битов PCFG3—PCFG0, которые содержатся в регистре ADCON1.

Модуль АЦП мультиплицируется с восемью независимыми аналоговыми каналами по которым могут подаваться напряжения разных уровней, но одновременно может осуществляться конвертирование напряжения только на одном из аналоговых каналов. Для выбора канала следует использовать биты CHS2—CHS0. Эти биты требуется устанавливать каждый раз перед началом нового аналого-цифрового преобразования, если используется более чем один

аналоговый канал. Биты CHS2—CHS0 конфигурируются один раз, если используется только один аналоговый канал.

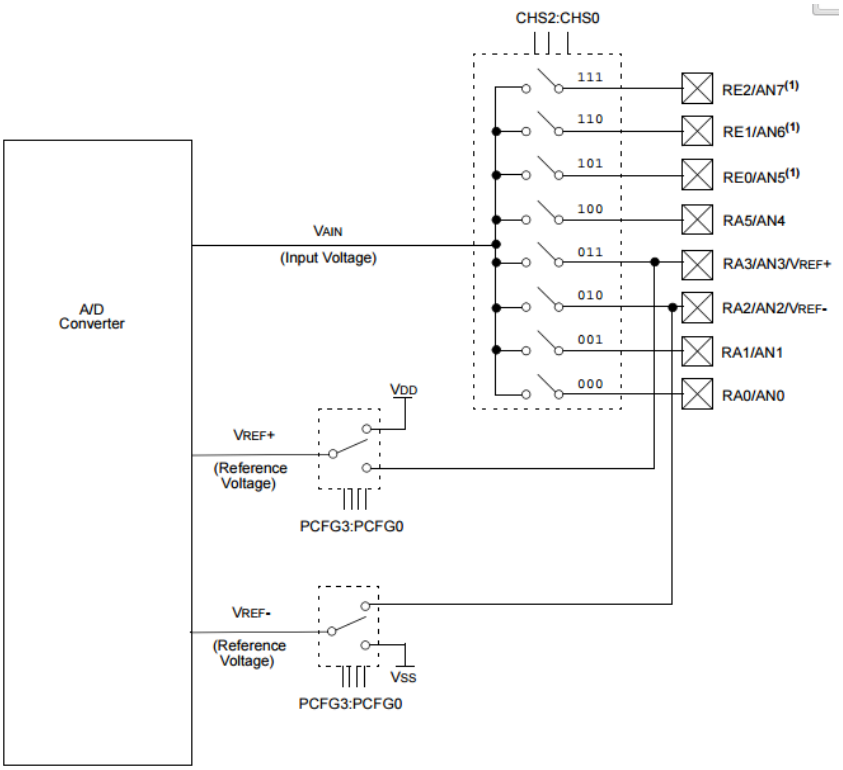

Рис. 4.1 — Структурная схема модуля АЦП

Источники опорных напряжений для модуля АЦП. Как видно из структурной схемы (см. рис. 4.1), данный модуль АЦП может использовать внутренние (выводы опорных напряжений модуля АЦП коммутируются с положительной и отрицательной шинами внутри самого микроконтроллера) или внешние источники опорных напряжений. Источники опорных напряжений задают приделы динамического диапазона измеряемых напряжений. Использование внешних опорных напряжений удобно для повышения разрешающей способности преобразования, за счет сужения динамического диапазона измеряемых напряжений. Если же разрешающая способность конвертирования достаточна, то можно использовать напряжение питания микроконтроллера в качестве опорного напряжения. В таком случае в качестве нижнего предела динамического диапазона используется общая шина микроконтроллера, имеющая нулевой потенциал, а в качестве верхнего предела используется положительная шина питания, имеющая напряжение +5 В. Использование питающего напряжения микроконтроллера как опорного удобно тем, что выводы RA3 и RA2 остаются доступными для работы в качестве аналоговых каналов.

Выбор источников опорных напряжений модуля АЦП осуществляется при помощи битов PCFG0—PCFG3 в регистре ADCON1.

Выбор частоты преобразования для модуля АЦП. Модуль АЦП позволяет преобразовать напряжение (аналоговый сигнал) в 10 разрядный пропорциональный двоичный код. Время получения одного бита занимает от 1,6—2 мкс, для обеспечения такого временного интервала, модулю АЦП требуется тактовый сигнал, который значительно медленнее относительно тактового сигнала микроконтроллера. Так как период тактовых импульсов микроконтроллера значительно меньше требуемого, то для корректного преобразования требуется замедлить сигнал тактирования для модуля АЦП.

Регистр ADCON1 содержит биты ADCS1 и ADCS0 которые позволяют программно определить тактовую частоту для модуля АЦП. Справочные данные по выбору тактовой частоты для АЦП исходя из тактовой частоты микроконтроллера приведены в Таблице 4.1.

Таблица 4.1 — Выбор тактовой частоты для модуля АЦП относительно тактовой частоты микроконтроллера

Тактовая частота для модуля АЦП	ADCS1—ADCS0	Максимальная тактовая частота микроконтроллера
Fosc/2	00	1,25 МГц
Fosc/8	01	5 МГц
Fosc/32	10	20 МГц
RC генератор	11	Собственный генератор АЦП

В таблице также приведены значения битов для использования собственного RC генератора. Преимущество в использовании такого генератора заключается в том, что в SLEEP режиме модуль АЦП продолжает выполнять преобразование.

Сохранение результата аналого-цифрового преобразования. Результат аналого-цифрового преобразования сохраняется в двух 8-разрядных регистрах ADRESH и ADRESL. Такой способ сохранения результата преобразования обуславливается тем, что регистры данных имеют всего 8 разрядов, и поэтому для сохранения 10-разрядного результата требуется два отдельных регистра. Важно помнить, о том, что регистр ADRESH находится в нулевом банке, а регистр ADRESL находится в первом банке памяти данных, поэтому для корректного доступа к регистрам, необходимо предварительно переключать банки памяти данных.

По усмотрению проектировщика сохранение результата преобразования может быть произведено с правым или левым выравниванием (рис. 4.2). Выравнивание задается сбросом или установкой бита ADFM, находящегося в регистре ADCON1.

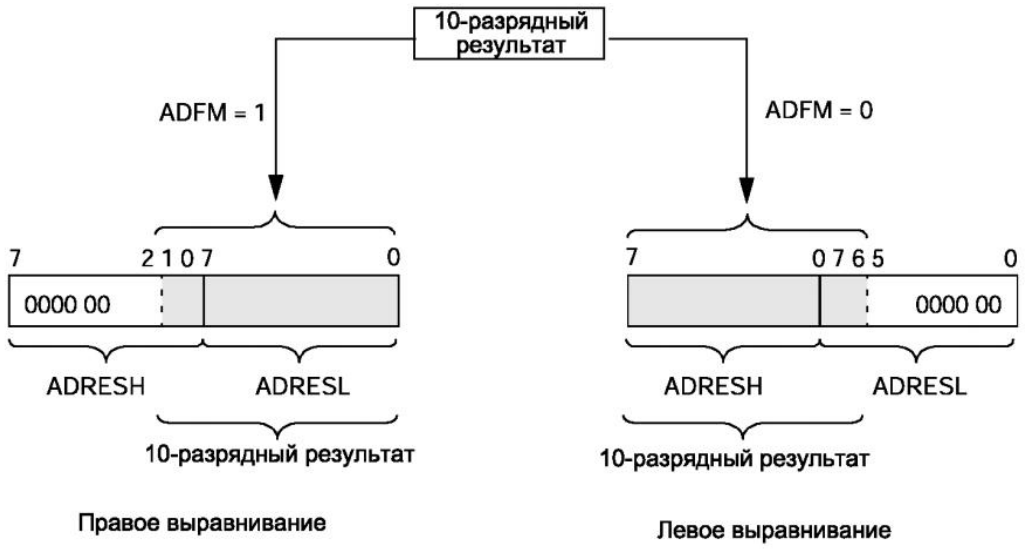

Рис. 4.2 — Левое и правое выравнивания результата аналого-цифрового преобразования

4.2.2. Работа с модулем АЦП

Конфигурация модуля АЦП с разрешением прерывания по завершению преобразования. Работа модуля АЦП с разрешением прерывания по завершению преобразования, удобна тем, что нет необходимости ожидать окончание преобразования проверяя состояние бита GO-DONE. Как только преобразование завершено, модуль АЦП устанавливает флаг ADIF в высокое логическое состояние, генерируя сигнал прерывания (если прерывания разрешены). По сигналу прерывания, в подпрограмме обработки прерываний требуется только прочесть результат преобразования и сбросить флаг ADIF.

Для использования модуля АЦП с генерацией прерывания требуется выполнить следующие шаги:

1) Выводы, которые нужно использовать в качестве аналоговых каналов следует настроить на вход, установкой соответствующих битов регистров TRISx и сконфигурировать как аналоговые каналы путем установки битов PCFG3—PCFG0.

2) Выбрать необходимый канал АЦП с помощью битов CHS2—CHS0 регистра ADCON0 (если требуется использовать более одного аналогового канала, то биты CHS2—CHS0 необходимо изменять в бесконечном цикле программы перед каждой установкой бита GO-DONE).

3) Задать частоту тактовых импульсов для АЦП.

4) Определить выравнивание битов результата преобразования установив или сбросив бит ADFM в регистре ADCON1.

5) Включить модуль АЦП установив бит ADON в регистре ADCON0.

6) Настроить прерывания от модуля АЦП:

— сбросить бит ADIF;

— установить бит ADIE;

— установить бит PEIE;

— установить бит GIE.

Следующие шаги необходимо выполнять в бесконечном цикле программы.

1) Выдержать паузу для зарядки конденсатора модуля АЦП.

2) Установить бит GO-DONE в единицу для начала конвертирования.

3) Ожидать прерывания по окончанию преобразования, затем считать результаты преобразования сохраненных в регистрах ADRESH и ADRESL и сбросить бит ADIF в «0».

4) Повторное преобразование необходимо начать с выбора аналогового канала.

Фрагмент программного кода с настройкой прерываний от модуля АЦП показан на рис. 4.3.

```
    b1
    CLRF    TRISA
    bsf     TRISA,0 ;настрока вывода RA0 на вход
    MOVLW   b'00001110'
    MOVWF   ADCON1  ;конфигурация RA0 на аналоговый вход
    b0
    bsf     ADCS1   ;установка тактовой частоты АЦП
    bsf     ADON    ;включение модуля АЦП
    bcf     ADIF    ;сброс флага
    b1
    bcf     ADFM    ;левое выравнивание результата
    bsf     ADIE    ;разрешение прерываний от АЦП
    b0
    bsf     PEIE    ;разрешение всех маскированных прерываний
    bsf     GIE     ;глобальное разрешение прерываний
```

Рис. 4.3 — Фрагмент кода для разрешения прерываний от модуля АЦП

Фрагмент подпрограммы обработки прерывания от модуля АЦП приведен на рис. 4.4.

```
ISR
    BTFSS   ADIF        ;проверка флага
    goto    OTHER_INT   ;переход на остальные прерывания
    b1              ;выбор банка 1
    MOVF    ADRESL,w    ;загрузка младшего результата преобразования
    MOVWF   L_RESULT    ; в регистр общего назначения
    b0              ;выбор банка 0
    MOVF    ADRESH,w    ;загрузка старшего результата преобразования
    MOVWF   H_RESULT    ; в регистр общего назначения

    bcf     ADIF        ;очистка флага прерывания
OTHER_INT
    retfie
```

Рис. 4.4 — Подпрограмма обработки прерывания для чтения результатов аналого-цифрового преобразования

Конфигурация модуля АЦП без разрешения прерывания. Для конфигурации модуля АЦП без разрешения прерываний следует выполнить следующие шаги.

1) Выводы которые требуется использовать в качестве аналоговых каналов настроить на вход, установкой соответствующих битов регистров TRISx и сконфигурировать их как аналоговые каналы путем установки битов PCFG3 — PCFG0 регистра ADCON1.

2) Выбрать частоту тактовых импульсов для АЦП.

3) Определить выравнивание битов результата преобразования установив или сбросив бит ADFM в регистре ADCON0.

Следующие пункты должны быть выполнены в подпрограмме.

1) Непосредственно перед вызовом или в тексте подпрограммы, требуется выбрать необходимый канал АЦП с помощью битов CHS2—CHS0 регистра ADCON0.

2) Включить модуль АЦП установив бит ADON регистра ADCON0.

3) Выдержать паузу для зарядки конденсатора модуля АЦП 4-6 мкс.

4) Установить бит GO-DONE для начала конвертирования.

5) Ожидать сброса бита GO-DONE.

6) Считать результата преобразования из регистров ADRESH и ADRESL (необходимо переключить банки памяти данных).

7) Выключить модуль АЦП сбросив бит ADON регистра ADCON0.

Для повторного аналого-цифрового преобразования требуется вызвать подпрограмму, предварительно выбрав требуемый аналоговый канал.

Фрагмент программы для конфигурации модуля АЦП без разрешения прерываний показан на рис. 4.5.

```
b1
CLRF    TRISA
bsf     TRISA,0 ;настрока вывода RA0 на вход
MOVLW   b'00001110'
MOVWF   ADCON1  ;конфигурация RA0 на аналоговый вход
b0
bsf     ADCS0   ;установка тактовой частоты АЦП
bsf     ADCS1   ;установка тактовой частоты АЦП
bcf     ADFM    ;левое выравнивание результата
```

Рис. 4.5 — Конфигурация модуля АЦП без разрешения прерываний

Фрагмент подпрограммы для аналого-цифрового преобразования показан на рис. 4.6.

```
ADC ;подпрограмма аналого-цифрового конвертирования
    bsf     ADON    ;включение АЦП
    ;здесь необходима задержка не менее 2-6 мкс
    ;для заряда конденсатора
    bsf     GO      ;запуск преобразования АЦП
    BTFSC   GO
    goto    $-1     ;ожидание окончания преобразования
    bcf     ADON    ;выключение модуля АЦП
    b1  ;выбор банка 1
    MOVF    ADRESL,w    ;считывание младшего результата преобразования
    b0  ;выбора банка 0
    MOVWF   L_RESULT    ;сохранение младшего результата
    MOVF    ADRESH,w    ;считывание старшего результата
    MOVWF   H_RESULT    ;сохранение страшего результата
    return
```

Рис. 4.6 — Подпрограмма для аналого-цифрового преобразования

Двоично-десятичное преобразование. Поскольку микроконтроллер работает с двоичными числами, то для представления цифровой информации в формате удобном для восприятия пользователем, необходимо использовать десятичный формат. В данной лабораторной работе требуется выводить на дисплей результат аналого-цифрового преобразования предварительно преобразовав в десятичный формат. Процесс перевода числа из двоичного формата в десятичный называется двоично-десятичным преобразованием.

Например, если рассматривать регистр данных, который имеет 8 битов, то максимальное бинарное число для такого регистра будет b’11111111’, что эквивалентно числу 255 в десятичном формате. Для того чтобы вывести число 255 на разряды семисегментного индикатора, требуется извлечь информацию о том сколько единиц, десятков и сотен данное число содержит. Самым простым способом для извлечения разрядов из числа 255 будет отнимать последовательно от него сотни, затем от остатка отнимать десятки, а вторичный остаток будет являться цифрой показывающий количество единиц в младшем разряде. Результат вычитания из числа 255 даст следующее:

— две сотни;

— пять десятков;

— пять единиц.

Количество сотен, десятков и единиц необходимо использовать в качестве адресов смещения для чтения транскодировочной таблицы.

В данной лабораторной работе требуется использовать команды ассемблера для вычитания сотен, десятков и единиц контролируя арифметический флаг C регистра STATUS, чтобы не допустить заём из старшего разряда. Пример программы для нахождения количества сотен в числе 200 показан на рис. 4.7.

```
MOVLW   .100          ;для вычитания сотен
SUBWF   H_RESULT,0    ;буфер хранения результата АЦП

BTFSS   C             ;проверка флага заема C
goto    $+4           ;пропуск если заем произошел

MOVWF   H_RESULT      ;сохранение пезультата вычитания
INCF    BUF100,f      ;инкремент разряда при каждой итерации
goto    $-6           ;повторное вычитание сотен
```

Рис. 4.7 — Фрагмент программы для высчитывания сотен в регистре

4.3. Порядок выполнения лабораторной работы

Требуется выбрать один из способов аналого-цифрового преобразования, с разрешением или без разрешения прерывания.

Реализуйте программу для преобразования уровня напряжения на входе RA0.

Выведите результат аналого-цифрового преобразования используя динамическую индикацию, предварительно выполнив двоично-десятичное преобразование (требуется разработать два варианта кода с использованием 8 и 10 разрядов результата преобразования).

Задействуйте одну из пар кнопочных контактов для переключения между аналоговыми каналами RA0 и RA1 (рекомендуется использовать внешнее прерывание).

4.4. Содержание отчета

Цель работы.

Электрическая принципиальная схема задействованной части макета.

Теоретическая часть.

Часть программы, где осуществляется конфигурация модуля АЦП и подпрограмму (обработчик прерываний если разрешены прерывания).

Часть программы, где осуществляется двоично-десятичное преобразование с подробным описанием для 8 и 10 битовых результатов.

Подробные выводы.

4.5. Контрольные вопросы

4.5.1. Периферийные особенности модуля АЦП в МК PIC16F877.

4.5.2. Разрешающая способность модуля АЦП.

4.5.3. Использование внешних опорных напряжений для управления разрешающей способностью модуля АЦП.

4.5.4. Сохранение результата аналого-цифрового преобразования. Форматирование.

4.5.5. Принцип двоично-десятичного преобразования.

4.5.6. Практическое использование модуля АЦП.

ЛАБОРАТОРНАЯ РАБОТА № 5.
МОДУЛЬ ЗАХВАТ- СРАВНЕНИЕ-ШИМ

5.1. Цель работы

Изучить функционирования модуля CCP;

Изучить функционирование модуля CCP в режиме захвата для измерения постоянной времени RC-цепочки;

Изучить функционирование модуля CCP в режиме ШИМ для изменения яркости свечения светодиода.

5.2. Теоретическая часть

Используемые части лабораторного макета

Для выполнения данной лабораторной работы потребуется использовать следующие части лабораторного макета:

— пару кнопочных контактов, подключенных ко входу внешнего прерывания;
— переменный резистор для регулирования постоянной времени RC цепочки;
— семисегментный индикатор.

5.2.1. Общие сведения о модулях CCP

Данный микроконтроллер имеет два идентичных модуля CCP (Capture - Compare - Pulse-Width-Modulation), которые могут независимо работать в трех режимах:

— захват;
— сравнение (не рассматривается в данной лабораторной работе);
— ШИМ (Широтно-импульсная модуляция).

Когда используются режимы захват и сравнение, модули CCP взаимодействуют с регистрами TMR1H и TMR1L модуля таймера TMR1. Поэтому, для использование этих режимов требуется задействовать модуль TMR1. Когда используется режим ШИМ, требуется задействовать модуль TMR2.

Для конфигурации модулей CCP используются регистры CCP1CON и CCP2CON;

Описание конфигурационных регистров приведено в Приложении Б. Ввиду топологии электрической принципиальной схемы лабораторного макета, для использования ШИМ сигнала требуется использовать модуль CCP1, а для использования функции захвата требуется использовать модуль CCP2.

5.2.2. Режим захвата

В данной лабораторной работе требуется использовать модуль CCP2 в режиме захвата для численной оценки постоянной времени RC-цепи (см. рис. 1.1). Сигнал высокого логического уровня, генерируемый на выводе RC0 проходит через задерживающую RC-цепь, поступает на вывод RC1, формируя на

этом выводе высокий логический уровень. В момент формирования высокого логического уровня на выводе RC0, регистр приращения модуля TMR1 (при этом модуль TMR1 функционирует в режиме счетчика) обнуляется для начала нового отсчета времени, через некоторое время (постоянная времени RC-цепи) сигнал высокого уровня поступит на вывод RC1, в этот момент модуль CCP2 осуществит захват значения модуля TMR1. После чего, вывод RC0 переводится в низкое логическое состояние для подготовки к следующему изменению. Захваченное значения TMR1 будет иметь значение прямо пропорциональное постоянной времени исследуемой RC-цепи.

Как упоминалось ранее, для использования модуля CCP2 в режиме захвата требуется также задействовать модуль таймер TMR1. Структурная схема модуля TMR1 представлена на рис. 5.1.

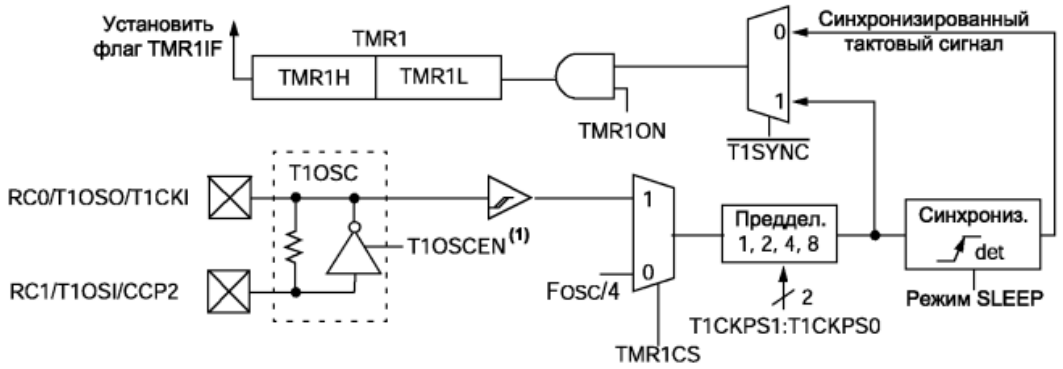

Рис. 5.1 — Структурная схема модуля TMR1

Модуль таймер TMR1 — это 16 разрядный таймер/счётчик состоящий из двух спаренных 8 разрядных регистров TMR1H и TMR1L. Для конфигурации модуля TMR1 следует использовать регистр T1CON. Для использования режима захвата, данный таймер должен быть сконфигурирован как таймер или синхронный счетчик.

Суть режима захвата заключается в том, что модуль CCP2 захватывает значения регистров TMR1H и TMR1L модуля TMR1 по заданному событию на выводе RC1 (данный вывод относится к модулю CCP2). Захваченные значения помещаются в соответствующие регистры CCPR1H и CCPR1L. Структурная схема модуля CCP2 в режиме захвата представлена на рис. 5.2.

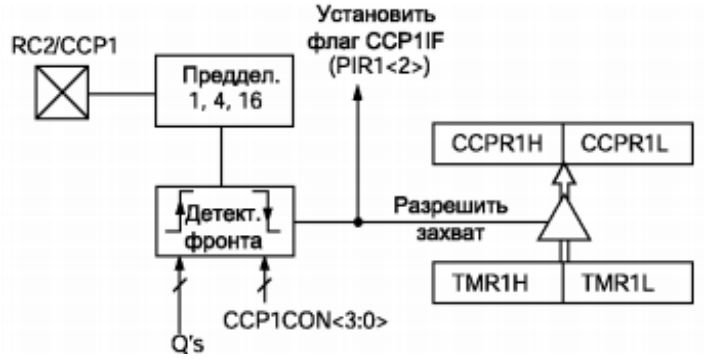

Рис. 5.2 — Структурная схема модуля CCP2 в режиме захвата

Условия событий на выводе RC1, по которым должен происходить захват, определяются программно через конфигурацию битов CCPxM3 — CCPxM0 регистра CCP2CON и могут иметь следующие опции:

— захват по каждому спадающему фронту сигнала (значения битов 0100);
— захват по каждому нарастающему фронту сигнала (0101 — значение битов необходимое для выполнения данной лабораторной работы);
— захват по каждому четвертому нарастающему фронту сигнала (0110);
— захват по каждому шестнадцатому спадающему фронту сигнала (0111).

Для конфигурации модуля CCP2 в режиме захвата, следует выполнить следующие шаги.

1) Настроить вывод RC1 на вход, установив соответствующий разряд регистра TRISC.

2) Включить модуль TMR1 (если требуется, то задается коэффициент деления).

3) Определив требуемое событие для захвата, задать необходимые значения битам CCPxM3 — CCPxM0 регистра CCP2CON.

4) Разрешить прерывания от модуля CCP2 установив бит CCP2IE.

Фрагмент программы с конфигурацией модуля CCP2 и разрешением прерывания показан на рис. 5.3.

```
b1
    CLRF    TRISC   ;все выводы порта C на выход
    bsf     TRISC,1 ;вывод для модуля CCP2 на вход
b0
    bsf     TMR1ON  ;включение модуля TMR2
    MOVLW   b'00000101' ;конфигурация
    MOVWF   CCP2CON     ;модуля CCP2 на захват
    bcf     CCP2IF      ;очистка флага захвата
b1
    bsf     CCP2IE  ;разрешение прерывания по захвату
b0
    bcf     INTF
    bsf     INTE
    bsf     GIE
```

Рис. 5.3 — Фрагмент программы для конфигурации модуля CCP2 в режиме захвата и разрешения прерывания

Для удобства можно разрешить прерывания от модуля CCP2 для считывания результатов захваченных значений. Модуль CCP2, при разрешенных прерываниях, связан с флагом CCP2IF. По каждому прерыванию от модуля CCP2 в подпрограмме прерываний рекомендуется выполнить следующие действия.

1) Сбросить вывод RC0.

2) Произвести форматирование захваченного значения в регистрах CCPR2H и CCPR2L (если требуется).

3) Осуществить двоично-десятичное преобразование для вывода результата захваченного значения на дисплей (может быть выполнено в бесконечном цикле основной программы).

4) Сбросить флаг CCP2IF.

Возможный фрагмент подпрограммы для обработки прерывания от модуля CCP2 функционирующего в режиме захвата приведен на рис. 5.4.

```
BTFSS       CCP2IF
goto    OTHER_INT1

bcf     PORTC,0

MOVF    CCPR2L
MOVLW   b'00011111'
ANDWF   CCPR2H,1

CALL    DECODING
bcf     CCP2IF

OTHER_INT1
```

Рис. 5.4 — Фрагмент подпрограммы для обработки прерывания от модуля CCP2

Для полной функциональности программы измерения постоянной времени *RC*-цепи, требуется использовать внешнее прерывание для ручного запуска измерения по замыканию или размыканию кнопочных контактов. В этот момент необходимо очищать спаренные регистры модуля TMR1H и TMR1L и формировать высокий логический уровень на выводе RC0. Затем, по прерыванию от модуля CCP2 отформатировать захваченные значения и преобразовать их в десятичный формат для вывода на дисплей.

Измерение постоянной времени может также инициироваться автоматически в основном цикле программы.

5.3.3. Режим ШИМ

ШИМ сигналы широко распространены в электронике для выполнения различных задач. В данной лабораторной работе ШИМ сигнал используется для регулирования яркости свечения светодиода, который подключен к выводу RC2. Управление этим выводом передается модулю CCP1, когда модуль используется в режиме ШИМ. Структурная схема модуля CCP1 в режиме ШИМ показана на рис. 5.5.

Функционирование модуля CCP1 в режиме ШИМ невозможно без использования модуля таймера TMR2. Структурная схема модуля таймера TMR2 показана на рис. 5.6.

Из структурной схемы видно, что при совпадении значений регистра PR2 и TMR2 выполняется сравнение и генерируется сигнал EQ, который используется модулем CCP1 для формирования периода ШИМ сигнала. Для того, чтобы задать требуемую частоту ШИМ сигнала, необходимо загрузить необходимое значение в регистр PR2.

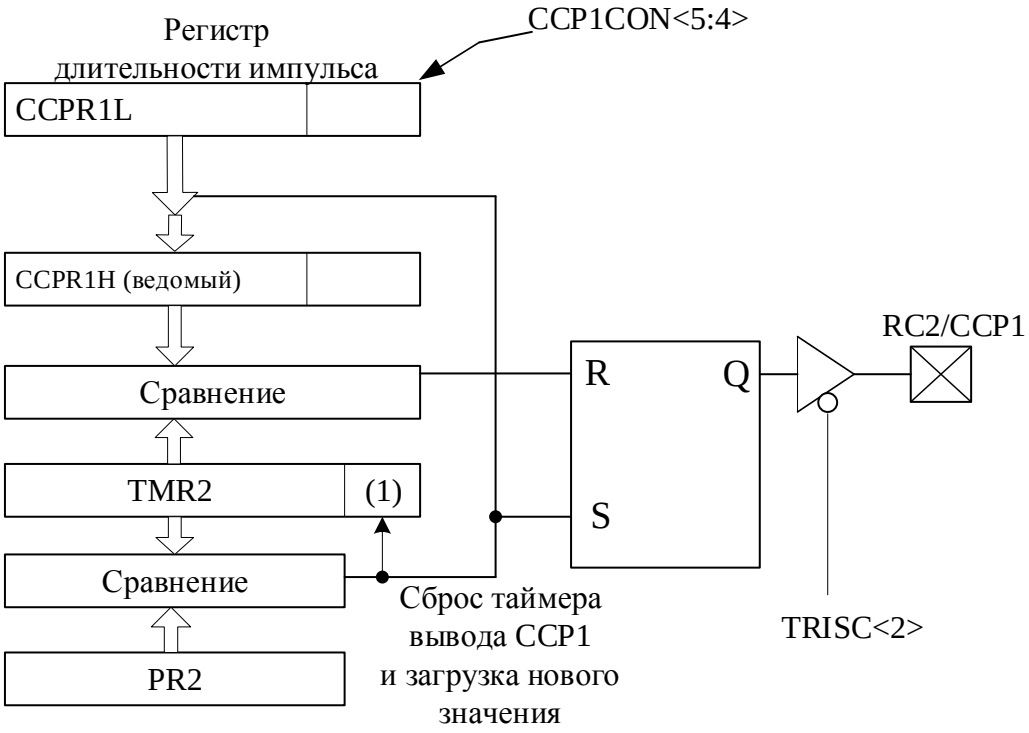

Рис. 5.5 — Структурная схема модуля CCP1 в режиме ШИМ

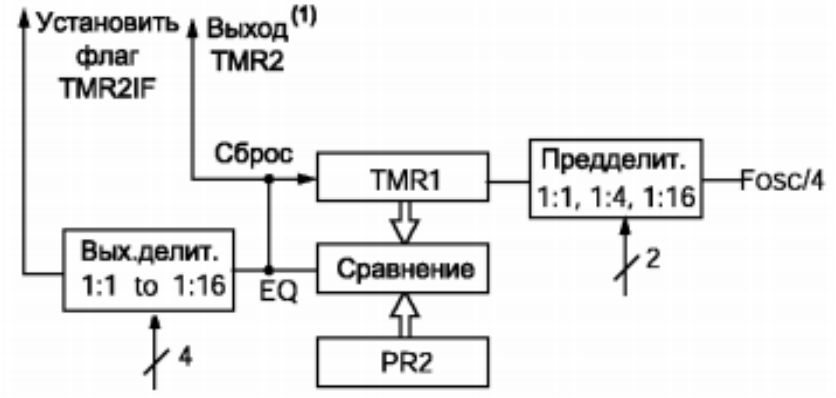

Рис. 5.6 — Структурная схема модуля TMR2

Для того, чтобы рассчитать частоту ШИМ сигнала необходимо воспользоваться следующей формулой:

$$PWM_{\text{period}} = [PR2 + 1]4T_{osc}TMR2_{prescaler\_value}.$$

Также важным элементом ШИМ сигнала является длительность импульсов. Для управления длительностью импульсов необходимо загружать 8 разрядов регистра CCPR1L и два младших 5 и 4 разряды регистра CCP1CON. Таким образом, получается, что длительность импульсов ШИМ сигнала имеет разрешение в 10 бит. Но при увеличении частоты следования импульсов ШИМ сигнала, разрешение по длительности уменьшается. Расчет разрешения по длительности может быть осуществлен по следующей формуле:

$$PWM_{resolution} = \frac{\log\left(\frac{F_{OSC}}{F_{PWM}}\right)}{\log\ 2}.$$

Для конфигурации модуля CCP1 в режиме ШИМ требуется выполнить следующие шаги.

1) Настройте вывод RC2 на выход.

2) Задайте требуемую частоту ШИМ сигнала загрузив регистр PR2.

3) Включите таймер TMR2, установив бит TMR2ON регистра T2CON.

3) Сконфигурировать модуль CCP1 в режиме ШИМ, установкой битов регистра CCP1CON.

4) Загружая значение в регистр CCPR1L с 4 и 5 битами регистра CCP1CON задается длительность импульсов ШИМ сигнала.

Пример конфигурации модуля CCP1 в режиме ШИМ показан на рис. 5.7.

```
b1
    CLRF    TRISC   ;
    bsf     TRISC,2 ;настройка на выход

    MOVLW   0xFF
    MOVWF   PR2 ;установка частоты ШИМ
b0
    CLRF    T2CON
    bsf     TMR2ON  ;включение таймера 2

    MOVLW   b'00111100' ;PWM set up + 2 LSb are set
    MOVWF   CCP1CON     ;включение модуля CCP

    MOVLW   .20         ;установка начальной длительности
    MOVWF   CCPR1L  ; Duty cycle with 8 Msb
```

5.7 — Конфигурация модуля CCP1 в режиме ШИМ

Далее, для управления длительностью импульсов ШИМ сигнала требуется изменять значение регистра CCPR1L с 5 и 4 битами регистра CCP1CON.

5.3. Порядок выполнения лабораторной работы

1) Реализуйте программу для измерения постоянной времени RC цепи с использованием модуля CCP2. Результат измерения длительности требуется отображать на индикаторе. Требуется реализовать ручной и автоматический режимы измерения постоянной времени.

2) Реализуйте программу для управления яркостью светодиода используя модуль CCP1. В программе требуется реализовать возможность оперативного управления длительностью импульсов (для изменения яркости) ШИМ сигнала с помощью кнопочных контактов или модуля АЦП в качестве устройства ввода. Значение длительности требуется отображать на светодиодном индикаторе.

5.4. Содержание отчета

Цель работы.

Схему задействованной части макета и объяснить, как реализована цепь задержки сигнала.

Часть кода, где осуществляется ШИМ регулирование и вывод длительности импульсов ШИМ сигнала на индикатор.

Часть кода, где осуществляется захват и двоично-десятичное преобразование захваченных значений (раздельные части кода для автоматического и мануального запуска измерения).

Подробные выводы.

5.5. Контрольные вопросы

5.5.1. Функционирование модуля CCP в режиме захвата. Примеры применения режима захвата.

5.5.2. Функционирования модуля CCP в режиме ШИМ. Параметры генерации ШИМ сигнала.

5.5.3. Периферийные модули, использующиеся совместно с модулем CCP.

5.5.4. Последовательность действий для настройки модуля CCP в режим ШИМ и захвата.

ЛАБОРАТОРНАЯ РАБОТА № 6. ЧАСЫ РЕАЛЬНОГО ВРЕМЕНИ DS1307 И ОБМЕНА ДАННЫМИ ПО ШИНЕ I2C

6.1. Цель работы

Ознакомиться с основными положениями I2C протокола.

Изучить работу модуля последовательной синхронной передачи (MSSP) в режиме ведущего устройства для PIC16F877.

Изучить аппаратное устройство интегральных часов реального времени DS1307.

Реализовать программу для отображения реального времени на семисегментном индикаторе используя I2C протокол (MSSP модуль) для опроса DS1307.

6.2. Теоретическая часть

Описание лабораторного макета

Для выполнения данной лабораторной работы будут задействованы следующие части макета:

— часы реального времени DS1307;

— семисегментный индикатор;

— кнопочные контакты.

6.2.1. Основные положения I2C протокола

Протокол передачи данных I2C (Inter Integrated Circuit) — это протокол последовательной передачи данных, позволяющий организовать компактную двухпроводную шину для двунаправленного обмена данных между цифровыми устройствами на относительно коротком расстоянии, где, по крайней мере, одно из цифровых устройств обязательно является ведущим (далее, ведущий), а остальные выступают в качестве ведомых (далее, ведомый) [2, 3].

I2C шина представляет собой две линии (проводов) именуемы как линия данных SDA (Serial Data) и линия синхронизации SCL (Serial Clock).

Поскольку данный протокол обеспечивает двусторонний обмен данных, то прямая и обратная передачи данных осуществляются по линии SDA, а синхронизация передаваемых данных по линии SCL.

I2C шина поддерживает несколько режимов передачи данных по скоростям (может варьироваться для разных устройств, определяется аппаратной поддержкой):

— высокоскоростная передача данных 3,4 Мбит/с;

— скоростной режим передачи данных от 400 кбит/с до 1 Мбит/с;

— стандартная скорость 100 кбит/с;

— низкоскоростная передача данных 10 кбит/с.

Устройства, поддерживающие I2C протокол, соединяются с линиями SDA и SCL параллельно как показано на рис. 6.1. Поскольку все устройства соединены с шиной через выводы с открытыми стоками (коллекторами), то необходимо использовать подтягивающие резисторы (3,3 — 5,1 кОм), для обеспечения высокого логического уровня в режиме простоя (присутствие высокого логического уровня на линиях показывает, что все устройства на шине находятся в режиме ожидания и шина свободна).

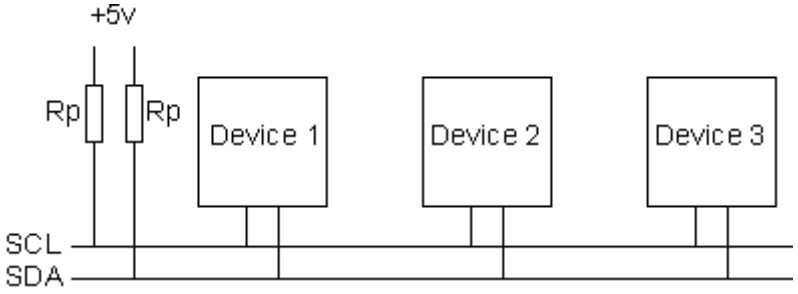

Рис. 6.1 — Соединение устройств с шиной I2C

Для информационного обмена между устройствами на I2C шине используются следующие сигнальные последовательности, именуемые как:

— условие старта Start (далее сигнал начала сеанса передачи);
— условие повторного старта Repeated Start (далее сигнал повторного страта);
— сигнал подтверждения и неподтверждения Acknowledgment, not acknowledgment (далее сигнал подтверждения);
— условие окончания обмена данными STOP с последующим переводом шины в неактивное состояние (далее сигнал окончания сеанса передачи);
— посылка адреса для выбора ведомого устройства на шине;
— передача данных ведомому;
— прием данных от ведомого.

Направления обмена данными между устройствами можно разделить на следующие:

— ведущий обращается к ведомому для передачи данных (рис. 6.2);
— ведущий обращается к ведомому для приема данных (рис. 6.3).

Любая транзакция приема или передачи данных обязательно начинается с формирования сигнала старта с последующей передачей 7 битов адреса ведомого (в данной лабораторной работе не рассматривается адресация ведомых с 10 битовым адресом). Если ведомый с переданным адресом присутствует на шине, тогда посылается сигнал подтверждения. Согласно I2C протоколу, все устройства на шине, которые принимают данные должны генерировать сигнал подтверждения или неподтверждения (только для ведущего в режиме приема). В свою очередь сигнал подтверждения позволяет передатчику (передатчиком может быть, как ведомый, так и ведущий) начать следующую передачу данных.

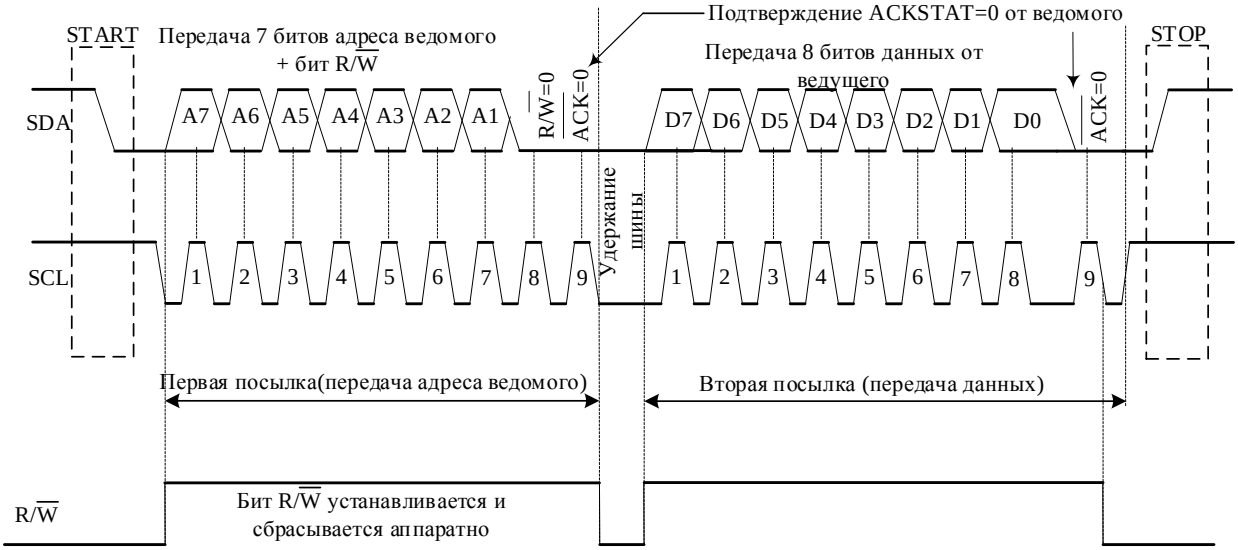

Рис. 6.2 — Временная диаграмма сигналов на I2C шине при обращении ведущего к ведомому для передачи данных

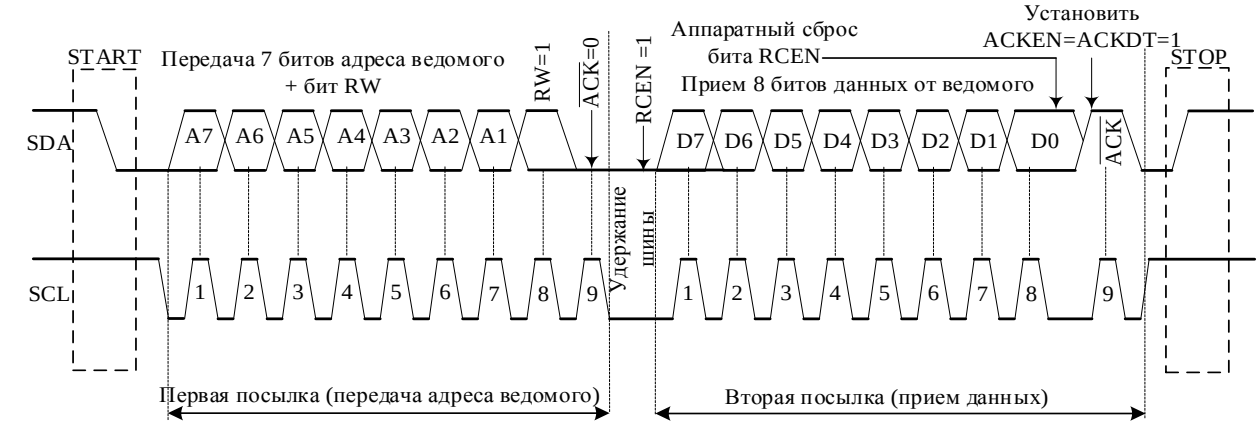

Рис. 6.3 — Временная диаграмма сигналов на шине при обращении ведущего к ведомому для приема данных

Для завершение сеанса передачи или приема данных ведущий должен сгенерировать сигнал окончания сеанса сигналом STOP.

6.2.2. Функционирование модуля MSSP в режиме ведущего

MSSP модуль (Master Synchronous Serial Port — модуль ведущего синхронного последовательного порта) данного микроконтроллера может быть сконфигурирован для работы с SPI или I2C шинами по выбору разработчика. Модуль MSSP также может быть сконфигурирован как ведущее или как ведомое устройство. Модуль MSSP данного микроконтроллера в режиме I2C поддерживает следующие режимы:

— ведомый с 7 битовым адресом;
— ведомый с 10 битовым адресом;
— ведущий с задаваемой частотой тактового сигнала ($F_{osc}/4(\text{SSPADD} + 1)$);

— программная поддержка ведущего режима (используется для совместимости с другими устройствами среднего класса).

Ведущее устройство не имеет собственного адреса, в то время как все ведомые имеют. В этой связи, ведущий должен иметь заранее запрограммированные адреса ведомых для корректной адресации.

Для конфигурации и управления модулем MSSP используются следующие регистры:

— конфигурационный регистр SSPOCN;
— конфигурационный регистр SSPCON2;
— статусный регистр SSPSTAT;
— буферный регистр последовательного приема/передачи SSPBUF;
— регистр сдвига SSPSR (не доступен из программы);
— регистр адреса SSPADD в режиме ведомого (в режиме ведущего используется для определения тактовой частоты на линии SCL).

Для выполнения настоящей лабораторной работы потребуется использовать модуль MSSP сконфигурированного в режиме ведущего (рис. 6.4).

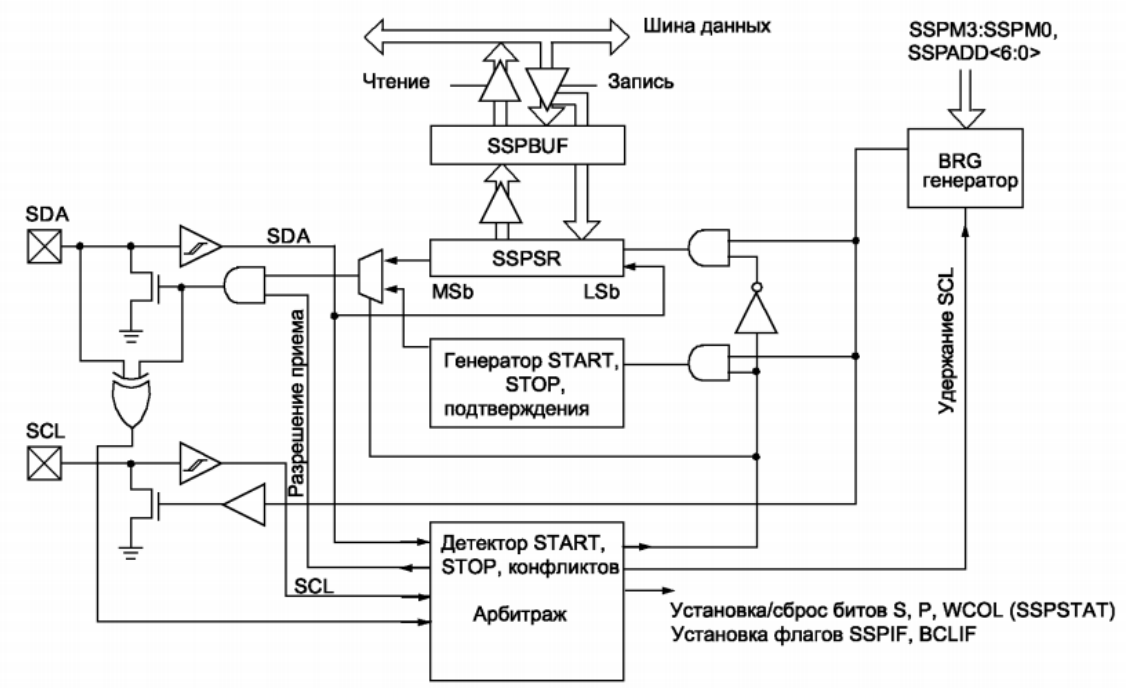

Рис. 6.4 — Структурная схема модуля MSSP в режиме ведущего

Для подключения к I2C шине, данный микроконтроллер использует два вывода порта C: RC3 для соединения с SCL и RC4 для соединения с линией SDA. При включении модуля MSSP использование этих выводов в общем режиме невозможно, так как управление этими выводами передается модулю MSSP.

Для того, чтобы сконфигурировать модуль MSSP в режиме ведущего требуется выполнить следующие шаги:

1) Выводы RC3 и RC4 настроить на вход установив соответствующие биты регистра TRISC.

2) Установить бит SMP регистра SSPSTAT для адаптации скорости нарастания сигнала для 100 кГц и 1 МГц или сбросить для скорости 10 кГц.

3) Загрузить значение в регистр SSPADD для определения тактовой частоты на шине.

4) Установить бит SSPEN регистра SSPCON для включения модуля MSSP и переключения выводов RC3 и RC4 в режимы SCL и SDA соответственно.

5) Задать режим работы модулю MSSP, задав требуемые значения битам SSPM3—SSPM0 (в режиме ведущего следует установить только бит SSPM3).

Фрагмент программы для инициализации последовательно синхронного порта показан на рис. 6.5.

```
b1
bsf     TRISC,3     ; подготовка выводов
bsf     TRISC,4     ;для I2C интерфейса
bsf     SMP         ;стандартный режим скорости

MOVLW   .5          ;определяем
MOVWF   SSPADD      ;тактовую частоту ведущего

b0
bsf SSPM3           ;включаем режим ведущего
BSF     SSPEN       ;включение послед порта
```

Рис. 6.5 — Фрагмент программы для конфигурации модуля MSSP в режиме ведущего

После выполнения шагов конфигурации, модуль MSSP готов к передаче или приему данных. Как упоминалось ранее, передача данных может осуществляться от ведомого к ведущему устройству и обратно. Прием и передача данных рассмотрены далее.

Для передачи данных от ведущего к ведомому (см. рис. 6.2) требуется сформировать следующие сигнальные последовательности на шине.

1) Сгенерировать сигнал STATR установив бит SEN регистра SSPCON2.

2) Через некоторое время, когда сигнал старта сформировался, бит SEN аппаратно сбрасывается, после чего 7 битов адреса ведомого устройства с нулевым значением бита запись/чтение R/W (см. рис. 6.2) следует загрузить в буферный регистр SSPBUF, при этом после записи регистра SSPBUF, бит R/W регистра SSPSTAT установится в «1», а после восьмого сдвига содержимого регистра SSPBUF, этот бит аппаратно сбросится.

3) По завершению передачи адреса, в случае присутствия ведомого устройства с соответствующим адресом, ведомый отправит сигнал подтверждение ACK (сигнал подтверждения имеет инверсное значение) ведущему, по девятому импульсу синхронизации.

4) После приема сигнала подтверждения бит R/W регистра SSPSTAT аппаратно сбросится, после чего ведущий может осуществить передачу данных, загрузив 8 битов в регистр SSPBUF.

5) По завершению приема данных, ведомый ответит сигналом подтверждения по девятому импульсу синхросигнала, после чего ведущий сможет осуществить передачу следующих 8 битов или завершить сеанс передачи данных сформировав сигнал STOP.

Генерация сигнала STOP осуществляется установкой бита PEN регистра SSPCON и ожиданием аппаратного сброса этого бита.

Фрагмент подпрограммы передачи данных b‘00000000’ ведомому с адресом b‘1101000’ показан на рис. 6.6.

```
I2C_WRITE
    Call    START_I2C   ;начало сеанса, сигнал START
    MOVLW   b'11010000' ;загрузка адреса ведомого + нулевой бит записи
    Call    SEND_I2C    ;ожидание отправки адреса
    MOVLW   b'00000000' ;загрузка данных
    Call    SEND_I2C    ;ожидание отсылки данных
    Call    STOP_I2C;завершение сеанса, сигнал STOP
    return
```

Рис. 6.6 — Фрагмент подпрограммы для передачи данных ведомому от ведущего

Для генерации сигнальных последовательностей на шине, в подпрограмме (см. рис. 6.5) использовались подпрограммы START_I2C, SEND_I2C и STOP_I2C, реализация которых приведена на рис. 6.7.

```
START_I2C
    b1
    BSF     SEN
    btfsc   SEN
    goto    $-1
    b0
    return
SEND_I2C
    movwf   SSPBUF
    b1
    btfsc   R_W
    goto    $-1
    b0
    return
STOP_I2C
    b1
    BSF     PEN
    btfsc   PEN
    goto    $-1
    b0
    return
```

Рис. 6.7 — Подпрограммы формирования сигналов START, STOP и ожидания передачи данных SEND_I2C

Для приема данных от ведомого, ведущему необходимо сформировать следующие сигнальные последовательности на шине.

1) Сгенерируйте сигнал STATR установив бит SEN регистра SSPCON2.

2) Через некоторое время, когда сигнал старта сформировался, бит SEN аппаратно сбрасывается, после чего 7 битов адреса ведомого устройства и бит запись/чтение R/W со значением «1» (см. рис. 6.3) следует загрузить в буферный регистр SSPBUF, при этом после записи регистра SSPBUF, бит R/W регистра SSPSTAT установится в «1», а после отсылки 8 битов, этот бит сбросится аппаратно;

3) По завершению передачи адреса, в случае присутствия ведомого устройства с соответствующим адресом, ведомый отправит подтверждение ACK (сигнал подтверждения имеет инверсное значение) ведущему, по девятому импульсу синхросигнала.

4) После приема подтверждения бит R/W регистра SSPSTAT аппаратно сбросится, после чего можно осуществить прием данных, для этого ведущий должен переключиться в режим приемника установкой бита RCEN регистра SSPCON2, после чего осуществляется прием 8 битов, и их загрузка в буферный регистр SSPBUF, по завершению приема, бит RCEN будет автоматически сброшен, после чего можно осуществить считывание данных из регистра SSPBUF.

5) Если требуется принять следующие 8 битов данных, то ведущий формирует подтверждение, для этого требуется сбросить бит ACKDT (если бит сброшен, тогда отсылается подтверждение, а если установлен, тогда отсылается неподтверждение) и установить бит ACKEN (для разрешения отсылки сигнала подтверждения) в регистре SSPCON2, для неразрывного приема данных следует повторно устанавливать бит RCEN, считать буферный регистр SSPBUF и отослать сигнал подтверждения.

6) В случае, если требуется завершить сеанс приема, то ведущий формирует сигнал неподтверждения, для этого требуется сбросить бит ACKDT, установить бит ACKEN и сгенерировать сигнала STOP.

Пример подпрограммы, позволяющий осуществить прием данных от ведомого, приведен на рис. 6.8.

Подпрограммы REC_I2C — receive mode, ACKNOL — acknowledge signal, ACKNOL_NOT — not acknowledge signal, использованные в подпрограмме для приема байтов данных от ведомого показаны на рис. 6.9.

В некоторых случаях для считывания памяти ведомого устройства требуется предварительно задать адрес указателю памяти ведомого устройства. Для этого необходимо выполнить следующую последовательность.

1) Ведущий должен сформировать сигнал START.

2) Затем ведущий передает адрес ведомого с нулевым значением бита записи/чтения R/W.

3) После получения сигнала подтверждения от ведомого, ведущий должен передать 8 битов данных для указателя адреса регистра памяти данных ведомого, с которого надо начать считывание.

```
DREAD
    CALL    START_I2C   ;сигнал начала
    MOVLW   b'11010001' ;загрузка и передача
    call    SEND_I2C    ;7 битов адреса с битом приема "1"
    call    REC_I2C     ;переключение ведущего в режим приемника
    MOVF    SSPBUF,W    ;сохранение принятого байта в W
    MOVWF   FIRST_BYTE  ;запись принятого байта в регистр
    call    ACKNOL_YES  ;высылвем "подтверждение" для солед прие
    call    REC_I2C
    MOVF    SSPBUF,W
    MOVWF   SECOND_BYTE
;;;продолжаем прием байтов
;;;от ведомого, окончание према,
;;;должно быть обозначено "не подтверждением"
    call    ACKNOL_NOT
    CALL    STOP_I2C
    return
```

Рис. 6.8 — Подпрограмма для приема байтов данных от ведомого

```
REC_I2C
    b1
    BSF     RCEN    ;включение ведущего в режим приемника
    btfsc   RCEN    ;ожидание приема байта
    goto    $-1
    b0
    return
ACKNOL_YES
    b1
    bcf     ACKDT   ;сформировать сигнал подтверждения
    bsf     ACKEN   ;сформировать сигнал на шине
    btfsc   ACKEN
    goto    $-1
    b0
    return
ACKNOL_NOT
    b1
    BSF     ACKDT   ;сформировать сигнал не подтверждения
    BSF     ACKEN   ;сформировать сигнал на шине
    btfsc   ACKEN
    goto    $-1
    b0
    return
```

Рис. 6.9 — Описание подпрограмм для приема данных

4) После получения сигнала подтверждения, ведущему требуется сформировать сигнал повторного старта.

5) Далее, ведущий должен передать адрес ведомого со значением бита записи/чтения R/W равным единице.

6) Далее, ведущий осуществляет прием данных от ведомого в стандартном режиме (см. рис. 6.8).

Подпрограмма для формирования сигнала повторного старта показана на рис. 6.10.

```
ReSTART_I2C
    b1
    BSF     RSEN
    btfsc   RSEN
    goto    $-1
    b0
    return
```

Рис. 6.10 — Подпрограмма для формирования сигнала повторного страта

6.3.3. Аппаратная организация часов реального времени DS1307

Часы реального времени DS1307 представляют собой двоично-десятичные часы и календарь. Передача данных между часами и опрашивающим устройством осуществляется по I2C шине, где DS1307 всегда выступает как ведомое устройство и поэтому имеет фиксированный 7 битовый адрес (1101000) запрограммированный заводом изготовителем. Часы/календарь ведут счет секунд, минут, часов, дней, недель, месяцев и годов. При этом осуществляется автоматическая коррекция количества дней в месяцах имеющих менее 31 дня, включая поправку для високосного года [4].

DS1307 имеет независимый внешний источник питания и генератор тактовой частоты, что позволяет продолжать счет времени даже при отсутствии основного питания.

DS1307 содержит 7 буферных регистров данных и 56 регистров общего назначения (рис. 6.11). Календарно-часовой модуль ведет счет времени и соответственно инкрементирует значения секунд, минут, часов и т.д., после чего значения заносятся в 7 буферных регистров. Информация, содержащаяся в буферных регистрах, может быть прочитана или перезаписана по I2C шине. При этом если произвести перезапись какого-либо из буферных регистров, то новое записанное значение будет инкрементироваться в нормальном порядке счета. Таким образом, удобно осуществлять корректировку или установку времени путем записи требуемых значений.

Для чтения и записи регистров, данных представленных на карте регистров памяти данных (см. рис. 6.5) следует формировать следующие сигнальные последовательности на I2C шине.

Для записи памяти данных DS1307 следует сформировать сигнальные последовательности, как показано на рис. 6.12.

ADDRESS	BIT 7	BIT 6	BIT 5	BIT 4	BIT 3	BIT 2	BIT 1	BIT 0	FUNCTION	RANGE
00h	CH	10 Seconds			Seconds				Seconds	00–59
01h	0	10 Minutes			Minutes				Minutes	00–59
02h	0	12 24	10 Hour PM/ AM	10 Hour	Hours				Hours	1–12 +AM/PM 00–23
03h	0	0	0	0	0	DAY			Day	01–07
04h	0	0	10 Date		Date				Date	01–31
05h	0	0	0	10 Month	Month				Month	01–12
06h	10 Year				Year				Year	00–99
07h	OUT	0	0	SQWE	0	0	RS1	RS0	Control	—
08h–3Fh									RAM 56 x 8	00h–FFh

Рис. 6.11 — Карта регистров памяти данных DS1307

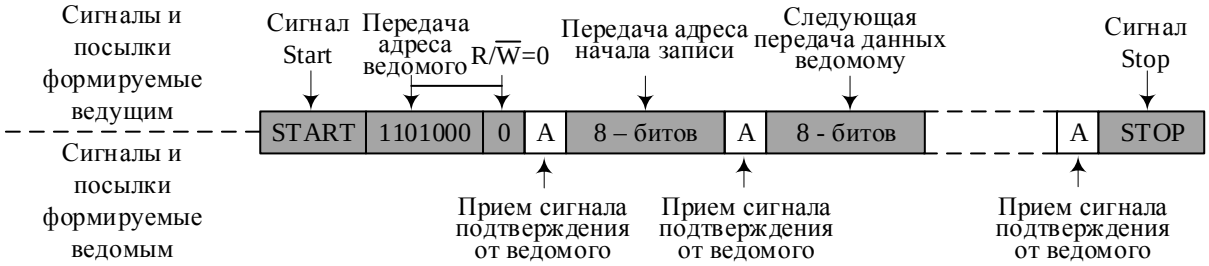

Рис. 6.12 — Запись в память данных DS1307

Для начала сеанса записи данных, ведущий должен сформировать сигнал START и передать 7 битовый адрес с нулевым битом R/W и получить сигнал подтверждения от ведомого. После чего, ведущий передает адрес начала чтения и затем передает байты данных столько раз сколько потребуется, при этом все регистры данных DS1307 будут записываться последовательно, так как после каждого принятого байта, указатель адреса автоматически инкрементируется. Если указатель не был определен ранее, то он начнет инкрементироваться с нулевого значения или с последнего значения, на котором завершился предыдущий сеанс записи или чтения. После каждого переданного байта, ведомый должен отсылать сигнал подтверждения, который сообщает ведущему, что данные приняты и ожидается следующая транзакции данных. Для окончания сеанса записи, ведущий должен сгенерировать на шине сигнал STOP, после чего шина перейдет в неактивное состояние.

Для чтения памяти данных DS1307 следует сформировать сигнальные последовательности, как показано на рис. 6.13.

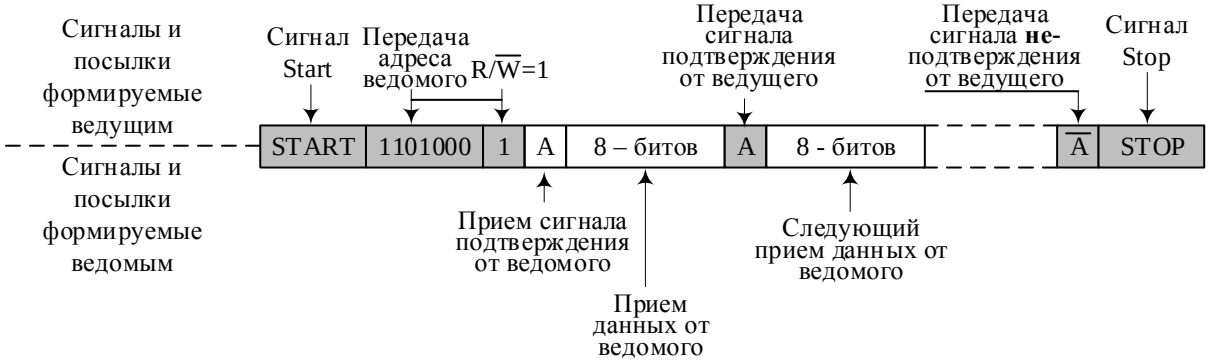

Рис. 6.13 — Чтение памяти данных DS1307

Для начала сеанса чтения памяти данных, ведущий должен сформировать сигнал START и передать адрес ведомого с битом R/W имеющим значение «1». Далее, ведомый передает сигнал подтверждения, после чего ведомый начинает передачу данных. После каждого принятого байта, ведущему надо генерировать сигнал подтверждения, для приема следующего байта данных. Процесс приема данных от ведомого может продолжаться до тех пор, пока ведущий не сгенерирует сигнал неподтверждения с последующим сигналом STOP. Следует учитывать, что с каждым передаваемым байтом данных, указатель адреса DS1307 автоматически инкрементируется.

Для того, чтобы считывать данные только из требуемого регистра, требуется предварительно передать значение для указателя адреса, такая последовательность называется «запись указателя, затем чтение» (рис. 6.14).

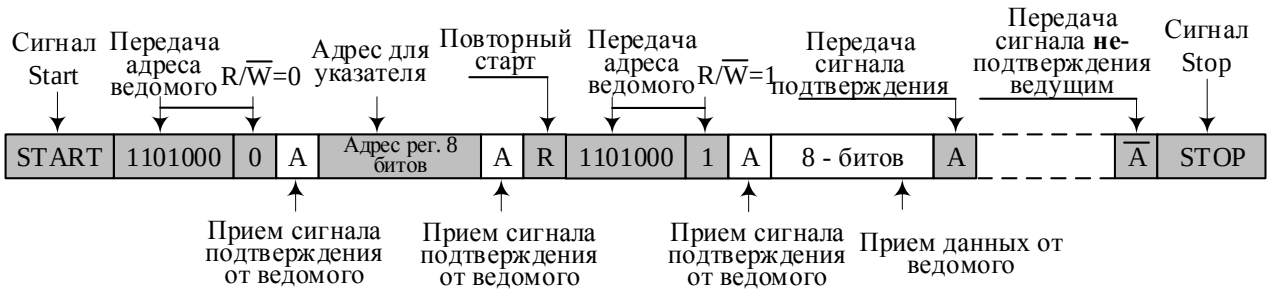

Рис. 6.14 — Последовательность сигналов для чтения памяти данных с загрузкой значения указателя адреса

Для начала сеанса чтения памяти DS1307 с загрузкой значения указателя адреса памяти данных, ведущий должен сгенерировать сигнал START, затем передать адрес ведомого устройства с нулевым значением для бита R/W. В случае совпадения адреса, ведомый передает сигнал подтверждения, после чего, ведущий должен передать 8 битов данных для адреса указателя и ожидать очередного сигнала подтверждения от ведомого. Далее, ведущему требуется сгенерировать сигнал повторного старта и передать адрес ведомого с битом R/W имеющим значение «1», затем ожидать подтверждения от ведомого. После чего, ведомый начнет передавать данные, начиная с адреса, который был загружен в указатель. По завершению приема данных, ведущий должен сгенерировать сигнал неподтверждения и сигнал STOP.

Важно обратить внимание, на то, что, по умолчанию, тактирование DS1307 от внешнего генератора выключено, поэтому для включения тактового сигнала, требуется сбросить самый старший разряд CH в регистре памяти данных по адресу 0x00, после чего начнется инкрементирование регистров, содержащих календарную и часовую информацию.

6.3. Порядок выполнения лабораторной работы

Написать программу, которая будет опрашивать DS1307, используя I2C протокол при помощи модуля MSSP. При этом модуль MSSP должен работать в режиме ведущего. Для конфигурации модуля и опроса DS1307 необходимо использовать описание и фрагменты программ в теоретической части. После

чего необходимо запрограммировать микроконтроллер и убедится, что программа выполняется правильно. На данном этапе на дисплее должны отображаться секунды и ежесекундно мигающая десятичная точка во 2 разряде.

После выполнения предыдущего задания, требуется доработать программу таким образом, чтобы, на дисплее отображались часы и минуты с ежесекундно мигающей точкой.

После успешного выполнения двух предыдущих пунктов, требуется доработать программу таким образом, чтобы можно было произвести корректировку времени, используя кнопочную клавиатуру.

6.4. Содержание отчета

Цель работы.

Схема задействованной части макета.

Фрагмент кода с реализацией настройки часов при помощи кнопочных контактов.

Фрагмент кода с извлечением минут и часов из принятых байтов от DS1307.

Подробные выводы

6.5. Контрольные вопросы

6.5.1. Протокол I2C. Основные принципы.

6.5.2. Организация шины I2C.

6.5.3. Максимальное количество устройств на шине I2C. Режимы адресации.

6.5.4. Конфигурация модуля MSSP. Режим ведущего устройства и выводы, использующиеся для последовательно интерфейса.

6.5.5. Биты формирования сигналов START и STOP на I2C шине.

ЛАБОРАТОРНАЯ РАБОТА № 7. ОБМЕН ДАННЫМИ ПО 1-WIRE ШИНЕ С ЦИФРОВЫМ ТЕМПЕРАТУРНЫМ ДАТЧИКОМ DS18B20

7.1. Цель работы

Ознакомиться с принципом работы коммуникационного протокола 1-Wire.

Изучить особенности работы цифрового датчика температуры DS18b20.

Выполнить программную реализацию протокола 1-Wire для измерения температуры окружающей среды.

7.2. Теоретические сведения

На практике при разработке радиоэлектронных устройств и систем часто возникает необходимость измерения температуры. Для решения этой задачи существует ряд способов. Самые распространенные из них это применение термопары, терморезисторов и цифровых датчиков.

При реализации МК системы использование термопар и терморезисторов требует усложнения схемы. Например, сигнал термопары имеет достаточно низкий уровень и его нужно усиливать. В свою очередь терморезисторы обладают нелинейной характеристикой, что усложняет процесс считывания показаний.

Цифровые термодатчики лишены указанных выше недостатков, не требуют калибровки, подключаются посредством простых и стандартных интерфейсов. Именно поэтому их использование предпочтительно в МК системах.

7.2.1. Основные положения температурного датчика DS18B20

В данной работе исследуется термодатчик DS18B20. К его достоинствам можно отнести простоту, доступность и достаточно высокую точность измерения температуры, а также возможность подключения множества датчиков на одну линию [5]. Внешний вид DS18B20 показан на рис. 7.1. Схема подключения показана на общей схеме макета.

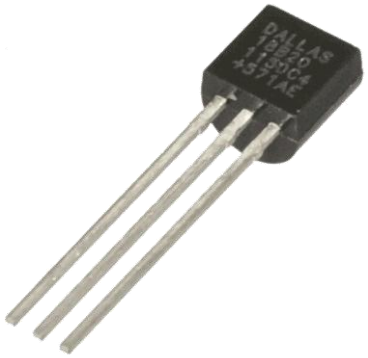

Рис. 7.1 — Внешний вид используемого термодатчика

Для обмена информацией между МК и термодатчиком используется последовательный 1-Wire интерфейс. Важно отметить, что процесс передачи данных контролируется МК, при этом используется временная синхронизация.

7.2.2. Основы работы 1-Wire шины

Рассмотрим, как передать логические уровни согласно протоколу 1-Wire. При этом следует понимать, что высокий логический уровень на линии автоматически формируется подтягивающим резистором, а низкий — одним из устройств, подключенных к линии.

На рис. 7.2 показано, как передать бит с низким логическим уровнем. Для этого МК должен установить на линии низкий логический уровень на интервал времени длиной от 60 мкс до 120 мкс, а после этого вернуть линию в исходное состояние (холостой ход).

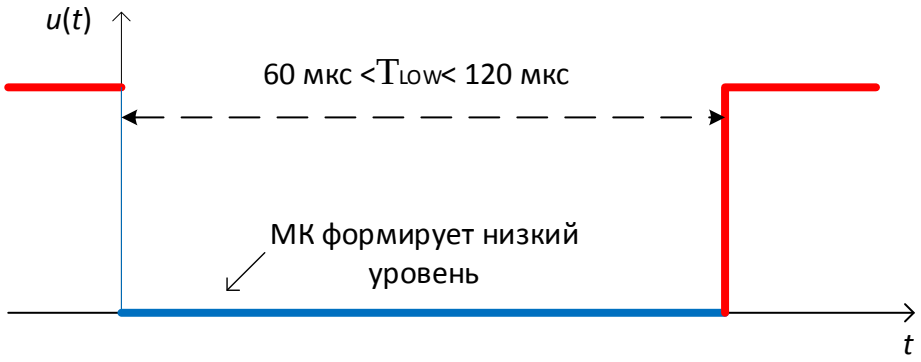

Рис. 7.2 — Тайм-слот для передачи «0» от МК

Для передачи бита с высоким логическим уровнем следует установить на линии низкий логический уровень на интервал времени более 1 мкс, после чего вернуть линию в исходное состояние. Сказанное проиллюстрировано на рис.7.3.

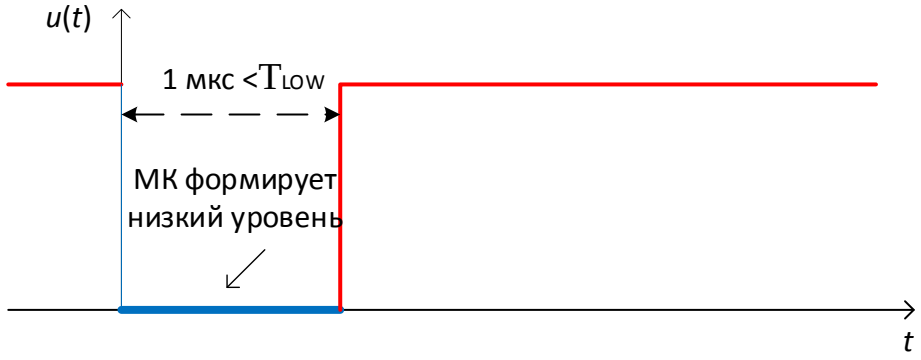

Рис.7.3 — Тайм-слот для передачи «1» от МК

Процесс передачи данных от датчика также контролируется МК. Прием каждого бита начинается с передачи ведущим устройством тайм-слота чтения. Рассмотрим, каким образом формируется тайм-слот чтения. МК должен установить на линии низкий логический уровень на интервал времени длиной от 1 мкс до 15 мкс, а после этого вернуть линию в исходное состояние, для того,

чтобы считать какой логический уровень присутствует на линии. При этом последующее состояние линии задает DS18B20. Если передаваемый от датчика бит «1», то на линии будет установлен высокий логический уровень, а если на линии установлен низкий логический уровень, то передается бит равный «0». Указанная процедура показана на рис. 7.4.

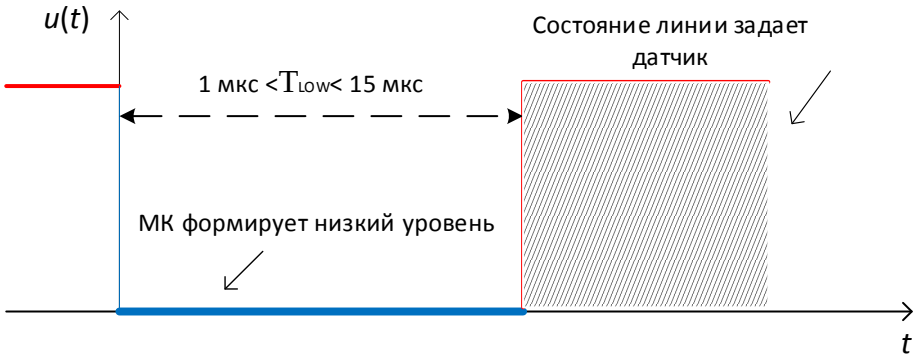

Рис.7.4 — Тайм-слот для передачи данных от датчика к МК

Важно понимать, что для правильной работы с термодатчиком необходимо строго соблюдать указанные временные интервалы каждого тайм-слота.

7.2.3. Порядок работы с DS18b20

Для обмена данными с термодатчиком следует придерживаться определенного порядка действий. Упрощенный алгоритм взаимодействия с датчиком, достаточный для выполнения данной лабораторной работы, показан на рис. 7.5.

Рис.7.5 — Алгоритм взаимодействия с датчиком для считывания температуры

Как видно из приведенного алгоритма, любое обращение к DS18B20 должно начинаться с сигнала инициализации. Рассмотрим, как передать такой сигнал (рис. 7.6.), для этого МК должен установить на линии низкий логический уровень на интервал времени длиной от 480 мкс, после чего вернуть линию в исходное состояние. Через 15—60 мкс, после того как МК возвращает

линию в состояние холостого хода, DS18B20 формирует сигнал присутствия, означающий что датчик исправен и готов к работе. Сигнал присутствия показан на рис. 7.6.

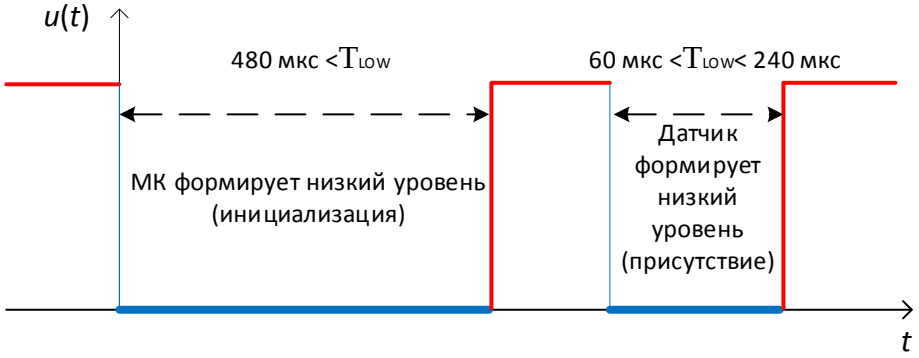

Рис.7.6 — Сигнал инициализации и сигнал присутствия

После того как получен сигнал присутствия от ведомого устройства можно передавать команды ведущим устройством. Здесь важно отметить, что существует возможность подключения множества DS18B20 на одну общую линию. Для независимого взаимодействия с каждым ведомым устройством все они имеют свой уникальный 64 битный код (ROM). В данной работе исследуется только один термодатчик, поэтому задача его идентификации не стоит.

Для того чтобы пропустить операции, связанные с ROM существует команда SkipROM, которую следует передать после инициализации.

7.2.4. Структура памяти DS18B20

Термодатчик DS18B20 имеет ОЗУ объемом 9 байт и ПЗУ объемом 2 байта. Карта памяти DS18B20 показана на рис. 7.7.

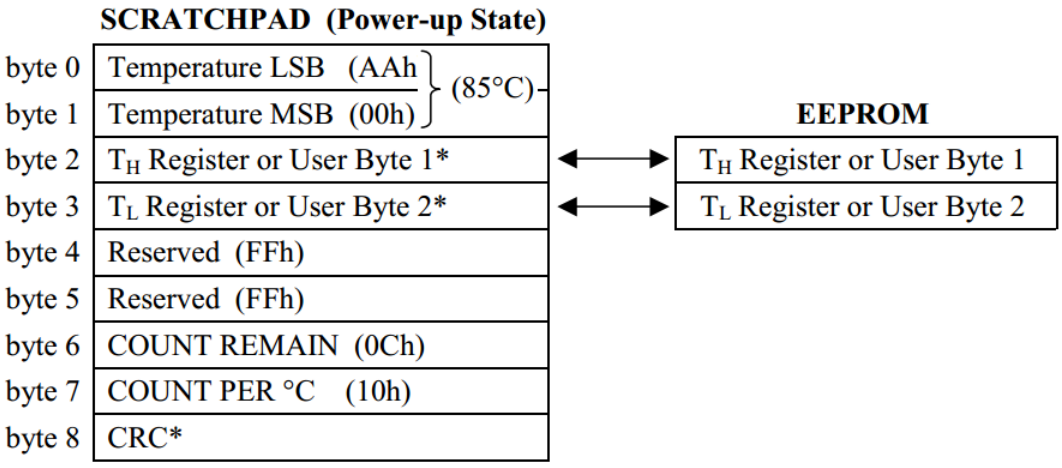

Рис.7.7 — Структура памяти DS18B20

Результаты измерения температуры хранятся в регистрах ОЗУ с адресами 0 и 1, при этом содержимое остальной памяти в ходе данной работы не потребуется.

Рассмотрим в каком формате хранятся данные о температуре в памяти датчика. Формат данных определяет то, с каким разрешением измеряется тем-

пература. Согласно документации DS18B20 может измерять температуру с разрешающей способностью от 9 до 12 битов, что в свою очередь определяет точность измерений. На рис. 7.8 показано, в каком виде хранятся данные в регистрах температуры при разрешении в 12 битов.

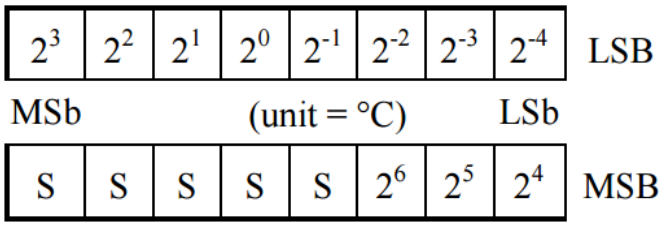

Рис.7.8 — Содержимое регистров температуры

Старшие биты, обозначенные буквой S, указывают на знак температуры, если они равны 1, то температура отрицательная, а если 0, то положительная. При этом биты 2^6—2^0 представляют целую часть, а биты 2^{-1} — 2^{-4} представляют дробную часть величины измеренной температуры. Дробная часть формируется путем анализа установленных битов с 2^{-1} по 2^{-4}. Если бит 2^{-1} имеет значение «1», то это эквивалентно 0,5 градуса, если бит 2^{-2} установлен в «1», то это эквивалентно 0,25 градуса и так далее с уменьшением в два раза. После того как все биты дробной части проанализированы, дробные части, соответствующие каждому установленному биту следует сложить, что и будет составлять дробную часть измеренной температуры, затем полученное число необходимо преобразовать в десятичный код и вывести на индикатор после десятичного знака. А все биты, представляющие целую часть измеренной температуры, следует преобразовать в десятичный формат без дополнительных преобразований и вывести полученное число перед десятичным знаком. Также необходимо помнить, что для корректного двоично-десятичного преобразования требуется наложить маску и осуществить циклический сдвиг битов, представляющих дробную и целые части измерения температуры.

Отметим, что согласно протоколу 1-Wire данные передаются младшим битом вперед.

7.2.5. Команды термодатчика

Для взаимодействия с термодатчиком существует набор определенных команд. Каждая команда представляет из себя один байт (далее будут приведены в HEX-формате). Рассмотрим основные команды, которые потребуются для взаимодействия с DS18B20 в ходе выполнения данной лабораторной работы.

SKIP ROM [0xCC] — позволяет обратиться ко всем подчиненным устройствам на линии одновременно.

CONVERT T [0x44] — запускает процесс измерения температуры, который занимает некоторое время, зависящее от заданной точности.

READ SCRATCHPAD [0xBE] — инициализирует передачу содержимого ОЗУ DS18b20 ведущему устройству (МК).

7.2.6. Программная реализация протокола 1-Wire

Для реализации алгоритма (см. рис. 7.5) целесообразно использовать следующие подпрограммы:

— передачи байта (команды и конфигурационного слова);
— приема байта от температурного датчика (для приема результатов измерения температуры, для опроса статуса тревоги и др.);
— инициализации и проверки присутствия датчика.

Для упрощения кода программы предлагается применить несколько макросов, которые показаны на рис. 7.9.

```
OUT     macro
        b1
        bcf     TRISA,4
        b0
        endm
IN      macro
        b1
        bsf     TRISA,4
        b0
        endm
#define sens  PORTA,4
```

Рис.7.9 — Макросы для упрощения программы

Макрос OUT переключает вывод МК RA4 (к которому подключен термодатчик) на выход, для того чтобы задавать на нем логические уровни. Макрос IN переключает вывод МК RA4 на вход, для того чтобы считывать логические уровни на линии.

Для реализации алгоритма (см. рис. 7.5) целесообразно использовать следующие подпрограммы:

— передачи байта (команды и конфигурационного слова) (рис. 7.10);
— приема байта от температурного датчика (для приема результатов измерения температуры, для опроса статуса тревоги и др.) (рис. 7.11);
— инициализации и проверки присутствия датчика (рис. 7.12).

Подпрограммы, приведенные выше необходимо записать в соответствующей последовательности при реализации основного кода измерителя температуры. Для анализа величины температуры, требуется принять два байта дважды, использовав подпрограмму приема байта (рис. 7.11).

```
SEND_BITE  ; ПОДПРОГРАММА ОТПРАВКИ БАЙТА ДАННЫХ ДАТЧИКУ
        b0
        MOVLW   .8 ;задаем сколько раз нужно сдвинуть
        MOVWF   CNT3;загружаем этозначение в переменную
send
        RRF     sendbite,f  ; сдвигаем регистр передаваемых данных
        ;чтобы крайний бит выпал в C
        BTFSS   C       ; проверяем что попало в C
        GOTO    SendNull ; если выпал 0,то передаем 0
        GOTO    SendOne  ; если выпал 1,то передаем 1
SendNull                ; посылаем ноль
        OUT     ; линию на вывод
        BCF     sens; выставляем ноль на линии
        CALL    WAIT100uS; ждем
        IN      ; линию на вход(отпускаем ее)
        GOTO    m1
SendOne                ;посылаем единицу
        OUT     ; линию на вывод
        BCF     sens; выставляем ноль
        call    WAIT2uS ; ждем
        IN      ; линию на вход(отпускаем ее)
        CALL    WAIT100uS;
m1
        DECFSZ  CNT3,F  ; увеличиваем счетчик и проверяем не закончен ли байт
        GOTO    send    ; если не закончили идем пересылать следующий бит
        RETURN          ; передача закончена
```

Рис.7.10 — Подпрограмма для передачи байта

```
GET_BITE
        b0
        MOVLW   .8   ;задаем сколько раз нужно сдвинуть
        MOVWF   CNT3 ;загружаем это значение в переменную
get
        OUT     ; установили линию на выход
        BCF     sens; установили линию в ноль
        CALL    WAIT2uS
        IN      ; отпустили линию
        CALL    WAIT30uS; ждем пока датчик установит состояние
        BTFSS   sens; проверяем состояние линии
        GOTO    GetNull; если на линии ноль
        GOTO    GetOne ; если на линии высокий
m2
        DECFSZ  CNT3,F  ; увеличиваем счетчик и проверяем не закончен ли байт
        GOTO    get    ; если не закончили прием идем принимать следующий бит
        RETURN
GetNull
        BCF     C; кладем полученный ноль в C
        RRF     result,f ; сдвигаем чтобы содержимое C
        ; попало на позицию старшего бита result
        CALL    WAIT70uS
        GOTO    m2
GetOne
        BSF     C; кладем полученную единицу в C
        RRF     result,f ;сдвигаем чтобы содержимое C
        ; попало на позицию старшего бита result
        CALL    WAIT70uS
        GOTO    m2
```

Рис.7.11 — Подпрограмма для приема байта

```
INIT    ; инициализация и проверка присутствия DS18b20
        OUT ;линию на выход
        BCF     sens ; прижимаем линию к земле
        CALL    WAIT500uS ; ждем
        IN      ; линию на вход
        CALL    WAIT100uS ;ждем
        BTFSC   sens ; проверяем состояние линии
        GOTO    ERR_no_sensor ; датчик не ответил
        CALL    WAIT500uS ; все в порядке
                RETURN
```

Рис.7.12 — Подпрограмма инициализации и проверки присутствия

7.3. Порядок выполнения лабораторной работы

Реализуйте программу для считывания температуры с температурного датчика используя коды подпрограмм, которые приведены в теоретической части.

Используйте динамическую индикацию для вывода измеренной температуры предварительно преобразовав полученный код от температурного датчика в пропорциональный десятичный с дробной частью.

7.4. Содержание отчета

Цель работы
Схема задействованной части макета.
Код программы с подробными объяснениями.
Подробные выводы.

7.5. Контрольные вопросы

7.5.1. Протокол 1-Wire. Принцип передачи данных.
7.5.2. Устройство памяти температурного датчика DS18b20.
7.5.3. Подпрограмма для передачи байта к температурному датчику.
7.5.4. Извлечение целой и дробной частей из двух байтов результата температурного конвертирования.

8. ЛАБОРАТОРНАЯ РАБОТА № 8. ОСНОВЫ ЯЗЫКА ПРОГРАММИРОВАНИЯ ВЫСОКОГО УРОВНЯ

8.1. Цель работы

Ознакомиться с основами языка «С» для программирования 8 разрядных микроконтроллеров.

Составить программу с реализацией динамической индикации с использованием программной и аппаратной задержек.

Составить программу регистрации замыканий кнопочных контактов матричной клавиатуры.

Изучить использование ассемблерных вставок.

8.2. Теоретические сведения

На практике компьютерные языки программирования низкого уровня широко применяются для составления программ 8 разрядных и реже 16 разрядных микроконтроллеров. Компьютерный язык программирования низкого уровня позволяет составить оптимальную программу с максимальной «прозрачностью», при компиляции, программа, составленная на низкоуровневом языке программирования, преобразовывается непосредственно в машинный код, который готов для загрузки в память программ микроконтроллера. В то время как компилятор компьютерного языка высокого уровня выполняет два преобразования: первое — программу языка высокого уровня в программу языка низкого уровня, а второе — программу языка низкого уровня в машинный код. На этапе первого преобразования каждый компилятор выполняет оптимизацию при формировании программы на низком уровне. Оптимизация позволяет повысить производительность и компактность программного кода, однако степень оптимизации у каждого компилятора различна.

В большинстве случаев высокоуровневый язык программирования «С» удобен для быстрой разработки программного кода и выполнения сложных программных конструкций в особенности связанных с математическими вычислениями. При этом программный код не изменяется при замене микроконтроллера одной разрядности на микроконтроллер с другой разрядностью.

Компьютерные языки программирования высокого уровня допускают использование специальных вставок программного кода на языке низкого уровня программирования (ассемблерные вставки). Ассемблерные вставки не подвергаются оптимизации, поэтому на первом этапе преобразования транслируются на язык низкого уровня без изменений.

8.2.1. Шаблон программы на языке С

Шаблон программы составленный на языке С представлен на рис. 7.1.

```
#include <htc.h>
#include <stdio.h>

void init()
{
// раздел инициализации и начальной конфигурации
}
void main(){
init();
while(1){

//основная часть программы

}}

void interrupt ISR(void){
//стандартный обработчик прерываний (высого приоритета)
}

void interrupt low_priority ISRL(void){
//обработчик прерываний низкого приоритета
}
```

Рис. 7.1 — Шаблон программы на языке С

В шаблоне программы (см. рис. 7.1) используются следующие элементы:

— **#include** <htc.h> — вспомогательный файл, содержащий имена регистров специального назначения с адресными константами;

— **void** init() — функция объявления переменных и начальной конфигурации периферийных модулей микроконтроллера;

— **void** main(**void**) — главная функция программы;

— **void** interrupt ISR(void) — функция обработчика прерываний (высокого приоритета);

— **void** interrupt low_priority ISRL(**void**) — функция обработчика прерываний низкого приоритета (используется для микроконтроллеров семейства PIC18Fxxx).

В функции **void** main(**void**) помещается функция инициализации и начальной конфигурации периферийных модулей МК init() и основной бесконечный цикл while(). Функция **void** interrupt ISR(**void**) имеет ключевое слово «interrupt» которое указывает, что данная функция выполняется только в случае возникновения прерывания, которое разрешено. Для микроконтроллеров се-

мейства PIC16Fxxx используется одна функция обработки прерываний так как в памяти программ предусмотрен только один вектор прерываний. В микроконтроллерах семейства PIC18Fxxx память программ содержит два вектора прерываний: высокого и низкого приоритетов. Для обработки прерываний низкого приоритета используется функция с ключевым словом low_priority. Приоритетность прерываний задается конфигурационными битами регистра управления прерываниями.

8.2.2. Реализация программы мигающего светодиода

Реализация программы заключается в изменении логического состояния одного вывода порта с некоторым интервалом времени для чего необходимо использовать программную или аппаратную задержку. Программную и аппаратную задержку можно реализовать управляемой по длительности, где длительность регулируется при помощи указания числа соответствующего длительности или доступных внешних устройств ввода.

```
while(1){

PORTE |= 0x01; // PORTE = 0b00000001;
__delay_ms(100);
PORTE &= 0x1E; // PORTE = 0b11111110;
__delay_ms(100);

}
```

Рис. 8.2 — Программа мигающего светодиода с программной задержкой

В примере программы (см. рис. 8.2) применяется функция задержки, которая представляет собой цикл выполняющий циклическое декрементирование переменной. Время, потраченное на декрементирование переменной является временем задержки.

8.2.4. Реализация динамической индикации с использованием программной задержки

Основной принцип динамической индикации с использованием программной задержки изложен в первой части лабораторной работы №2. В данном пункте рассматривается аналогичная реализация программы динамической индикации на языке программирования высокого уровня.

```
while(1){
__delay_ms(100);
n++;
switch(n)
{
case 1:
RA2 = 0;     //выкобчение предыдущего разряда
PORTD = digit[1]; //вывод цифры
RA3 = 1;     //включение текущего разряда
break;
case 2:
RA3 = 0;
PORTD = digit[0];
RA2 = 1;
break;
}
if (n==2) n =0; // выбор разряда
}}
```

Рис. 8.3 — Фрагмент программы реализации динамической индикации с программной задержкой.

8.2.5. Реализация динамической индикации с использованием аппаратной задержки

Основной принцип динамической индикации с использованием аппаратной задержки изложен во второй части лабораторной работы №2. В данном пункте рассматривается один из возможных способов реализации динамической индикации с использованием аппаратной задержки реализованной с применением языка программирования «С».

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

void interrupt ISR(void){
//обработчик прерываний высого приоритета
if (T0IF){
n++;
switch(n)
{
case 1:
RA2 = 0;     //выкобчение предыдущего разряда
PORTD = digit[1]; //вывод цифры
RA3 = 1;     //включение текущего разряда
break;
case 2:
RA3 = 0;
PORTD = digit[0];
RA2 = 1;
break;
}
if (n==2) n =0; // выбор разряда
T0IF = 0;
}}
```

Рис. 8.4 — Фрагмент программной реализации динамической индикации

В приведенном фрагменте программы (см. рис. 8.4) не показан раздел инициализации модуля таймера, объявления переменных и массива.

8.2.6. Реализация обработчика прерываний матричной клавиатуры с устранением влияния дребезга

Принцип реализации матричной клавиатуры подробно объяснен в лабораторной работе №3.

Пример обработчика прерываний матричной клавиатуры (рис. 8.5) реализован с использованием переменной «n» которая представляет собой одновременно и буфер (защелку) — отдельную переменную, и задержку в виде числа, которое присваивается переменной «n». В случае регистрации прерывания, проверяется флаг прерывания «RBIF», а затем буфер «n». Если буфер «n» имеет значение равное 0, тогда переменной буфера присваивается число 200 (величина соответствующая длительности времени для подавления эффекта дребезга) и выполняется детектирование нажатой кнопки. В случае возникновения дребезга, вход в обработчик прерываний матричной клавиатуры блокирован, до тех пор, пока буфер «n» не примет значение 0. Таким образом достигается регистрация только первого изменения фронта, а остальных изменений в результате дребезга игнорируется (подавляется). В зависимости от длительности дребезга опытным путем подбирается число, присваиваемое буферу «n».

Значение, присвоенное «n» постоянно декрементируется при помощи аппаратного модуля таймера. Такая организация подпрограммы позволяет наиболее эффективно выполнять регистрацию нажатий кнопок матричной клавиатуры.

```
|
void interrupt ISR(void){

if(RBIF){
if (n == 0) { n = 200; //программная защелка
if(RB4 == 0) {          //где число указывает длительность
    RB2 = 1;
    if (RB4 == 1) {/* описание действия по нажатию*/ }
    RB2 = 0;
    RB3 = 1;
    if (RB4 == 1) {/* описание действия по нажатию*/ }
    RB3 = 0;
}

if(RB5 == 0) {
    RB2 = 1;
    if (RB5 == 1) {/* описание действия по нажатию*/ }
    RB2 = 0;
    RB3 = 1;
    if (RB5 == 1) {/* описание действия по нажатию*/ }
    RB3 = 0;
}
}
RBIF = 0;
}
if(T0IF){ // аппаратная задержка для подавления дребезга
if (n > 0) n --; // декрементирование
T0IF = 0;
}}
```

Рис. 8.5 — Обработчик прерываний матричной клавиатуры с подавлением «дребезга» контактов

8.2.6. Реализация счетчика с совместным использованием программной и аппаратной задержек

В основе аппаратной задержки используется один из доступных модулей таймеров. Интервал между прерываниями от модуля таймера является минимальным временем аппаратной задержки. Используя дополнительный регистр счетчик, в обработчике прерывания таймера, можно увеличить длительность аппаратной задержки кратную минимальной длительности.

Для реализации простейшего счетчика в основном цикле программы используется программная задержка с инкрементированием или декрементированием регистра в диапазоне от 0 до 99. Вывод значения счетчика осуществляется на семисегментный индикатор в режиме динамической индикации с использованием аппаратной задержки. Аппаратная задержка имеет приоритет по отношению к программной задержке, поэтому процесс счета программной за-

держки может быть многократно прерван аппаратной задержкой для вывода текущего значения счетчика на индикатор.

```
while(1){
__delay_ms(100);
if (n<100) n++;
else n = 0;
a = n/10%10; // извлечение единиц
b = n%10;    // извлечение десятков
}

void interrupt ISR(void){
//обработчик прерываний высого приоритета
if (T0IF){
n++;
switch(n)
{
case 1:
RA2 = 0;     //выкобчение предыдущего разряда
PORTD = digit[a]; //вывод цифры
RA3 = 1;     //включение текущего разряда
break;
case 2:
RA3 = 0;
PORTD = digit[b];
RA2 = 1;
break;
}
if (n==2) n =0; // выбор разряда
T0IF = 0;
}}
```

Рис. 8.6 — Фрагмент программы счетчика от 0 до 99

8.2.7. Использование ассемблерных вставок

Использование ассемблерных вставок позволяет сформировать код который выполнятся за определенное количество программных тактов. Такие вставки не подвергаются оптимизации подобно строкам языка С.

Пример программного кода с использованием ассемблерных вставок (рис. 8.7).

```
main(){
int a=0; // объявление переменной на С
while(){
#asm
    MOVLW 0x10  // загрузка константы
    MOVWF (_a+1)// во второй регистр переменной а
#endasm
}}

asm (" CLRF (_a+1) "); // одиночная ассемблерная вставка
```

Рис. 8.7 — Пример программы с использованием ассемблерных вставок

Как видно из представленного фрагмента программного кода, ассемблерные вставки могут быть блоковые (#asm…..#endasm) или однострочные (asm("…..")).

Объявленные переменные на языке «С» в ассемблерной вставке записываются с нижним подчеркиванием.

Необходимо также учитывать разрядности переменных, если переменная состоит из двух регистров, то в ассемблерном формате записи необходимо к имени переменной прибавлять единицу для изменения адреса регистра переменной.

8.3. Порядок выполнения лабораторной работы

Используя шаблон программы на языке «С» выборочно выполните приведенные программы с программированием лабораторного макета для проверки работоспособности составленной программы.

8.4. Содержание отчета

Цель работы
Схема задействованной части макета.
Код программы с подробными объяснениями.
Подробные выводы.

8.5. Контрольные вопросы

8.5.1. Основные отличия языков программирования высокого и низкого уровней. Необходимость их совместного использования.

8.5.2. Программный принцип устранения влияния дребезга контактов.

8.5.3. Способ извлечения единиц, десятков, сотен и тысяч из переменной.

8.5.4. Виды ассемблерных вставок и цель их применения.

8.5.5. Сравнительный анализ шаблонов программы на языках С и ассемблера.

8.5.6. Перечислите виды циклов языка С и приведите примеры их применения.

8.5.7. Представьте аналоги циклов языка С и оператора условия на низкоуровневом языке.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Microchip PIC16F877 / Информационный листок : 2008 — 2013.
2. Кениг А., Кенег М. Полное руководство по PIC микроконтроллерам. — МК Пресс, Киев, 2007.
3. Таверенье А. PIC микроконтроллеры практика и применение / — ДМК Пресс, 2003.
4. Real Clock Timer DS1307 / Maxim Integrated. — Б. м., Maxim Integrated, б.г.
5. Temperature sensor DS18B20 / Maxim Integrated. — Б. м., Maxim Integrated, б. г.

ПРИЛОЖЕНИЕ А.
СИСТЕМА КОМАНД МИКРОКОНТРОЛЛЕРА RISC-АРХИТЕКТУРЫ

Мнемоника, операнды	Описание	Колич. тактов	14 битовый код инструкций	Статус биты
Байт-ориентированные команды				
ADDWF f,d	Сложить W и f	1	00 0111 dfff ffff	C, DC, Z
ANDWF f,d	Логическое И W с f	1	00 0101 dfff ffff	Z
CLRF f	Очистить регистр f	1	00 0001 1fff ffff	Z
CLRW -	Очистить аккумулятор W	1	00 0001 0xxx xxxx	Z
COMF f,d	Инвертировать биты f	1	00 1001 dfff ffff	Z
DECF f,d	Декрементировать f	1	00 0011 dfff ffff	Z
DECFSZ f,d	Дек. f, пропустить если 0	1(2)	00 1011 dfff ffff	
INCF f,d	Инкрементировать f	1	00 1010 dfff ffff	Z
INCFSZ f,d	Инк. f, пропустить если 0	1(2)	00 1111 dfff ffff	
IORWF f,d	Логическое ИЛИ W с f	1	00 0100 dfff ffff	Z
MOVF f,d	Переместит f	1	00 1000 dfff ffff	Z
MOVWF f	Переместить W в f	1	00 0000 1fff ffff	
NOP -	Пустая операция	1	00 0000 0xx0 0000	
RLF f,d	Сдвигать влево f	1	00 1101 dfff ffff	C
RRF f,d	Сдвигать вправо f	1	00 1100 dfff ffff	C
SUBWF f,d	Вычесть W из f	1	00 0010 dfff ffff	C, DC, Z
SWAPF f,d	Обмен половин в f	1	00 1110 dfff ffff	
XORWF f,d	Исключающее ИЛИ W с f	1	00 0110 dfff ffff	Z
Бит-ориентированные команды				
BCF f, b	Очистить бит регистра f	1	01 00bb bfff ffff	
BSF f, b	Установить бит регистра f	1	01 01bb bfff ffff	
BTFSC f, b	Проп. если бит сброшен	1(2)	01 10bb bfff ffff	
BTFSS f, b	Проп. если бит установлен	1(2)	01 11bb bfff ffff	
Команды управления и работы с постоянными величинами (константами)				
ADDLW k	Сложить константу и W	1	11 111x kkkk kkkk	C, DC, Z
ANDLW k	Лог. И константы с W	1	11 1001 kkkk kkkk	Z
CALL k	Вызов подпрограммы	2	10 0kkk kkkk kkkk	
CLRWDT -	Очистить сторож. таймер	1	00 0000 0110 0100	TO, PD
GOTO k	Переход по адресу	2	10 1kkk kkkk kkkk	
IORLW k	Лог. ИЛИ константы и W	1	11 1000 kkkk kkkk	Z
MOVLW k	Переместить конст. W	1	11 00xx kkkk kkkk	
RETFIE -	Возврат из прерывания	2	00 0000 0000 1001	
RETLW k	Возврат со значение в W	2	11 01xx kkkk kkkk	
RETURN -	Возврат из подпрограммы	2	00 0000 0000 1000	
SLEEP -	Режим ожидания	1	00 0000 0110 0011	TO, PD
SUBLW k	Вычесть W из константы	1	11 110x kkkk kkkk	C, DC, Z
XORLW k	Искл. ИЛИ константы и W	1	11 1010 kkkk kkkk	Z

ПРИЛОЖЕНИЕ Б.
ОПИСАНИЕ РЕГИСТРОВ СПЕЦИАЛЬНОГО НАЗНАЧЕНИЯ

Регистр **STATUS** содержит биты управления банками памяти данных, флаги (биты значения которых управляются непосредственно ядром микроконтроллера) отображающие арифметические состояния АЛУ и флаги, отображающие причины сброса микроконтроллера.

RW-0	RW-0	RW-0	R-1	R-1	RW-x	RW-x	RW-x
IRP	RP1	RP0	TO	PD	Z	DC	C
7 бит							0 бит

IRP	Бит выбора банка при косвенной адресации 1 = банк 2, 3 0 = банк 0, 1
RP1:RP0	Биты выбора банка памяти данных при непосредственной адресации 11 = банк 3 10 = банк 2 01 = банк 1 00 = банк 0
TO (инв)	Флаг переполнения сторожевого таймера 1 = после POR или команд SLEEP, CLRWDT 0 = после переполнения WDT
PD (инв)	Флаг включения питания 1 = после POR или команды CLRWDT 0 = после выполнения команды SLEEP
Z	Флаг нулевого результата 1 = результат выполнения логической или арифметической операции равен нулю 0 = результат выполнения логической или арифметической операции не равен нулю
DC	Флаг десятичного переноса/заема для команд SUBLW, ADDWF, SUBWF, ADDLW 1 = был перенос из младшего полубайта 0 = не было переноса из младшего полубайта
C	Флаг переноса/заема для команд SUBLW, ADDWF, SUBWF, ADDLW 1 = был перенос из старшего бита 0 = не было переноса из старшего бита При использовании команд сдвига RRF и LRF бит C загружается старшим или младшим битом сдвигаемого регистра.

Регистр **OPTION_REG** содержит биты для конфигурации подтягивающих резисторов для порта B, выбора активного фронта для внешнего прерывания, выбора источника тактового сигнала для TMR0 и настройки предварительно делителя для TMR0 или WDT.

RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1	RW-1
RPBU	INTEDG	T0CS	T0SE	PSA	PS2	PS1	PS0
7 бит							0 бит

RPBU	Бит включения подтягивающих резисторов для порта B 1 = подтягивающие резисторы отключены 0 = подтягивающие резисторы подключены		
INTEDG	Бит выбора активного фронта сигнала на входе внешнего прерывания 1 = прерывание по переднему фронту сигнала 0 = прерывание по заднему фронту сигнала		
T0CS	Бит выбора тактового сигнала для TMR0 1 = внешний тактовый сигнал с вывода RA4/T0CKI 0 = внутренний тактовый сигнал CLKOUT		
T0SE	Бит выбора фронта внешнего тактового сигнала для приращения TMR0 1 = приращение TMR0 по заднему фронту тактового сигнала 0 = приращение TMR0 по переднему фронту тактового сигнала		
PSA	Бит назначения предварительного делителя 1 = Предварительный делитель используется для WDT 0 = Предварительный делитель используется для TMR0		
PS2:PS0	Биты установки коэффициента деления предварительного делителя для:		
	TMR0	WDT	
000	1:2	1:1	
001	1:4	1:2	
010	1:8	1:4	
011	1:16	1:8	
100	1:32	1:16	
101	1:64	1:32	
110	1:128	1:64	
111	1:256	1:128	

Регистр **INTCON** содержит биты разрешения прерываний и флаги, которые устанавливаются, когда выполняется условие прерывания от определенного устройства. Флаги устанавливаются при выполнении условия прерывания независимо от состояний битов разрешения прерываний.

RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	RW-x
GIE	PEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF
7 бит							0 бит

GIE	Бит глобального разрешения прерывания 1 = разрешены все немаскированные прерывания 0 = все прерывания запрещены
PEIE	Бит разрешения прерывания от периферийных устройств 1 = разрешены все немаскированные прерывания от периферийных устройств 0 = все прерывания от периферийных устройств запрещены
T0IE	Бит разрешения прерывания по переполнению TMR0 1 = прерывание разрешено 0 = прерывание запрещено
INTE	Бит разрешения внешнего прерывания 1 = прерывание разрешено 0 = прерывание запрещено
RBIE	Бит разрешения прерывания по изменению логических состояний на одном из RB7 : RB4 1 = прерывание разрешено 0 = прерывание запрещено
T0IF	Флаг переполнения TMR0 1 = произошло переполнение TMR0 (программный сброс) 0 = переполнение TMR0 не было
INTF	Флаг внешнего прерывания 1 = внешнее прерывание произошло (программный сброс) 0 = внешнего прерывания не было
RBIF	Флаг прерывания по изменению уровня сигнала на одном из входов RB7:RB4 PORTB 1 = изменение уровня сигнала зафиксировано на одном из входов RB7 : RB4 (программный сброс) 0 = изменения уровня сигнала не было ни на одном из входов не было

Регистр **ADCON0**

RW-0	RW-0	RW-0	RW-0	RW-0	RW-0	U-0	RW-0
ADCS1	ADCS0	CHS2	CHS1	CHS0	GO-DONE	-	ADON
7 бит							0 бит

ADCS1: ADCS0	Выбор источника тактового сигнала, и установка частоты преобразования 00 = Fosc/2 01 = Fosc/8 10 = Fosc/32 11 = FRC (включение собственный генератор модуля АЦП)
CHS2: CHS0	Выбор канала для модуля АЦП- 000 = Канал 0 (RA0/AN0) 001 = Канал 1 (RA1/AN1) и так далее.
GO- DONE	Бит состояния АЦП Чтобы модуль АЦП начал преобразование необходимо установить данный бит, а затем проверять его состояние, если: 1 = модуль АЦП выполняет преобразование 0 = преобразование еще не завершено (по завершению преобразования аппаратно сбрасывается в 0)
ADON	Бит для включения модуля АЦП 1 = модуль АЦП включен 0 = модуль АЦП выключен

Регистр **ADCON1**

RW-0	U-0	U-0	U-0	RW-0	RW-0	RW-0	RW-0
ADFM	-	-	-	PCFG3	PCFG2	PCFG1	PCFG0
7 бит							0 бит

ADFM	Формат сохранения результата преобразования модуля АЦП 1 = правое выравнивание, 6 старших битов ADRESH читаются как 0 0 = левое выравнивание, 6 младших битов ADRESL читаются как 0									
PCFG3: PCFG0	AN7 RE2	AN6 RE1	AN5 RE0	AN4 RA5	AN3 RA3	AN2 RA2	AN1 RA1	AN0 RA0	VRF+	VRF–
0000	A	A	A	A	A	A	A	A	VDD	Vss
0001	A	A	A	A	VREF+	A	A	A	RA3	Vss
0010	D	D	D	A	A	A	A	A	VDD	Vss
0011	D	D	D	A	VREF+	A	A	A	RA3	Vss
0100	D	D	D	D	A	D	A	A	VDD	Vss
0101	D	D	D	D	VREF+	D	A	A	RA3	Vss
011x	D	D	D	D	D	D	D	D	VDD	Vss
1000	A	A	A	A	VREF+	VREF-	A	A	RA3	RA2
1001	D	D	A	A	A	A	A	A	VDD	Vss
1010	D	D	A	A	VREF+	A	A	A	RA3	Vss
1011	D	D	A	A	VREF+	VREF-	A	A	RA3	RA2
1100	D	D	D	A	VREF+	VREF-	A	A	RA3	RA2
1101	D	D	D	D	VREF+	VREF-	A	A	RA3	RA2
1110	D	D	D	D	D	D	D	A	VDD	Vss
1111	D	D	D	D	VRF+	VRF-	D	A	RA3	RA2

Регистр **T1CON** (адрес 10h)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	T1CKPS1	T1CKPS0	T1OSCEN	-T1SYNC	TMR1CS	TMR1ON
7 бит							0 бит

T1CKPS1: T1CKPS0	Выбор коэффициента деления предделителя TMR1 11 = 1:8 10 = 1:4 01 = 1:2 00 = 1:1
T1OSCEN	Включение тактового генератора TMR1 1 = генератор включен 0 = генератор выключен (инвертирующий элемент и резистивная обратная связь включены для уменьшения тока потребления)
T1SYNC	Синхронизация внешнего тактового сигнала TMR1CS=1 1 = не синхронизировать внешний тактовый 0 = синхронизировать внешний тактовый TMR1CS=0 Значение бита игнорируется
TMR1CS	Выбор источника тактового сигнала 1 = внешний источник вывода RC0/T1OSO/T1CKI (активным является передний фронт сигнала) 0 = внутренний источник Fosc/4
TMR1ON	Включение модуля TMR1 1 = включен 0 = выключен

Регистр **T2CON** (адрес 12h)

U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	TOUTPS3	TOUTPS2	TOUTPS1	TOUTPS0	TMR2ON	T2CKPS1	T2CKPS0
7 бит							0 бит

TOUTPS3: TOUTPS0	Выбор коэффициента входного делителя TMR2 0000 = 1:1 0001 = 1:2 : : 1111 = 1:16
TMR2ON	Включение модуля TMR2 1 = включен 0 = выключен
T2CKPS1: T2CKPS0	Выбор коэффициента деления предделителя TMR2 00 = 1:1 01 = 1:4 1x = 1:16

Регистр **CCPxCON** (адрес 17h/1Dh)

U-0	U-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
-	-	CCPxX	CCPxY	CCPxM3	CCPxM2	CCPxM1	CCPxM0
7 бит							0 бит

CCPxX: CCPxY	Младшие биты скважности ШИМ Режим захвата Не используются Режим сравнения Не используются Режим ШИМ Два младших бита скважности. Восемь мтарших находяться в CCPRxL
CCPxM3: CCPxM0	Режим работы модуля CCPx 0000 = модуль CCPx выключен (сброс модуля CCPx) 0100 = захват по каждому заднему фронту сигнала 0101 = захват по каждому переднему фронту сигнала 0110 = захвата по каждому 4-му переднему фронту сигнала 0111 = захвата по каждому 16-му переднему фронту сигнала 1000 = сравнение, устанавливает выходной сигнал (устанавливается флаг CCPxIF в ‘1’) 1001 = сравнение, сбрасивает выходной сигнал (устанавливается флаг CCPxIF в ‘1’) 1010 = сравнение, на выходной сигнал не влияет (устанавливается флаг CCPxIF в ‘1’) 1011 = сравнение, триггер специальных функций (устанавливается флаг CCPxIF в ‘1’; на вывод CCPx не влияет). CCP1 — сброс таймера TMR1. CCP2 — сброс таймера TMR1, запуск преобразования АЦП (если АЦП включено). 11xx = ШИМ режим

Регистр **SSPSTAT** (адрес 94h). Регистр статуса модуля MSSP

R/W-0	R/W-0	R-0	R-0	R-0	R-0	R-0	R-0
SMP	CKE	D/-A	P	S	R/-W	UA	BF

7 бит 0 бит

SMP	Фаза выборки бита Ведущий режим SPI 1 = опрос входа в конце периода вывода данных 0 = опрос входа в середине периода вывода данных Ведомый режим SPI Для режима ведомого SPI этот бит всегда должен быть сброшен в ‘0’ Ведущий или ведомый режим I2C 1 = управление длительности фронта выключено в стандартном режиме (100кГц и 1МГц) 0 = управление длительности фронта включено в скоростном режиме (400кГц)
CKE	Выбор фронта тактового сигнала SPI режим, CKP = 0 1 = данные передаются по переднему фронту сигнала на выводе SCK 0 = данные передаются по заднему фронту сигнала на выводе SCK SPI режим, CKP = 1 1 = данные передаются по заднему фронту сигнала на выводе SCK 0 = данные передаются по переднему фронту сигнала на выводе SCK Ведущий или ведомый режим I2C 1 = входные уровни соответствуют спецификации SMBus 0 = входные уровни соответствуют спецификации I2C
D/-A	Бит Данные/Адрес (только для режима I2C) 1 = последний принятый или переданный байт является информационным 0 = последний принятый или переданный байт является адресным
P	Бит STOP (только для режима I2C) Этот бит сбрасывается в ‘0’ когда модуль MSSP выключен, SSPEN=0. 1 = указывает, что бит STOP был обнаружен последним (этот бит равен ‘0’после сброса) 0 = бит STOP не является последним

S	Бит START (только для режима I2C) Этот бит сбрасывается в ‘0’ когда модуль MSSP выключен, SSPEN=0. 1 = указывает, что бит START был обнаружен последним (этот бит равен ‘0’после сброса) 0 = бит START не является последним
R/-W	Бит чтения/записи (только для режима I2C) Значение бита действительно только после совпадения адреса и до приема бита START, STOP или –ACK. Ведомый режим I2C 1 = чтение 0 = запись Ведущий режим I2C 1 = выполняется передача данных 0 = передачи данных не происходит Логическое ИЛИ этого бита с битами SEN, RSEN, PEN, RCEN или ACKEN укажет на неактивное состояние модуля MSSP.
UA	Флаг обновления адреса устройства (только для режима10-разрядного I2C) 1 = необходимо обновить адрес в регистре SSPADD 0 = обновление адреса не требуется
BF	Бит статуса буфера Прием (SPI и I2C режимы) 1 = прием завершен, буфер SSPBUF полон 0 = прием не завершен, буфер SSPBUF пуст Передача (только I2C режима) 1 = выполняется передача данных (исключая биты –ACK и STOP), буфер SSPBUF полон 0 = передача данных завершена (исключая биты –ACK и STOP), буфер SSPBUF пуст

Регистр **SSPCON** (адрес 14h). Регистр управления модуля MSSP

R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0
WCOL	SSPOV	SSPEN	CKP	SSPM3	SSPM2	SSPM1	SSPM0
7 бит							0 бит

WCOL	Бит конфликта записи Ведущий режим 1 = запись в SSPBUF была выполнена при не выполнении условий шины I2C 0 = конфликта не было Ведомый режим 1 = была предпринята попытка записи в SSPBUF во время передачи предыдущего байта 0 = конфликта не было
SSPOV	Бит переполнения приемника SPI режим 1 = принят новый байт в то время как SSPBUF содержит предыдущие данные (байт в SSPSR будет потерян). В ведомом режиме пользователь должен прочитать содержимое регистра, т.к. каждая операция инициализируется записью в SSPBUF. (сбрасывается в ‘0’ программно) 0 = нет переполнения I2C режим 1 = принят новый байт в то время как SSPBUF содержит предыдущие данные. Значение бита не действительно при передаче данных. (сбрасывается в ‘0’ программно) 0 = нет переполнения
SSPEN	Бит включения модуля MSSP Когда модуль включен, соответствующие порты ввода/вывода настраиваются на выход или вход SPI режим 1 = модуль MSSP включен, выводы SCK, SDO, SDI, -SS используются модулем MSSP 0 = модуль MSSP выключен, выводы работают как цифровые порты ввода/вывода I2C режим 1 = модуль MSSP включен, выводы SDA, SCL используются модулем MSSP 0 = модуль MSSP выключен, выводы работают как цифровые порты ввода/вывода
CKP	Бит выбора полярности тактового сигнала SPI режим 1 = пассивный высокий уровень сигнала

	0 = пассивный низкий уровень сигнала Ведомый режим I2C Управление тактовым сигналом SCK 1 = не управлять тактовым сигналом 0 = удерживать тактовый сигнал в низком логическом уровне (используется для подготовки данных) Ведущий режим I2C Не имеет значения
SSPM3: SSPM0	Режим работы модуля MSSP 0000 = ведущий режим SPI, тактовый сигнал = Fosc/4 0001 = ведущий режим SPI, тактовый сигнал = Fosc/16 0010 = ведущий режим SPI, тактовый сигнал = Fosc/64 0011 = ведущий режим SPI, тактовый сигнал = выход TMR2/2 0100 = ведомый режим SPI, тактовый сигнал с вывода SCK. Вывод –SS подключен к MSSP 0101 = ведомый режим SPI, тактовый сигнал с вывода SCK. Вывод –SS не подключен к MSSP 0110 = ведомый режим I2C, 7-разрядная адресация 0111 = ведомый режим I2C, 10-разрядная адресация 1000 = ведущий режим I2C, тактовый сигнал = Fosc/(4*(SSPADD+1)) 1011 = программная поддержка ведущего режима I2C (ведомый режим выключен) 1110 = программная поддержка ведущего режима I2C, 7-разрядная адресация с разрешением прерываний по приему бит START и STOP 1111 = программная поддержка ведущего режима I2C, 10-разрядная адресация с разрешением прерываний по приему бит START и STOP 1001, 1010, 1100, 1101 = резерв

ЛАБОРАТОРНЫЙ ПРАКТИКУМ ПО ДИСЦИПЛИНЕ «ПЕРИФЕРИЙНЫЕ ИНТЕГРИРОВАННЫЕ КОНТРОЛЛЕРЫ В РАДИОЭЛЕКТРОННЫХ СРЕДСТВАХ»

Учебно-методическое пособие

Составители: **Скорик** Иван Викторович,
Широков Игорь Борисович

В авторской редакции

Изд. № 42/2020. Объем 6 п.л.
РИИЦМ ФГАОУ ВО «Севастопольский государственный университет»