

СОДЕРЖАНИЕ

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, ЕДИНИЦ, СОКРАЩЕНИЙ И ТЕРМИНОВ	5
ВВЕДЕНИЕ	7
1 ОПИСАНИЕ ЛАБОРАТОРНОГО СТЕНДА	9
2 СРЕДА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ IDE IAR EMBEDDED WORKBENCH	13
2.1 Интерфейс компилятора языка Си IDE IAR Embedded Workbench	14
2.2 Порядок создания и настройки проекта в среде IDE IAR Embedded Workbench.....	19
2.3 Отладка проекта в среде IDE IAR Embedded Workbench	24
2.4 Встроенные функции компилятора языка Си IDE IAR Embedded Workbench.....	35
3 РУКОВОДСТВО ПО РАЗРАБОТКЕ ПРОГРАММ НА ЯЗЫКЕ Си ДЛЯ МИКРОКОНТРОЛЛЕРНЫХ СИСТЕМ	38
4 ЛАБОРАТОРНЫЙ ПРАКТИКУМ.....	76
4.1 ИЗУЧЕНИЕ ПРИНЦИПОВ ПРОГРАММНОГО УПРАВЛЕНИЯ ВНЕШНИМИ УСТРОЙСТВАМИ НА ПРИМЕРЕ ВЫВОДА ИНФОРМАЦИИ НА ЦИФРОВОЙ ИНДИКАТОР	76
4.1.1 Указания по организации самостоятельной работы	76
4.1.2 Порядок проведения работы и указания по ее выполнению	93
4.1.3 Содержание отчета.....	93
4.1.4 Контрольные вопросы и задания.....	93
4.2 ИЗУЧЕНИЕ ПРИНЦИПОВ ОРГАНИЗАЦИИ ОБМЕНА ДАННЫМИ ПО ПОСЛЕДОВАТЕЛЬНОМУ ИНТЕРФЕЙСУ USB МЕЖДУ МИКРОКОНТРОЛЛЕРОМ MSP430F1611 И ПЭВМ.....	95
4.2.1 Указания по организации самостоятельной работы	95
4.2.1.1 Организация модулей USART в микроконтроллере MSP430	95
4.2.2 Описание лабораторной установки	105
4.2.3 Порядок проведения работы и указания по ее выполнению	110
4.2.4 Содержание отчета.....	114
4.2.5 Контрольные вопросы и задания.....	114
4.3 ИЗУЧЕНИЕ ПРИНЦИПОВ ОРГАНИЗАЦИИ ОБМЕНА ДАННЫМИ ПО ПОСЛЕДОВАТЕЛЬНОМУ ИНТЕРФЕЙСУ I2C НА ПРИМЕРЕ УПРАВЛЕНИЯ БЛОКОМ СВЕТОДИОДОВ И ПРОГРАММНОГО ОПРОСА КЛАВИАТУРЫ	115
4.3.1 Указания по организации самостоятельной работы	115

4.3.2 Описание лабораторной установки	125
4.3.3 Порядок проведения работы и указания по ее выполнению	131
4.3.4 Содержание отчета.....	135
4.3.5 Контрольные вопросы и задания.....	135
4.4 ИЗУЧЕНИЕ ПРИНЦИПОВ ОБРАБОТКИ ПРЕРЫВАНИЙ НА ПРИМЕРЕ УПРАВЛЕНИЯ ВСТРОЕННЫМИ В МИКРОКОНТРОЛЛЕР ТАЙМЕРАМИ- СЧЕТЧИКАМИ И КОМПАРАТОРОМ	136
4.4.1 Указания по организации самостоятельной работы	136
4.4.2 Описание лабораторной установки	150
4.4.3 Порядок проведения работы и указания по ее выполнению	153
4.4.4 Содержание отчета	155
4.4.5 Контрольные вопросы и задания.....	156
4.5 ИЗУЧЕНИЕ ПРИНЦИПОВ РАБОТЫ СО ВСТРОЕННЫМ В МИКРОКОНТРОЛЛЕР АНАЛОГО-ЦИФРОВЫМ ПРЕОБРАЗОВАТЕЛЕМ НА ПРИМЕРЕ ИЗМЕРЕНИЯ ОТНОСИТЕЛЬНОЙ ВЛАЖНОСТИ ВОЗДУХА И ПОТРЕБЛЯЕМОГО СТЕНДОМ ТОКА.....	157
4.5.1 Указания по организации самостоятельной работы.....	157
4.5.2 Описание лабораторной установки.....	166
4.5.3 Порядок проведения работы и указания по ее выполнению	175
4.5.4 Содержание отчета.....	178
4.5.5 Контрольные вопросы и задания.....	178
ПЕРЕЧЕНЬ ССЫЛОК.....	179
ПРИЛОЖЕНИЕ А	180
А.1 Лабораторный стенд. Схема электрическая принципиальная	180
ПРИЛОЖЕНИЕ Б.....	187
Б.1 Параметры датчиков НН-4000-003	187

ПЕРЕЧЕНЬ УСЛОВНЫХ ОБОЗНАЧЕНИЙ, СИМВОЛОВ, ЕДИНИЦ, СОКРАЩЕНИЙ И ТЕРМИНОВ

- ACLK – Auxiliary Clock (вспомогательное тактирование);
- ADC – Analog-to-Digital Converter (аналого-цифровой преобразователь, АЦП);
- BOR – Brown-Out Reset (Сброс при пониженном питающем напряжении);
- BSL – Bootstrap Loader (начальный загрузчик программной памяти или ОЗУ);
- CPU – Central Processing Unit (центральное процессорное устройство, ЦПУ);
- DAC – Digital-to-Analog Converter (цифро-аналоговый преобразователь, ЦАП);
- DCO – Digitally Controlled Oscillator (осциллятор с цифровым управлением);
- Dst – Destination (Назначение);
- FLL – Frequency Locked Loop (система автоматической подстройки частоты);
- GIE – General Interrupt Enable (общее разрешение прерываний);
- I/O – Input/Output (вход / выход);
- ISR – Interrupt Service Routine (процедура обработки прерывания);
- LSB – Least-Significant Bit (младший бит);
- LSD – Least-Significant Digit (младший разряд);
- LPM – Low-Power Mode (режим пониженного энергопотребления);
- MAB – Memory Address Bus (адресная шина памяти);
- MCLK – Master Clock (главное тактирование);
- MDB – Memory Data Bus (шина данных памяти);
- MSB – Most-Significant Bit (старший бит);
- MSD – Most-Significant Digit (старший разряд);
- NMI – Non-Maskable Interrupt (немаскируемое прерывание);
- PC – Program Counter (программный счетчик);
- POR – Power-On Reset (сброс при включении питания);
- PUC – Power-up clear (очистка при включении питания);
- RAM – Random Access Memory (оперативное запоминающее устройство, ОЗУ);
- SCG – System Clock Generator (генератор системного тактирования);
- SFR – Special Function Register (регистр специального назначения);

SMCLK – Sub-System Master Clock (подсистема главного тактирования);
 SP – Stack Pointer (указатель стека);
 SR – Status Register (регистр статуса);
 Src – Source (источник);
 TOS – Top-of-Stack (вершина стека);
 WDT – Watchdog Timer (сторожевой таймер);

Тип доступа к битам регистров и их исходное состояние

Ключ	Тип доступа к биту
rw	Чтение / запись
r	Только чтение
r0	Читается как «0»
r1	Читается как «1»
w	Только запись
w0	Записывается как «0»
w1	Записывается как «1»
(w)	Бит в регистре не реализован; запись 1 приводит к импульсу. Всегда читается как «0»
h0	Очищается аппаратно
h1	Устанавливается аппаратно
-0, -1	Состояние после сигнала PUC
-(0), -(1)	Состояние после сигнала POR

ВВЕДЕНИЕ

Развитие микроэлектроники и широкое применение ее изделий в промышленном производстве, в устройствах и системах управления самыми разнообразными объектами и процессами является в настоящее время одним из основных направлений научно-технического прогресса.

Использование микроконтроллеров в изделиях не только приводит к повышению технико-экономических показателей (стоимости, надежности, потребляемой мощности, габаритных размеров), но и позволяет сократить время разработки изделий и делает их модифицируемыми, адаптивными. Использование микроконтроллеров в системах управления обеспечивает достижение высоких показателей эффективности при низкой стоимости.

Микроконтроллеры представляют собой эффективное средство автоматизации разнообразных объектов и процессов. Они широко используются для создания систем по обработке сигналов, сбору данных, выполнению функций обработки и управления в промышленных, медицинских, пользовательских, автомобильных электронных устройствах и контрольно-измерительных приборах.

Все это определяет необходимость изучения микроконтроллерных систем. Лабораторный практикум по курсу «Проектирование микроконтроллерных систем» нацелен на закрепление теоретических знаний по архитектуре микроконтроллеров серии MSP430. В нем рассматриваются схемы подключения разнообразных периферийных модулей, особенности сопряжения микроконтроллера с различными датчиками и акцент ставится на разработке программного обеспечения для подсистем датчиков и ввода-вывода.

Разработка программного обеспечения является центральным моментом общего процесса проектирования. Центр тяжести функциональных свойств современных цифровых систем находится именно в программных средствах.

Применение лабораторного стенда возможно в различных областях, требующих автоматизации процесса сбора, обработки данных, дистанционного управления. Подсистема датчиков, представленная датчиками температуры, влажности, освещенности, расширяет области применения стенда, он может быть использован для контроля климатических параметров производственных процессов. Реализованные преобразователи USB-COM, драйверы оптической линии связи, ZigBee-модуль на базе CC2430/CC2431, интерфейс для подключения к Ethernet на базе микросхем от WIZnet (W3100, W3150), обеспечивают обмен данными, удаленный контроль и управление устройством

по различным протоколам.

В стенде присутствуют: знакосинтезирующий ЖКИ, который используется для организации вывода текущей информации непосредственно на объекте в реальном времени; клавиатура может использоваться для управления устройством; flash-память объемом 2 Мб дает возможность хранить внушительные объемы информации, например, показания датчиков, или исходные данные для выполнения действий – генерации цифровых и аналоговых сигналов, управления другими устройствами по заданной программе, выдачу статической и динамической информации в качестве Web-сервера и так далее.

1 ОПИСАНИЕ ЛАБОРАТОРНОГО СТЕНДА

Лабораторный стенд (рисунок 1.1) реализован на базе микроконтроллера MSP430F1611 производства компании Texas Instruments.



Рисунок 1.1 – Лабораторный стенд и JTAG-программатор

Он включает в себя следующие функциональные элементы:

- микроконтроллер MSP430F1611;
- USB-интерфейс (FT232RL);
- датчик температуры (TMP275);
- датчик освещенности (TSL2561T);
- датчик влажности (HIN-4000-003);
- датчик тока (INA139); резистивный датчик;
- драйверы оптической линии связи (HFBR-2522 и HFBR-1522);
- 8 светодиодов;
- сетевой модуль для подключения к ЛВС Ethernet 10/100Base T (ПМ7010А);
- 16 Мбит flash-память (AT45DB161В);
- аналоговые входы/выходы (микрофонный вход, 2 линейных входа, 2

выхода ЦАП);

- клавиатуру 3x4 (AK-304-N-BBW);
- LCD индикатор (WH1602A-NGG-CT);
- трансивер стандарта ZigBee для построения беспроводных сетей (PSIS2430);
- динамик, подключенный к ЦАП, через усилитель мощности;
- схему для измерения сопротивления.

Структурная схема лабораторного стенда представлена на рисунке 1.2.

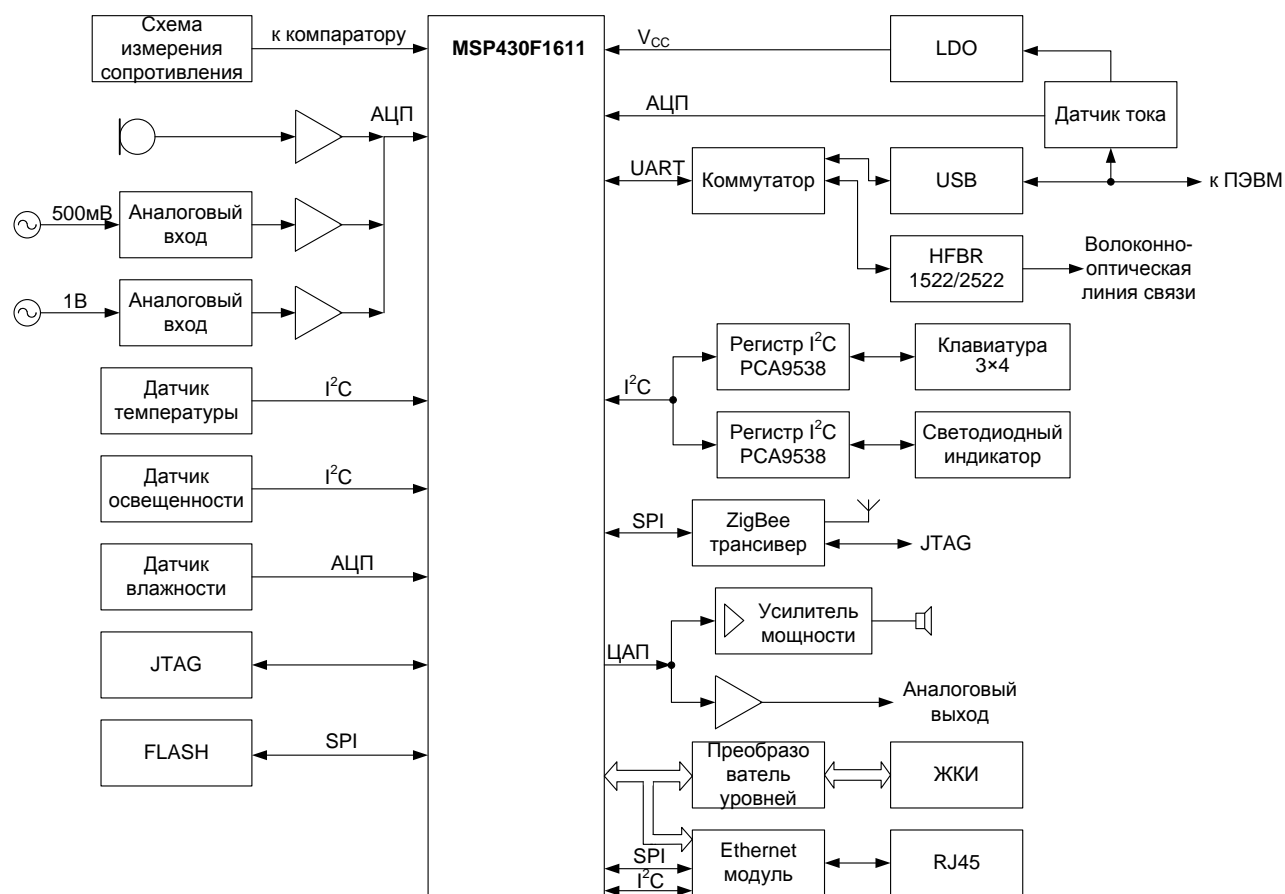


Рисунок 1.2 – Структурная схема лабораторного стенда

Функции управления и синхронизации выполняет микроконтроллер, который работает под управлением программы, хранящейся во flash-памяти.

В задачи микроконтроллера входит прием и обработка данных, поступающих от ПЭВМ, управление режимом работы цифро-аналогового преобразователя (ЦАП) и аналого-цифрового преобразователя (АЦП), обмен данными через локальную сеть Ethernet или беспроводную сеть стандарта ZigBee.

Согласование электрических уровней сигналов между ПЭВМ и стендом обеспечивает преобразователь уровней из КМОП-уровней в уровни стандарта USB.

Двенадцати разрядный ЦАП обеспечивает формирование выходного напряжения, пропорционального коду, а подключенный к усилителю мощности динамик позволяет формировать звуковые сигналы. Наличие микрофонного входа и flash-памяти позволяет построить на базе стенда цифровой магнитофон.

Клавиатура 3х4 подключается к микроконтроллеру через последовательный регистр PCA9534, что позволяет сократить число линий для обмена данными с микроконтроллером и значительно упростить программу обработки дребезга контактов за счет использования сигналов прерываний от регистра, которые генерируются в момент изменения входных сигналов.

Светодиодные индикаторы также подключаются к контроллеру через последовательные регистры для сокращения числа необходимых линий ввода/вывода.

Разнообразные датчики, установленные в стенде, позволяют реализовать на нем систему сбора и обработки информации.

JTAG-интерфейс позволяет выполнять внутрисхемное программирование контроллера, отлаживать программы в реальном устройстве с оперативным доступом ко всем регистрам и периферийным модулям контроллера.

ZigBee трансивер обеспечивает возможность построения беспроводных сетей со скоростью передачи до 250 кБод.

Коммутатор предназначен для выбора последовательно интерфейса (USB или волоконно-оптическая линия связи).

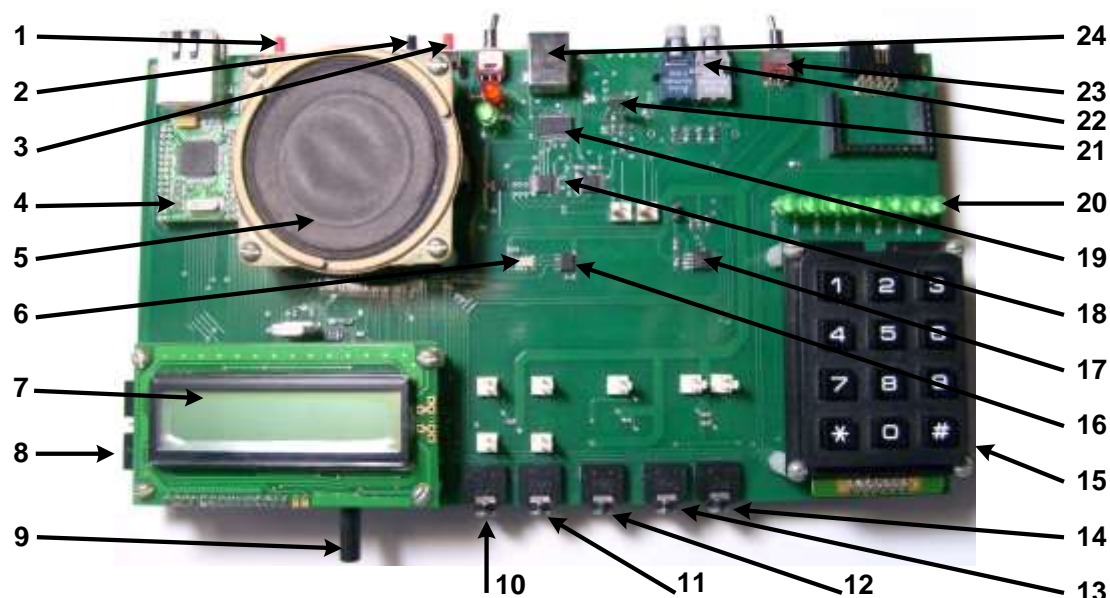
Блок питания (LDO) обеспечивает все напряжения, необходимые для работы схемы.

Размещение модулей на печатной плате изображено на рисунках 1.3 и 1.4.

Особенностью стенда является отсутствие внешнего источника питания (питание осуществляется от USB-порта), контроль тока потребления и возможность программирования по USB через BSL-загрузчик без использования JTAG программатора.

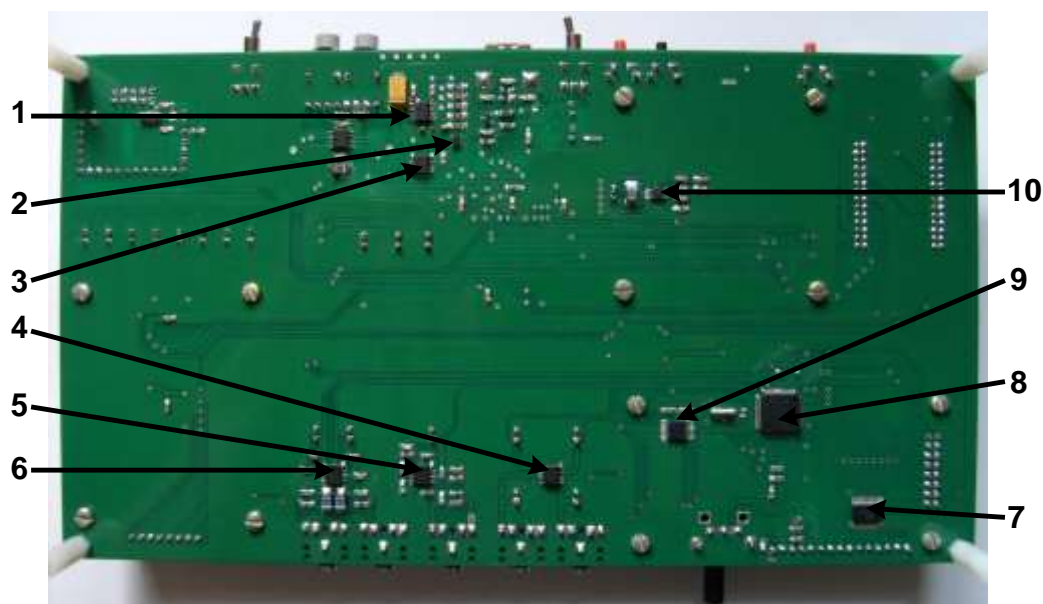
Принципиальная схема лабораторного стенда приведена в ПРИЛОЖЕНИИ А.

Разработка программного обеспечения для лабораторного стенда ведется в интегрированной среде IDE IAR Embedded Workbench на языке высокого уровня Си и Ассемблера.



1 – кнопка сброса; 2 – кнопка выхода из ждущего режима; 3 – кнопка сброса USB-интерфейса; 4 – модуль Ethernet; 5 – динамик; 6 – датчик освещенности; 7 – ЖКИ; 8 – JTAG-интерфейс; 9 – подстроечный резистор; 10 – выходы ЦАП1; 11 – вход ЦАП0; 12 – микрофонный вход; 13 – линейный вход ($\pm 1V$); 14 – линейный вход ($\pm 0,5V$); 15 – клавиатура; 16 – датчик температуры; 17 – Flash-память; 18 – дешифраторы; 19 – USB-интерфейс; 20 – светодиоды; 21 – датчик влажности; 22 – драйверы оптической линии связи; 23 – тумблер выбора режима загрузки; 24 – USB-порт.

Рисунок 1.3 – Размещение модулей на печатной плате (вид сверху)



1 – линейный стабилизатор; 2 – датчик тока; 3, 4, 5, 6 – операционные усилители; 7 – преобразователь уровней; 8 – микроконтроллер; 9 – дешифратор; 10 – усилитель мощности.

Рисунок 1.4 – Размещение модулей на печатной плате (вид снизу)

2 СРЕДА РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ IDE IAR EMBEDDED WORKBENCH

Интегрированная среда разработки и отладки IDE IAR Embedded Workbench (IAR EW) представляет собой мощное средство разработки и отладки, позволяющее создавать законченные прикладные проекты на базе различных 8-, 16- и 32-битных микроконтроллеров, в том числе микроконтроллеров MSP430. IDE включает в себя набор инструментальных средств, интегрированных в единую программу-оболочку с удобным оконным интерфейсом, работающую под Microsoft Windows. IAR Embedded Workbench хорошо документирована и имеет простой, интуитивно понятный пользовательский интерфейс. Демоверсия IDE IAR Embedded Workbench с ограниченным сроком действия компилятора C/C++ (1 месяц) доступна на сайте www.iar.com.

IDE IAR Embedded Workbench для MSP430 включает в себя следующие инструментальные средства:

- компилятор IAR C/C++;
- ассемблер IAR;
- универсальный компоновщик IAR XLINK Linker;
- программа построения библиотек IAR Library Builder;
- набор библиотек IAR XLIB Librarian;
- текстовый редактор;
- менеджер проектов;
- утилита построения командной строки;

– отладчик языка высокого уровня IAR C-SPY Debugger. Компилятор, ассемблер и компоновщик могут запускаться на выполнение не только из оболочки IDE, но из командной строки в случае, если есть необходимость использовать их как внешние инструментальные средства в уже установленной проектной среде. IDE IAR Embedded Workbench поддерживает развитые функции управления проектами, дающие возможность пользователю управлять всеми проектными модулями, например, файлами исходного текста на C или C++, ассемблерными файлами, подключаемыми файлами и другими связанными модулями. Файлы могут быть сгруппированы с различными опциями, заданными на уровне всего проекта, группы или только файла.

Для управления проектами IDE предоставляет следующие основные средства и возможности:

- шаблоны для создания проектов;

- иерархическое представление проекта;
- браузер исходного файла с иерархическим символьным представлением;
- установка опций глобально для групп исходных файлов или для индивидуальных исходных файлов;
- утилита Make, которая перетранслирует, повторно ассемблирует и компоует файлы, когда это необходимо;
- текстовые файлы проектов;
- утилита Custom Build, разворачивающая стандартный инструментальный набор простым способом;
- командная строка, формирующая проектный файл на входе.

IDE IAR Embedded Workbench предоставляет пользователю возможность удобного управления размещением окон, произвольной организации окон в группах, произвольного изменения размеров окон.

2.1 Интерфейс компилятора языка Си IDE IAR Embedded Workbench

На рисунке 2.1 показано главное окно IAR Embedded Workbench IDE и его различные компоненты.

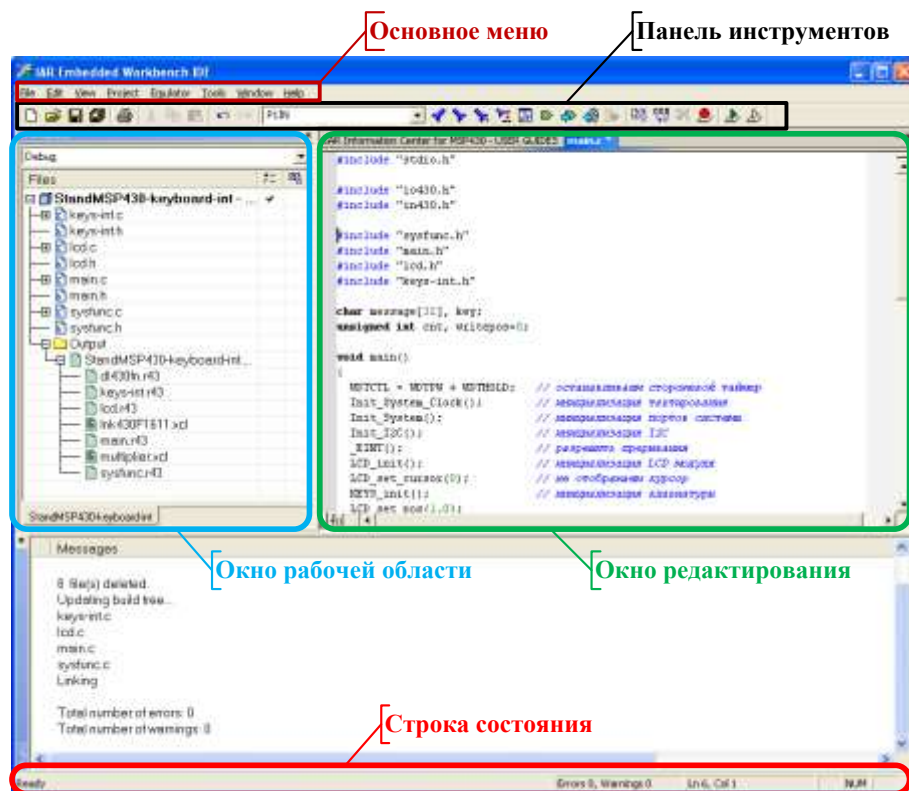


Рисунок 2.1 – Главное окно IAR Embedded Workbench IDE

Окно может выглядеть по-разному, в зависимости от используемых сменных модулей. Описание пунктов меню приведено в таблице 2.1.

Таблица 2.1 – Описание пунктов меню

Меню	Описание
File	В меню File представлены команды для открытия новых файлов и файлов проекта, сохранение, печать и выход из IAR Embedded Workbench IDE.
Edit	Меню Edit содержит команды редактирования и поиска в окнах редактора, включения и отключения контрольных точек в C-SPY.
View	Команды меню View используют для того, чтобы открыть окна и выбрать для отображения необходимые инструментальные панели.
Project	Меню Project содержит команды для того, чтобы добавить файлы к проекту, создать группы, и использовать инструментальные средства IAR в текущем проекте.
Tools	Меню Tools – пользовательское конфигурируемое меню, к которому Вы можете добавить инструментальные средства для использования с IAR Embedded Workbench IDE.
Window	С помощью команд в меню Window вы можете управлять внедренными окнами IAR Embedded Workbench IDE и изменять их расположение относительно экрана.
Help	Меню Help содержит справку о IAR Embedded Workbench IDE.

Toolbar. Панель инструментов (**View->Toolbar**), обеспечивает доступ к наиболее часто используемым командам IAR.

Описание любой кнопки можно посмотреть, наведя на нее курсор мыши. Когда команда не доступна, соответствующая кнопка панели будет недоступна, и нельзя щелкнуть на нее.

На рисунке 2.2 показано описание команд панели инструментов.



Рисунок 2.2 – Панель инструментов

Status Bar. Строка состояния внизу активного окна, вызываемая из меню

View, отображает состояние IAR.

В процессе редактирования, строка состояния показывает текущую линию и номер столбца, содержащего курсор, состояние клавиш Caps Lock, Num Lock, и состояние ввода.

Окно рабочей области. Окно Workspace, доступное из меню View, показывает название текущей рабочей области и древовидное представление проектов, групп и файлов, включенных в рабочую область (рисунок 2.3).

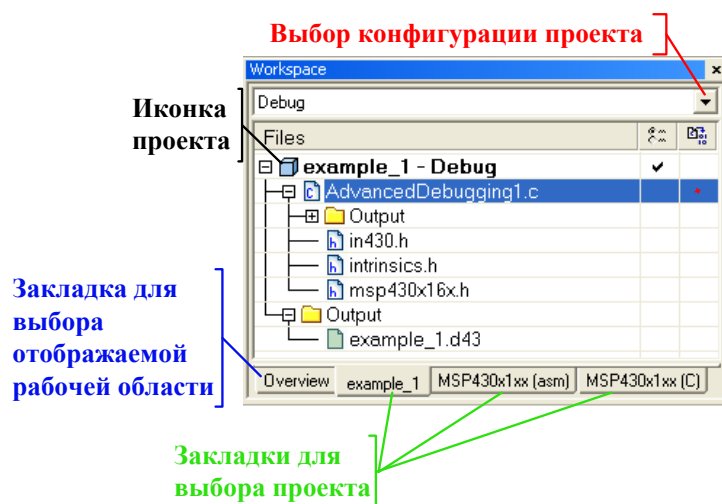


Рисунок 2.3 – Окно рабочей области

В раскрывающемся списке сверху окна можно выбрать конфигурацию построения текущего проекта (Debug или Release).

Для получения дополнительной информации об управлении проектом и использовании окна Workspace, см. *MSP430 IAR Embedded Workbench User Guide* (пункт меню Help).

Контекстное меню окна Workspace. Щелчок правой кнопкой мыши в окне Workspace отображает контекстное меню, которое дает доступ к командам, описание которых приведено в таблице 2.2.

Таблица 2.2 – Контекстное меню окна Workspace

Команда меню	Описание
1	2
Options	Отображает диалоговое окно, где можно установить параметры для каждого из используемых инструментов в выбранном элементе окна рабочей области. Параметры можно установить во всем проекте, в группе файлов, или в отдельном файле

Продолжение таблицы 2.2

1	2
Make	Собирает, транслирует, и связывает текущие файлы в соответствии с последними изменениями в проекте.
Compile	Компилирует или транслирует открытый файл. Можно выбрать файл в рабочем окне или окно редактора, содержащего файл, который необходимо компилировать.
Rebuild All	Повторно собирает и повторно связывает все файлы в выбранной конфигурации.
Clean	Удаляет промежуточные файлы.
Stop Build	Останавливает текущую операцию.
Add>Add Files	Открывает диалоговое окно, в котором можно добавить файлы к проекту.
Add>Add Group	Открывает диалоговое окно, в котором можно добавить новые группы к проекту.
Remove	Удаляет выбранные элементы из окна Workspace.
Source Code Control	Открывает подменю с командами для управления исходным текстом
File Properties	Открывает стандартное диалоговое окно свойств для выбранного файла.
Set as Active	Устанавливает активным выбранный проект в дисплее краткого обзора.

Меню управления исходным кодом (Source Code Control). Меню управления исходным кодом доступно из меню Project и из контекстного меню в окне Workspace.

Для получения дополнительной информации о взаимодействии с внешней системой управления исходного кода см. *MSP430 IAR Embedded Workbench User Guide* (пункт меню Help).

Описание пунктов меню управления исходным кодом приведено в таблице 2.3.

Таблица 2.3 – Меню управления исходным кодом







Команда меню	Описание
1	2
Check In (регистрация)	Открывает диалоговое окно Check In Files , в котором можно зарегистрировать выбранные файлы. Любые изменения, которые сделаны в файлах, будут сохранены в архиве. Выполнение этой команда возможно, если выделенные файлы отображены в окне рабочей области.

Продолжение таблицы 2.3

1	2
Check Out (проверка)	Проверяет выбранный файл или файлы. Это означает, что можно получить местную копию файла(ов), который можно редактировать. Эта команда возможна после выполнения команды регистрации
Undo Check out (отмена проверки)	Отобранные файлы возвращаются к последней заархивированной версии. Любые изменения (замены), сделанные в файлах, будут потеряны. Эта команда доступна после выполнения команды проверки.
Get Latest Version	Заменяет выбранные файлы последней заархивированной версией.
Compare	Показывает в окне SCC различия между существующей версией и новой заархивированной версией.
History	Отображает информацию о хронологии проверки выбранного файла.
Properties	Отображает информацию о выбранном файле
Refresh	Обновляет состояния для всех файлов, которые являются частью проекта. Эта команда всегда допускается для всех проектов под SCC.
Add Project To Source Control	Открывает диалоговое окно, которое позволяет создавать подключение между выбранным IAR и SCC проектами.
Remove Project From Source Control	Удаляет подключение между выбранными IAR проектом и проектом SCC.

Состояния при управлении исходным кодом. Каждый исходный файл может быть в одном из нескольких состояний. Описание пунктов меню состояний управления исходным кодом приведено в таблице 2.4.

Таблица 2.4 – Состояния при управлении исходным кодом

SCC состояние	Описание
	Проверенный. Файл доступен для редактирования
	Проверенный. Файл доступен для редактирования и был изменен.
 (серый замок)	Зарегистрированный. Это означает, что файл защищен от записи.
 (серый замок)	Зарегистрированный. Есть новая версия, доступная в архиве.
 (красный замок)	Это означает, что нельзя проверить файл.
 (красный замок)	Есть новая версия, доступная в архиве. Это означает, что нельзя проверить файл.

2.2 Порядок создания и настройки проекта в среде IDE IAR Embedded Workbench

В среде IDE IAR Embedded Workbench каждая программа для микроконтроллера должна оформляться в виде проекта, представляющего собой совокупность файлов, содержащих исчерпывающую информацию для программатора. Файлы каждого проекта рекомендуется сохранять в отдельном подкаталоге.

2.2.1 Создание проекта на языке Си

При создании нового проекта необходимо выполнить следующую последовательность действий:

- запустить программный модуль IAR Embedded Workbench for MSP430;
- используя верхнее меню, выполнить команду Project>Create New Project (рисунок 2.4);
- установить **Tool chain** в “MSP430”;
- выбрать проект **C > main**;
- нажать “**ОК**” и в открывшемся диалоговом окне выбрать путь, куда будет сохранен проект.

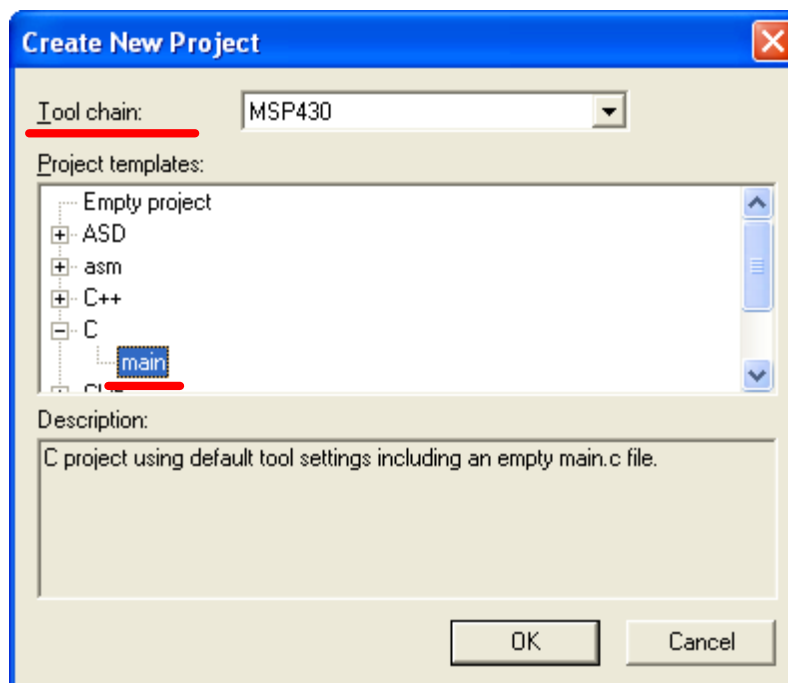


Рисунок 2.4 – Диалоговое окно для выбора типа создаваемого проекта

Проект добавится в рабочее пространство, по умолчанию к нему добавится файл **main.c**. Перед тем как добавлять файлы к проекту, необходимо сохранить рабочее пространство **File>Save Workspace**. Исходный текст программы для микроконтроллера на языке Си записывается в отдельном текстовом файле.

Для создания дополнительных файлов, необходимо выполнить следующие действия:

- в главном рабочем окне программы, используя верхнее меню, выполнить команду **File>New>File**;

- используя верхнее меню, выполнить команду **File>Save** (сохранить файл с расширением *.c).

- используя верхнее меню, выполнить команду **Project>Add Files...** и в открывшемся диалоговом окне выбрать сохраненный файл.

Далее необходимо выполнить команду **Project>Options** (при этом в окне **Workspace** необходимо выделить имя проекта (рисунок 2.5)).

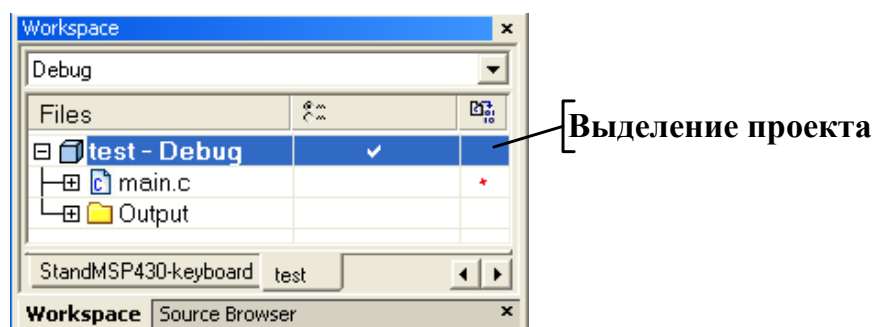


Рисунок 2.5 – Окно Workspace

В появившемся окне выбрать категорию **General Options** в закладке **Target** установить **Device** = “MSP430F1611” (рисунок 2.6).



Рисунок 2.6 – Окно Options>General Options>Target

В закладке **Library Options** установить параметры **Printf formatter** и **Scanf formatter** в **Large** (рисунок 2.7).

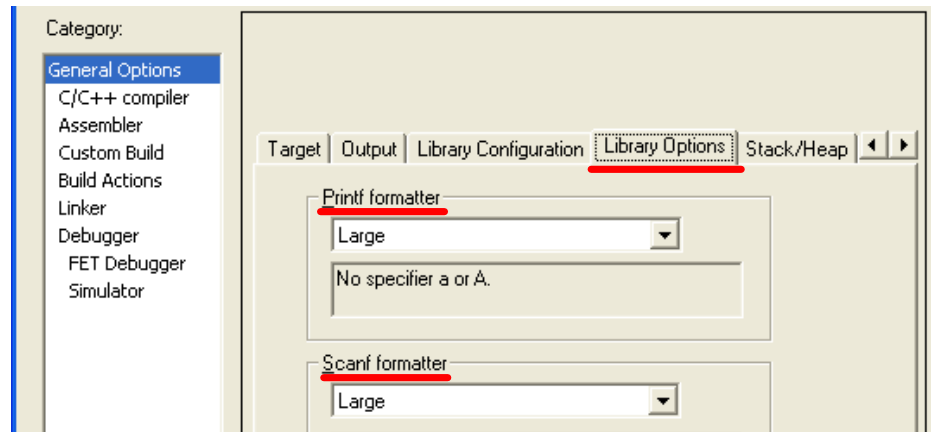


Рисунок 2.7 – Окно **Options>General Options>Library Options**

Затем необходимо выбрать категорию **C/C++ Compiler** в закладке **Optimizations** в выпадающем списке установить значение **None** (рисунок 2.8).

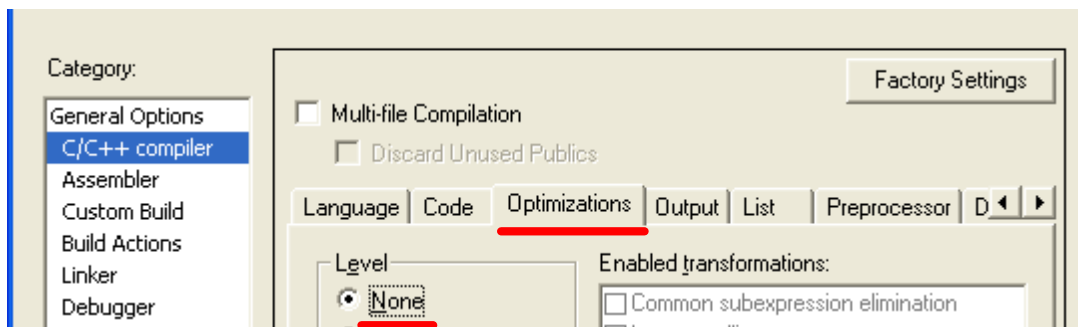


Рисунок 2.8 – Окно **Options>C/C++ Compiler>Optimizations**

В категории **Debugger** в закладке **Setup** установить **Driver** в **FET Debugger** (рисунок 2.9).

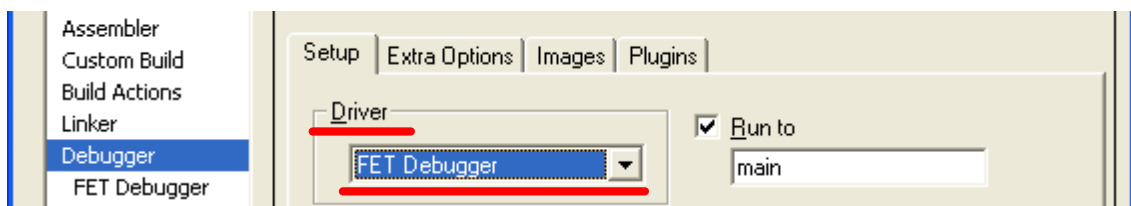


Рисунок 2.9 – Окно **Options>Debugger>Driver**

В категории **FET Debugger** в закладке **Setup** выбрать тип соединения **Texas Instruments USB-IF** (рисунок 2.10).

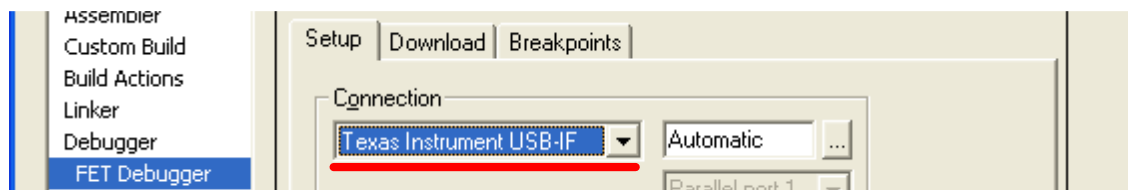


Рисунок 2.10 – Окно **Options> FET Debugger > Connection**

После того, как исходный текст программы для микроконтроллера будет набран, необходимо выполнить компиляцию проекта и загрузку кода и данных программы в память микроконтроллера:

- для компиляции приложения выбрать Project>Compile;
- выявленные в результате компиляции сообщения об ошибках отображаются в окне Build. При активизации сообщения об ошибке компилятор выводит подробные сведения о локализации и возможных причинах ошибки (рисунок 2.11).

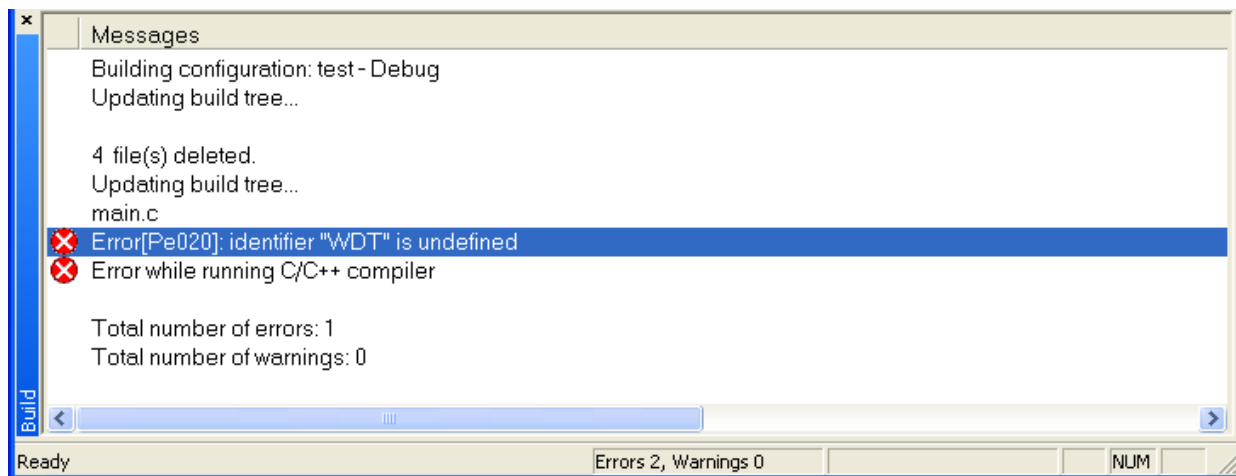


Рисунок 2.11 – Окно отображения информации о результатах компиляции проекта

2.2.2 Создание проекта на языке Ассемблера

При создании ассемблерного проекта в среде IDE IAR Embedded Workbench необходимо выполнить следующую последовательность действий:

- запустить программный модуль IAR Embedded Workbench for MSP430.
- используя верхнее меню, выполнить команду **Project>Create New Project** (рисунок 2.12);
- установить **Tool chain** в “MSP430”;
- выбрать проект **asm > asm**;
- нажать “**OK**” и в открывшемся диалоговом окне выбрать путь, куда будет сохранен проект.

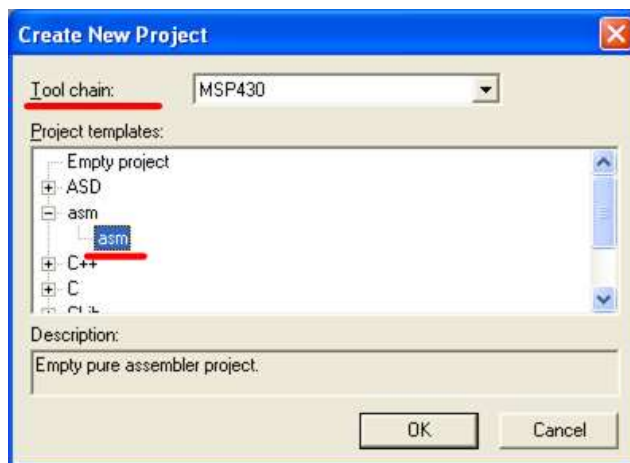


Рисунок 2.12 – Диалоговое окно для выбора типа создаваемого проекта

Проект добавится в рабочее пространство, по умолчанию к нему добавится файл **asm.s43** (рисунок 2.13).

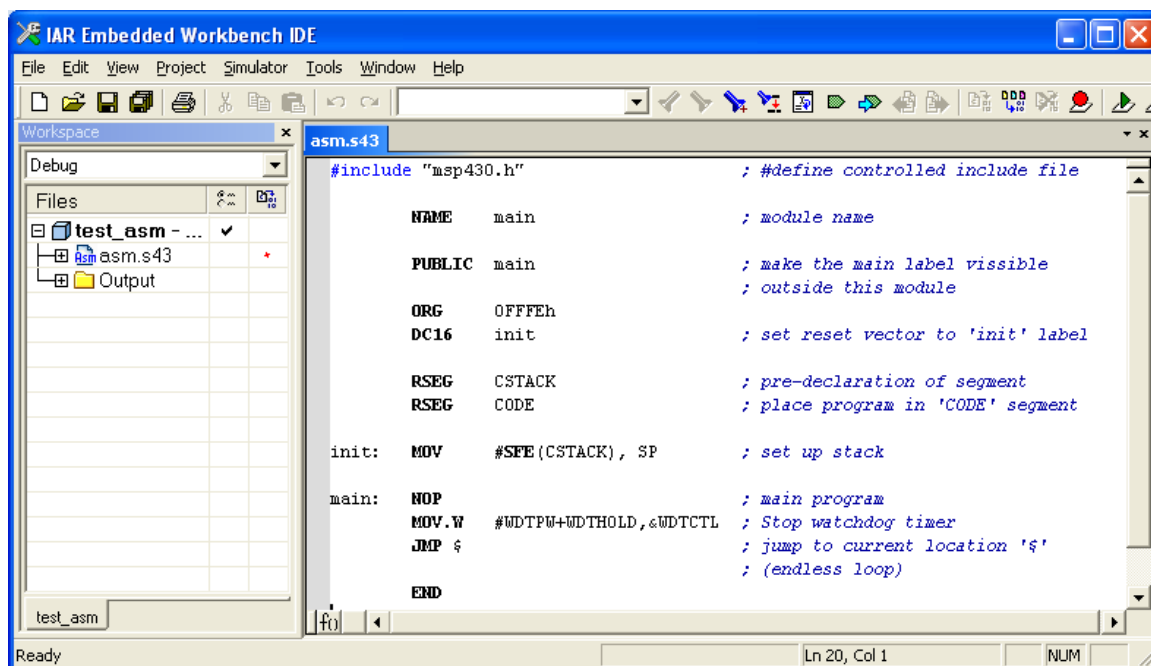


Рисунок 2.13 – Рабочее пространство созданного проекта

Дальнейшие настройки проекта выполняются в той же последовательности как при настройке проекта на языке Си.

2.3 Отладка проекта в среде IDE IAR Embedded Workbench

В интегрированной среде разработки программного обеспечения IAR Embedded Workbench есть встроенный отладчик C-SPY. При запуске IAR C-SPY отладчика (**Project>Debug**) в главном окне IAR Embedded Workbench IDE появляются следующие элементы:

- специальное меню отладки с командами для выполнения и отладки приложения;
- меню драйвера (в зависимости от того, какой C-SPY драйвер используется);
- специальная панель инструментов отладки;
- несколько окон и диалоговых окон, специфичных для C-SPY.

Меню отладки. В дополнение к меню, доступному в среде разработки, когда запущен C-SPY, доступно и меню отладки. В нем содержатся команды для выполнения и отладки приложения. Большинство из них также доступны на панели инструментов отладки. Пункты меню, которые являются активными во время работы отладчика, приведены в таблице 2.5.

Таблица 2.5 – Описание пунктов меню отладки

Пункт меню	Описание
Debug	Выполнение и отладка приложения
Simulator	Доступ к диалоговым окнам для моделирования прерываний

Панель инструментов отладки (рисунок 2.14) состоит из кнопок, соответствующих наиболее часто используемым командам меню отладки.

В таблице 2.6 приведены команды, соответствующие каждой из кнопок.

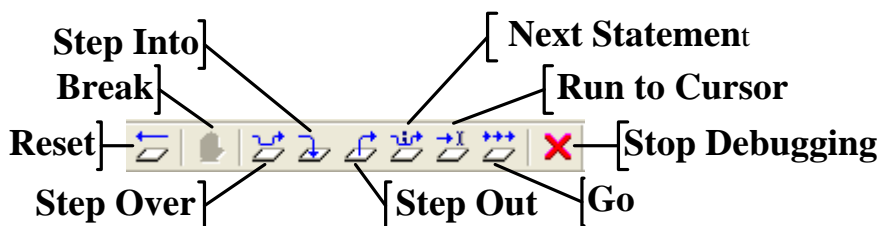











Рисунок 2.14 – Вид панели инструментов отладки

Таблица 2.6 – Описание кнопок панели инструментов отладки

Пункт меню	Описание
 Go (F5)	Выполняет программу от текущего выражения или оператора до точки прерывания или до тех пор, пока не будет достигнут конец программы
 Break	Останавливает выполнение приложения
 Reset	Сбрасывает целевой процессор
 Stop Debugging	Останавливает процесс отладки
 Step Over (F10)	Выполняет следующее выражение или оператор без входа в C или C++ функции или подпрограммы ассемблера
 Step Into (F11)	Выполняет следующее выражение или оператор с входом в C или C++ функции или подпрограммы ассемблера
 Step Out (SHIFT+F11)	Выполняет программу от текущего выражения до выражения, которое следует после вызова текущей функции
 Next Statement	Если шаги по входу/выходу из функций являются неоправданно медленными, целесообразно использовать эту команду для прямого перехода к следующему выражению
 Run to Cursor	Выполняет программу от текущего выражения или оператора до выбранного пользователем выражения или оператора

Окно дизассемблирования C-SPY (рисунок 2.15) доступно из меню **View>Disassembly** и позволяет совершать отладку через дизассемблированный код приложения.

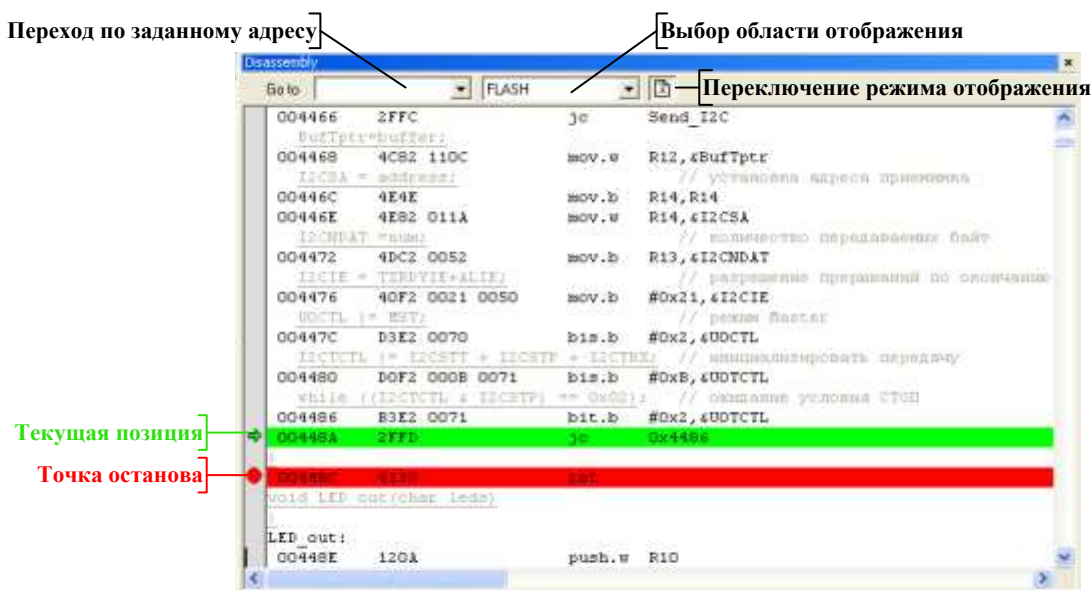


Рисунок 2.15 – Пример работы с окном дизассемблирования

В верхней части окна находится панель управления, описание основных пунктов которой приведено в таблице 2.7.

Таблица 2.7 – Описание элементов панели управления окна дизассемблирования

Действие	Описание
Go to	Переход к требуемому пользователю месту памяти
Zone display	Отображение доступных для отображения областей памяти
Toggle Mixed-Mode	Переключение между отображением только дизассемблированного кода или дизассемблированного кода вместе с соответствующим исходным кодом.

Текущая позиция (подсвечено зеленым) показывает ассемблерную команду, которая будет выполнена следующей. Двойной щелчок левой клавиши мыши в области серого цвета, расположенной слева, поставит точку останова, отображаемую красным цветом.

Для просмотра соответствующего ассемблерного кода для функции, можно выбрать ее в окне редактора и перетащить в окно дизассемблирования.

Окно памяти (рисунок 2.16) можно открыть из меню **View>Memory**. В этом окне отображается содержимое выбранной области памяти. Описание доступных команд приведено в таблице 2.8. Можно открывать несколько экземпляров этого окна, что очень удобно, если необходимо отслеживать несколько типов памяти или диапазоны регистров, или просматривать состояние различных частей памяти.

Таблица 2.8 – Описание элементов панели управления окна памяти

Действие	Описание
Go to	Переход по заданному адресу в памяти
Zone display	Выбор области отображаемой памяти
Context-menu button	Вызов контекстного меню

Область отображения показывает просматриваемые в данный момент адреса, содержимое памяти в формате, выбранном пользователем, и содержимое памяти в ASCII формате. Содержимое окна памяти можно редактировать и в шестнадцатеричной, и в ASCII части окна.

Режим доступа к данным отображается следующими цветами:

– желтый цвет указывает на прочитанную информацию;

- синий цвет указывает на записанную информацию;
- зеленый цвет указывает на информацию, которая была как считана, так и записана.

Чтобы просмотреть память, соответствующую определенной переменной, требуется выбрать ее (переменную) в окне редактора и перетащить в окно памяти.

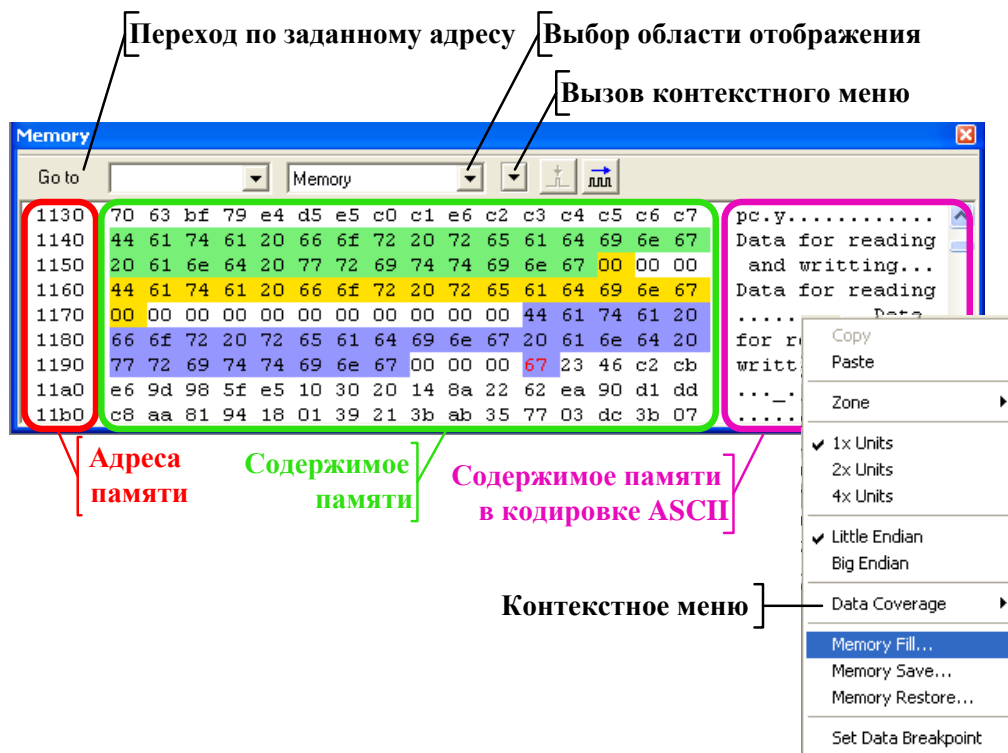


Рисунок 2.16 – Пример работы с окном памяти

В диалоговом окне заполнения (доступно из контекстного меню в окне памяти **View>Memory>Memory Fill**) можно записать в выбранную пользователем область памяти некоторое значение (таблицы 2.9, 2.10).

Таблица 2.9 – Описание опций по работе с диалоговым окном заполнения

Опция	Описание
Start Address	Определение стартового адреса – в двоичной, восьмеричной, десятичной или шестнадцатеричной системе счисления
Length	Определение размера – в двоичной, восьмеричной, десятичной или шестнадцатеричной системе счисления
Zone	Выбор области памяти
Value	Определение 8-битного значения, используемого для заполнения каждой ячейки памяти

Таблица 2.10 – Описание операций заполнения памяти

Операция	Описание
Copy	Значение Value будет скопировано в указанную область памяти
AND	Операция И будет выполнена над значением Value и текущим содержимым памяти перед записью результата в память
XOR	Операция ИСКЛЮЧАЮЩЕЕ ИЛИ будет выполнена над значением Value и текущим содержимым памяти перед записью результата в память
OR	Операция ИЛИ будет выполнена над значением Value и текущим содержимым памяти перед записью результата в память

Диалоговое окно сохранения памяти доступно из меню (Debug>Memory>Save) или из контекстного меню в окне памяти. Используется для сохранения содержимого выбранного участка памяти в файл.

Диалоговое окно восстановления памяти доступно из меню (Debug>Memory>Restore) или из контекстного меню в окне памяти и применяется для загрузки содержимого файла в msp430-txt-format, Intel-extended или Motorola s-record формате в выбранную область памяти.

Окно символьной памяти доступно во время работы IAR C-SPY отладчика из меню (**View>Symbolic Memory**). Показывает, как переменные со статической длительностью хранения выгружаются из памяти. Это может быть полезно для понимания проблем, вызванных переполнением буферов.

Таблица 2.11 – Описание операций, доступных при работе с окном символьной памяти

Операция	Описание
Go to	Переход к интересующей пользователя области памяти или символу
Zone display	Отображение доступных для отображения областей памяти
Previous	Возврат к предыдущему символу
Next	Переход к следующему символу

Область отображения показывает пространство памяти, в которой информация представлена столбцами, приведенными в таблице 2.12.

Окно регистров (на рисунке 2.17 выделено красным прямоугольником), доступное из меню (**View>Register**), отображает информацию о содержимом регистров процессора и позволяет пользователю их редактировать. Когда значение меняется, оно подчеркивается. Есть возможность открыть несколько

экземпляров этого окна, что является очень удобным, если необходимо наблюдать за различными группами регистров.

Таблица 2.12 – Формат представления информации в пространстве памяти

Столбец	Описание
Location	Адрес памяти
Data	Содержимое памяти в шестнадцатеричной форме. Информация группируется в соответствии с размером символа. Столбец можно редактировать
Variable	Имя переменной, требуется, чтобы переменная имела фиксированное местоположение в памяти. Локальные переменные не отображаются
Value	Значение переменной. Столбец можно редактировать
Type	Тип переменной

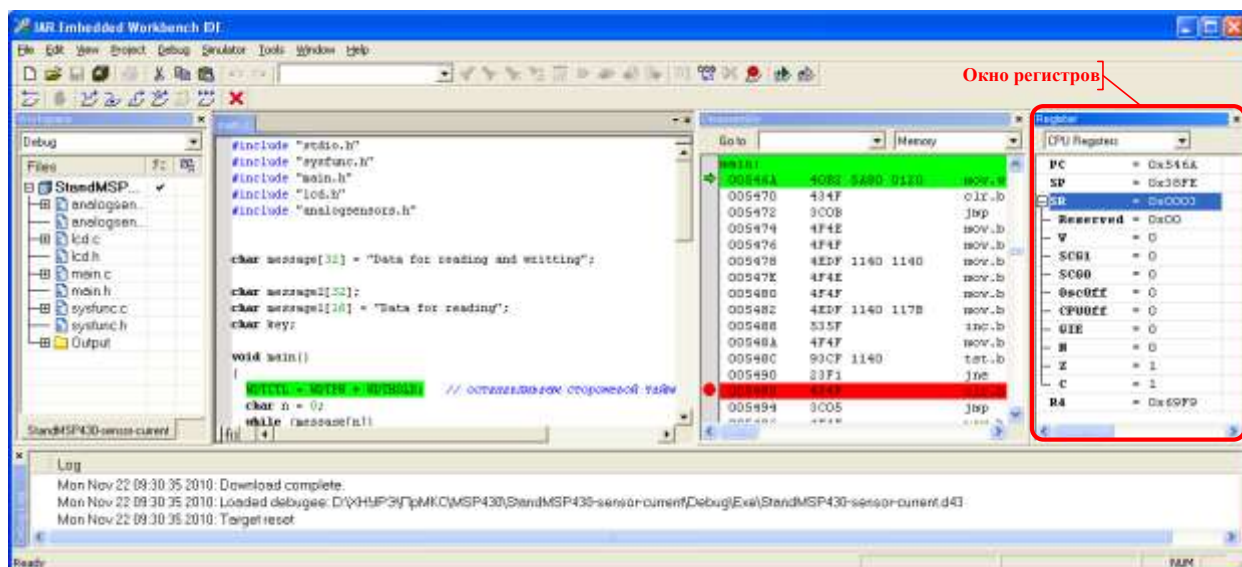


Рисунок 2.17 – Работа с отладчиком C-SPY

Пользователь может выбрать интересующую его группу регистров в окне регистров, используя выпадающий список.

Окно просмотра, доступное из меню (**View>Watch**), позволяет наблюдать за значениями C-SPY выражений или переменных. Можно просматривать, добавлять, изменять и удалять выражения в ОКНЕ ПРОСМОТРА. Древовидные структуры массивов, структур и объединений расширяемы, что дает возможность просмотреть любой из их элементов.


Каждый раз, когда выполнение в C-SPY останавливается, значение, которое изменилось с момента последнего останова, выделяется

подчеркиванием. В действительности, в момент каждого изменения памяти производится пересчет значений в ОКНЕ ПРОСМОТРА.

Окно локальных данных доступно из меню (**View>Locals**). В этом окне автоматически отображаются все локальные переменные и параметры функций.

Окно Auto доступно из меню (**View>Auto**). Автоматически отображает полезные наборы переменных и выражений, находящихся в текущем блоке, либо находящихся в непосредственной близости от него.

Окно текущего просмотра может быть вызвано из меню **View>Live Watch**. В этом окне отображаются значения выражений во время выполнения приложения. Переменные в выражениях должны быть расположены статически, так же, как и глобальные переменные.

В окне быстрого просмотра, доступного из меню (**View>Quick Watch**), можно просматривать значения переменной или выражения и вычислять выражения. Для вычисления выражения требуется ввести само выражение в текстовое поле **Expressions** и нажать на кнопку **Recalculate** .

Окно статических переменных доступно из меню (**View>Statics**) и отображает значения статических переменных. Как правило, это переменные в функциях и классах.

В области отображения содержится информация о значениях статических переменных, представленная в виде столбцов, показанных в таблице 2.13.

Таблица 2.13 – Формат представления информации о переменных

Столбец	Описание
Expression	Имя переменной. Этот столбец нельзя редактировать
Value	Значение переменной. Значения, которые были изменены, выделяются красным цветом. Этот столбец можно редактировать
Location	Место в памяти, где хранится переменная
Type	Тип переменной

Диалоговое окно выбора статических переменных, доступное из контекстного меню в окне статических переменных, используется для выбора требуемых пользователю переменных, которые будут отображены в окне статических переменных.

Опция «*Show all variables with static storage duration*» используется для отображения всех переменных в окне статических переменных, в том числе и новых переменных, которые добавились к приложению между отладочными

сеансами.

Опция «*Show selected variables only*» применяется для выбора требуемых переменных, которые будут отображены в окне статических переменных. Следует заметить, что в случае добавления переменной в приложение между двумя сеансами отладки, она не будет автоматически отображена. Только, если выбран флажок рядом с переменной, она будет отображена.

Окно стека вызовов доступно из меню (**View>Call Stack**). Это окно (на рисунке 2.18 выделено красным прямоугольником) показывает стек вызовов Си функций с текущей функцией на его вершине. Чтобы проверить вызов функции, на ней нужно дважды щелкнуть левой клавишей мыши. Если команда **Step Into** заходит в вызов функции, имя этой функции будет отображено на серой панели в верхней части окна. Это особенно полезно для неявных вызовов функций, таких как C++ конструкторы, деструкторы и операторы.

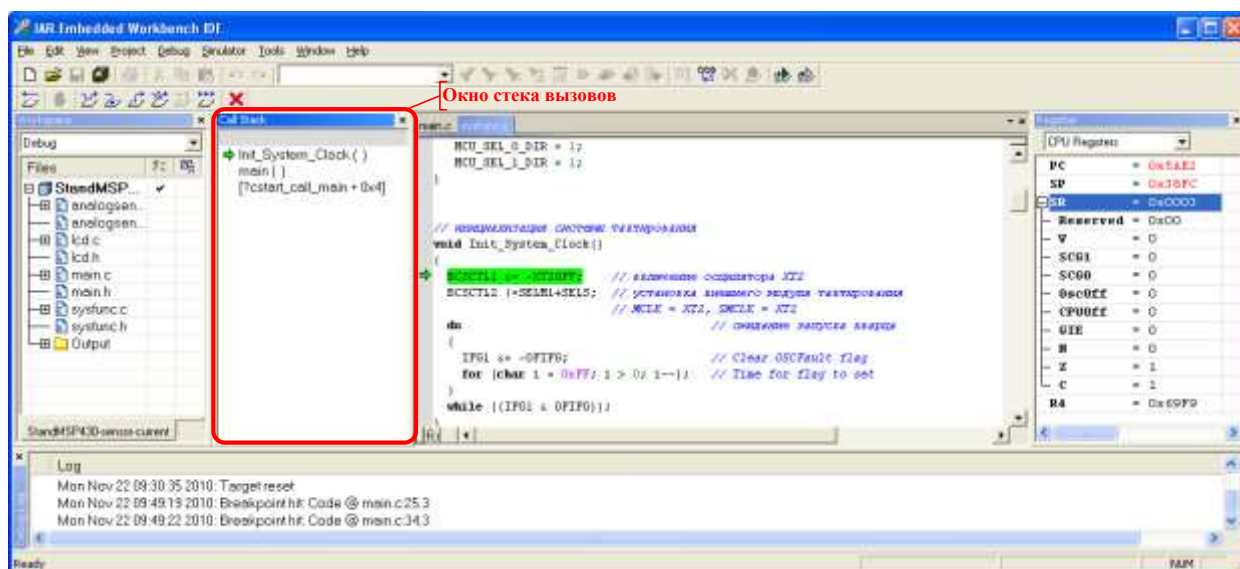


Рисунок 2.18 – Пример работы с окном стека вызовов

Окно анализа кода (на рисунке 2.19 выделено красным прямоугольником) доступно из меню (**View>Code Coverage**) сообщает о состоянии текущего анализа прохождения кода, то есть, какие части кода были выполнены хотя один раз с момента начала анализа. Компилятор генерирует подробную пошаговую информацию в форме точек шага в каждом выражении, а также при каждом вызове функции. В отчете содержится информация обо всех модулях и функциях. В нем сообщается сумма всех точек шага, в процентах, которые были выполнены и выдается список всех точек шага, которые не были выполнены до той точки, где приложение было остановлено.

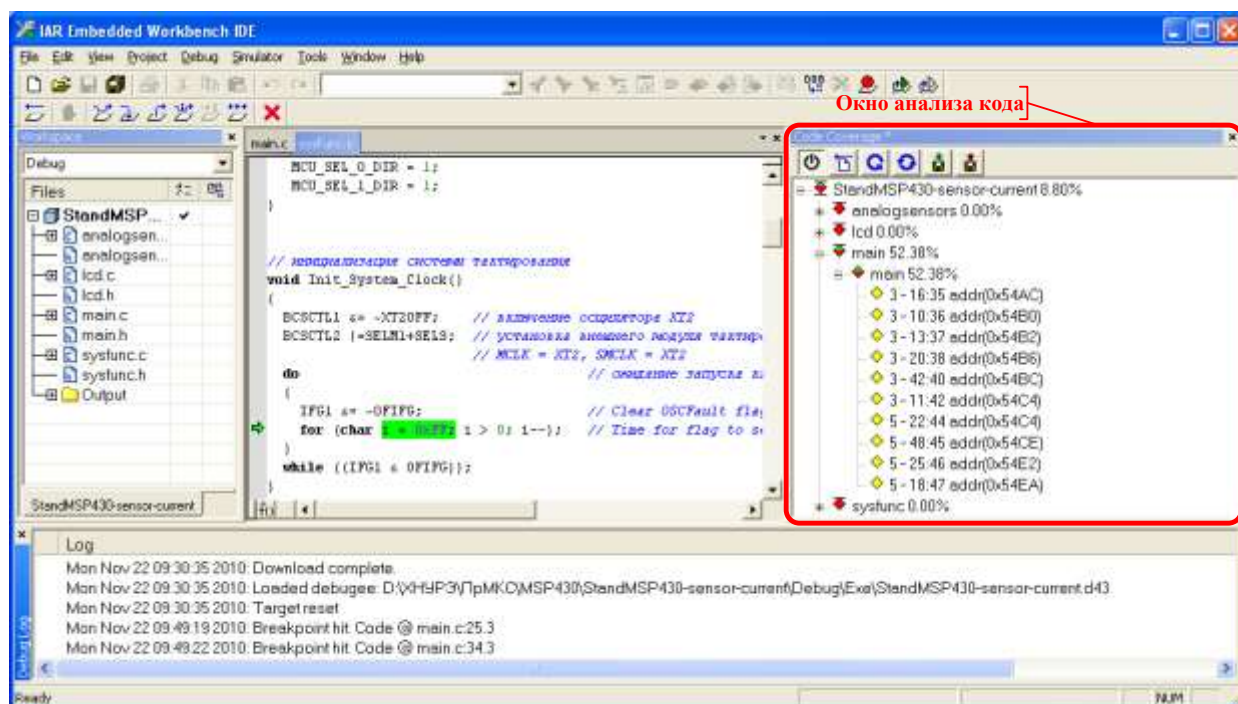






Рисунок 2.19 – Пример отладки проекта

Таблица 2.14 – Описание пунктов меню окна анализа кода






Пункт меню	Описание
Activate 	Включение / выключение прохождения кода во время выполнения
Clear 	Удаление информации о прохождении кода. Все точки шага помечены как невыполненные
Refresh 	Обновление информации о прохождении кода и обновляет само окно. Все точки шага, выполненные с момента последнего обновления, удаляются из дерева
Auto-refresh 	Включение / выключение автоматической перезагрузки информации о прохождении кода
Save As	Сохранение текущей информации о прохождении кода в текстовый файл

Окно профилирования доступно из меню (**View>Profiling**) и отображает информацию о времени выполнения функций в приложении (таблица 2.15). Время измеряется от момента входа в функцию и до момента возврата из функции.

Окно стека является окном памяти, которое отображает содержимое стека. Кроме того, некоторые проверки целостности стека могут быть

выполнены для обнаружения и предупреждения проблем с переполнение стека. Например, окно стека является очень полезным для определения оптимального размера стека. На рисунке 2.20 окно стека выделено красным прямоугольником.

Таблица 2.15 – Описание пунктов меню окна профилирования

Пункт меню	Описание
Activate 	Включение / выключение измерения времени во время выполнения
New measurement 	Начало нового измерения. После нажатия этой кнопки отображаемые значения обнуляются
Show details 	Отображение более детальной информации о выбранных из списка функциях
Refresh 	Обновление информации в окне
Auto refresh 	Включение / выключение автоматической перезагрузки информации о временных характеристиках функций
Save As	Сохранение текущей информации о временных характеристиках функций в файл

В таблице 2.16 показано, каким именно образом представляются данные в окне профилирования.

Таблица 2.16 – Формат представления данных в окне профилирования

Столбец	Описание
Function	Имя каждой функции
Calls	Число вызовов каждой из функций
Flat Time	Общее время, проведенное в каждой из функций и в циклах без учета всех вызовов функций, сделанных из этой функции
Accumulated Time	Время, проведенное в каждой из функций и в циклах без учета всех вызовов функций, сделанных из этой функции

Прежде чем открыть окно стека, нужно убедиться в том, что оно доступно (т.е. активен пункт меню): **Project>Options>Debugger>Plugins** и выбрать **Stack** из списка плагинов. В C-SPY можно открыть окно стека, выбрав **View>Stack**. Существует возможность открытия нескольких окон стека, в каждом из которых будут отображены различные стеки (если они доступны) либо один и тот же стек с различными настройками отображения.

Выпадающее меню стека позволяет выбрать необходимый пользователю

стек, если используемый микроконтроллер имеет несколько стеков.

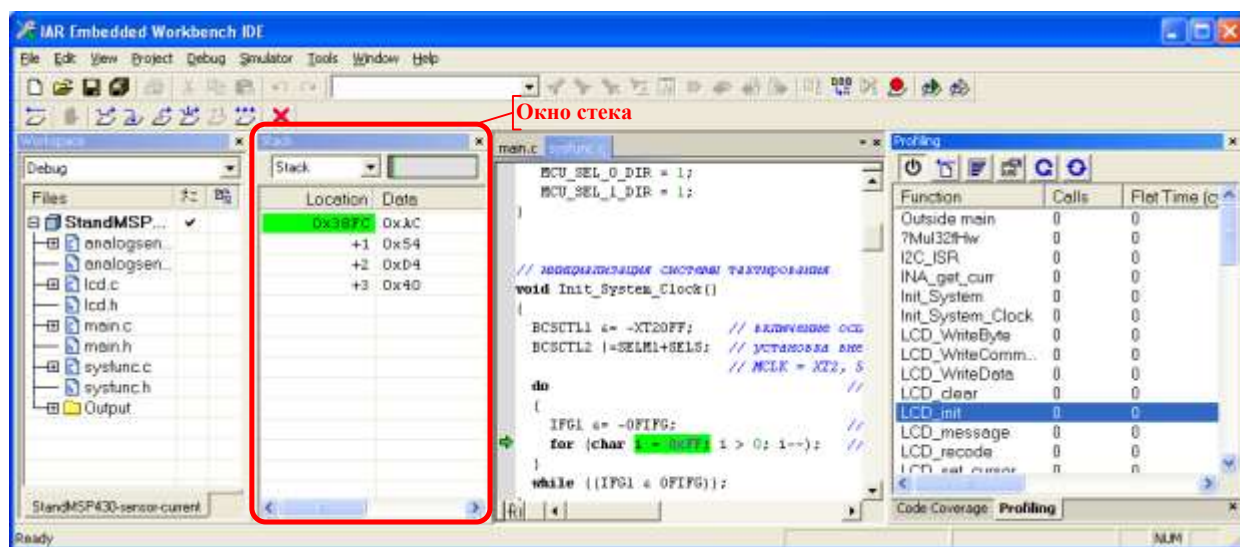


Рисунок 2.20 – Пример работы с окном стека

Графическая панель стека. В верхней части окна, панель стека отображает состояние стека в графическом виде. Описание формата представления стековой памяти приведено в таблице 2.17. Для того, чтобы просмотреть панель стека, нужно убедиться в то, что она доступна: **Tools>Options>Stack** и выбрать опцию **Enable graphical stack display and stack usage tracking**.

Таблица 2.17 – Формат представления стековой памяти

Столбец	Описание
Location	Отображение места в памяти. Адреса расположены в порядке возрастания. Адрес, на который ссылается указатель стека, другими словами, вершина стека, подсвечивается зеленым цветом
Data	Отображение содержимого ячейки памяти в выбранной области
Variable	Отображение имени переменной, если есть локальная переменная в выбранной области. Переменные отображаются только в том случае, если они локально объявлены в функции и располагаются в стеке, а не в регистрах
Value	Отображение значения переменной, которая находится в столбце Variable
Frame	Отображение имени функции, соответствующей вызову.

В левой части панели стека представлено дно стека, другими словами,

положение указателя стека, когда стек пуст. Справа показано окончание памяти, зарезервированной для стека. Зеленая линия соответствует текущему значению указателя стека. Часть стековой памяти, которая была использована во время выполнения, выделена темно серым цветом, а неиспользованная – светло серым. Графическая панель стека выделяется красным цветом, когда превышает пороговый уровень стека, который можно задать.

2.4 Встроенные функции компилятора языка Си IDE IAR Embedded Workbench

В таблице 2.18 приведено описание некоторых встроенных функций компилятора IAR Embedded Workbench (файл `intrinsics.h`).

Таблица 2.18 – Встроенные функции компилятора IAR Embedded Workbench

Встроенная функция	Описание
1	2
<code>__bcd_add_type</code>	Сложение чисел в формате BCD (<code>type</code> задает тип данных и может быть <code>short</code> , <code>long</code> или <code>long long</code>) <code>/* c = 0x19 */</code> <code>c = __bcd_add_short(c, 0x01);</code> <code>/* c = 0x20 */</code>
<code>__bic_SR_register</code>	Сброс битов в регистре статуса SR <code>void __bic_SR_register(unsigned short);</code> аргумент функции является битовой маской.
<code>__bic_SR_register_on_exit</code>	Сброс битов в регистре статуса SR при выходе из обработчика прерывания или из функции монитора <code>void __bic_SR_register_on_exit(unsigned short);</code> аргумент функции является битовой маской.
<code>__bis_SR_register</code>	Установка битов в регистре статуса SR <code>void __bis_SR_register(unsigned short);</code> аргумент функции является битовой маской

Продолжение таблицы 2.18

1	2
<code>__bis_SR_register_on_exit</code>	Установка битов в регистре статуса SR при выходе из обработчика прерывания или из функции монитора void <code>__bis_SR_register_on_exit(unsigned short);</code> аргумент функции является битовой маской.
<code>__data16_read_addr</code>	Чтение 20-ти битных данных из SFR регистра, расположенного по 16-ти битному адресу address unsigned long <code>__data16_read_addr(unsigned short address);</code>
<code>__data16_write_addr</code>	Запись 20-ти битных данных data в регистр SFR, расположенный по 16-ти битному адресу address void <code>__data16_write_addr(unsigned short address, unsigned long data);</code>
<code>__data20_read_type</code>	Чтение данных, расположенных в 1-Мбайтном адресном пространстве (type задает тип данных и может быть unsigned char, unsigned short или unsigned long) unsigned type <code>__data20_read_type(unsigned long address);</code>
<code>__data20_write_type</code>	Запись данных в 1-Мбайтном адресном пространстве (type задает тип данных и может быть unsigned char, unsigned short или unsigned long) void <code>__data20_write_type(unsigned long address, unsigned type);</code>
<code>__delay_cycles</code>	Вставка временной задержки <code>__delay_cycles(unsigned long n);</code> Параметр n определяет длительность задержки (в циклах процессора)
<code>__disable_interrupt</code>	Запрет прерываний, вставка команды CLI void <code>__disable_interrupt(void);</code>

Продолжение таблицы 2.18

1	2
<code>__enable_interrupt</code>	Разрешение прерываний, вставка команды SEI <code>void __enable_interrupt(void);</code>
<code>__get_R4_register</code>	Возвращает значение регистра R4
<code>__get_R5_register</code>	Возвращает значение регистра R5
<code>__get_SP_register</code>	Возвращает значение указателя стека SP
<code>__get_SR_register</code>	Возвращает значение регистра статуса SR
<code>__get_SR_register_on_exit</code>	Возвращает значение регистра статуса SR, которое будет восстановлено из стека при выходе из текущего прерывания
<code>__low_power_mode_n</code>	Переход в режим пониженного энергопотребления
<code>__low_power_mode_off_on_exit</code>	Выход из режима пониженного энергопотребления при возврате из текущего прерывания
<code>__no_operation</code>	Команда NOP
<code>__set_R4_register</code>	Записать значение в регистр R4
<code>__set_R5_register</code>	Записать значение в регистр R5
<code>__set_SP_register</code>	Записать значение в указатель стека SP
<code>__swap_bytes</code>	Команда обмена байтами SWPB

Для использования встроенных функций в приложении нужно подключить заголовочный файл `intrinsics.h`. Обратите внимание на то, что имена встроенных функций начинаются с двойного символа подчеркивания, например:

`__no_operation`

Более подробная информация о встроенных функциях компилятора IAR Embedded Workbench содержится в документе «MSP430 IAR C/C++ Compiler Reference Guide» файл `EW430_COMPILERREFERENCE.PDF` [с. 215-225].

3 РУКОВОДСТВО ПО РАЗРАБОТКЕ ПРОГРАММ НА ЯЗЫКЕ Си ДЛЯ МИКРОКОНТРОЛЛЕРНЫХ СИСТЕМ

Вводные понятия

Прежде всего, рассмотрим такие вводные понятия как комментарии, ключевые слова, идентификаторы, литералы, операторы и знаки пунктуации.

Комментарий – это некоторый поясняющий текст, который при компиляции не учитывается. Комментарии бывают многострочными (начинаются с символов `/*` и заканчиваются символами `*/`) и однострочными (начинаются с символов `//`). В последнем случае комментарием считается вся часть строки, расположенная справа от символов `//`.

Примеры:

```
/* Многострочный комментарий часто размещают в начале
файла, где он содержит имя автора и описание программы */
#include "stdio.h"//Подключаем заголовочный файл
stdio.h
```

Идентификатор – это последовательность букв, цифр и символов подчеркивания `"_"`, которая не должна начинаться с цифры, используемая для именования различных программных элементов, наподобие переменных, констант, функций, типов и т.д. Регистр букв имеет значение.

Ключевое слово – это зарезервированное слово четко определенного назначения. Ключевые слова не могут использоваться в качестве идентификаторов:

```
asm, auto;
bit, bool, break;
case, char, const, continue;
default, defined, do, double;
else, enum, explicit, extern;
false, float, for;
goto; if, inline, int;
long;
register, return;
short, signed, sizeof, static, struct, switch;
true, typedef;
union, unsigned;
void; while.
```

Литерал – постоянное значение некоторого типа, используемое в

выражениях. Примеры числовых литералов:

10 – число 10 в десятичной форме;
0xA – число 10 в шестнадцатеричной форме (префикс 0x);
0b1010 – число 10 в двоичной форме (префикс 0b);
012 – число 10 в восьмеричной форме (префикс 0);
10.5 – число с плавающей точкой;
105e-1 – число 10,5 в экспоненциальной форме;
10U – беззнаковая константа (суффикс U);
10L – знаковая константа (суффикс L).

Символьные литералы заключаются в одинарные кавычки, например, 'B','* '. Для обозначения непечатаемых и специальных символов в литералах используются так называемые escape-последовательности:

'\a ' – звуковой сигнал;
'\b ' – клавиша <Backspace>;
'\f ' – прогон листа;
'\n ' – символ перевода строки;
'\r ' – возврат каретки;
'\t ' – горизонтальная табуляция;
'\v ' – вертикальная табуляция;
'\0 ' – нулевой символ;
'\\ ' – обратная косая;
'\'' – апостроф.

Кроме того, любой символ можно представить с помощью литерала по его ASCII-коду, например, литерал '\t' равнозначен '\x09' (в шестнадцатеричном представлении).

Строковые литералы ограничиваются двойными кавычками, а в памяти хранятся как последовательности символов, заканчивающиеся нулевым символом '\0'. Специальные символы внутри строки должны предваряться обратной косой ("\""). Примеры строковых литералов:

" " – пустая строка: один символ '\0';
"B" – два символа: 'B' и '\0';
"A\tB\n" – пять символов: 'A', табуляция, 'B', перевод строки, '\0'.

Оператор – это символ, указывающий компилятору, какие действия выполнить над операндами. Некоторые символы могут трактоваться по-разному в зависимости от контекста. Например, знак "-" может использоваться для изменения знака числа или в качестве оператора вычитания. Операторы, соединяющие операнды, представляют собой **выражения**. Выражения могут

быть заключены в круглые скобки и отделяются друг от друга символом точки с запятой (";").

Приоритетность выполнения операторов в выражениях языка Си указана в таблице 3.1 (приоритет с меньшим номером уровня – выше).

Таблица 3.1 – Приоритетность выполнения операторов в выражениях языка Си

Уровень	Операторы	Категория	Описание
1	2	3	4
1	()		Круглые скобки
	[]		Элемент массива
	.	Доступ к данным	Обращение к элементу структуры, например: PORTB .1 – разряд 1 порта В
	->		Обращение к элементу структуры, определенной указателем, например: pStruct->x – элемент x структуры, на которую указывает pStruct
	++, -- (постфиксы)	Арифметические	Операторы автоинкремента и автодекремента после того как выражение, в котором задействованы соответствующие операнды, вычислено. Примеры: a = b++; равнозначно a = b; b = b+1; a = b--; равнозначно a = b; b = b-1;
2	++,-- (префиксы)		Операторы автоинкремента и автодекремента перед тем как выражение, в котором задействованы соответствующие операнды, будет вычислено. Примеры: a = ++b; равнозначно b = b+1; a = b; a = --b; равнозначно b = b-1; a = b;
	!		Логические Логическое(унарное) отрицание
	~		Поразрядные Поразрядное отрицание
	&		Доступ к данным Адрес
3	+, - (унарные)	Арифметические	Изменение знака операнда
	* (унарный)	Доступ к данным	Разыменование указателя

Продолжение таблицы 3.1

1	2	3	4
4	*	Арифметические	Умножение
	(бинарный)		Деление
	/		Остаток от деления
5	+	Арифметические	Сложение и вычитание
	-		
6	<<	Поразрядные	Поразрядный сдвиг влево
	>>		Поразрядный сдвиг вправо
7	<, >, <=, >=	Сравнения	Меньше, больше и т.д.
8	==, !=		Равно, не равно
9	&	Поразрядные	Поразрядное "И"
10	^	Поразрядные	Поразрядное "Исключающее ИЛИ"
11		Поразрядные	Поразрядное "ИЛИ"
12	&&	Логические	Логическое "И"
13		Логические	Логическое "ИЛИ"
14	=	Присваивание	Возможны также сочетания оператора присваивания с арифметическими и поразрядными операторами, например: a += b; равнозначно a = a + b; a *= b+c; равнозначно a = a * (b + c);

Объединенные по некоторому признаку последовательности выражений заключаются в фигурные скобки { }. Так, к примеру, обозначаются границы функций, а также блоки выражений в циклических и условных конструкциях.

Структура программы на Си

Структуру программы, написанной на языке Си, рассмотрим на примере test.c.

```
//Директивы препроцессора
#include "stdio.h"//Подключаем заголовочный файл
stdio.h
//Объявления глобальных типов, переменных и констант
...
//Функции
Function1
```

```

{
//Объявления локальных типов, переменных и констант
...
//Операторы
...
}

...
FunctionN
{
//Объявления локальных типов, переменных и констант
...
//Операторы
...
}

//Главная функция программы
void main()
{
//Объявления локальных типов, переменных и констант
...
//Операторы
...
}

```

Программы обычно начинаются с **директив препроцессора** (начинаются с символа "#"), которые, по сути, не являются конструкциями языка Си и обрабатываются до фактической компиляции программы. Их смысл – подстановка некоторого кода в программу. Так, к примеру, очень часто используется директива `#include`, которая включает в файл с исходным кодом программы текст внешнего заголовочного файла (с расширением `.h`). **Заголовочные файлы** содержат определения глобальных типов, констант, переменных и функций.

Типы данных, переменные, константы

Тип данных определяет диапазон допустимых значений и пространство, отводимое в памяти данных для переменных, констант и результатов, возвращаемых функциями. В различных компиляторах языка С могут

использоваться собственные типы данных, однако все они базируются на стандартных типах, перечисленных в таблице 3.2.

Таблица 3.2 – Стандартные типы данных языка Си для микроконтроллерных систем

Тип	Название	Размер, в битах	Диапазон значений
bit	Бит	1	0, 1
char	Символ	8	-128...127
int	Целое число	16	-32768...32767
float	Вещественное число	32	$\pm 1,175 \cdot 10^{-38} \dots \pm 3,402 \cdot 10^{38}$
long int	Длинное целое	32	-2147483648...2147483647
unsigned char	Беззнаковый символ или логическое значение	8	0...255, TRUE, FALSE
unsigned int	Беззнаковое целое	16	0...65535
unsigned long int	Беззнаковое длинное целое	32	0...4294967295

Пользовательские типы

Язык Си позволяет, кроме стандартных, объявлять собственные типы. Для этого используется ключевое слово `typedef`, например:

```
typedef unsigned char byte;    //объявляем тип byte
typedef unsigned int word;    //объявляем тип word
```

Другими словами, для объявления пользовательского типа используется конструкция вида

```
typedef                                стандартный_тип
идентификатор_пользовательского_типа;
```

Переменная – это именованная величина определенного типа, которая может изменяться в ходе выполнения программы. Для объявления переменных (т.е., выделения для них памяти) в программе на С используется следующая конструкция:

```
тип переменной идентификатор1, идентификатор2, ...;
```

Например:

```
int i;                                //Объявление целочисленной
переменной i
```

```
char c1, c2;           //Объявление          СИМВОЛЬНЫХ
переменных c1 и c2
```

Пользовательский тип, используемый при объявлении переменной, должен быть объявлен выше в тексте программы.

Для доступа к переменной в программе используется соответствующий идентификатор (обязательно после объявления переменной). Значения, присваиваемые переменной, должны соответствовать ей по типу (или правилам приведения типов, рассматриваемым далее). Примеры:

```
i = 2;      //Ошибка! Переменная i еще не объявлена
int i;      //Объявление целочисленной переменной i
float f;    //Объявление вещественной переменной f
i = 2;      //Переменной i присвоено значение 2
f = 3.3     //Переменной f присвоено значение 3,3
f = i;      //Переменной f присвоено значение
переменной i
// (в данном случае будет выполнено автоматическое
// приведение типов, т.е. f = 2.0)
```

По области видимости переменные могут быть глобальными и локальными. К **глобальным переменным** имеют доступ все функции программы. Такие переменные объявляются в программе перед объявлением всех функций. К **локальным переменным** имеет доступ только та функция, в которой они объявлены.

Область видимости переменных

Имена переменных обладают определенной **областью видимости**, которая подразумевает, что компилятор использует переменные в соответствии с тем, где они находятся. Имена переменных, объявленных внутри функции, имеют область видимости, ограниченную конкретной функцией. Например, в нескольких функциях можно объявить переменную `int i`, которая в каждой функции не будет иметь никакой связи с аналогичными переменными в других функциях. Точно так же и переменная, объявленная внутри блока (ограничен фигурными скобками `{}`), остается локальной по отношению к этому блоку.

Глобальные переменные имеют область видимости, которая начинается от места их объявления и продолжается до конца программного файла. Для

того чтобы глобальную переменную можно было использовать в других файлах, ее нужно объявить с помощью ключевого слова `extern`:

```
extern int n;
```

Объявленную таким образом переменную, прежде, чем ее использовать, следует обязательно инициализировать во внешнем файле некоторым значением.

Использование переменных, объявленных с помощью ключевого слова `extern`, часто приводит к ошибкам в программах, поэтому применяйте их только тогда, когда нет другой альтернативы, и будьте при этом очень внимательны.

Константа – это именованная величина определенного типа, которая, в отличие от переменной, не может изменяться в ходе выполнения программы, а имеет конкретное значение, определенное в момент объявления. Для объявления констант в программе на С используется следующая конструкция:

```
const тип_константы идентификатор = значение;
```

Например:

```
const int i = 10;           //Объявление целочисленной
константы i
```

Величина, объявленная как константа, будет размещена компилятором в памяти программ, а не в ограниченной области переменных в RAM. Для доступа к константе используется ее идентификатор.

Перечислимый тип – это объявление списка целочисленных констант, которые можно явно не инициализировать (в этом случае компилятор считает, что первая константа в списке принимает значение 0, вторая – 1 и т.д.). Для подобного объявления используется ключевое слово `enum`:

```
int n;
enum (zero, one, two);    //zero = 0;   one = 1;
two = 2
n = one;                  //n = 1
```

Если требуется изменить начальное значение для списка констант, то можно указать его явно при объявлении, например:

```
enum (three = 3, four, five); //three = 3; four = 4;  
five = 5
```

Приведение типов – это принудительное преобразование значения одного типа к другому, совместимому с исходным. Это важно при выполнении арифметических операций, когда полученные значения могут выходить за допустимые пределы.

Приведение типов бывает явным и неявным. Неявное приведение типов используется в операторах присваивания, когда компилятор сам выполняет необходимые преобразования без участия программиста. Для явного приведения типа некоторой переменной перед ней следует указать в круглых скобках имя нового типа, например:

```
int X;  
int Y = 200;  
char C = 30;  
X = (int)C * 10 + Y; //Переменная C приведена к типу  
int
```

Если бы в этом примере не было выполнено явное приведение типов, то компилятор предположил бы, что выражение $C * 10$ – это восьмиразрядное умножение (разрядности типа `char`) и вместо корректного значения 300 (0x12C) в стек было бы помещено урезанное значение 44 (0x2C). Таким образом, в результате вычисления выражения $C * 10 + Y$ переменной `X` было бы присвоено значение 244, а не корректное 500. В результате приведения типа переменная `C` распознается компилятором как 16-тиразрядная, и описанной выше ошибки не возникает.

Оператор `sizeof` применяется для вычисления размера области памяти (в байтах), отводимой под некоторую переменную, результат выражения или тип. Например:

```
int a;  
float f;
```

```
a = sizeof(int); //a = 2
a = sizeof(f);   //a = 4
f = 3.3;
a = sizeof(f + a); //a = 4,           поскольку тип
результата - float
```

Функция представляет собой "контейнер", в котором выполняется некоторый фрагмент программного кода. Использование функций упрощает написание и отладку программ, поскольку в них удобно размещать повторяющиеся группы операторов. Любая программа на языке С содержит главную функцию под названием `main ()`. Эта функция при запуске программы выполняется первой.

Пользовательские функции определяются после директив препроцессора и глобальных объявлений типов, переменных и констант в файле с исходным кодом или в заголовочном файле. При этом используется следующий синтаксис:

```
Тип_возвращаемого_значения
Имя_функции(Список_параметров)
{
    //Тело функции
}
```

Предварительное объявление функции может отсутствовать. В этом случае она доступна только внутри того файла, в котором определена.

В качестве типа возвращаемого значения может использоваться ключевое слово `void`. Это означает, что функция не возвращает никакого значения (в некоторых языках программирования такие функции называют процедурами).

Функцию вызывают по ее имени с указанием в круглых скобках перечня передаваемых параметров (если их нет, то в скобках ничего не указывается), например:

```
void Function1(int n, char c)
{
    ...
}
int Function2()
```

```

{
    ...
}
int main()
{
    int x;
    char y;
    Function1(x, y);
    x = Function2();
    return x;
}

```

Параметры – это идентификаторы, которые могут использоваться внутри функции. Вместо них подставляются соответствующие значения, указанные при вызове функции (если в функцию передается более одного параметра, то они отделяются друг от друга запятыми как при объявлении, так и при вызове).

Значения, переданные в функцию, фактически не изменяются, а просто копируются в параметры, который в этом смысле выполняют роль локальных переменных. При этом следует следить за тем, чтобы тип передаваемых значений соответствовал типу параметров, объявленных в заголовке функции.

Возвращаемые значения

Если функция предназначена для возврата значения некоторого типа, то для этого в ее теле используют ключевое слово `return`, после которого (через пробел) указывают возвращаемое значение. При этом все операторы после слова `return` игнорируются, и происходит возврат в вызывающую функцию.

Пример:

```

...
int power3(int n)
{
    return n*n*n;
}
void main()
{

```

```
int x;
x = power3(2);    //x = 8
}
```

Слово `return` может также использоваться без указания возвращаемого выражения. В этом случае оно просто означает выход из функции.

Прототипы функций

В обычном варианте функции используются только после их определения, однако бывают случаи, когда функции вызывают друг друга, и организовать их "правильное" определение невозможно. Обойти подобную проблему позволяют **прототипы функций**, которые представляют собой объявление до определения. Такое объявление представляет собой только заголовок функции, причем в списке параметров указывают только типы, без идентификаторов, например:

```
...
int f1(int);
void f2(int, int);
int f1(int x)
{
    ...
    return x;
}
void f2(int a, int b)
{
    ...
}
```

Прототипы функций часто используются в заголовочных файлах, включаемых в текст программы с помощью директивы препроцессора `#include`.

Классы памяти при объявлении локальных переменных

Локальные переменные могут быть объявлены внутри функций как принадлежащие к одному из трех классов памяти:

auto (значение по умолчанию, можно явно не указывать) – при объявлении переменная не инициализируется никаким значением (значение –

текущее содержимое области памяти, отведенной под переменную); при выходе из функции переменная удаляется из памяти;

static – статическая переменная доступна только в пределах функции, хотя память для нее выделяется в пространстве глобальных переменных; при первом обращении к функции инициализируется нулевым значением, и после выхода из функции из памяти не удаляется (таким образом, при последующих обращениях к функции в ней содержится старое значение);

register – аналог автоматической локальной переменной за тем исключением, что компилятор попытается выделить для нее не область памяти данных, а рабочий регистр микроконтроллера, что значительно ускоряет обращение к значению переменной.

Пример использования статической переменной:

```
...
int plus5()
{
    static int x;
    return x + 5;
}
void main()
{
    int y;
    y = plus5();      //y = 5
    y = plus5();      //y = 10
    y = plus5();      //y = 15
}
```

Рекурсия – это вызов функцией самой себя. Эта возможность бывает трудна в понимании, и потому не удивительно, что эксперты по языку С так любят рекурсивные функции. Пример рекурсивного вызова:

```
void f1(int n)
{
    int x;
    f1(x);
}
```


Несмотря на сложность восприятия, рекурсия довольно часто используется в стандартных библиотечных функциях, а также во многих алгоритмах сортировки. Тем не менее, при программировании микроконтроллеров использование рекурсии, как правило, чревато проблемами из-за ограниченного объема оперативной памяти. Дело в том, что при каждом вызове рекурсивной функции часть памяти расходуется на сохранение данных, помещаемых в стек. Эти данные хранятся там до тех пор, пока не будет выполнен возврат из функции. Таким образом, когда рекурсивная функция снова и снова вызывает саму себя, в стеке остается все меньше и меньше свободной памяти.

Можно сказать, что в подавляющем большинстве случаев использование рекурсии при программировании микроконтроллеров – это надежный способ быстрого и непредсказуемого заполнения стека. Это очень часто приводит к возникновению ошибочного состояния, называемого переполнением стека. Область стека обычно размещается в верхней части памяти и растет "вниз", тогда как область переменных размещается в нижней части памяти и растет "вверх". Поэтому если объем данных, помещаемых в стек, превысит размер области стека, эти данные могут достигнуть области переменных и затереть собой ее значения. При этом ошибку переполнения стека не всегда легко выявить, поскольку она может проявляться лишь периодически.

Структура – это особый тип данных, состоящий из нескольких разнотипных переменных (полей). В общем случае объявление структуры имеет следующий вид:

```
struct имя_структуры {  
    тип поле_1;  
    ...  
    тип поле_N;  
};
```

Как и любой другой тип, структуру можно в дальнейшем использовать для объявления переменных, например:

```
struct MyStructure      {           //Объявление структуры  
MyStructure  
    int Field1;
```

```

char Field2;
float Field3;
};
// Объявление переменных YourStruct и
// OurStruct типа MyStructure
struct MyStructure YourStruct, OurStruct;

```

Допускается инициализация полей непосредственно при объявлении переменных-структур с помощью перечня значений в фигурных скобках, например:

```

struct DATE {
int Day;
int Month;
int Year;
}
struct DATE MyBirthday = {7, 8, 1974};

```

Для доступа к полям структуры в программе используют запись вида имя_структуры.поле. То есть, в представленном выше примере структуры DATE для инициализации полей можно было воспользоваться следующими операторами:

```

MyBirthday.Day = 11;
MyBirthday.Month = 2;
MyBirthday.Year = 1976;

```

Структуры, в свою очередь, могут быть полями других структур, например:

```

struct ME {
char MyName[30]; //Строка длиной 30 символов - имя
struct DATE MyBirthday; //Структура, хранящая день
рождения
}
...
struct ME MyData;

```

```
MyData.MyName = "John Smith";
MyData.MyBirthday.Day = 7;
MyData.MyBirthday.Month = 8;
MyData.MyBirthday.Year = 1974;
```

Структуры могут выступать в качестве параметров функций, а также возвращаемого результата. Ниже представлен пример функции, возвращающей структуру типа DATE:

```
struct DATE January1(int CurYear)
{
    struct DATE Jan01;
    Jan01.Day = 1;
    Jan01.Month = 1;
    Jan01.Year = CurYear;
    return Jan01;
}
...
struct DATE BeginOfTheYear;
BeginOfTheYear = January1(2009) ;
```

Указатели и адреса переменных

Указатель – это переменная, содержащая адрес некоторого элемента данных (переменной, константы, функции, структуры). В языке С указатели тесно связаны с обработкой массивов и строк, которые будут рассмотрены далее.

Для объявления переменной как указателя используется оператор *:

```
int *p;    //p - указатель на целое число
```

Для присвоения адреса некоторой переменной указателю используется оператор &, например:

```
char *p;    //указатель на символ
char c;    //символьная переменная
c = ' A ';
p = &c;    // p содержит адрес переменной c (указывает
```

на 'A')

Для того чтобы извлечь значение переменной, на которую указывает указатель, используется оператор разыменования *.

```
char *p;  
char c, b;  
c = ' A ' ;  
p = &c;  
b = *p;    //Теперь b = ' A '
```

Аналогичным образом, этот оператор можно использовать и для присвоения некоторого значения переменной, на которую указывает указатель:

```
char *p;  
char c, b;  
b = ' A ' ;  
p = &c;  
*p = b;    //Теперь c = 'A '
```

Применительно к программированию микроконтроллеров, указатели можно использовать, к примеру, для записи данных в порт ввода/вывода. Предположим, регистр данных порта расположен в памяти по адресу 0x16. В этом случае, для записи в него значения 0xFF можно воспользоваться следующим фрагментом программного кода:

```
unsigned char *thePort;  
thePort = 0x16;  
*thePort = 0xFF;
```

Передача в функции параметров по ссылке

С помощью указателей, используемых в качестве параметров функций, можно организовать возврат более одного значения. Рассмотрим следующий пример:

```
int SumAndDiv(int *a, int *b)  
{
```

```

int bufA,  bufB;
bufA = *a;
bufB = *b;
*a = bufA / bufB; //Указатель ссылается на результат
                  //целочисленного деления без остатка
*b = bufA % bufB; //Указатель ссылается на остаток от
                  //целочисленного деления
return bufA + bufB; //Функция возвращает сумму
    }
    ...
int v1, v2, sum;
v1=10;
v2=3
sum = SumAndDiv(&v1, &v2); //sum = 13; v1 = 3; v2 = 1

```

В функции SumAndDiv вначале сохраняются в буферных переменных значения, на которые указывают указатели a и b. Затем в переменную, на которую указывает указатель a записывается результат деления переданных в функцию значений без остатка, а в переменную, на которую указывается указатель b – остаток от такого деления. При вызове функции SumAndDiv в нее передаются адреса переменных v1 и v2, которым в теле функции соответствуют указатели a и b.

Указатели на структуры

На структуры можно создавать указатели точно так же, как и для любого другого типа данных. Пример применения такого указателя:

```

struct  DATE  {
int Day;
int Month;
int Year;
    }
    ...
struct DATE MyBirthday, *dateP;
dateP = &MyBirthday; //dateP указывает на
                     //структуру MyBirthday

```

В таком случае для доступа к полям структуры через указатель используется оператор `->`:

```
dateP->Day = 7;
dateP->Month = 8;
dateP->Year = 1974;
```

Можно также использовать и разыменованное указателя:

```
(*dateP).Day = 7;
(*dateP).Month = 8;
(*dateP).Year = 1974;
```

Указатели также могут быть полями структуры. Это часто используется для создания в памяти цепочек однотипных структур, когда каждая предыдущая указывает на следующую:

```
struct DATE {
    int Day;
    int Month;
    int Year;
    struct DATE *next_date;    //указатель на
                               //другую структуру
                               //того же типа
}
...
struct DATE date1, date2;
date1.next_date = &date2;
```

Теперь, к примеру, оператору `date1.next_date->Year` соответствует доступ к полю `Year` структуры `date2` (то есть, это эквивалентно записи `date2.Year`).

Массивы и строки

Массив – это тип данных, который используется для представления последовательности однотипных значений. Массивы объявляются подобно обычным переменным, с указанием в квадратных скобках размерности

(количества элементов в массиве):

```
int digits[10]; //Массив из десяти элементов типа int
int char str[10]; //Массив из десяти символов (строка)
```

Доступ к элементам массива реализуется с помощью индекса (порядкового номера элемента, начиная с 0):

```
digits[0] = 0;
digits[1] = 1;
str[0]    = 'A';
str[1]    = 'B';
```

По сути, имя массива – это указатель на его первый элемент. Так в последнем примере, оператор `str[0]='A';` можно было бы заменить оператором `*str = 'A' ;` а оператор `str[1] = 'B';` – оператором `*(str + 1) = 'B';`.

Зачастую гораздо удобнее инициализировать массив непосредственно при его объявлении, например:

```
int digits [10] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9};
char str[10] = {'T', 'h', 'e', ' ', 'l', 'i', 'n', 'e'};
int n;
char c;
n = digits[2];    //n = 2
c = str [1] ;     //c = 'h'
```

При разработке программ следует учитывать, что компилятор не всегда может отследить выход индекса за пределы массива. Другими словами, если бы в представленном выше примере мы использовали оператор `c = str[n+10]`, то на этапе компиляции ошибки не возникнет, но в ходе выполнения программы переменной `c` будет присвоено значение "13-го", несуществующего элемента массива, то есть извлечено содержимое области памяти, адрес которой на 12 элементов смещен относительно начального адреса массива.

Строка в Си – это массив типа `char`. Выше был рассмотрен пример объявления такого массива, соответствующего строке "The line". Это

объявление можно было бы выполнить и с помощью строкового литерала:

```
char str[10] = "The line";
```

Однако в таком случае следует помнить, что компилятор неявно завершает строковые литералы символом `'\0'`, и, таким образом, реальная длина строки больше на 1. Это следует учитывать, чтобы при инициализации массива не выйти за его пределы.

При инициализации строк размерность массива можно явно не указывать:

```
char str[] = "The line";
```

В этом случае компилятор определит размерность самостоятельно. (Это правило применимо и к инициализации любых других массивов.)

Для того чтобы строковые константы размещались не в памяти SRAM, а в флэш-памяти микроконтроллера, их следует объявлять с использованием слова `const`:

```
const char str[] = "MyString".
```

Многомерные массивы

Язык Си допускает использование **многомерных массивов**, то есть массивов, элементами которых являются массивы. Примеры объявления таких массивов:

```
int a2[10][2];           //Двухмерный массив 10x2
int a3[3][2][5];        //Трёхмерный массив 3x2x5
```

Фактически, все элементы многомерного массива хранятся в памяти последовательно, поэтому представленные ниже примеры инициализации аналогичны:

```
int a[2][3] = { {1, 2, 3},
                {4, 5, 6} };
```

и

```
int a[2][3] = { 1, 2, 3, 4, 5, 6 };
```


Для доступа к элементам многомерного массива используются индексы по каждой из размерностей или операции разыменования указателя. Например, чтобы извлечь в некоторую переменную *n* цифру 4 из представленного выше массива *a*, можно воспользоваться одним из двух вариантов:

```
n = a[1][0]; //Извлекаем элемент из "строки" 1
           //(вторая строка),
           //"столбца" 0 (первый столбец),
           //то есть - цифру 4
n = *(a + 3); //a - указатель на первый
           // элемент массива, следовательно,
           // a + 3 - указатель на 4-й элемент
```

Операторы ветвления используются для выполнения того или иного блока кода в зависимости от некоторого условия. К таким операторам в языке C относятся `if-else` и `switch-case`.

Оператор if-else

В простейшем случае оператор `if-else` имеет следующую структуру:

```
if      (условное_выражение)      блок_кода_1;          else
блок_кода_2 ;
```

Если условное выражение истинно, то выполняется блок кода 1, в противном случае – блок кода 2. При этом в качестве блока кода 2 допускается использовать последовательность операторов `else-if`:

```
if      (условное_выражение1)      блок_кода_1;
else if  (условное_выражение2)      блок_кода_2;
else if  (условное_выражение3)      блок_кода_3;
... else блок_кода_N;
```

В данном случае каждое выражение будет вычисляться поочередно до тех пор, пока не будет найдено выражение, давшее истинный результат. Если же результаты вычислений всех выражений окажутся ложными, то будет

выполнен блок_кода_N.

В тех случаях, когда при ложных результатах всех условных выражений не требуется выполнять никаких операторов, можно опустить завершающий блок кода вместе с последней ветвью `else`. Например:

```
if (a==1) b = a*3;
else if (a==2) b = a+10;
else if (a==3) b=0;
```

Если выражение в первой строке будет истинным, остальные операторы будут пропущены, в противном случае начнется последовательная проверка выражений, указанных в следующих строках, до тех пор, пока не будет найдено выражение, дающее ненулевой результат, или до тех пор, пока не будет проверено последнее выражение. Если ни одно из выражений не будет истинным, никакие действия с переменной `b` выполнены не будут.

Условные выражения

Язык C позволяет вместо оператора `if-else` использовать условные выражения. Так, конструкцию вида

```
if (условное_выражение)      блок_кода_1;      else
блок_кода_2;
```

можно заменить следующим условным выражением:

```
условное_выражение ? блок_кода_1 : блок_кода_2;
```

Например:

```
(a == 1) ? b = a*3 : b = 0;    //Если a=1, то b=a*3,
                               //иначе b=0
```

Оператор switch-case

В операторах `if-else` можно использовать только выражения, которые сводятся к значению `TRUE` или `FALSE`. В тех случаях, когда необходимо применять выражения, дающие произвольный числовой результат, удобнее

воспользоваться оператором `switch-case`. Этот оператор позволяет с помощью некоторой переменной выбрать одно из выражений, соответствующее заданной константе. Его синтаксис:

```
switch    (выражение)    {  
case константа_выражение1  :  блок_кода  
case константа_выражение2  :  блок_кода  
case константа_выражение3  :  блок_кода  
default:  блок_кода }
```

Продemonстрируем использование оператора `switch-case` на примере рассмотренного выше фрагмента, реализованного с помощью оператора `if-else`:

```
switch(a) {  
    case 1 : b=a*3; break;  
    case 2 : b=a+10;  
    case 3 : b=0; break;  
    default: }
```

Оператор `break` приводит к немедленному выходу из блока `switch`. Если по каким-то причинам необходимо продолжить проверку соответствия выражения выбора тем константам, которые указаны в ветвях `case`, следует убрать операторы `break`.

Ко всему прочему, ветви `case` можно каскадировать. Такой прием особенно удобен в тех случаях, когда нужно, например, проверить введенный символ, не заботясь о том, представляет ли он прописную букву или строчную, или когда нужно получить одну и ту же реакцию на несколько разных чисел. Рассмотрим это на примере фрагмента некоторой абстрактной программы:

```
switch (input)  
{  
    case 'a' : case 'A' : DoA(); break;  
    case 'b' : case 'B' : DoB(); break;  
    case '0' : case '1' : case '2' : case '3' :  
        Do0123(); break;
```

```

        case '4' : case '5' : case '6' : case '7' :
                                Do4567(); break;
        default :                DoDefault(); break;
    }

```

Циклические конструкции применяют для повторения некоторого блока кода на основании условия цикла. В языке С используются циклические конструкции `while`, `for` и `do-while`.

Конструкция **while**

Цикл `while` имеет следующий синтаксис:

```

while (условное_выражение)
{
    // Выполнение тела цикла, если
    // условие_выражение истинно
}

```

Другими словами, циклы `while` имеет смысл использовать в тех случаях, когда соответствующий оператор или блок операторов необходимо выполнять до тех пор, пока условие выражение истинно. Пример формирования строки, состоящей из нечетных цифр:

```

int c,    i;
const char str[] = "0123456789";
char OddNums[5]; //Строка для хранения нечетных цифр
c = 0;           //Счетчик циклов
i = 0;           //Индекс массива OddNums
while (c < 10)   //До тех пор, пока c меньше 10, ...
{ //Если остаток от деления c на 2 = 1, то
    // записываем в i-ю позицию массива OddNums c-й
    // элемент строки str, после чего значение i
    // автоматически инкрементируется
    if ((c % 2) == 1) OddNums[i++] = str[c];
    c++;           //c = c + 1
}

```

Конструкция **for**

Цикл `for` имеет следующий синтаксис:

```
for    (выражение1;  выражение2;  выражение3)
{
    // Выполнение тела цикла
}
```

Выражение1 выполняется только один раз при входе в цикл, и обычно представляет собой оператор присваивания некоторого начального значения счетчику цикла. Выражение2 – это условное выражение, определяющее момент выхода из цикла (цикл выполняется до тех пор, пока оно равно TRUE или 1). Выражение3 – еще один оператор присваивания, в котором обычно изменяется счетчик цикла или некоторая переменная, влияющая на выполнение условия в выражении2. Выражения могут быть представлены любыми операторами, включая пустые (то есть, вместо выражения можно поставить только символ точки с запятой).

Циклы `while` и `for` в большинстве случаев взаимозаменяемы. Так, представленный пример для цикла `while`, можно переписать в следующем виде:

```
int c,  i;
const char str[] = "0123456789";
char OddNums[5]; //Строка для хранения нечетных цифр
i = 0;           //Индекс массива OddNums
for(c = 0;  c < 10;  c++) //До тех пор,
                        //пока c меньше 10, ...
if ((c % 2)  == 1) OddNums[i++] = str[c];
```

В одних ситуациях удобнее применять циклы `for`, в других – `while`. Достоинством циклов `for` является более наглядная инициализация и организация изменения счетчика цикла. С другой стороны, циклы `while` более гибкие и обеспечивают больше возможностей для реализации нестандартных программных решений при организации повторяющихся вычислений.

Конструкция **do-while**

Кроме циклов `while` и `for`, в которых вначале выполняется проверка

истинности условия цикла, и только потом управление передается блоку операторов цикла, в языке Си имеется также конструкция `do-while`. Она отличается от первых двух тем, что в ней вначале выполняется блок операторов, и только потом проверяется выполнение условия. Другими словами, цикл `do-while` всегда выполняется как минимум один раз, вне зависимости от условия цикла.

Цикл `do-while` имеет следующий синтаксис:

```
do
{
// Выполнение блока операторов цикла
}
while (условное_выражение) ;
```

Организация бесконечных циклов

Для организации бесконечного цикла в качестве условного выражения в конструкции `while` или `do-while` можно просто указать значение `TRUE` или `1`:

```
while(1) блок_операторов;
```

В случае циклов `for` это будет выглядеть следующим образом:

```
for (;;) блок_операторов;
```

Операторы `break` и `continue`

Если в теле любого цикла встречается оператор `break`, управление тут же передается на оператор, следующий за оператором цикла, вне зависимости от истинности или неистинности условного выражения. При этом во вложенных циклах выход осуществляется не на самый верхний уровень вложенности, а лишь на один уровень вверх.

При выполнении оператора `continue` все находящиеся после него операторы блока пропускаются, а управление передается в начало цикла для следующей итерации. На практике операторы `continue` используются гораздо реже, чем операторы `break`, однако в сложных циклах, требующих принятия решений на основании многих факторов, использование операторов

`continue` может быть весьма удобным.

Стандартные функции ввода/вывода

Стандартные функции ввода/вывода позволяют обмениваться информацией с выполняемой программой, что, к примеру, очень удобно в процессе отладки. Все они построены на основе функций посимвольного ввода/вывода, которые легко приспособить к аппаратным требованиям системы. Если при обычном применении языка Си эти функции используются для ввода данных с клавиатуры и вывода на экран или принтер, то применительно к микроконтроллерам MSP430 речь идет об обмене данными через приемопередатчик UART/USART (по умолчанию) или последовательный порт. Таким образом, прежде чем начать ввод/вывод в программе должны быть выполнены соответствующие настройки.

Ввод/вывод символов с помощью функций `getchar()` и `putchar()`

Простейшими функциями ввода/вывода в языке С являются `getchar()` и `putchar()`, которые предназначены для посимвольного обмена данными (обе объявлены в стандартное заголовочном файле `stdio.h`). Функция `getchar()` возвращает символ, принятый от UART/USART или по последовательному интерфейсу, а функция `putchar()`, наоборот, выводит символ. Все остальные стандартные функции ввода/вывода базируются на них.

Функции `getchar()` и `putchar()` в компиляторах для микроконтроллеров реализуют ожидание готовности именно приемопередатчика UART/USART для передачи/приема символа. В том случае, если требуется организовать обмен данными по другому интерфейсу (например, по SPI), необходимо разработать собственные функции аналогичного содержания.

Функции вывода строк `puts()` и `printf()`

Функции `puts()` и `printf()` используют функцию `putchar()` для вывода посимвольно строки в порт RS232, назначенным последним. Предположим, у нас есть определение строки, предназначенной для вывода через последовательный интерфейс:

```
char s[6] = "12345";
```

Вызов функции `puts()` для вывода этой строки выглядит просто как

```
puts(s);
```

Функция `printf()` несколько сложнее, поскольку позволяет применить форматированный вывод. Она имеет следующий синтаксис вызова:

```
printf("Строка, в которую подставляются переменные",  
переменная1, переменная2, ... );
```

В выводимой строке обычно используются спецификации форматирования значений переменных, переданных в качестве параметров. Каждая такая спецификация начинается со знака "%", после которого следуют обозначения параметров форматирования:

`c` – вывод параметра в виде символа ASCII;

`d` – вывод параметра в виде целого числа;

`e` – вывод параметра в экспоненциальном формате;

`f` – вывод параметра в виде вещественного числа;

`id` – вывод параметра в виде длинного целого со знаком;

`lu` – вывод параметра в виде беззнакового длинного целого;

`Lx` – вывод параметра в виде беззнакового длинного целого в шестнадцатеричном представлении, используя нижний регистр символов;

`LX` – вывод параметра в виде беззнакового длинного целого в шестнадцатеричном представлении, используя верхний регистр символов;

`s` – вывод параметра в виде строки;

`u` – вывод параметра в виде беззнакового целого;

`x` – вывод параметра в виде беззнакового целого в шестнадцатеричном представлении, используя нижний регистр символов;

`X` – вывод параметра в виде беззнакового целого в шестнадцатеричном представлении, используя верхний регистр символов;

`%` – вывод знака процента.

Количество спецификаций должно совпадать с количеством переменных. При этом первая встретившаяся спецификация соответствует первой переменной в списке, вторая – второй и т.д.

В случае вывода числовых значений сразу же после знака "%" может быть указано количество символов и формат для отображения чисел, например:

`8` – восемь знаков;

0 8 – восемь знаков с дополнением ведущими нулями;

8.2 – восемь знаков, два знака после десятичной точки.

Примеры использования спецификаций форматирования:

```
int n = 123;
char c = '$';
float f = 23.5;
char str[] = "Hello";
printf("n = %d, str = %s", n, str); //n = 123,
                                   //str = Hello
printf("n = %d : 0x%04X", n, n);    //n = 123:0x007B
printf("Salary = %c%8.2f", c, f);   //Salary = $23.50
```

Функции ввода строк `gets ()` и `scanf ()`

Функции `gets ()` и `scanf ()` выполняют действия, обратные функциям `puts ()` и `printf ()`: считывают посимвольно строку через назначенный последним последовательный интерфейс.

Функция `scanf ()` считывает из входного потока значения в формате, определенном в первом параметре, и сохраняет их в переменных, адреса которых переданы ей в качестве последующих параметров. Спецификации форматирования для этой функции такие же, как и для `printf ()`. Обычно функция `scanf ()` используется в паре с `printf ()`:

```
int x, y;
puts("Enter two integers: ");
scanf("%d,%x\n", &x, &y); //Считывает два числа,
                           //введенные через запятую:
                           //одно – в десятичной, а
                           //другое – в
                           //шестнадцатеричной форме
printf("x = %d, y = %04X", x, y);
```

Директивы препроцессора

Как уже было отмечено ранее, директивы препроцессора, по сути, не являются составной частью языка Си, а используются для подстановки кода в текст программы. **Препроцессор** – это особая программа, которая выполняет предварительную обработку данных до начала компиляции. Рассмотрим

стандартные директивы препроцессора.

Директива #include

Ранее, в разделе "Структуры программы на С" уже упоминалась самая распространенная директива `#include`, которая используется фактически во всех программах для включения в текст программы заголовочных файлов (с определениями), имеющих расширение `.h`, или файлов `.c` (не содержащих функцию `main ()`).

Эта директива может использоваться в двух формах:

```
#include <имя_файла>
```

или

```
#include "имя_файла"
```

Если имя файла заключено между знаками "<" и ">", то он считается частью стандартной библиотеки, поставляемой вместе с компилятором. Если же имя файла заключено в двойные кавычки, то считается, что он расположен в той же папке, что и файл с исходным кодом.

Директива #define

Директива `#define` указывает препроцессору на необходимость выполнить подстановку в тексте программы определенной последовательности символов другой последовательностью. Формат директивы:

```
#define заменяемая_последовательность фрагмент_подстановки
Например:
#define MyName "John Smith"
#define Condition (a > b)
#define Operation a = b
char str[] = MyName; // Равнозначно
                        // char str[] = "John Smith";
int a, b;
if Condition Operatrion; //Равнозначно
                        //if (a > b) a = b;
```

В директиве `#define` могут использоваться параметры, благодаря чему она становится очень мощный и гибким инструментом, позволяющим заменять один простой текстовый элемент сложным выражением. Такие подстановки называют **макросами**.

Например, выражение, в котором выбирается большее из двух значений, можно представить в виде следующего макроса:

```
#define larger(x, y) ( (x)>(y) ? (x) : (y) )
```

Если определен такой макрос, то код, использующий его, может иметь следующий вид.

```
int a = 9;
int b = 7;
int c = 0;
c = larger(a, b);
```

Напоминает вызов функции, однако это не функция – компилятор получит от препроцессора последнюю строку в следующем виде.

```
c = ( (a)>(b) ? (a) : (b) );
```

Основное отличие макроса от функции заключается в том, что код макроподстановки вставляется препроцессором в программный код везде, где встречается заданный текстовый элемент, код же функции определяется только в одном месте, а в тех местах, где указано ее имя, осуществляется вызов этого кода.

Есть и еще одно отличие, особенно важное при программировании микроконтроллеров, – при использовании макросов, в отличие от функций, ничего не помещается в стек, что позволяет сэкономить оперативную память.

Кроме того, макросы преобразуются в обычный код, который выполняется быстрее, чем код функции, на вызов которого и возврат управления в вызывающую функцию уходит дополнительное машинное время. Наконец, при использовании макросов не требуется формального объявления типов данных.

Перечислим некоторые правила использования директивы `#define`:

– при создании комментариев в строке с `#define` всегда используется

комментарий вида `/* ... */`;

– следует помнить, что конец строки – это конец `#define`, и весь текст слева заменит текст справа;

– для преобразования параметра макроса в текстовую строку можно указать перед ним символ `"#"`, например:

```
#define OutString(s)  puts(#s)
OutString(Line);    //Равнозначно puts("Line");
```

– для конкатенации двух параметров можно воспользоваться оператором `##`, например:

```
#define Concat(x, y)  x ## y
int i = Concat(2,1);  //Равнозначно int i = 21;
```

– для переноса текста подстановки на другую строку используется символ обратной косой `"\"`, например:

```
#define LongStr "0 1 2 3 4 5 6 7 8 9 10 \
11 12 13 14 15 16 17 18 19 20 "
```

– для отмены определения используется директива `#undef`, например:

```
#define A_Char  'A'
...
#undef A_Char
#define A_Char  'a'
```

Директивы условной компиляции

Многие микроконтроллеры отличаются лишь некоторыми параметрами, количеством выводов, размером памяти и размещением регистров. Это позволяет создавать на языке C программный код для всего модельного ряда. Однако для этого следует каким-то образом заменить те параметры, которые отличаются у разных моделей. Для таких целей используются директивы условной компиляции `#ifdef`, `#ifndef`, `#else` и `#endif`, а также `#if` и `#elif`.

Синтаксис для директивы #ifdef:

```
#ifdef имя_макроса  
последовательность_операторов_1  
#else  
последовательность_операторов_2  
#endif
```

Если имя макроса определено в программе, то компилируется первая последовательность операторов, в противном случае – вторая последовательность (ветка #else может и отсутствовать).

Синтаксис для директивы #ifndef:

```
#ifndef имя_макроса  
последовательность_операторов_1  
#else  
последовательность_операторов_2  
#endif
```

В данном случае, в отличие от директивы #ifdef, первая последовательность операторов выполняется в том случае, если имя макроса в программе не определено.

Для условной компиляции можно также воспользоваться директивами #if, #elif. Их синтаксис:

```
#if выражение_1  
последовательность_операторов_1  
#elif выражение_2  
последовательность_операторов_2  
#else  
последовательность_операторов_3  
#endif
```

Эта конструкция работает аналогично условному оператору if-else. Компилятор оценивает выражения после #if и #elif до тех пор, пока одно из них не даст в результате TRUE, после чего в текст программы подставляется соответствующая последовательность операторов. Если оба выражения дают

FALSE, то подставляется последовательность операторов после директивы `#else` (если она присутствует).

Допустим, для передачи и приема данных через UART у абстрактного микроконтроллера `SomeMic16` используются выводы 12 и 13, у микроконтроллера `SomeMic8` – выводы 6 и 14, а у `SomeMic4` – выводы 1 и 2. Тогда мы можем создать заголовочный файл `SomeMic.h` и включить в него следующие директивы:

```
#if SomeMicX == 16
#define TXD 12
#define RXD 13
#elif SomeMicX == 8
#define TXD 6
#define RXD 14
#elif SomeMicX == 4
#define TXD 1
#define RXD 2
#else
#error "Pins TXD and RXD for SomeMicX are not
defined"
#endif
```

Теперь, если программный проект будет построен на микроконтроллере `SomeMic8`, то в заголовочный файл следует поместить следующий текст:

```
#ifndef SomeMicX
#define SomeMicX = 8
#include <SomeMic.h>
#endif
```

Встретив директиву `#ifndef`, препроцессор включит в текст программы `#define SomeMicX = 8` и `#include <SomeMic.h>`. Поскольку после этого элемент `SomeMicX` получит значение, равное 8, то любая повторная обработка директивы `#ifndef` не приведет к дублированию в выходном тексте соответствующей информации. Другими словами, содержимое заголовочного файла `SomeMic.h` будет помещено в исходный код файлов,

использующих заголовочный файл с `#ifndef`, только один раз.

Использование директивы `#ifndef` с последующими директивами `#define` и `#include` – стандартный прием, позволяющий избежать дублирования информации из заголовочных файлов в исходном коде проекта. Если этого не сделать, то при дублировании компилятор обычно выдает множество сообщений об ошибках, связанных с многократным объявлением переменных.

Директива `#error`

Директива `#error` используется совместно с директивами условной компиляции. Встретив ее, компилятор сгенерирует сообщение об ошибке, указанное справа от директивы.

Обработка прерываний в микроконтроллерных системах на базе MSP430

В различных компиляторах обработка прерываний реализована по-разному. Так, в среде IAR функция обработчик прерывания, в общем случае, имеет вид:

```
#pragma vector = идентификатор_прерывания
__interrupt void irqHandler(void)
{
    Код, выполняемый при вызове разрешенного локально и
    глобально прерывания
}
```

Например, так выглядит определение обработчика для прерывания от АЦП:

```
#pragma vector=ADC_VECTOR
__interrupt void ADC12ISR (void)
{
    if (ADC12MEM0 <= FirstADCVal + ADCDeltaOn)
        P1OUT &= ~0x01;           // LED off
    else P1OUT |= 0x01;           // LED on
}
```

Идентификатор прерывания представляет собой смещение адреса прерывания относительно начала таблицы векторов прерываний, определяемой архитектурой контроллера и расположенной по адресам от 0xFFE0 до 0xFFFFE. Смещения векторов определены в заголовочных файлах `io430x???h`

Так в файле `io430x16x.h` определены следующие идентификаторы прерываний:

- `DACDMA_VECTOR` – объединенный вектор прерывания от контроллера ПДП и модуля ЦАП12;
- `PORT2_VECTOR` – вектор прерывания от линий порта P2;
- `USART1TX_VECTOR` – вектор прерывания при передаче от последовательного интерфейса USART1;
- `USART1RX_VECTOR` – вектор прерывания при приеме от последовательного интерфейса USART1;
- `PORT1_VECTOR` – вектор прерывания от линий порта P1;
- `TIMERA1_VECTOR` – объединенный вектор прерывания таймера A для флагов `TACCR1 CCIFG`, `TACCR2 CCIFG` и `TAIFG`;
- `TIMERA0_VECTOR` – вектор прерывания таймера A для флага `TACCR0 CCIFG`;
- `ADC12_VECTOR` – вектор прерывания от АЦП12;
- `USART0TX_VECTOR` – вектор прерывания при передаче от последовательного интерфейса USART0;
- `USART0RX_VECTOR` – вектор прерывания при приеме от последовательного интерфейса USART0;
- `WDT_VECTOR` – вектор прерывания от сторожевого таймера, работающего в интервальном режиме;
- `COMPARATORA_VECTOR` – вектор прерывания от компаратора A;
- `TIMERB1_VECTOR` – объединенный вектор прерывания таймера B для флагов `TBCCR1 CCIFG`, `TBCCR2 CCIFG`, `TBCCR3 CCIFG`, `TBCCR4`, `TBCCR5 CCIFG`, `TBCCR6 CCIFG` и `TBIFG`;
- `TIMERB0_VECTOR` – вектор прерывания таймера B для флага `TBCCR0 CCIFG`;
- `NMI_VECTOR` – вектор немаскируемого прерывания;
- `RESET_VECTOR` – вектор сброса.

В библиотечном файле `intrinsics.h` объявлены макрофункции `__enable_interrupt()` и `__disable_interrupt()`. Первая устанавливает флаг общего разрешения прерываний в регистре статуса SR, а вторая, наоборот, – сбрасывает его.

Исполнение ассемблерного кода для микроконтроллерных систем на базе MSP430

Иногда в программах, разрабатываемых на языке C, требуется напрямую использовать команды ассемблера с целью оптимизации кода или просто из-за невозможности выполнить те или иные операции средствами C. Компилятор IAR поддерживает вставку в исходный текст программы фрагментов на ассемблере.

Кроме написания ассемблерных функций в ассемблерных файлах, встроенный ассемблер позволяет писать ассемблерный код внутри C-файла. Синтаксис встроенного ассемблерного кода следующий:

```
asm("<string>");
```

Множественные операторы ассемблера могут отделяться символом новой строки \n:

```
bool flag;
```

```
asm ("Label: nop\n"  
    "      mov.b &0x20,&flag\n"  
    "      jmp Label");
```

Можно использовать конкатенацию строк, чтобы определить множественные операторы без использования дополнительных ключевых слов asm. Встроенный ассемблер может использоваться внутри или вне функции Си.

4 ЛАБОРАТОРНЫЙ ПРАКТИКУМ

4.1 ИЗУЧЕНИЕ ПРИНЦИПОВ ПРОГРАММНОГО УПРАВЛЕНИЯ ВНЕШНИМИ УСТРОЙСТВАМИ НА ПРИМЕРЕ ВЫВОДА ИНФОРМАЦИИ НА ЦИФРОВОЙ ИНДИКАТОР

Цель работы: изучить принципы функционирования и возможности программного управления цифровым индикатором, разработать алгоритм и программу для вывода информации на цифровой индикатор.

4.1.1 Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе [1] и конспект лекций, ознакомиться с принципами функционирования и возможностями программного управления выводом символов на экран 32 позиционного цифрового ЖК-индикатора с использованием микроконтроллера MSP430F1611.

Цифровые порты ввода-вывода

Устройства MSP430 имеют до 6 портов цифровых входов/выходов от P1 до P6. Каждый порт имеет 8 выводов входа/выхода. Каждый вывод индивидуально конфигурируется как вход или выход и каждая линия ввода/вывода может быть индивидуально считана или записана.

Порты P1 и P2 имеют возможность вызывать прерывание. Для каждой линии ввода/вывода портов P1 и P2 можно индивидуально разрешить прерывания и сконфигурировать их так, чтобы прерывание происходило по фронту или спаду входного сигнала.

Регистры ввода PxIN

Каждый бит в каждом регистре PxIN отражает величину входного сигнала на соответствующей ножке ввода/вывода, когда она сконфигурирована на функцию ввода.

Бит = 0: Входной сигнал имеет низкий уровень;

Бит = 1: Входной сигнал имеет высокий уровень.

Регистры вывода PxOUT

Каждый бит в каждом регистре PxOUT содержит значение, которое будет выведено на соответствующую ножку ввода/вывода, сконфигурированную на функцию вывода и имеющую направление на вывод.

Бит = 0: Выходной сигнал имеет низкий уровень;

Бит = 1: Выходной сигнал имеет высокий уровень.

Регистры направления PxDIR

Каждый бит в каждом регистре PxDIR позволяет выбрать направление соответствующей ножки ввода/вывода, независимо от выбранной для этой ножки функции. Биты PxDIR для ножек ввода/вывода, выбранные для других функций модуля должны быть установлены так, как это требуется для другой функции.

Бит = 0: Ножка порта переключается на ввод;

Бит = 1: Ножка порта переключается на вывод.

Регистры выбора функции PxSEL

Ножки порта часто мультиплексированы с другими функциями периферийных модулей. Каждый бит PxSEL определяет, как будет использована ножка – в качестве порта ввода/вывода или в качестве функции периферийного модуля.

Бит = 0: Для ножки выбирается функция ввода/вывода

Бит = 1: Для ножки выбирается функция периферийного модуля

Установка PxSEL=1 автоматически не определяет направление передачи информации для ножки. Некоторые функции периферийных модулей требуют конфигурирования битов PxDIR для выбора направления, необходимого для правильной работы этой функции.

Прерывания P1 и P2

Каждая ножка портов P1 и P2 имеет возможность вызова прерывания, конфигурируемую регистрами PxIFG, PxIE и PxIES. Все ножки P1 – источник одного вектора прерывания, а все выводы P2 – источник другого одиночного вектора прерывания. Определить источник прерывания – P1 или P2 можно путем проверки регистра PxIFG.

Регистры флагов прерывания P1IFG, P2IFG

Каждый бит PxIFG – это флаг прерывания соответствующей ножки ввода/вывода, устанавливаемый, когда происходит перепад выбранного входного сигнала на ножке. Все флаги прерывания PxIFG запрашивают прерывание, когда установлен соответствующий им бит в PxIE и установлен бит GIE. Каждый флаг PxIFG должен быть сброшен программно. Программное обеспечение также может устанавливать каждый флаг PxIFG, обеспечивая возможность генерации программно-инициированного прерывания.

Бит = 0: Прерывание не ожидается

Бит = 1: Прерывание ожидается

Прерывания вызывают только перепады уровней, а не статические

уровни. Если любой флаг P_xIFG оказывается установленным во время выполнения процедуры обработки прерывания P_x или устанавливается после команды RETI выполняемой процедуры обработки прерывания P_x, установка флага P_xIFG_x генерирует другое прерывание. Таким образом, гарантируется, что каждый перепад уровня будет учтен.

Запись в P1OUT, P1DIR, P2OUT или P2DIR может привести к установке соответствующих флагов P1IFG или P2IFG.

Любое событие вызова внешнего прерывания должно иметь длительность, по крайней мере, равную 1,5 MCLK или дольше, чтобы быть гарантировано принятым и вызвать установку соответствующего флага прерывания.

Регистры выбора фронта прерывания P1IES, P2IES

Каждый бит P_xIES позволяет выбрать, по какому фронту сигнала будет происходить прерывание для соответствующей ножки ввода/вывода.

Бит = 0: Флаг P_xIFG устанавливается при изменении уровня сигнала с низкого на высокий;

Бит = 1: Флаг P_xIFG устанавливается при изменении уровня сигнала с высокого на низкий.

Разрешение прерываний P1IE, P2IE

Каждый бит P_xIE разрешает прерывание от соответствующего флага прерываний регистра P_xIFG.

Бит = 0: Прерывание запрещено

Бит = 1: Прерывание разрешено

Конфигурирование неиспользуемых выводов порта

Неиспользуемые ножки ввода/вывода должны быть сконфигурированы на функцию ввода/вывода, в направлении вывода и оставаться неподключенными на печатной плате для уменьшения потребляемой мощности. Значение бита P_xOUT может быть любым, поскольку ножка не подключена.

Регистры цифровых входов/выходов

Для конфигурирования P1 и P2 используются семь регистров. Четыре регистра необходимы для конфигурирования портов P3-P6. Регистры цифровых портов ввода-вывода приведены в таблице 4.1.1.

Таблица 4.1.1 – Регистры цифровых портов ввода-вывода

Порт	Регистр	Обозначение	Адрес	Тип регистра	Исходное состояние
P1	Ввод	P1IN	020h	Только чтение	–
	Вывод	P1OUT	021h	Чтение/запись	Не изменяется
	Направление	P1DIR	022h	Чтение/запись	Сброс с PUC
	Флаг прерывания	P1IFG	023h	Чтение/запись	Сброс с PUC
	Выбор фронта прерывания	P1IES	024h	Чтение/запись	Не изменяется
	Разрешение прерывания	P1IE	025h	Чтение/запись	Сброс с PUC
	Выбор порта	P1SEL	026h	Чтение/запись	Сброс с PUC
P2	Ввод	P2IN	028h	Только чтение	-
	Вывод	P2OUT	029h	Чтение/запись	Не изменяется
	Направление	P2DIR	02Ah	Чтение/запись	Сброс с PUC
	Флаг прерывания	P2IFG	02Bh	Чтение/запись	Сброс с PUC
	Выбор фронта прерывания	P2IES	02Ch	Чтение/запись	Не изменяется
	Разрешение прерывания	P2IE	02Dh	Чтение/запись	Сброс с PUC
	Выбор порта	P2SEL	02Eh	Чтение/запись	Сброс с PUC
P3	Ввод	P3IN	018h	Только чтение	-
	Вывод	P3OUT	019h	Чтение/запись	Не изменяется
	Направление	P3DIR	01Ah	Чтение/запись	Сброс с PUC
	Выбор порта	P3SEL	01Bh	Чтение/запись	Сброс с PUC
P4	Ввод	P4IN	01Ch	Только чтение	-
	Вывод	P4OUT	01Dh	Чтение/запись	Не изменяется
	Направление	P4DIR	01Eh	Чтение/запись	Сброс с PUC
	Выбор порта	P4SEL	01Fh	Чтение/запись	Сброс с PUC
P5	Ввод	P5IN	030h	Только чтение	-
	Вывод	P5OUT	031h	Чтение/запись	Не изменяется
	Направление	P5DIR	032h	Чтение/запись	Сброс с PUC
	Выбор порта	P5SEL	033h	Чтение/запись	Сброс с PUC
P6	Ввод	P6IN	034h	Только чтение	-
	Вывод	P6OUT	035h	Чтение/запись	Не изменяется
	Направление	P6DIR	036h	Чтение/запись	Сброс с PUC
	Выбор порта	P6SEL	037h	Чтение/запись	Сброс с PUC

Принципы программного управления выводом символов на экран цифрового индикатора с помощью микроконтроллера MSP430F1611

В лабораторном стенде используется индикатор WH1602A-NGG-CT на базе контроллера HD44780. Данный ЖКИ позволяет отображать 2 строки по 16 символов. Символы отображаются в матрице 5x8 (или 5x10) точек. Между символами имеются интервалы шириной в одну отображаемую точку. Каждому отображаемому на ЖКИ символу соответствует его код в ячейке ОЗУ модуля.

Модуль содержит два вида памяти – кодов отображаемых символов и пользовательского знакогенератора, а также логику для управления ЖК-панелью.

Модуль позволяет:

- работать как по 8-ми, так и по 4-х битной шине данных (задается при инициализации);
- принимать команды с шины данных;
- записывать данные в ОЗУ с шины данных;
- читать данные из ОЗУ на шину данных;
- читать статус состояния на шину данных;
- запоминать до 8-ми изображений символов, задаваемых пользователем;
- выводить мигающий (или не мигающий) курсор двух типов;
- управлять контрастностью и подсветкой;

Программирование и управление. Перед началом рассмотрения принципов управления ЖКИ-модулем, обратимся к внутренней структуре контроллера HD44780.

Упрощенная структурная схема контроллера HD44780 приведена на рисунке 4.1.1. Можно сразу выделить основные элементы, с которыми приходится взаимодействовать при программном управлении: регистр данных (DR), регистр команд (IR), видеопамять (DDRAM), ОЗУ знакогенератора (CGRAM), счетчик адреса памяти (AC), флаг занятости контроллера (BF).

Другие элементы не являются объектом прямого взаимодействия с управляющей программой – они участвуют в процессе регенерации изображения на ЖКИ: знакогенератор, формирователь курсора, сдвиговые регистры и драйверы.

Управление контроллером ведется посредством интерфейса управляющей системы. Основными объектами взаимодействия являются регистры DR и IR. Выбор адресуемого регистра производится линией D/\bar{C}_{LCD} , если $D/\bar{C}_{LCD} = 0$ – адресуется регистр команд (IR), если

$D/\overline{C}_{LCD} = 1$ – регистр данных (DR).

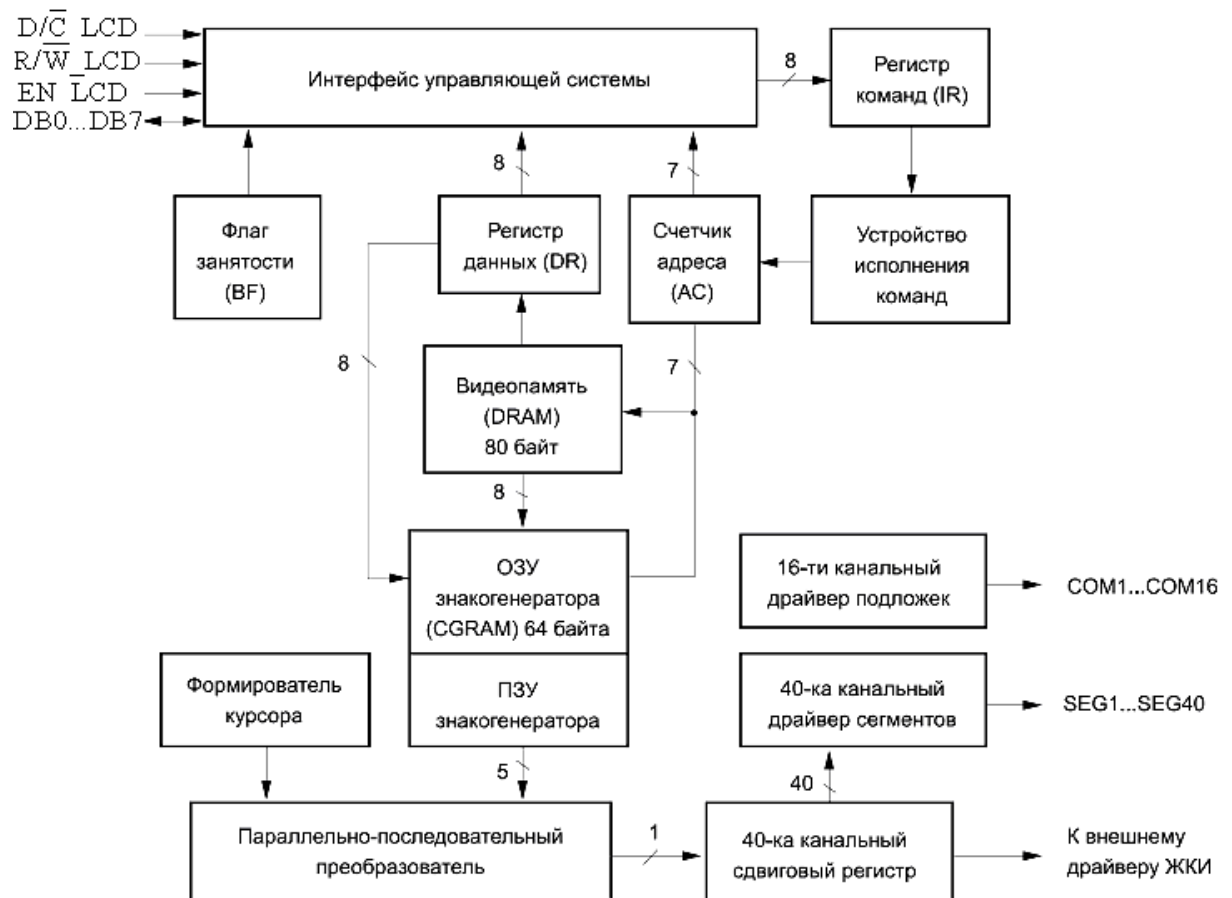


Рисунок 4.1.1 – Упрощенная структурная схема контроллера HD44780

Данные через регистр DR, в зависимости от текущего режима, могут помещаться (или прочитываться) в видеопамять (DRAM) или в ОЗУ знакогенератора (CGRAM) по текущему адресу, указываемому счетчиком адреса (AC). Информация, попадающая в регистр IR, интерпретируется устройством выполнения команд как управляющая последовательность. При чтении регистра IR в 7-ми младших разрядах возвращается текущее значение счетчика AC, а в старшем разряде флаг занятости (BF).

Видеопамять, имеющая общий объем 80 байтов, предназначена для хранения кодов символов, отображаемых на ЖКИ. Видеопамять организована в две строки по 40 символов в каждой. Эта привязка является жесткой и не подлежит изменению. Другими словами, независимо от того, сколько реальных строк будет иметь каждый конкретный ЖКИ-модуль, скажем, 80 x 1 или 20 x 4, адресация видеопамети всегда производится как к двум строкам по 40

СИМВОЛОВ.

Модуль WH1602A-NGG-CT содержит ОЗУ размером 32 байт по адресам 00h–0Fh и 40h–4Fh для хранения данных (DDRAM), выводимых на ЖКИ. Адреса отображаемых на ЖКИ символов распределены, как показано в таблице 4.1.2.

Таблица 4.1.2 – Адреса знакомест в памяти модуля WH1602A-NGG-CT

Номер символа	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Строка 1	00	01	02	03	04	05	06	07	08	09	0A	0B	0C	0D	0E	0F
Строка 2	40	41	42	43	44	45	46	47	48	49	4A	4B	4C	4D	4E	4F

У контроллера HD44780 существует набор внутренних флагов, определяющих режимы работы различных элементов контроллера (таблица 4.1.3). В таблице 4.1.4 приведены значения управляющих флагов непосредственно после подачи на ЖКИ-модуль напряжения питания. Переопределение значений флагов производится специальными командами, записываемыми в регистр IR, при этом комбинации старших битов определяют группу флагов или команду, а младшие содержат собственно флаги.

Таблица 1.1.3 – Флаги, управляющие работой контроллера HD44780

Флаг	Описание
1	2
I/D:	режим смещения счетчика адреса AC, 0 - уменьшение, 1 - увеличение.
S:	флаг режима сдвига содержимого экрана. 0 - сдвиг экрана не производится, 1 - после записи в DDRAM очередного кода экран сдвигается в направлении, определяемым флагом I/D: 0 - вправо, 1 - влево. При сдвиге не производится изменение содержимого DDRAM. изменяются только внутренние указатели расположения видимого начала строки в DDRAM.
S/C:	флаг-команда, производящая вместе с флагом R/L операцию сдвига содержимого экрана (так же, как и в предыдущем случае, без изменений в DDRAM) или курсора. Определяет объект смещения: 0 - сдвигается курсор, 1 - сдвигается экран.

Продолжение таблицы 1.1.3

Флаг	Описание
1	2
R/L:	флаг-команда, производящая вместе с флагом S/C операцию сдвига экрана или курсора. Уточняет направление сдвига: 0 - влево, 1 - вправо.
D/L:	флаг, определяющий ширину шины данных: 0 - 4 разряда, 1 - 8 разрядов.
N:	режим развертки изображения на ЖКИ: 0 - одна строка, 1 - две строки
F:	размер матрицы символов: 0 - 5x8 точек, 1 - 5x10 точек.
D:	наличие изображения: 0 - выключено, 1 - включено
C:	курсор в виде подчеркика: 0 - выключен, 1 - включен
B:	курсор в виде мерцающего знакоместа: 0 - выключен, 1 - включен.

Таблица 4.1.4 – Значения управляющих флагов после подачи питания

Флаг	Описание
I/D = 1:	режим увеличения счетчика на 1
S = 0:	без сдвига изображения
D/L = 1:	8-ми разрядная шина данных
N = 0:	режим развертки одной строки
F = 0:	символы с матрицей 5x8 точек
D = 0:	отображение выключено
C = 0:	курсор в виде подчеркика выключен
B = 0:	курсор в виде мерцающего знакоместа выключен

Список управляющих комбинаций битов регистра IR и выполняемые ими команды приведены на рисунке 4.1.2. Так как на момент включения ЖКИ-модуль ничего не отображает (флаг D = 0), то для того, чтобы вывести какой-либо текст необходимо, как минимум, включить отображение, установив флаг D = 1.

Пример последовательности для инициализации ЖКИ-модуля показан на рисунке 4.1.3: \$38, \$0C, \$01, \$06 (знак «\$» перед числом указывает на шестнадцатеричное основание). \$38 устанавливает режим отображения 2-х строк с матрицей 5x8 точек и работу с 8-ми разрядной шиной данных; \$0C включает отображение на экране ЖКИ-модуля, без отображения курсоров; \$01 – очищает дисплей и помещает курсор в левую позицию, \$06 устанавливает режим автоматического перемещения курсора слева направо после вывода каждого символа.

Команда	Код команды										Описание	Время выполнения (270 кГц)
	D/ \overline{C} _LCD	R/ \overline{W}	DB7	DB6	DB5	DB4	DB3	DB2	DB1	DB0		
Clear Display	0	0	0	0	0	0	0	0	0	1	В DDRAM записываются нули, и курсор помещается в самую левую позицию (AC=0)	1.53 мс
Return Home	0	0	0	0	0	0	0	0	1	-	Перемещает курсор в самую левую позицию (AC=0) Содержимое DDRAM не изменяется	1.53 мс
Entry Mode Set	0	0	0	0	0	0	0	1	I/D	SH	Установка направления сдвига курсора (I/D=0/1 - влево/вправо) и разрешение сдвига дисплея (SH=1) при записи в DDRAM	39 мкс
Display ON/OFF Control	0	0	0	0	0	0	1	D	C	B	Включает дисплей (D=1) и выбирает тип курсора (C, B)	39 мкс
Cursor or Display Shift	0	0	0	0	0	1	S/C	R/L	-	-	Выполняет сдвиг дисплея или курсора (S/C=0/1 – курсор/дисплей, R/L=0/1 – влево/вправо)	39 мкс
Function Set	0	0	0	0	1	DL	N	F	-	-	Устанавливает разрядность интерфейса (DL=0/1 - 4/8 бит), число строк (N=0/1 - 1/2 строки) и размер матрицы символа (F=0/1 - 5×8 точек / 5×11 точек)	39 мкс
Set CGRAM Address	0	0	0	1	AC5	AC4	AC3	AC2	AC1	AC0	Установка адреса для последующих операций и выбор области CGRAM	39 мкс
Set DDRAM Address	0	0	1	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Установка адреса для последующих операций и выбор области DDRAM.	39 мкс
Read Busy Flag and Address	0	1	BF	AC6	AC5	AC4	AC3	AC2	AC1	AC0	Прочитать флаг занятости и содержимое счетчика адреса	0 мкс
Write Data to RAM	1	0	D7	D6	D5	D4	D3	D2	D1	D0	Запись данных в активную область внутреннего ОЗУ(DDRAM/CGRAM).	43 мкс
Read Data from RAM	1	1	D7	D6	D5	D4	D3	D2	D1	D0	Чтение данных из активной области внутреннего ОЗУ (DDRAM/CGRAM).	43 мкс

Рисунок 4.1.2 – Описание команд модуля ЖКИ

Контроллер HD44780 поддерживает как операции записи, так и операции чтения. Чтение регистра DR приводит к загрузке содержимого DDRAM или CGRAM, в зависимости от текущего режима, при этом курсор смещается на одну позицию, как и при записи. Чтение регистра IR возвращает 8 значащих разрядов, причем в 7-ми младших содержится текущее значение счетчика AC (7 разрядов, если адресуется DDRAM, и 6 – если CGRAM), а в старшем – флаг занятости BF. Этот флаг имеет значение 1 когда контроллер занят и 0 – когда свободен. Необходимо учитывать, что большинство операций, выполняемых контроллером, занимают значительное время, около 40 мкс, а время выполнения некоторых доходит до единиц миллисекунд, поэтому цикл

ожидания снятия флага BF должен обязательно присутствовать в программах драйвера ЖКИ-модуля и предшествовать совершению любой операции (естественно, кроме операции проверки флага BF).

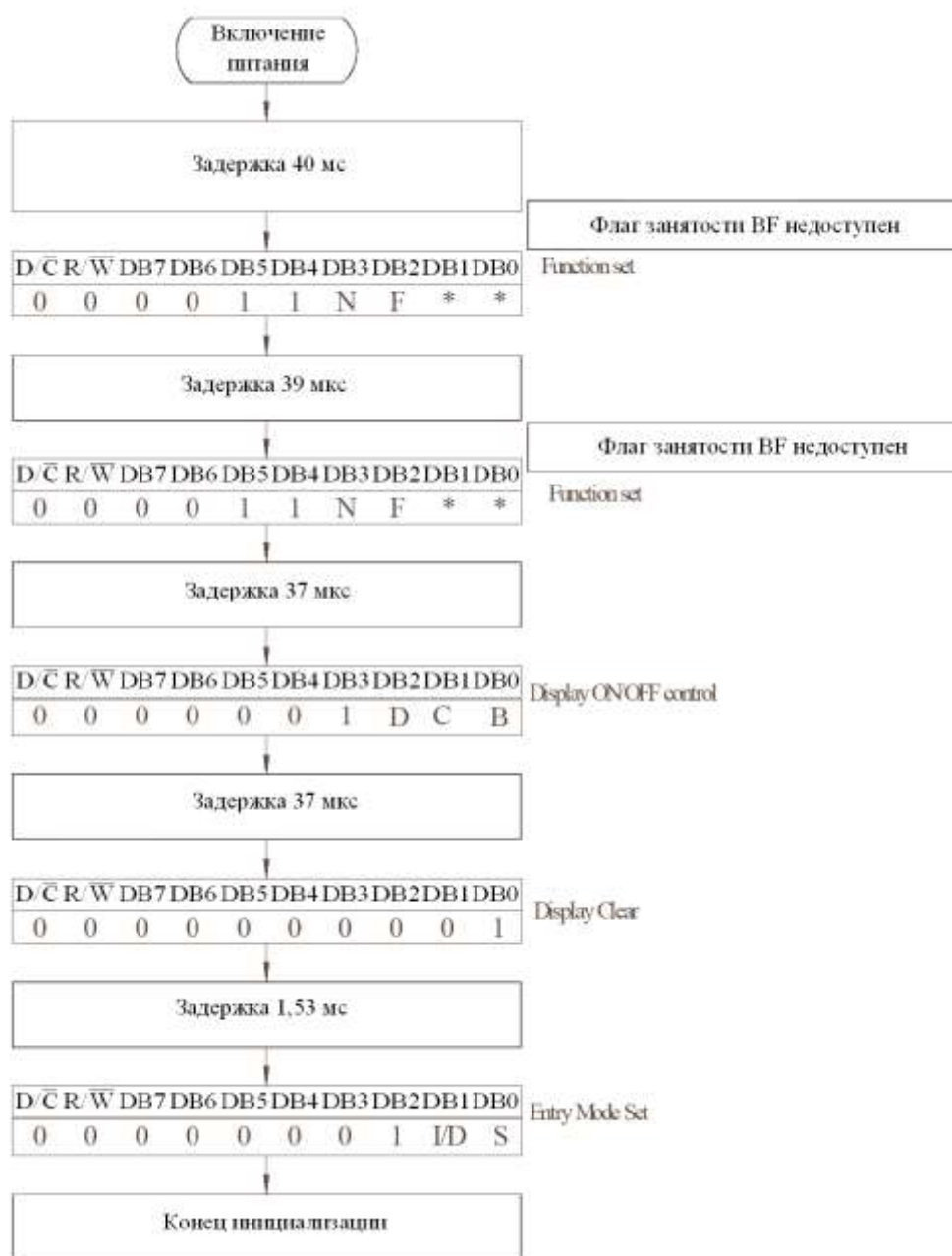


Рисунок 4.1.3 – Инициализация модуля ЖКИ

Вывод на экран символа производится записью его кода в регистр DR. При этом символ размещается в DDRAM по текущему адресу, указываемому AC, а значение AC увеличивается или уменьшается на 1. Чтобы произвести переустановку курсора на нужную позицию, необходимо присвоить AC

соответствующее значение (см. рисунок 1.2). Здесь есть одна тонкость. Когда производится последовательная запись символов и в результате заполняется вся строка, курсор автоматически переходит на вторую строку, но если необходимо принудительно установить курсор, скажем, на начало второй строки, то будет неверным присвоить АС казалось бы, логичное значение \$28 (40), правильным является значение \$40 (64). Значения адресов DDRAM в диапазоне \$28...\$3F (а равно и \$68...\$7F) являются неопределенными и результаты работы с ними могут быть непредсказуемыми. Необходимо учитывать, что контроллеры, устанавливаемые на ЖКИ-модули, могут иметь различные наборы символов, причем это может зависеть как от производителя контроллера, так и от модификации данной конкретной модели. Например, фирма Powertip выпускает ЖКИ-модули с четырьмя базовыми модификациями наборов символов: японской, европейской, французской и русской.

Русские символы из встроенной в модуль WH1602A-NGG-CT таблицы знакогенератора (рисунок 4.1.4) не совпадают с кодировкой ASCII, поэтому необходимо программно перекодировать выводимые на экран символы, для корректного отображения.

Из допустимых для размещения в DDRAM кодов символы с кодами \$00...\$07 (и их дубликат с кодами \$08...\$0F) имеют специальное назначение – это переопределяемые символы, графическое изображение которых может назначить сам пользователь, разместив соответствующую информацию в области CGRAM. Для программирования доступны 8 переопределяемых символов в режиме с матрицей 5x7 точек и 4 с матрицей 5x10 (в режиме 5x10 переопределяемые символы адресуются кодами DDRAM через один: \$00, \$02, \$04, \$06). Для кодирования матрицы используются горизонтально «уложенные» байты, пять младших битов которых несут информацию о рисунке (причем 1 означает, что сегмент будет включен), 4-й разряд каждого из 8-ми (или 11-ти в режиме 5x10) байтов матрицы определяет левую колонку символа, а 0-й – правую. Старшие три бита не используются, равно как и старшие пять байтов, составляющих полную область матрицы символа (16 байтов) в режиме 5x10 (обратите внимание, что матрица программируемых символов допускает использование полной высоты строки (8 строчек для режима 5x7 и 11 строчек для режима 5x10), то есть можно размещать точки в области подчеркивающего курсора). Чтобы определить собственный символ необходимо установить счетчик АС на адрес начала матрицы требуемого символа в CGRAM - \$00, \$08, \$10 и т.д. (\$00, \$10, \$20 для режима 5x10 точек) - произвести перезапись всех байтов матрицы, начиная с верхней строки. После

этого, записав в DDRAM код запрограммированного символа: \$00, \$01, \$02 (\$00, \$02, \$04 для режима 5x10 точек), на экране в соответствующем месте будет отображаться переопределенный символ.

Upper 4 bit Lower 4 bit	LLLL	LLH	LLHL	LLHH	LHLL	LHLH	LHHL	LHHH	HLLL	HLLH	HLHL	HLHH	HHLL	HHHL	HHHH
LLLL	CG RAM (1)			0aP`P									BA4.2K		
LLH	CG RAM (2)		!	1A0a9									ГЯш.Л9		
LLHL	CG RAM (3)		"	2BRbr									Е6ь.шK		
LLHH	CG RAM (4)		#	3DScs									Кв.ш.дK		
LHLL	CG RAM (5)		\$	4DTdt									Эгь.7.4B		
LHLH	CG RAM (6)		%	5EUeu									Кеах.л.7		
LHHL	CG RAM (7)		&	6FUFU									Вхш.7.ш9		
LHHH	CG RAM (8)		'	7BU9u									Л3я.1.7		
HLLL	CG RAM (1)		(8HXhx									Пх.ш.7		
HLLH	CG RAM (2))	9IYiy									Ух.7.7		
HLHL	CG RAM (3)		*	JZjz									Фк.7.7		
HLHH	CG RAM (4)		+	KCKk									Чл.7.7		
HHLL	CG RAM (5)		,	<L*1e									Шх.7.7		
HHLH	CG RAM (6)		-	=MOnB									Ых.7.7		
HHHL	CG RAM (7)		.	>N^n4									Ыг.7.7		
HHHH	CG RAM (8)		/	?0_Loe									ЭтЕ.7.7		

Рисунок 4.1.4 – Таблица символов знакогенератора

Производитель ЖКИ рекомендует выполнять следующую последовательность действий для инициализации. Выдержать паузу не менее 40 мс между установлением рабочего напряжения питания ($> 4,5$ В) и выполнением каких-либо операций с контроллером. Первой операцией выполнить команду, выбирающую разрядность шины (это должна быть команда \$30 независимо от того, какой разрядности интерфейс будет

использовать в дальнейшем), причем перед выполнением этой операции не проверять значение флага BF. Далее выдержать паузу не менее 39 мкс и повторить команду выбора разрядности шины, причем перед подачей команды вновь не производить проверку флага BF. Эти две операции являются инициализирующими и призваны вывести контроллер в исходный режим работы (то есть перевести в режим работы с 8-ми разрядной шиной) из любого состояния. Следующим шагом необходимо вновь выдержать паузу, на этот раз 37 мкс, и подать команду включения модуля \$0C, за которой после паузы в 37 мкс подать команду очистки дисплея \$01, выдержать паузу 1,53 мс и подать команду выбора режима \$06.

Необходимо помнить, что выбор режима работы с 4-х разрядной шиной, то есть выдача команды \$20, выполняется из 8-ми разрядного режима, который устанавливается автоматически после подачи напряжения питания, ввиду чего не возможно адекватно объявить необходимое значение флагов N и F, располагающихся в младшей тетраде команды установки разрядности шины. Поэтому команду необходимо повторить в уже установившемся 4-х разрядном режиме путем последовательной передачи двух тетрад.

Схема подключения ЖКИ к микроконтроллеру MSP430F1611

Для согласования электрических уровней в схеме применен преобразователь SN74LVC4245A (3,3V в 5V).

Размещение модуля на плате показано на рисунке 4.1.5, а принципиальная схема подключения отображена на рисунке 4.1.6.

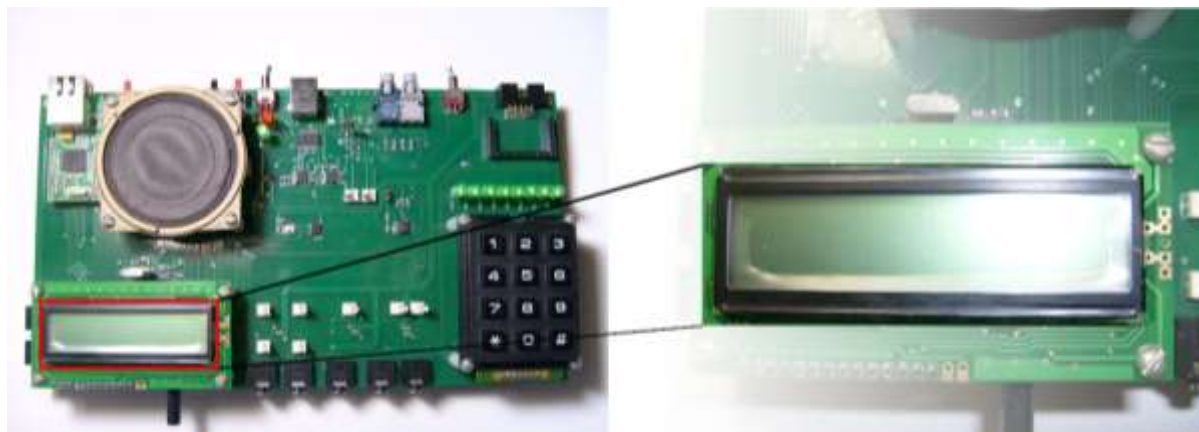


Рисунок 4.1.5 – Размещение модуля ЖКИ на плате

Сигналы и их подключение описаны в таблицах 4.1.5 и 4.1.6. Сигналы $\overline{OE_BF_LCD}$ и $R/\overline{W_LCD}$ поступают от дешифратора DD7, если на нем выбран

модуль LCD (A0=1, A1=1).

Таблица 4.1.5 – Описание сигналов и выводов модуля WH1602A (LCD1)

Вывод LCD1	Название	Подключение к цепи / [номер вывода]	Примечание
1	GND	GND	Подключение к шине 0 В
2	Vcc	+5V	Напряжение питания
3	Vo	+5V	Операционное напряжение
4	Data/Command	D/C_LCD / 32	Режим: данные/команды
5	R / \overline{W}	DD7 / 9	Сигнал чтение/запись
6	EN	EN_LCD DB0...DB7/ 33	Строб данных
7	DB0	DD8 / 3	Шина данных, подключенная к преобразователю уровней DD8
8	DB1	DD8 / 4	
9	DB2	DD8 / 5	
10	DB3	DD8 / 6	
11	DB4	DD8 / 7	
12	DB5	DD8 / 8	
13	DB6	DD8 / 9	
14	DB7	DD8 / 10	

Таблица 4.1.6 – Описание сигналов и выводов преобразователя SN74LVC4245A

Вывод DD8	Название	Подключение к цепи / [номер вывода]	Примечание
1	2	3	4
1	VccA(5V)	+5V	Напряжение питания +5В
2	DIR	DD7 / 9	Сигнал чтение/запись R/ \overline{W} _LCD
3	A1	LCD1 / 7	Шина данных, подключенная к модулю LCD
4	A2	LCD1 / 8	
5	A3	LCD1 / 9	
6	A4	LCD1 / 10	
7	A5	LCD1 / 11	
8	A6	LCD1 / 12	
9	A7	LCD1 / 13	
10	A8	LCD1 / 14	
11	GND	GND	Подключение к шине 0 В
12	GND	GND	
13	GND	GND	
14	B8	DB7 / 43	Шина данных, подключенная к MSP430
15	B7	DB6 / 42	
16	B6	DB5 / 41	
17	B5	DB4 / 40	
18	B4	DB3 / 39	
19	B3	DB2 / 38	
20	B2	DB1 / 37	
21	B1	DB0 / 36	
22	\overline{OE}	DD7 / 7	Сигнал разрешения ввода/вывода $\overline{OE_BF_LCD}$
23	VccB(3,3V)	+3,3 V	Напряжение питания +3,3В
24	VccB(3,3V)	+3,3 V	

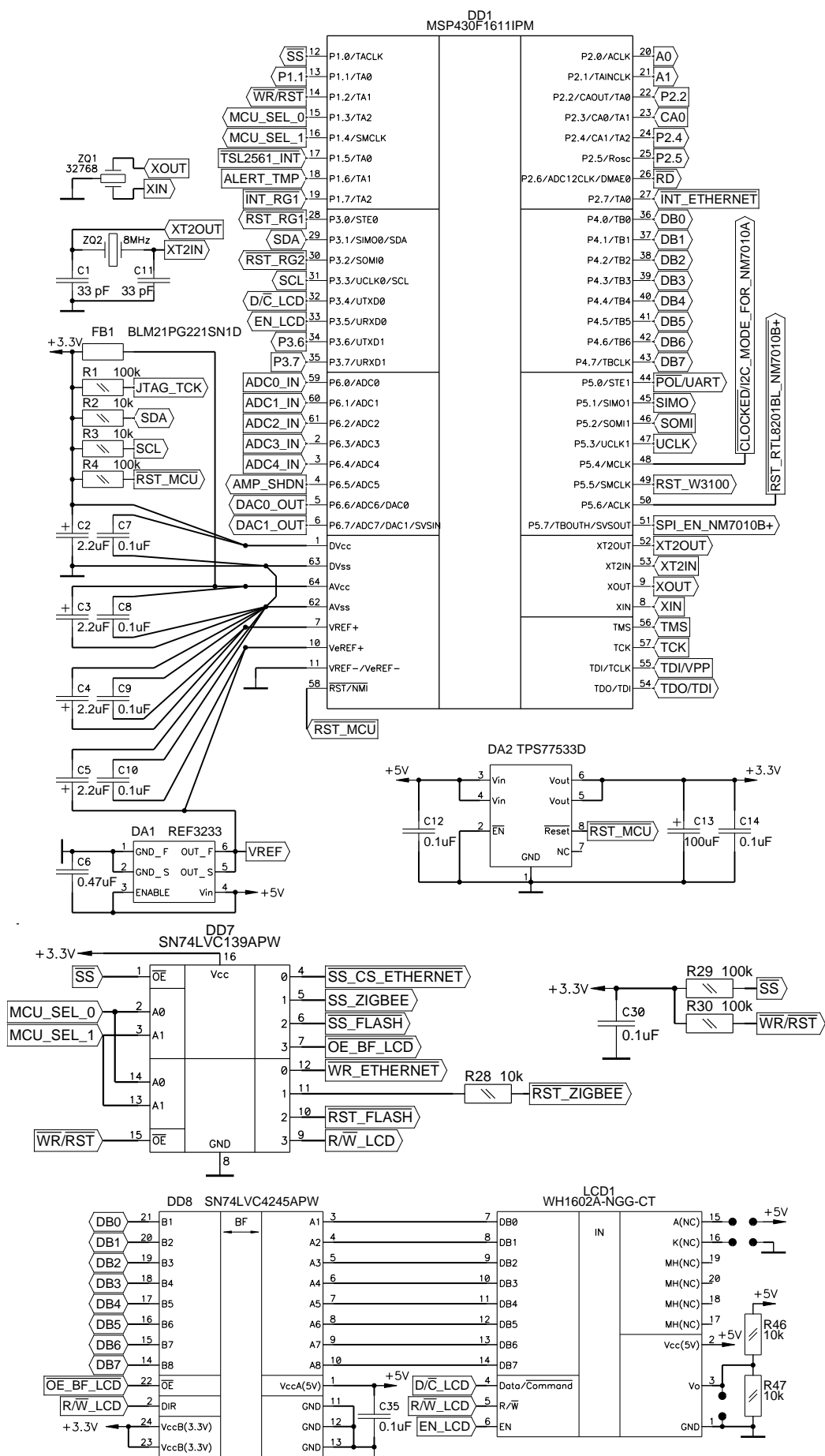


Рисунок 4.1.6 – Принципиальная схема подключения модуля ЖКИ

При работе с ЖКИ необходимо строго соблюдать последовательность формирования управляющих сигналов во избежание повреждения управляющего контроллера или преобразователя уровней. Ниже приведен текст процедуры для записи байта в модуль ЖКИ:

```
// Вывод байта на индикатор, параметры:      byte - выводимый байт,
// dnc=0 - режим передачи команд, dnc=1 - данных
void LCD_WriteByte(char byte, char dnc)
{
    DB_DIR = 0x00;          // Шина данных на прием
    MCU_SEL_0_OUT = 1;      // Выбор модуля LCD
    MCU_SEL_1_OUT = 1;      // при помощи дешифратора DD7
    D_NC_LCD_OUT = 0;       // Выбор режима передачи команд для LCD
    NWR_NRST_OUT = 1;       // Сформировать сигнал "R/W_LCD"
    NSS_OUT = 0;            // Сформировать сигнал "OE_BF_LCD"
    EN_LCD_OUT = 1;         // Сформировать строб данных для LCD
    EN_LCD_OUT = 1;         // Сформировать строб данных для LCD
    EN_LCD_OUT = 1;         // Сформировать строб данных для LCD
    while (DB_IN & BIT7);    // ожидание сброса флага занятости BUSY
    EN_LCD_OUT = 0;         // Перевести сигнал "EN_LCD" в неактивное состояние
    NSS_OUT = 1;            // Перевести сигнал "OE_BF_LCD" в неактивное состояние
    D_NC_LCD_OUT = dnc;     // Выбрать режим записи данных (dnc=1)
                           // или записи команды (dnc=0)
    NWR_NRST_OUT = 0;       // Сформировать сигнал "R/W_LCD"
    NSS_OUT = 0;            // Сформировать сигнал "OE_BF_LCD"
    DB_DIR = 0xFF;          // Шина данных на выход
    DB_OUT = byte;          // Выставить данные на шину данных
    EN_LCD_OUT = 1;         // Сформировать строб данных для LCD
    EN_LCD_OUT = 1;         // Сформировать строб данных для LCD
    EN_LCD_OUT = 1;         // Сформировать строб данных для LCD
    EN_LCD_OUT = 0;         // Перевести сигнал "EN_LCD" в неактивное состояние
    NSS_OUT = 1;            // Перевести сигнал "OE_BF_LCD" в неактивное состояние
    DB_DIR = 0x00;          // Шина данных на вход
    NWR_NRST_OUT = 1;       // Перевести сигнал "R/W_LCD" в неактивное состояние
}
```

Функция перекодировки обеспечивает преобразование ASCII кодов символов в коды знакогенератора индикатора:

```
//Функция перекодировки символа в кириллицу
char LCD_recode(char b)
{
    if (b<192) return b;
    else return LCD_table[b-192];
}
//Таблица кириллицы
```

```

char LCD_table[64]={
    0x41,0xA0,0x42,0xA1,    //0xC0...0xC3 <=> А Б В Г
    0xE0,0x45,0xA3,0x33,    //0xC4...0xC7 <=> Д Е Ж З
    0xA5,0xA6,0x4B,0xA7,    //0xC8...0xCB <=> И Й К Л
    0x4D,0x48,0x4F,0xA8,    //0xCC...0xCF <=> М Н О П
    0x50,0x43,0x54,0xA9,    //0xD0...0xD4 <=> Р С Т У
    0xAA,0x58,0xE1,0xAB,    //0xD5...0xD7 <=> Ф Х Ц Ч
    0xAC,0xE2,0xAC,0xAE,    //0xD8...0xDB <=> Ш Щ Ъ Ы
    0x62,0xAF,0xB0,0xB1,    //0xDC...0xDF <=> Ь Э Ю Я
    0x61,0xB2,0xB3,0xB4,    //0xE0...0xE4 <=> а б в г
    0xE3,0x65,0xB6,0xB7,    //0xE5...0xE7 <=> д е ж з
    0xB8,0xA6,0xBA,0xBB,    //0xE8...0xEB <=> и й к л
    0xBC,0xBD,0x6F,0xBE,    //0xEC...0xEF <=> м н о п
    0x70,0x63,0xBF,0x79,    //0xF0...0xF4 <=> р с т у
    0xE4,0xD5,0xE5,0xC0,    //0xF5...0xF7 <=> ф х ц ч
    0xC1,0xE6,0xC2,0xC3,    //0xF8...0xFB <=> ш щ ъ ы
    0xC4,0xC5,0xC6,0xC7    //0xFC...0xFF <=> ь э ю я
};

```

Объявление используемых констант (lcd.h):

```

#ifndef __LCD_H__
#define __LCD_H__
#include <io430.h>

// direction
#define D_NC_LCD_DIR P3DIR_bit.P3DIR_4 //Режим 1-данные, 0-команда для LCD
#define EN_LCD_DIR P3DIR_bit.P3DIR_5 //Разрешение обращений к модулю LCD
    //(а также строб данных)
#define NWR_NRST_DIR P1DIR_bit.P1DIR_2
#define NSS_DIR P1DIR_bit.P1DIR_0
#define MCU_SEL_0_DIR P1DIR_bit.P1DIR_3
#define MCU_SEL_1_DIR P1DIR_bit.P1DIR_4
#define DB_DIR P4DIR

// output
#define NWR_NRST_OUT P1OUT_bit.P1OUT_2
#define NSS_OUT P1OUT_bit.P1OUT_0
#define MCU_SEL_0_OUT P1OUT_bit.P1OUT_3
#define MCU_SEL_1_OUT P1OUT_bit.P1OUT_4
#define DB_OUT P4OUT
#define D_NC_LCD_OUT P3OUT_bit.P3OUT_4 //Режим 1-данные, 0-команда для LCD
#define EN_LCD_OUT P3OUT_bit.P3OUT_5 //Разрешение обращений к модулю LCD
    //(а также строб данных)

// input
#define DB_IN P4IN

#endif

```

4.1.2 Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс-контроль знаний по принципам функционирования микроконтроллера MSP430, системе команд и возможностям организации программного управления выводом символов на экран цифрового индикатора.

При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствие с индивидуальным заданием.

Задание. Разработать в среде программирования IAR Embedded Workbench программу для микроконтроллера MSP430, которая выполняет вывод информации (фамилию, имя и отчество студента, и группу) на экран цифрового индикатора.

Порядок выполнения задания:

- запустить компилятор **IAR Embedded Workbench**.
- создать пустой проект.
- создать файл ресурса для кода программы и подключить его к проекту.
- выполнить компиляцию исходного модуля программы и устранить ошибки, полученные на данном этапе.
- проверить работоспособность программы и показать результаты работы преподавателю.

4.1.3 Содержание отчета

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный модуль разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности управления выводом данных на экран цифрового индикатора с помощью микроконтроллера MSP430x4xx.

4.1.4 Контрольные вопросы и задания

1. Поясните принцип функционирования цифрового индикатора, подключаемого к лабораторному макету.

2. Поясните алгоритм программного управления контроллером цифрового индикатора.

3. Каким образом можно осуществлять вывод информации на цифровой индикатор в фиксированные позиции?

4. Поясните принципы использования команд установки и сброса отдельных битов; приведите примеры.

5. Приведите алгоритм универсальной программы управления цифровым индикатором.

4.2 ИЗУЧЕНИЕ ПРИНЦИПОВ ОРГАНИЗАЦИИ ОБМЕНА ДАННЫМИ ПО ПОСЛЕДОВАТЕЛЬНОМУ ИНТЕРФЕЙСУ USB МЕЖДУ МИКРОКОНТРОЛЛЕРОМ MSP430F1611 И ПЭВМ

Цель работы: Изучить возможности сопряжения лабораторного стенда на базе микроконтроллера MSP430F1611 и ПЭВМ с помощью последовательного интерфейса USB, принципы программного управления двунаправленным обменом данных по последовательному интерфейсу USB.

4.2.1 Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе [1-4] и конспекту лекций, ознакомиться с основными возможностями и принципами функционирования последовательного интерфейса USB, возможностями сопряжения лабораторного макета на базе микроконтроллера MSP430F1611 и ПЭВМ с помощью последовательного интерфейса USB, принципами программного управления двунаправленным обменом данных по последовательному интерфейсу USART.

4.2.1.1 Организация модулей USART в микроконтроллере MSP430

Универсальный синхронно/асинхронный приемопередающий (USART) периферийный интерфейс поддерживает два последовательных режима. USART0 реализован в устройствах MSP430x12xx, MSP430x13xx и MSP430x15x. В дополнение к USART0 в устройствах MSP430x14x и MSP430x16x реализован второй идентичный USART модуль – USART1.

В асинхронном режиме USART подключает MSP430 к внешней системе через два внешних вывода: URXD и UTXD. Режим UART выбирается при очистке бита SYNC.

На рисунке 4.2.1 показан модуль USART, настроенный на режим работы UART.

Режим UART

В режиме UART модуль USART передает и принимает символы на скорости, асинхронной другому устройству. Синхронизация каждого символа основана на выбранной скорости передачи USART. Для выполнения функций передачи и приема используется одинаковая скорость в бодах.

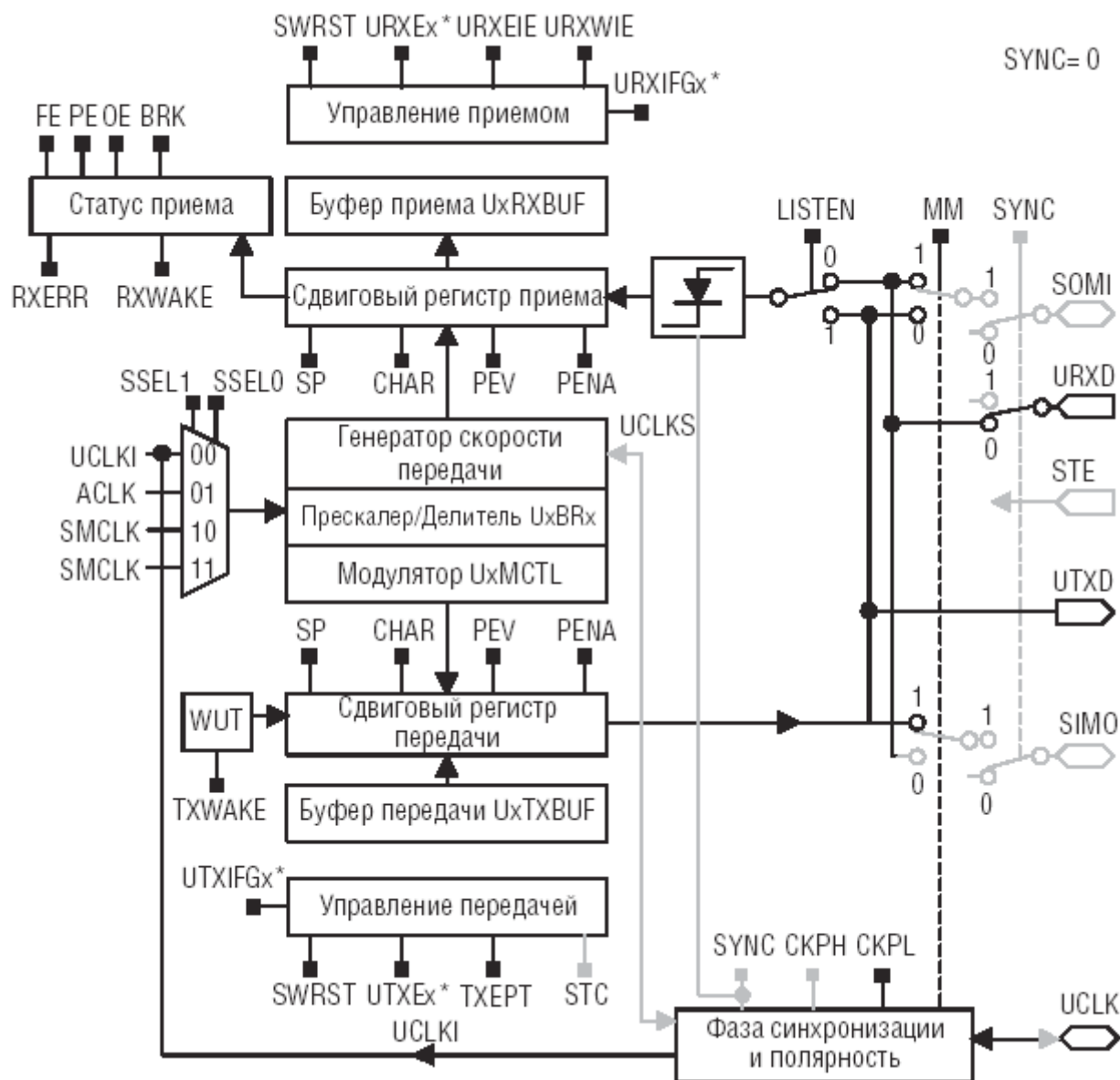


Рисунок 4.2.1 – Модуль USART в режиме UART

Инициализация и сброс USART

Модуль USART сбрасывается сигналом PUC или при установке бита SWRST. После PUC бит SWRST автоматически устанавливается, переводя USART в состояние сброса. Когда бит SWRST установлен, биты URXIE_x, UTXIE_x, URXIFG_x, RXWAKE, TXWAKE, RXERR, BRK, PE, OE, FE сбрасываются, а биты UTXIFG_x и TXEPT – устанавливаются. Флаги разрешения приема и передачи URXEx и UTXEx не изменяются битом SWRST. Очистка SWRST переводит модуль USART в активный режим.

Процесс инициализации/реконфигурирования USART необходимо выполнять в следующей последовательности:

- установить SWRST (BIS.B #SWRST,&UxCTL);
- проинициализировать все регистры USART при установленном SWRST=1 (включая UxCTL);
- включить модуль USART через MEx SFRs (URXEx и/или UTXEx);
- программно очистить SWRST (BIC.B #SWRST,&UxCTL);
- разрешить прерывания (если необходимо) через IEx SFRs (URXIEх и/или UTXIEх).

Невыполнение этой последовательности может привести к непредсказуемому поведению USART.

Формат символа

Формат символа USART (рисунок 4.2.2) содержит стартовый бит, семь или восемь битов данных, бит контроля четности, адресный бит (в адресном режиме) и один или два стоповых бита. Период битов определяется выбранным источником тактовых импульсов и настройкой регистров скорости передачи.

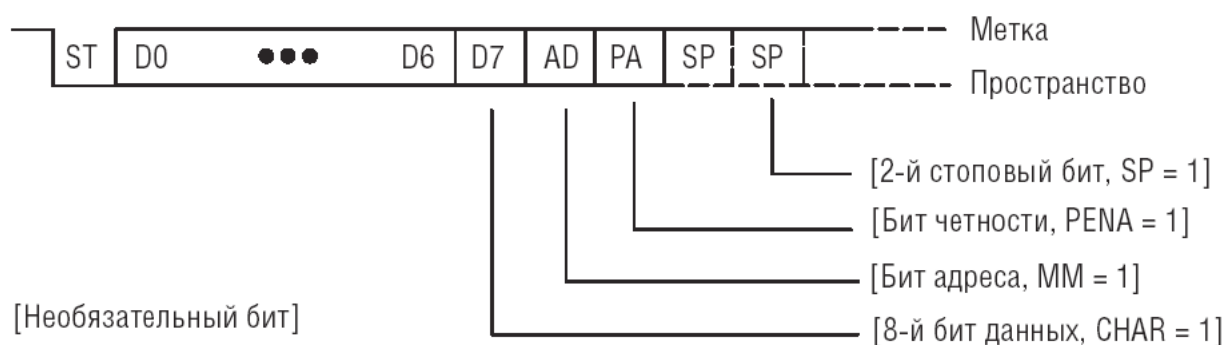


Рисунок 4.2.2 – Формат символа

Асинхронные коммуникационные форматы

Когда два устройства обмениваются информацией асинхронно, в качестве протокола используется формат «свободная линия». Для обмена данными между тремя или более устройствами, USART поддерживает многопроцессорные коммуникационные форматы: формат со свободной линией и формат с адресным битом.

Многопроцессорный формат со свободной линией

При MM=0 выбирается многопроцессорный формат со свободной линией. Блоки данных на линиях передачи или приема должны быть разделены временем простоя (рисунок 4.2.3). Простой линии приема обнаруживается, когда приняты 10 или более непрерывных логических единиц (меток) после первого стопового бита символа. Если используются два стоповых бита, то

второй стоповый бит воспринимается как первый маркерный бит периода простоя.

Первый символ, принятый после периода простоя, распознается как адрес. Бит RXWAKE используется как адресный тэг для каждого фрейма. В многопроцессорном формате свободной линии этот бит устанавливается в 1, когда принятый адрес помещается в UxRXBUF.

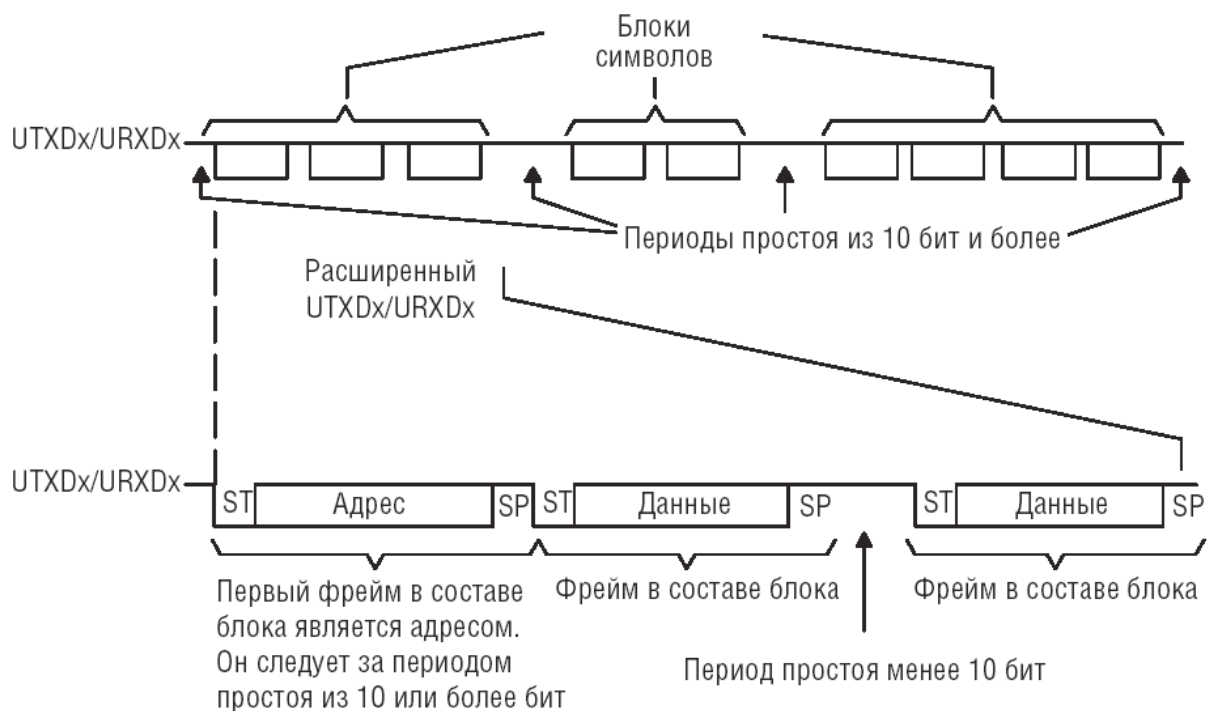


Рисунок 4.2.3 – Формат со свободной линией

Бит URXWIE используется для приема управляющих данных в многопроцессорном формате со свободной линией. Когда бит URXWIE установлен, все неадресные символы обрабатываются, но не перемещаются в UxRXBUF и прерывания не генерируются. Когда принят адресный символ, приемник временно активизируется для переноса символа в UxRXBUF и установки флага прерывания URXIFGx.

После приема адреса программное обеспечение пользователя должно проверить его корректность и сбросить URXWIE для продолжения приема данных. Если URXWIE остается установленным, будут приниматься только адресные символы. Бит URXWIE не изменяется аппаратно.

При передаче адреса в многопроцессорном формате со свободной линией точный период простоя для генерации идентификаторов адресного символа на

UTXD_x может быть сгенерирован модулем USART с использованием флага временного пробуждения (WUT). Когда в передатчик загружаются данные из UxTXBUF, в WUT сохраняется состояние бита TXWAKE, а сам бит TXWAKE сбрасывается.

Для формирования фрейма простоя используется следующая процедура:

1. Устанавливается TXWAKE, что приводит к записи любого символа в UxTXBUF, при этом UxTXBUF должен быть готов для новых данных (UTXIFG_x=1).

2. Значение TXWAKE сдвигается в WUT, а содержимое UxTXBUF сдвигается в сдвиговый регистр передачи, когда он готов для передачи новых данных. Это приводит к установке бита WUT, который препятствует нормальной передаче битов старта, данных и контроля четности, поэтому происходит передача периода простоя длительностью точно 11 бит, после чего бит TXWAKE сбрасывается автоматически.

3. Требуемый адресный символ записывается в UxTXBUF.

4. Новый символ, представляющий требуемый адрес, сдвигается наружу после периода простоя.

Многопроцессорный формат с адресным битом

При MM=1 выбирается многопроцессорный формат с адресным битом. Каждый обрабатываемый символ содержит дополнительный бит, используемый как указатель адреса (рисунок 4.2.4). Первый символ в блоке фреймов несет установленный бит адреса, который указывает, что этот символ является адресом. Бит USART RXWAKE устанавливается, когда в UxRXBUF записывается принятый адресный символ фрейма.

Бит URXWIE используется для приема управляющих данных в многопроцессорном формате с адресным битом. Когда бит URXWIE установлен, символы данных (бит адреса равен 0) обрабатываются приемником, но не перемещаются в UxRXBUF и прерывания не генерируются. Когда принятый символ содержит установленный адресный бит, приемник временно активизируется для переноса символа в UxRXBUF и установки флага прерывания URXIFG_x.

Если адрес принят, программное обеспечение пользователя должно сбросить URXWIE для продолжения приема данных. Если URXWIE остается установленным, будут приниматься только адресные символы (адресный бит равен 1). Бит URXWIE не изменяется аппаратно.

При передаче адреса в многопроцессорном режиме с адресным битом, адресный бит символа может изменяться путем записи бита TXWAKE.

Значение бита TXWAKE загружается в адресный бит символа, перемещенного из UxTXBUF в сдвиговый регистр передачи, при этом бит TXWAKE автоматически очищается.

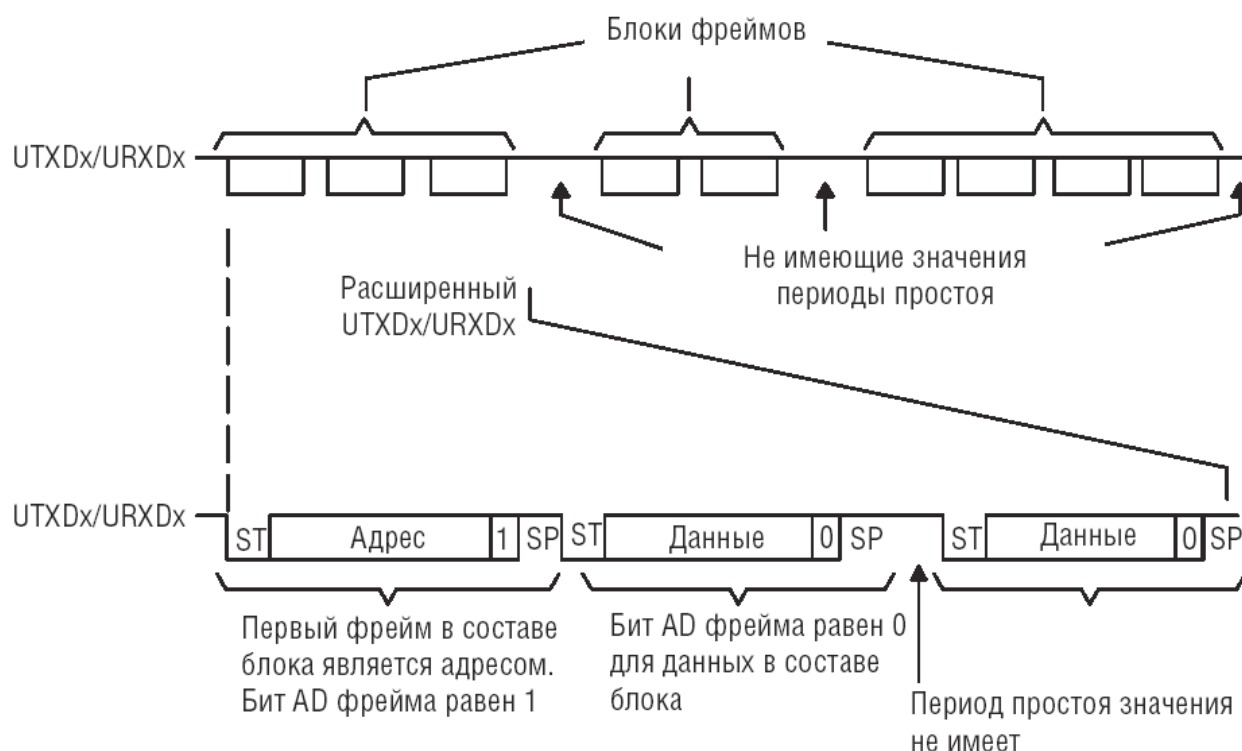


Рисунок 4.2.4 – Многопроцессорный формат с адресным битом

Автоматическое обнаружение ошибок

Подавление импульсных помех предотвращает случайный запуск USART. Любой сигнал низкого уровня на URXDx продолжительностью менее $t_i = 300$ нс игнорируется.

Если длительность сигнала низкого уровня на URXDx превышает t_i , этот сигнал мажоритарно принимается за стартовый бит. Если стартовый бит не будет распознан, то модуль USART приостанавливает прием символа и ожидает следующего периода низкого уровня на URXDx. Мажоритарный принцип также используется для предотвращения поразрядных ошибок при приеме каждого бита символа.

Модуль USART при приеме символов автоматически обнаруживает ошибки фрейма, четности, переполнения и прерывания (разрыва). Обнаружение ошибки приводит к установке соответствующих битов FE, PE, OE и BRK. При установке любого из этих флагов также устанавливается RXERR. Ситуации сбоя описаны в таблице 4.2.1.

Если обнаружена ошибка фрейма, четности или состояние разрыва и URXEIE=0, то символы приниматься не будут. Когда URXEIE=1, символы принимаются и сохраняются в UxRXBUF, при этом устанавливаются соответствующие биты ошибок.

Если любой из битов FE, PE, OE, BRK или RXERR был установлен, то он сохраняет свое состояние до сброса программным обеспечением или до чтения UxRXBUF.

Таблица 4.2.1 – Ошибки приема

Ошибочное состояние	Описание
Ошибка фрейма	Ошибка фрейма (кадровой синхронизации) происходит при обнаружении стопового бита с низким уровнем. Когда используется два стоповых бита, на ошибку фрейма проверяется только первый стоповый бит. При обнаружении ошибки фрейма устанавливается бит FE.
Ошибка четности	Ошибка четности – несоответствие между числом единиц в фрейме и значением бита четности. Когда бит адреса включен в фрейм, он учитывается при определении четности. При обнаружении ошибки четности устанавливается бит PE.
Ошибка переполнения приема	Ошибка переполнения появляется в случае, когда символ загружается в UxRXBUF до прочтения предыдущего символа. Когда происходит переполнение, устанавливается бит OE.
Ошибка прерывания (разрыва)	Состояние разрыва – это период 10 или более нулевых битов, принятых на URXDx после пропущенного стопового бита. Когда обнаруживается состояние разрыва, устанавливается бит BRK. Состояние разрыва также устанавливает флаг прерывания URXIFGx.

Контроллер скорости передачи UART

Контроллер скорости передачи включает в себя один прескалер/делитель и модулятор (рисунок 4.2.5). Такая структура позволяет получить дробные коэффициенты деления при генерации скорости передачи в бодах. Максимальная скорость передачи USART составляет одну треть от тактовой частоты BRCLK.

Синхронизация каждого бита показана на рисунке 4.2.6. Для каждого полученного бита используется мажоритарный принцип определения значения бита. Мажоритарные выборки происходят в $N/2-1$, $N/2$ и $N/2+1$ периоды BRCLK, где N – число импульсов BRCLKs на один импульс BITCLK.

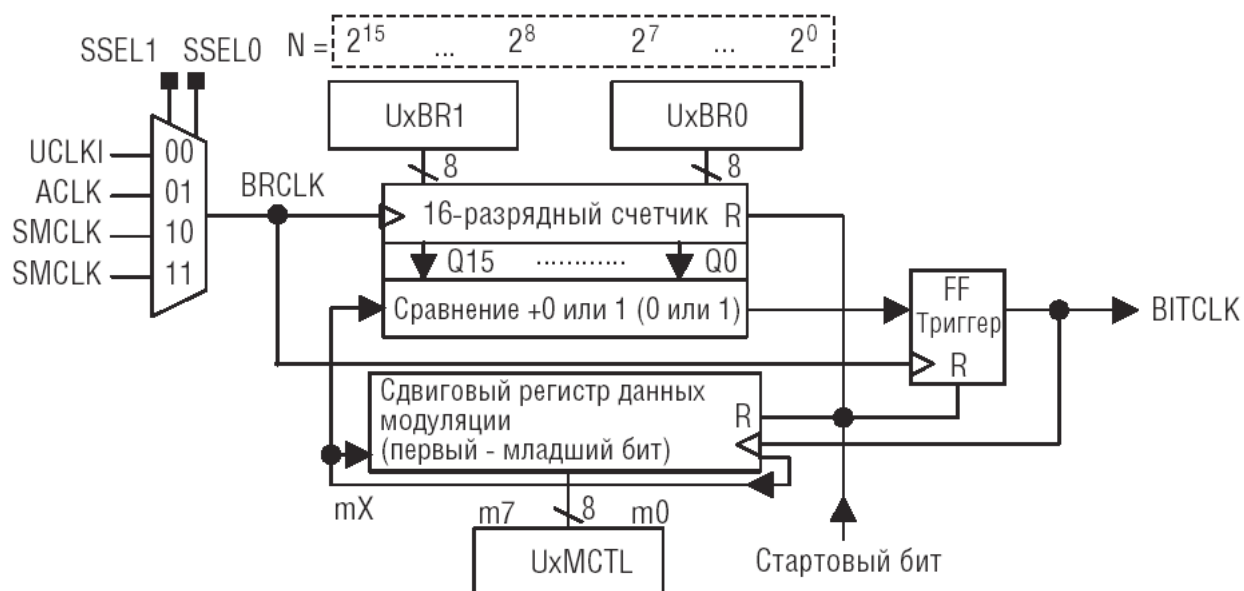


Рисунок 4.2.5 – Контроллер скорости передачи MSP430

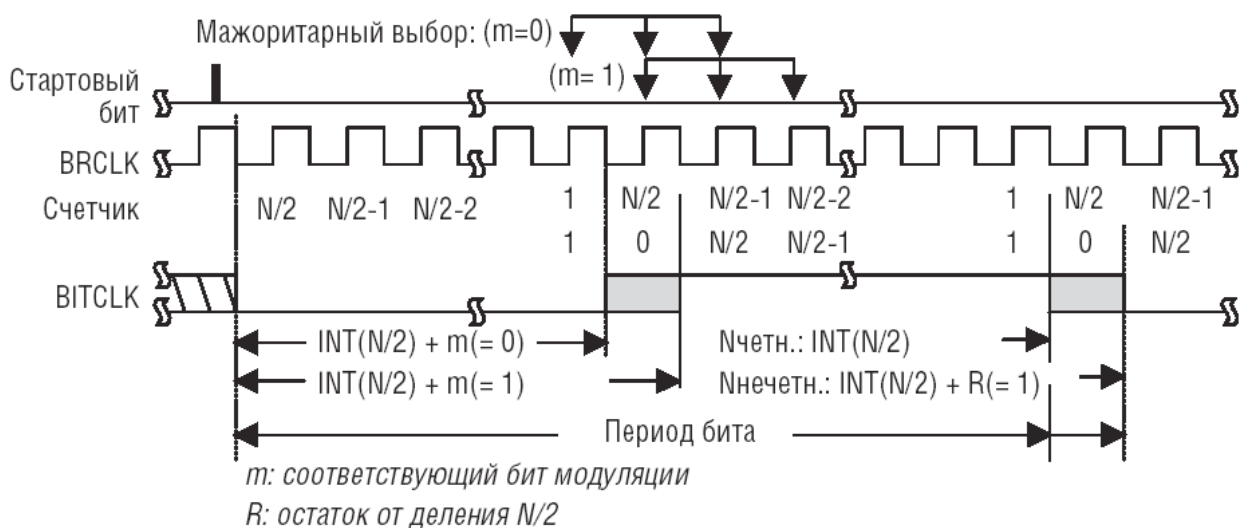


Рисунок 4.2.6 – Синхронизация скорости передачи BITCLK

Синхронизация скорости передачи бит

Первая ступень контролера скорости передачи – 16-разрядный счетчик и компаратор. В начале передачи или приема каждого бита счетчик загружается величиной $INT(N/2)$, где N – значение, сохраненное в регистрах UxBR0 и UxBR1. Счетчик перезагружает $INT(N/2)$ каждый полупериод периода бита, обеспечивая полный период бита N BRCLK. Для данного источника тактирования BRCLK, скорость передачи определяется требуемым коэффициентом деления N :

$$N = \frac{BRCLK}{\text{baud rate}}$$

Коэффициент деления N зачастую является нецелым числом, целочисленная часть которого может быть принята прескалером/делителем. Вторая ступень генератора скорости передачи – модулятор, используется для максимально точного учета дробной части. Коэффициент деления N в этом случае определяется так:

$$N = UxBR + \frac{1}{n} \sum_{i=0}^{n-1} m_i$$

где N - получаемый коэффициент деления;

$UxBR$ - 16-разрядное представление регистров $UxBR0$ и $UxBR1$;

i - позиция бита в фрейме;

n - общее количество битов в фрейме;

m_i - данные каждого соответствующего модуляционного бита (1 или 0).

$$\text{Baud rate} = \frac{BRCLK}{N} = \frac{BRCLK}{UxBR + \frac{1}{n} \sum_{i=0}^{n-1} m_i}$$

Определение модуляционного значения

Определение модуляционного значения – интерактивный процесс. Использование формулы ошибки синхронизации, начиная со стартового бита, позволяет рассчитать ошибку для каждого бита с последующей установкой или сбросом соответствующего бита модуляции. Модуляционный бит устанавливается с наименьшей выбранной и рассчитанной ошибкой следующего бита. Этот процесс продолжается до минимизации ошибок всех битов. Если фрейм содержит более 8 бит, модуляционные биты повторяются. К примеру, 9-ый бит фрейма использует бит модуляции 0.

Типовые скорости передачи и ошибки

Стандартные скорости передачи данных в бодах для $UxBR$ и $UxMCTL$ приведены в таблице 4.2.2 для часового кристалла (ACLK) на 32768 Гц и для типичного значения $SMCLK$ 1048576 Гц.

Ошибка приема – это накопленное время в сравнении с идеальным временем загрузки сдвигового регистра в середине каждого бита. Ошибка

передачи – накопленное время ошибки в сравнении с идеальным временем периода бита.

Таблица 4.2.2 – Наиболее часто используемые величины скорости передачи, скорость передачи данных в бодах и ошибки

Скорость передачи, бод	Деление на		А: BRCLK = 32768 Гц						В: BRCLK = 1048576 Гц				
	А:	В:	UxBR1	UxBR0	UxMCTL	Макс. ошибка TX, %	Макс. ошибка RX, %	Ошибка синхр. RX, %	UxBR1	UxBR0	UxMCTL	Макс. ошибка TX, %	Макс. ошибка RX, %
1200	27.31	873.81	0	1B	03	-4/3	-4/3	±2	03	69	FF	0/0.3	±2
2400	13.65	436.91	0	0D	6B	-6/3	-6/3	±4	01	B4	FF	0/0.3	±2
4800	6.83	218.45	0	06	6F	-9/11	-9/11	±7	0	DA	55	0/0.4	±2
9600	3.41	109.23	0	03	4A	-21/12	-21/12	±15	0	6D	03	-0.4/1	±2
19200		54.61							0	36	6B	-0.2/2	±2
38400		27.31							0	1B	03	-4/3	±2
76800		13.65							0	0D	6B	-6/3	±4
115200		9.1							0	09	08	-5/7	±7

Прерывания USART

USART имеет один вектор прерывания для передачи и один вектор прерывания для приема.

Функционирование прерывания USART при передаче

Флаг прерывания UTXIFGx устанавливается передатчиком для индикации готовности UxTXBUF к приему другого символа. Запрос прерывания генерируется, если установлены флаги UTXIEх и GIE. UTXIFGx автоматически сбрасывается при входе в обработчик прерывания или при записи нового символа в UxTXBUF.

UTXIFGx устанавливается после PUC или при SWRST=1. UTXIEх сбрасывается после PUC или когда SWRST=1 (рисунок 4.2.7).

Функционирование прерывания USART при приеме

Флаг прерывания URXIFGx устанавливается каждый раз при приеме символа и его загрузки в UxRXBUF. Запрос прерывания генерируется, если установлены флаги URXIEх и GIE. URXIFGx и URXIEх сбрасываются сигналом системного сброса PUC или при SWRST=1. URXIFGx сбрасывается автоматически при входе в обработчик прерывания (если URXSE=0) или при чтении данных из UxRXBUF (рисунок 4.2.8).

Форматы управляющих регистров Модуля UART в [1] стр. 240-249.

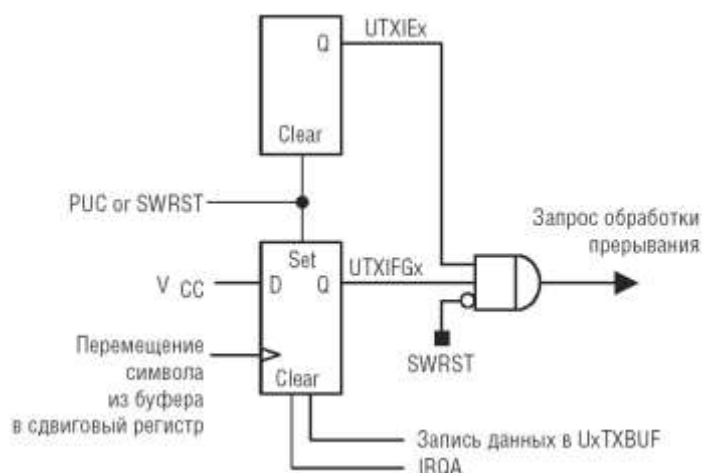


Рисунок 4.2.7 – Прерывание при передаче

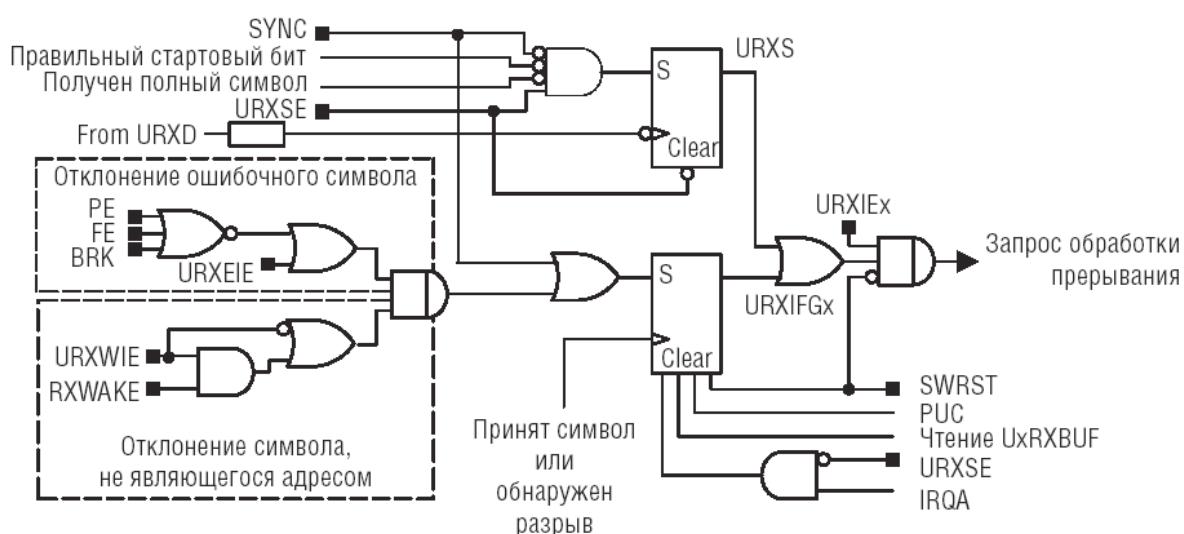


Рисунок 4.2.8 – Прерывание при приеме

4.2.2 Описание лабораторной установки

Задания выполняются на лабораторном стенде на базе 16-ми разрядного микроконтроллера MSP430. Дополнительно в работе используется USB для соединения лабораторного стенда с ПЭВМ через последовательный интерфейс USB.

В лабораторном стенде реализовано подключение последовательного порта UART1 к интерфейсу USB и оптоволоконной линии (POL). Переключение между интерфейсами выполняется с помощью сигнала $\overline{POL/UART}$, при высоком его уровне мультиплексор коммутирует вход и выход UART1 на USB, при низком уровне – на оптические преобразователи.

Для подключения по протоколу USB используется микросхема FT232RL, которая обеспечивает преобразования формата данных из USB в UART. Данная

микросхема содержит внутренний генератор 24 МГц и энергонезависимую память, что не требует установки внешних компонентов, как для предыдущих версий этих микросхем. Начиная с ОС Windows 2000, в ядре системы имеются драйвера, автоматически распознающие данную микросхему как дополнительный COM-порт со скоростью передачи до 3 МБод.

Оптические преобразователи HFBR1522/2522 позволяют организовать обмен данными через оптоволоконную линию связи на скорости до 1МБод. Подключение оптических преобразователей позволяет реализовать все преимущества ВОЛС: обеспечение высокой скорости передачи, гальваническую развязку, помехозащищенность и защиту данных от несанкционированного доступа.

На рисунках 4.2.9 и 4.2.10 показано размещение интерфейсов USB и ВОЛС на плате станда. Схемы подключения интерфейса USB, оптических приемопередатчиков и мультиплексора, осуществляющего выбор необходимого модуля, показаны на рисунке 4.2.11. В таблицах 4.2.5 - 4.2.7 приведено описание сигналов этих модулей.



Рисунок 4.2.9 – Размещение интерфейса USB на плате

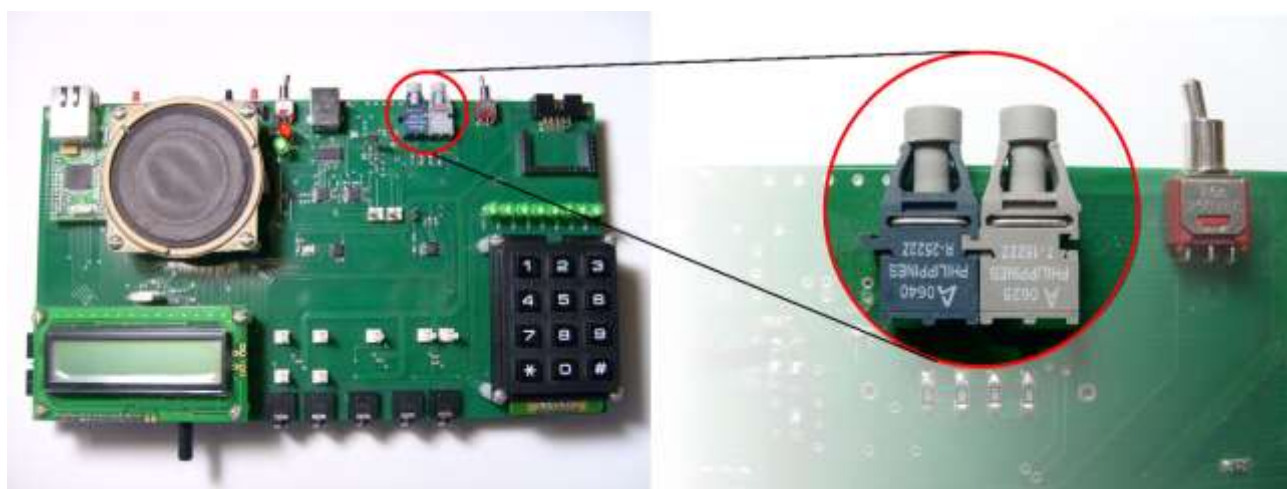


Рисунок 4.2.10 – Размещение оптических преобразователей на плате

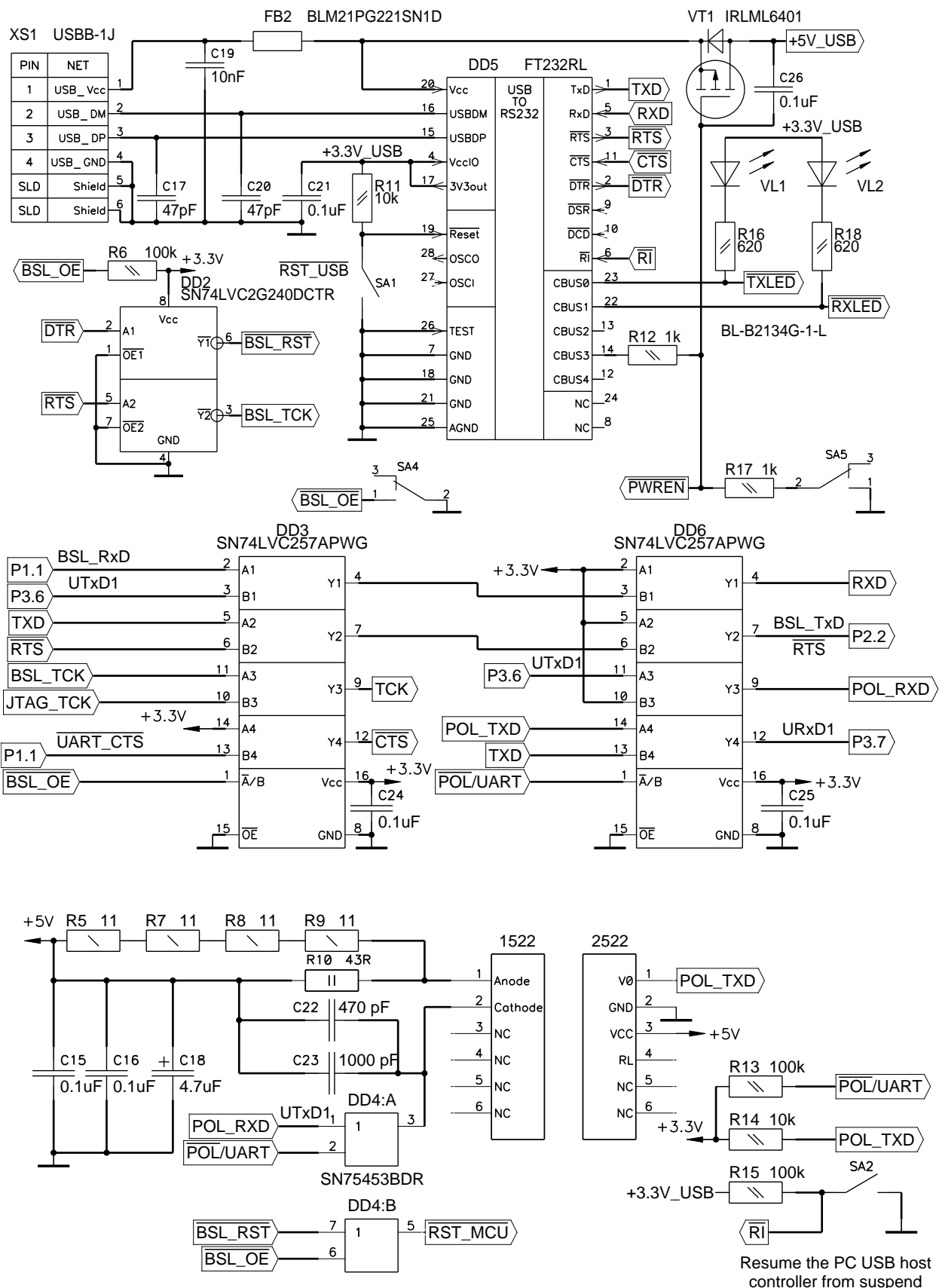


Рисунок 4.2.11 – Принципиальная схема подключения интерфейса USB, мультиплексора и оптических преобразователей

Таблица 4.2.5 – Описание сигналов и назначение выводов FT323RL

Вывод DD5	Название	Подключение к цепи / [номер вывода]	Примечание
1	2	3	4
1	TxD	DD3 / 5, DD6 / 13	Линия передачи
2	$\overline{\text{DTR}}$	$\overline{\text{DTR}}$	Готовность установить соединение
3	$\overline{\text{RTS}}$	DD3 / 6	Готовность обмена данными
4	VccIO	+3,3V	Опорное напряжение для ввода/вывода
5	RxD	DD6 / 4	Линия приема
6	$\overline{\text{RI}}$	SA2	Сигнал пробуждения PC USB
7	GND	GND	Подключение к шине 0 В
8	NC	–	
9	$\overline{\text{DSR}}$	–	Готовность последовательного порта
10	$\overline{\text{DCD}}$	–	Обнаружена несущая частота
11	$\overline{\text{CTS}}$	DD3 / 12	Готовность обмена данными
12	CBUS4	–	
13	CBUS2	–	
14	CBUS3	R12	
15	USBDP	XS1 / 3	USB Data+
16	USBDM	XS1 / 2	USB Data-
17	3V3out	+3,3 V	Выход от встроен. стабилизатора
18	GND	GND	Подключение к шине 0 В
19	$\overline{\text{Reset}}$	$\overline{\text{Reset}}$	Сигнал сброса модуля
20	Vcc	+5V	Напряжение питания ядра
21	GND	GND	Подключение к шине 0 В
22	CBUS1	$\overline{\text{RXLED}}$	Светодиод приема
23	CBUS0	$\overline{\text{TXLED}}$	Светодиод передачи
24	NC	–	
25	GND	GND	Подключение к шине 0 В
26	TEST	GND	Подключение к шине 0 В
27	OSCI	–	
28	OSCO	–	

Таблица 4.2.6 – Описание сигналов и назначение выводов мультиплексора SN74LVC257 (DD3)

Вывод DD3	Название	Подключение к цепи / [номер вывода]	Примечание
1	\overline{A}/B	BSL_OE	Сигнал выбора
2	A1	P1.1	
3	B1	P3.6	
4	Y1	DD6 / 3	
5	A2	TXD	
6	B2	RTS	
7	Y2	DD6 / 6	
8	GND	GND	Подключение к шине 0 В
9	Y3	TCK	
10	B3	JTAG_TCK	
11	A3	BSL_TCK	
12	Y4	CTS	
13	B4	P1.1	
14	A4	+3,3 В	Подключение к шине 3,3 В
15	\overline{OE}	GND	Подключение к шине 0 В
16	Vcc	+3,3 В	Напряжение питания

Таблица 4.2.7 – Описание сигналов и назначение выводов мультиплексора SN74LVC257 (DD6)

Вывод DD6	Название	Подключение к цепи / [номер вывода]	Примечание
1	\overline{A}/B	POL/UART	Сигнал выбора
2	A1	+3,3 В	Подключение к шине 3,3 В
3	B1	DD3 / 4	
4	Y1	RXD	
5	A2	+3,3 В	Подключение к шине 3,3 В
6	B2	DD3 / 7	
7	Y2	P2.2	
8	GND	GND	Подключение к шине 0 В
9	Y3	POL_RXD	
10	B3	+3,3 В	Подключение к шине 3,3 В
11	A3	P3.6	
12	Y4	P3.7	
13	B4	TXD	
14	A4	POL_TXD	
15	\overline{OE}	GND	Подключение к шине 0 В
16	Vcc	+3,3 В	Напряжение питания

Работа с программой CommTest (рисунок 4.2.12) выполняется путем настройки соответствующих параметров протокола обмена в верхней части рабочего окна и ввода отправляемых (в области Transmit) или наблюдения принимаемых (в области Receive) данных в десятичной, шестнадцатеричной или двоичной кодировке.

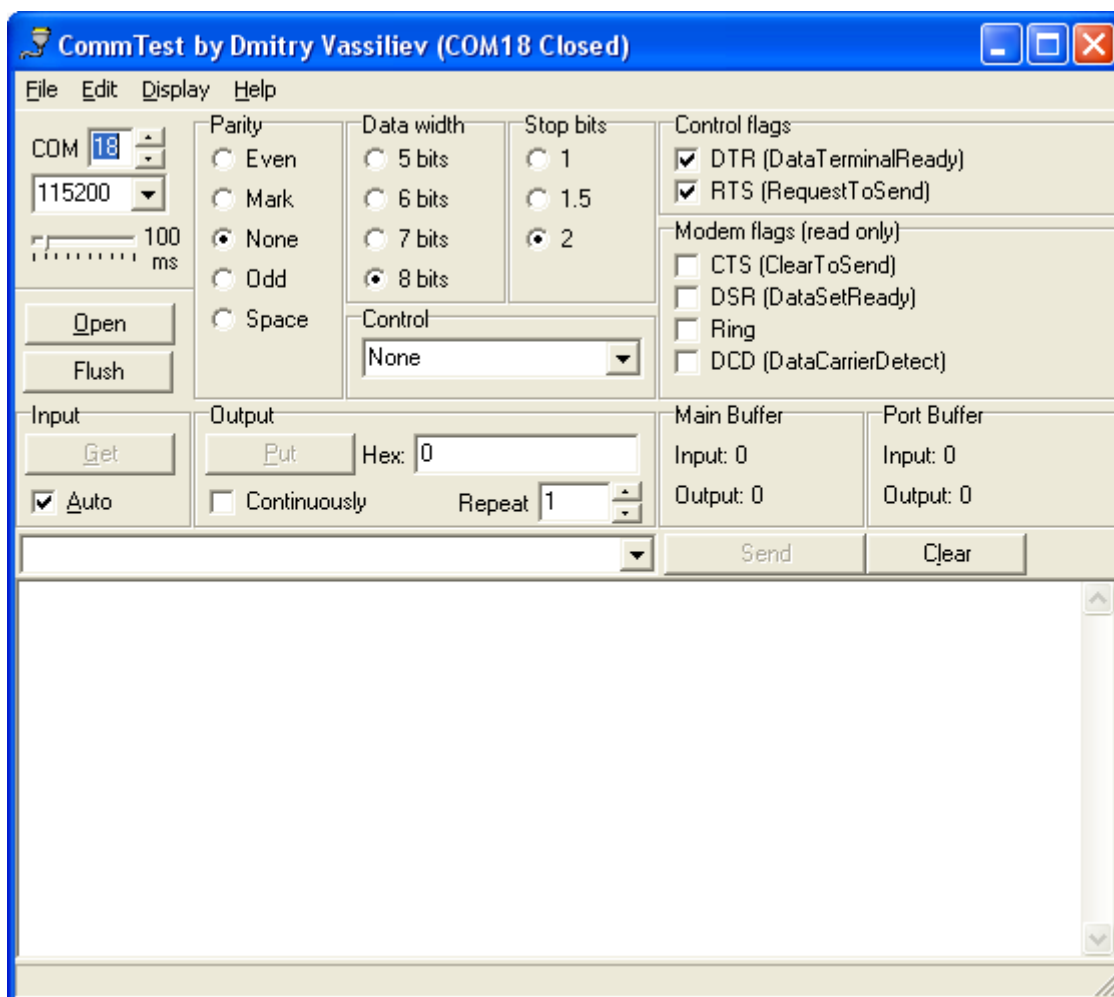


Рисунок 4.2.12 – Рабочее окно программы Terminal

4.2.3 Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс-контроль знаний по принципам функционирования модулей USART, входящих в состав микроконтроллера MSP430, а также по протоколу обмена данными по интерфейсу USB. При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствие с индивидуальным заданием (таблица 4.2.8).

Задание. Разработать в среде программирования IAR Embedded Workbench программу на языке C для связи микроконтроллера MSP430F1611 с ПЭВМ по интерфейсу USB в соответствии с параметрами протокола обмена, приведенными в таблице 4.2.8.

Для решения задачи можно использовать следующие функции:

```
//-----
// Инициализация режима UART
// speed = 0 - 38400, 1 - 57600, 2 - 115200 - скорость обмена
// databits = 7, 8 - длина символа
// stopbits = 1, 2 - кол-во передаваемых стоповых бит
// parity = 0 - без контроля четности, 1 - контроль четности, нечетный, 2 -
четный
// iface = 0 - USB, 1 - оптика
void UART_init(byte speed, byte databits, byte stopbits, byte parity, byte
iface)
{
    P3SEL |= BIT6 | BIT7;          // выбор функции USART1
    U1CTL = 0;                     // инициализация состояния USART
    ME2 |= UTXE1 + URXE1;          // включить приемник и передатчик USART1

    if (databits == 7) U1CTL &= ~CHAR;      // 7-разрядная длина символа
    if (databits == 8) U1CTL |= CHAR;       // 8-разрядная длина символа
    if (stopbits == 1) U1CTL &= ~SPB;       // 1 стоповый бит
    if (stopbits == 2) U1CTL |= SPB;        // 2 стоповых бита
    if (parity == 0) U1CTL &= ~PENA;        // контроль четности отключен
    if (parity == 1) U1CTL = (U1CTL & ~PEV) | PENA; // контроль четности,
нечетный
    if (parity == 2) U1CTL |= PENA | PEV;    // контроль четности, четный

    P5DIR |= BIT0;                  // переключение мультимплектора на USB/оптику
    if (iface == 0)
        P5OUT |= BIT0;
    if (iface == 1)
        P5OUT &= ~BIT0;

    U1TCTL |= SSEL1;                // BRCLK = SMCLK
    U1BR0 = 69;                     // 8Mhz / 115200 = 69.44 (по-
умолчанию)
    if (speed == 0) U1BR0 = 208;     // 8МГц / 38400 = 208.33
    if (speed == 1) U1BR0 = 139;    // 8МГц / 57600 = 138,89
    U1BR1 = 0x00;
    U1MCTL = 0x2C;                  // модуляция
}
//-----
// отключение режима UART
void UART_off()
{
    P3SEL |= BIT6 | BIT7;          // выбор функции USART1
    ME2 &= ~(UTXE1 + URXE1);       // выключить приемник и передатчик USART1
    U1CTL = SWRST;                  // отключение USART1
}
```

```

}
//-----
// вывод строки символов (символ с кодом 0 - конец строки)
void UART_message(char * buf)
{
    word i=0;
    while (buf[i])
        UART_sendbyte(buf[i++]);    // передача символа
}
//-----
// передача байта
void UART_sendbyte(char byte)
{
    while (!(IFG2 & UTXIFG1));    // проверка готовности буфера передачи
USART1
    U1TXBUF = byte;                // передача байта
}
//-----
// получение байта
char UART_getbyte()
{
    while (!(IFG2 & URXIFG1));    // проверка готовности буфера приема
USART1
    return U1RXBUF;                // возврат полученного байта
}
//-----

```

Порядок выполнения задания:

- включить лабораторный макет.
- запустить компилятор IAR Embedded Workbench.
- создать пустой проект.
- создать файл ресурса для кода программы и подключить его к проекту.
- ввести код исходного модуля программы обмена данными между микроконтроллером MSP430F1611 с ПЭВМ по интерфейсу USB в соответствии с индивидуальным заданием, приведенным в таблице 4.2.8.
- выполнить компиляцию исходного модуля программы и устранить ошибки, полученные на данном этапе.
- настроить параметры программатора.
- проверить правильность подключения интерфейсного USB кабеля к разъемам лабораторного макета и ПЭВМ.
- запустить на ПЭВМ программу Terminal, установить необходимые параметры протокола обмена данными, выбрать номер последовательного порта (COMx), соответствующего виртуальному COM-порту, и нажать на кнопку **Connect** в верхнем левом углу рабочего окна программы.

– создать загрузочный модуль программы и выполнить программирование микроконтроллера.

– проверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.

В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы.

Таблица 4.2.8 – Варианты индивидуальных заданий

№ п.п.	Задание
1	Разработать программу передачи 100 чисел (от 0 до 99) из микроконтроллера MSP430F1611 в ПЭВМ по интерфейсу USB в соответствие с протоколом: модуль USART0, скорость обмена данными 19200 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.
2	Разработать программу передачи 50 чисел (от 20 до 69) из микроконтроллера MSP430F1611 в ПЭВМ по интерфейсу USB в соответствие с протоколом: модуль USART0, скорость обмена данными 38400 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.
3	Разработать программу передачи 20 чисел (от 10 до 29) из микроконтроллера MSP430F1611 в ПЭВМ по интерфейсу USB в соответствие с протоколом: модуль USART0, скорость обмена данными 57600 бит/с, режим обмена асинхронный, 7 битов данных без бита четности.
4	Разработать программу передачи 10 чисел (от 0 до 9) из микроконтроллера MSP430F1611 в ПЭВМ по интерфейсу USB в соответствие с протоколом: модуль USART0, скорость обмена данными 14400 бит/с, режим обмена асинхронный, 7 битов данных без бита четности.
5	Разработать программу передачи 20 чисел (от 10 до 29) из микроконтроллера MSP430F1611 в ПЭВМ по интерфейсу USB в соответствие с протоколом: модуль USART0, скорость обмена данными 19200 бит/с, режим обмена асинхронный, 7 битов данных без бита четности, данные передаются через каждую секунду.*
6	Разработать программу передачи 50 чисел (от 10 до 59) из ПЭВМ в микроконтроллер MSP430F1611 по интерфейсу USB в соответствие с протоколом: модуль USART0, скорость обмена данными 14400 бит/с, режим обмена асинхронный, 7 битов данных без бита четности. При получении последнего информационного кадра выдать сигнал завершения приема на светодиод.
7	Разработать программу передачи 200 чисел (от 0 до 199) из микроконтроллера MSP430F1611 в ПЭВМ по интерфейсу USB в

	соответствие с протоколом: модуль USART0, скорость обмена данными 38400 бит/с, режим обмена асинхронный, 8 битов данных без бита четности.
--	--

4.2.4 Содержание отчета

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный модуль разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности использования встроенных в микроконтроллер модулей USART при реализации обмена данными между лабораторным стендом и ПЭВМ.

4.2.5 Контрольные вопросы и задания

1. Поясните принципы передачи информации по последовательным и параллельным интерфейсам.
2. Назовите современные универсальные интерфейсы и приведите их основные характеристики.
3. Поясните принципы обмена данными по интерфейсу USB.
4. Какие регистры используются для настройки параметров передачи данных с помощью встроенного в микроконтроллер MSP430F1611 блока USART?
5. Какие сигналы прерываний могут генерироваться блоком USART?
6. Поясните формат кадра при обмене данными в форматах со свободной линией и адресным битом.

4.3 ИЗУЧЕНИЕ ПРИНЦИПОВ ОРГАНИЗАЦИИ ОБМЕНА ДАННЫМИ ПО ПОСЛЕДОВАТЕЛЬНОМУ ИНТЕРФЕЙСУ I2C НА ПРИМЕРЕ УПРАВЛЕНИЯ БЛОКОМ СВЕТОДИОДОВ И ПРОГРАММНОГО ОПРОСА КЛАВИАТУРЫ

Цель работы: изучить принципы программного управления двунаправленным обменом данных по последовательному интерфейсу I2C.

4.3.1 Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе [1-4] и конспекту лекций, ознакомиться с основными возможностями и принципами функционирования последовательного интерфейса I2C, принципами программного управления двунаправленным обменом данных по последовательному интерфейсу I2C.

4.3.1.1 Использование модуля USART в режиме I2C

Универсальный синхронно/асинхронный приемопередающий (USART) периферийный интерфейс поддерживает связь по I2C в модулях USART0 устройств MSP430x15x и MSP430x16x (рисунок 4.3.1).

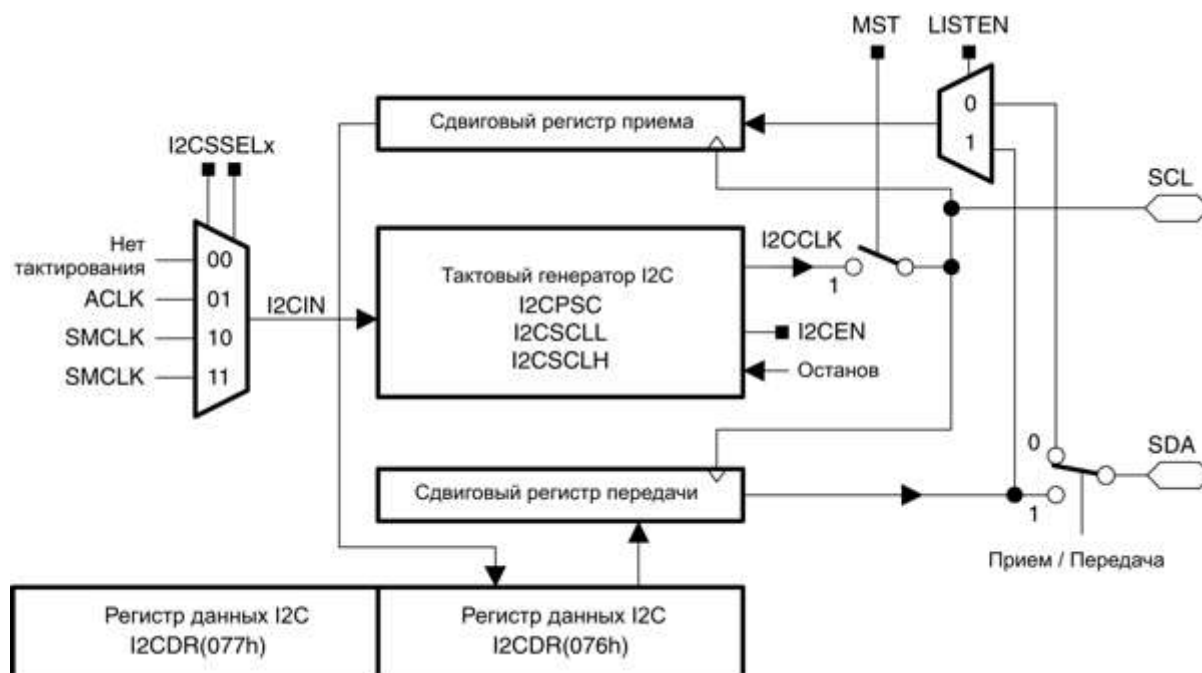


Рисунок 4.3.1 – USART0 в режиме I2C

Модуль управления взаимодействием между интегральными схемами (I2C) обеспечивает интерфейс между MSP430 и I2C-совместимыми устройствами через последовательную двухпроводную шину I2C. Внешние компоненты, подключенные к шине I2C, последовательно передают и принимают последовательные данные в/из USART через 2-х проводной I2C-интерфейс.

Функционирование модуля I2C

Модуль I2C поддерживает любые ведущие и ведомые устройства, совместимые с I2C. На рисунке 4.3.2 показан пример шины I2C. Каждое устройство обладает уникальным адресом и может работать и как передатчик и как приемник. Устройство, подключенное к шине I2C, во время передачи данных может рассматриваться как ведущее или ведомое. Ведущий инициирует передачу данных и генерирует тактовый сигнал SCL. Любое устройство, адресованное ведущим, рассматривается как ведомое.

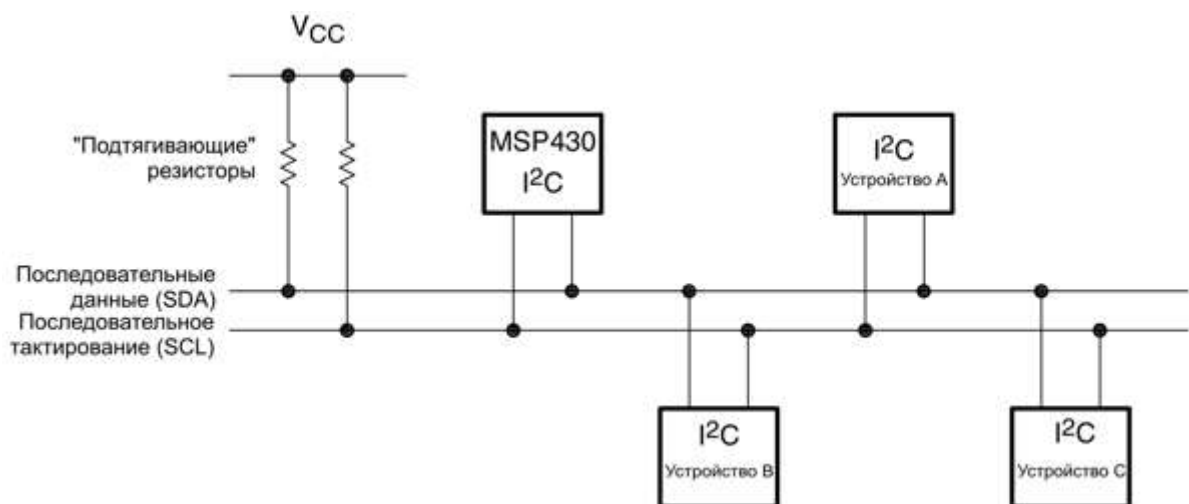


Рисунок 4.3.2 – Схема подключений на шине I2C

Последовательные данные I2C

При передаче каждого бита ведущим устройством генерируется один тактовый импульс. Модуль I2C работает с данными побайтно. Сначала перемещается старший значащий разряд, как показано на рисунке 4.3.3.

Первый, после условия «СТАРТ», байт состоит из 7-разрядного адреса ведомого и бита R/\overline{W} . Когда $R/\overline{W} = 0$, ведущий передает данные ведомому. Когда $R/\overline{W} = 1$, ведущее устройство принимает данные от ведомого. Бит ACK посылается приемником после каждого байта на 9-ом такте SCL.

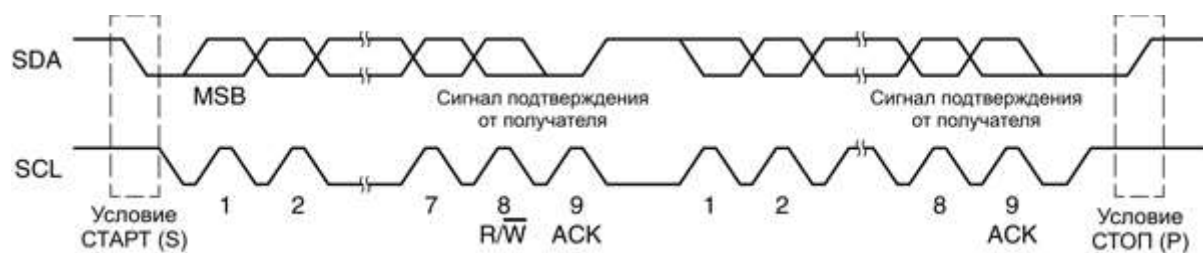


Рисунок 4.3.3 – Передача данных модулем I2C

Данные на SDA должны быть неизменны в течение периода высокого уровня SCL, как показано на рисунке 4.3.4. Высокий и низкий уровень SDA может изменяться, только когда SCL имеет низкий уровень, в противном случае будет сгенерировано условие «старт» или «стоп».



Рисунок 4.3.4 – Передача бита на шине I2C

Условия «СТАРТ» и «СТОП» I2C

Условия «СТАРТ» и «СТОП», показанные на рисунке 4.3.5, генерируются ведущим. Условие «СТАРТ» возникает при переходе с высокого уровня на низкий на линии SDA, когда SCL имеет высокий уровень. Условие «СТОП» появляется при переходе с низкого уровня на высокий на линии SDA при высоком уровне на SCL. Бит занятости I2CBV устанавливается после условия «СТАРТ» и сбрасывается после «СТОП».

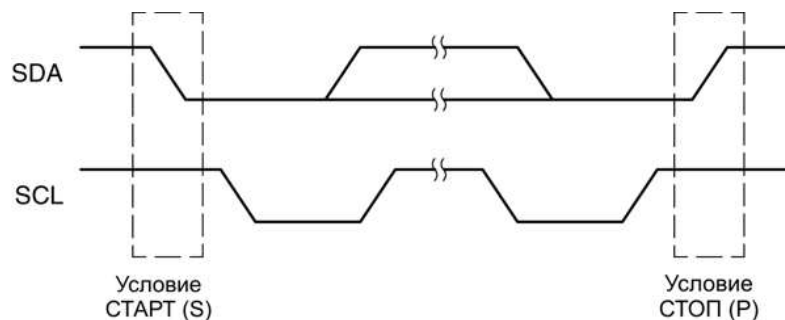


Рисунок 4.3.5 – Условия «СТАРТ» и «СТОП» модуля I2C

Режимы адресации I2C

Модуль I2C поддерживает 7-разрядный и 10-разрядный режимы адресации.

7-разрядная адресация

В 7-разрядном формате адресации (рисунок 4.3.6), первый байт – это 7-разрядный адрес ведомого и бит R/\overline{W} . Бит ACK посылается приемником после каждого байта.



Рисунок 4.3.6 – 7-разрядный формат модуля I2C

10-разрядная адресация

В 10-разрядном адресном формате (рисунок 4.3.7), первый байт содержит 11110b плюс два старших бита 10-разрядного адреса ведомого и бит R/\overline{W} . Бит ACK посылается приемником после каждого байта. Следующий байт содержит оставшиеся 8 бит 10-разрядного адреса ведомого, завершающиеся битом ACK, за которыми следуют байты данных и бит ACK.

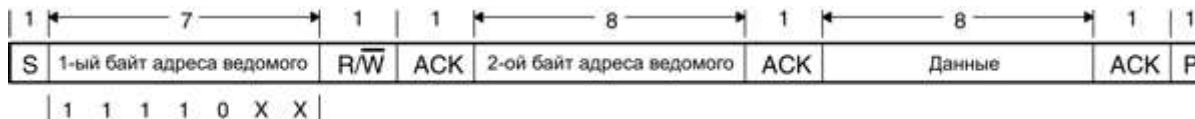


Рисунок 4.3.7 – 10-разрядный адресный формат модуля I2C

Повторные условия «СТАРТ»

Направление потока данных на SDA может быть изменено ведущим без первоначального останова передачи, что приведет к повторению условия «СТАРТ». Это вызовет «РЕСТАРТ». После выполнения «РЕСТАРТА» адрес ведомого отправляется снова, но уже с новым направлением данных, заданным битом R/\overline{W} . Условие «РЕСТАРТ» показано на рисунок 4.3.8.

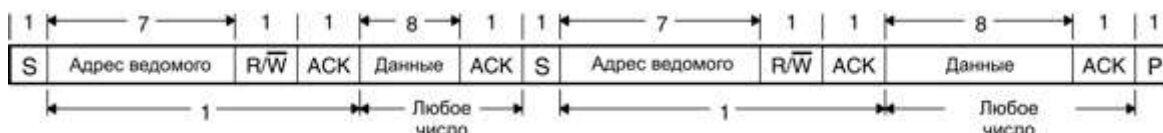


Рисунок 4.3.8 – Формат адресации модуля I2C с повторным условием «СТАРТ»

Режимы работы модуля I2C

Модуль I2C работает в режимах «ведущий передатчик», «ведущий приемник», «ведомый передатчик» или «ведомый приемник».

Режим ведущего

В режиме ведущего выполнение передачи и приема управляется с помощью битов I2CRM, I2CSTT и I2CSTP, как описано в таблице 4.3.1. Режим ведущего приемника вводится установкой I2CTRX=0 после передачи адресного байта ведомого и установленного бита R/\overline{W} . SCL удерживается в низком состоянии, когда необходимо вмешательство ЦПУ после передачи байта.

Таблица 4.3.1 – Функционирование ведущего

I2CRM	I2CSTP	I2CSTT	Состояние или активность шины
1	2	3	4
X	0	0	Модуль I ² C находится в режиме ведущего, но свободен. Условия «СТАРТ» и «СТОП» не генерируются.
0	0	1	Активность инициируется установкой I2CSTT. I2CNDAT используется для определения длины передачи. Условие «СТОП» автоматически не генерируется после перемещения байт, количество которых задано в I2CNDAT. Программное обеспечение должно установить I2CSTP для генерации условия «СТОП» в конце передачи. Это используется для условия «РЕСТАРТ».
0	1	1	I2CNDAT используется для установки длины передачи. Установка I2CSTT инициирует активность. Условие «СТОП» автоматически генерируется после передачи количества байт, заданного в I2CNDAT.
1	0	1	I2CNDAT не используется для установки длины передачи. Длинной передачи должно управлять программное обеспечение. Установка бита I2CSTT инициирует активность. Для инициирования условия «СТОП» или останова активности программное обеспечение должно установить бит I2CSTP. Этот режим используется, если необходимо передать более 255 байт.

Продолжение таблицы 4.3.1

1	2	3	4
0	1	0	Установка бита I2CSTP генерирует условие «СТОП» на шине после отправки количества байт, заданного в I2CNDAT или немедленно, если уже было передано то количество байт, которое заданное в I2CNDAT.
1	1	0	Установка бита I2CSTP генерирует условие «СТОП» на шине после завершения текущей передачи или немедленно, если текущая передача не выполняется.
1	1	1	Зарезервировано, шина неактивна.

Арбитраж

Если два или более передатчиков одновременно начинают передачу на шине, запускается процедура арбитража. На рисунке 4.3.9 поясняется процедура арбитража между двумя устройствами. Процедура арбитража использует данные, представленные на SDA конкурирующими передатчиками. Первый ведущий передатчик, генерирующий логическую единицу, отвергается противостоящим ведущим, генерирующим сигнал низкого уровня. Процедура арбитража дает приоритет устройству, которое передает поток последовательных данных с наименьшим двоичным значением. Ведущий передатчик, потерявший арбитраж, переключается в режим ведомого приемника и устанавливает флаг потери арбитража ALIFG. Если два или более устройства посылают одинаковые первые байты, арбитраж продолжается на последующих байтах.

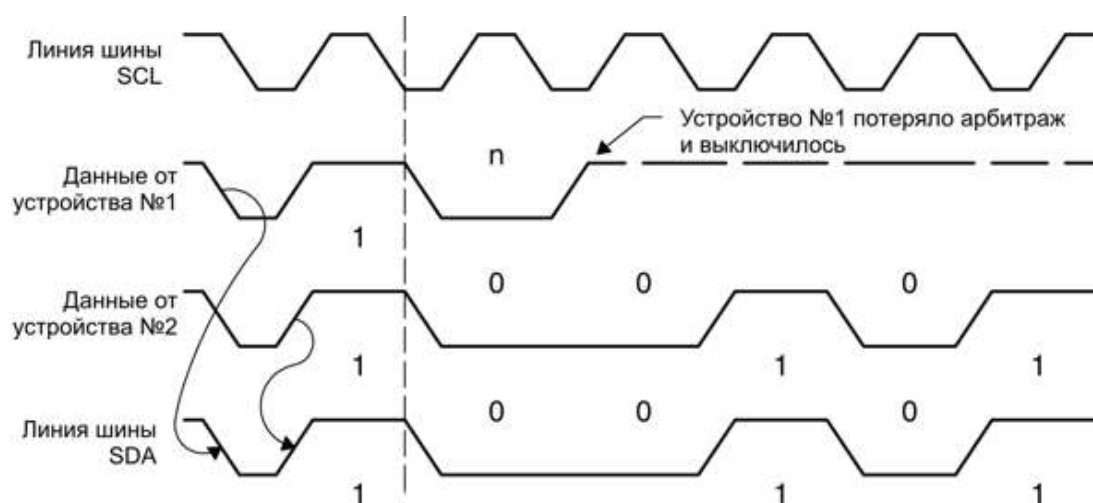


Рисунок 4.3.9 – Процедура арбитража между двумя ведущими передатчиками

Если выполняется процедура арбитража, когда на SDA повторяются условия «СТАРТ» или «СТОП», ведущие передатчики, вовлеченные в арбитраж, должны послать повторные условия «СТАРТ» или «СТОП» в том же самом месте в формате фрейма. Арбитраж не разрешается между:

- повторным условием «СТАРТ» и битом данных;
- условием «СТОП» и битом данных;
- повторным условием «СТАРТ» и условием «СТОП».

Режим ведомого

В режиме ведомого операции передачи и приема управляются автоматически модулем I2C.

В режиме ведомого приемника биты данных принимаются на SDA и сдвигаются по тактовым импульсам, генерируемым ведущим устройством. Ведомое устройство не генерирует тактовый сигнал, но может удерживать линию SCL в состоянии низкого уровня, если после приема байта необходимо вмешательство ЦПУ.

В режим ведомого передатчика можно войти только из режима ведомого приемника. Вход в режим ведомого передатчика происходит, если байт адреса ведомого, переданный ведущим, является таким же адресом, как и его собственный и был послан установленный бит R/\overline{W} , указывая на запрос отправки данных ведущему. Ведомый передатчик сдвигает последовательные данные из устройства на SDA по импульсам тактирования, генерируемым ведущим устройством. Ведомое устройство не генерирует тактовых сигналов, но может удерживать линию SCL в состоянии низкого уровня, если после передачи байта необходимо вмешательство ЦПУ.

Регистр данных I2CDR шины I2C

Регистр I2CDR может быть доступен как 8-разрядный или 16-разрядный регистр, что определяется битом I2CWORD. Функции регистра I2CDR описаны в таблице 4.3.2.

Опустошение при передаче

В режиме ведущего опустошение происходит, когда сдвиговый регистр передачи и буфер передачи пусты, а $I2CNDAT > 0$. В режиме ведомого опустошение происходит, когда сдвиговый регистр передачи и буфер передачи пусты, а внешний ведущий I2C все еще запрашивает данные. Когда происходит опустошение при передаче, устанавливается бит I2CTXUDF. Запись данных в регистр I2CDR или сброс бита I2CEN сбрасывает I2CTXUDF. I2CTXUDF используется только в режиме передачи.

Таблица 4.3.2 – Функции регистра I2CDR

I2CWORD	I2CTR_X	Функция I2CDR
0	1	Режим передачи байта: Используется только младший байт. Байт дважды буферизируется. Если новый байт записан до передачи предыдущего байта, новый байт ожидает во временном буфере до момента защелкивания в младшем байте регистра I2CDR. Когда I2CDR доступен, устанавливается бит TXRDYIFG.
0	0	Режим приема байта: Используется только младший байт. Байт дважды буферизируется. Если новый байт принят до прочтения предыдущего байта, новый байт ожидает во временном буфере до момента защелкивания в младшем байте регистра I2CDR. Когда I2CDR готов для чтения, устанавливается бит RXRDYIFG.
1	1	Режим передачи слова: Первым передается младший байт слова, затем старший байт. Регистр дважды буферизируется. Если новое слово записано до передачи предыдущего слова, новое слово ожидает во временном буфере до момента защелкивания в регистре I2CDR. Когда I2CDR доступен, устанавливается бит TXRDYIFG.
1	0	Режим приема слова: Первым принимается младший байт слова, затем старший байт. Регистр дважды буферизируется. Если новое слово принято до прочтения предыдущего слова, новое слово ожидает во временном буфере до момента защелкивания в регистре I2CDR. Когда I2CDR готов к доступу, устанавливается бит RXRDYIFG.

Переполнение при приеме

Переполнение при приеме происходит, когда сдвиговый регистр приема и буфер приема заполнены. Когда происходит переполнение при приеме, устанавливается бит I2CRXOVR. Потери данных не происходит, поскольку в этом случае линия SCL удерживается в состоянии низкого уровня, которое приостанавливает дальнейшую активность на шине. Чтение регистра I2CDR или сброс бита I2CEN сбрасывает бит I2CRXOVR. Бит I2CRXOVR используется только в режиме приема.

Генерация тактовых сигналов I2C и синхронизация

Модуль I2C работает с источником тактовой частоты, выбираемым

битами I2CSSELx. Прескалер I2CPSC и регистры I2CSCLH и I2CSCLL определяют частоту и скважность тактового сигнала SCL для режима ведомого, как показано на рисунке 4.3.10. Источник тактовых импульсов для модуля I2C должен иметь частоту, по крайней мере в 10 раз больше частоты SCL в обоих режимах ведущего и ведомого.

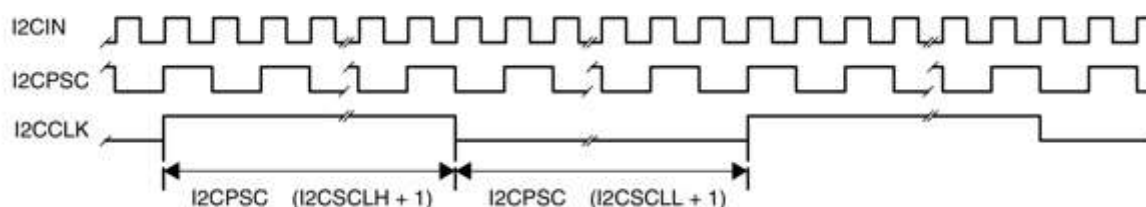


Рисунок 4.3.10 – Генерация сигналов на линии SCL модуля I2C

Конфигурирование USART для функционирования I2C

Контроллер I2C является частью периферии USART. Для работы в режимах SPI или I2C необходимо установить бит SYNC. Установка бита SYNC при SWRST=1 выбирает режим SPI. Установка бита I2C, когда SYNC=1 приводит к выбору режима I2C. Биты SYNC и I2C могут быть установлены вместе в одной команде для выбора режима I2C в модуле USART0.

После инициализации модуль I2C готов для выполнения операций передачи и приема. Очистка I2CEN прекращает работу модуля.

Прерывания I2C

Модуль I2C имеет один вектор прерывания для восьми флагов прерывания. Каждый флаг прерывания имеет собственный бит разрешения прерывания. Когда прерывание разрешено и установлен бит GIE, флаг прерывания будет генерировать запрос прерывания. События, перечисленные в таблице 4.3.3, вызывают I2C прерывание.

Генератор вектора прерывания I2CIV

Флаги прерывания I2C разделены по приоритетам и объединены в источник одного вектора прерывания. Регистр вектора прерывания I2CIV используется для выяснения, какой флаг запросил прерывание. Разрешенное прерывание с наивысшим приоритетом генерирует число в регистре I2CIV. Это число может быть оценено или добавлено к программному счетчику для автоматического входа в соответствующую программную процедуру. Запрещенные I2C прерывания не воздействуют на содержимое I2CIV.

Таблица 4.3.3 – Прерывания модуля I2C

Флаг	Условие прерывания
ALIFG	Потеря арбитража. Арбитраж может быть потерян, когда два или более передатчиков начинают передачу одновременно или когда программное обеспечение пытается инициировать I ² C перенос при I2CBB=1. Флаг ALIFG устанавливается, когда арбитраж был потерян. Когда ALIFG устанавливается, биты MST и I2CSTP очищаются и контроллер I ² C становится ведомым приемником.
NACKIFG	Прерывание при отсутствии подтверждения. Этот флаг устанавливается, когда подтверждение ожидается, но не получено.
OAIFG	Прерывание собственного адреса. Этот флаг устанавливается, когда другой ведущий имеет адрес модуля I ² C. OAIFG используется только в режиме ведомого.
ARDYIFG	Прерывание «регистр доступен для чтения». Этот флаг устанавливается, когда ранее запрограммированный перенос завершен, и биты статуса обновлены. Это прерывание используется для уведомления ЦПУ о том, что регистры I ² C готовы к доступу.
RXRDYIFG	Прерывание/статус готовности приема. Этот флаг устанавливается, когда модуль I ² C принял новые данные. RXRDYIFG автоматически очищается, когда I2CDR прочитан и буфер приема пуст. Переполнение приемника показывается, если бит I2CRXOVR=1. RXRDYIFG используется только в режиме приема.
TXRDYIFG	Прерывание/статус готовности передачи. Этот флаг устанавливается, когда модуль I ² C готов для новой передачи данных (режим ведущего передатчика) или когда другой ведущий запрашивает данные (режим ведомого передатчика). TXRDYIFG автоматически очищается, когда I2CDR и буфер передачи полны. Опустошение передачи показывается, если I2CTXUDF=1. Не используется в режиме приема.
GCIFG	Прерывание общего вызова. Этот флаг устанавливается, когда модуль I ² C принял адрес общего вызова (00h). GCIFG используется только в режим приема.
STTIFG	Прерывание при обнаружении условия «СТАРТ». Этот флаг устанавливается, когда модуль I ² C обнаружил условие «СТАРТ» в режиме ведомого. Это позволяет MSP430 находиться в режиме пониженного энергопотребления с неактивным источником тактирования I ² C, пока ведущий не иницирует связь по I ² C. STTIFG используется только в режиме ведомого.

При любом доступе (чтение или запись) к регистру I2CIV автоматически сбрасывается флаг ожидающего прерывания с наивысшим приоритетом. Если устанавливается другой флаг прерывания, после обработки начального прерывания немедленно генерируется другое прерывание.

Форматы управляющих регистров Модуля I2C в [1] стр. 285-295.

4.3.2 Описание лабораторной установки

Задания выполняются на лабораторном макете на базе 16-ми разрядного микроконтроллера MSP430.

Подключение клавиатуры

Для обеспечения возможности ручного управления микроконтроллерным стендом в его состав включена клавиатура 3x4 (AK-304-N-BBW), которая подключена через регистр PCA9538 к шине I²C.

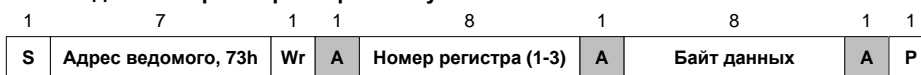
PCA9538 – это 8-разрядный регистр ввода/вывода для последовательной двухпроводной шины данных I²C, с рабочим диапазоном напряжением питания от 2,3В до 5,5В. Он обеспечивает общие функции ввода/вывода для большинства типов микроконтроллеров через интерфейс I²C (сигнал тактирования SCL и сигнал данных SDA).

Адрес регистра на шине I²C определяется пятью фиксированными старшими разрядами – 11100, и двумя адресными линиями A0 и A1, которые для управляющего клавиатурой регистра подключены к шине питания, поэтому адрес регистра на шине I²C – 73h (1110011).

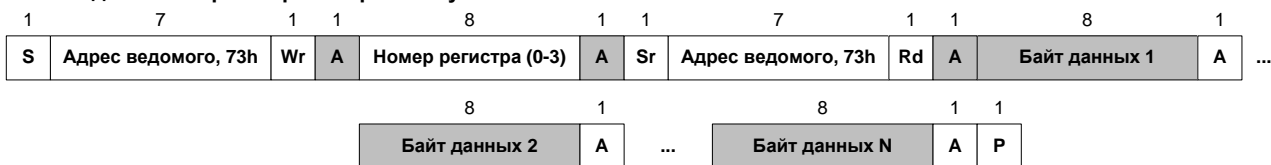
Формат обмена данными между ЦП и последовательным регистром по протоколу I²C изображен на рисунке 4.3.11. Условия «СТАРТ» и «СТОП» генерируются ведущим устройством (ЦП). Под условием «СТАРТ» понимается переход с высокого уровня линии SDA на низкий при высоком уровне на SCL. Первый после условия «СТАРТ» байт состоит из 7 разрядного адреса устройства (73h) и бита R/\overline{W} . Когда $R/\overline{W}=0$, ЦП передает данные регистру, а когда $R/\overline{W}=1$, то ЦП принимает данные от регистра. Бит ACK посылается приемником после каждого байта на 9-ом такте SCL. Два младших разряда второго байта (B1, B0) указывают номер внутреннего регистра датчика, к которому идет обращение. Во третьем байте передаются непосредственно данные. В конце обмена данными ЦП генерирует условие «СТОП» – переход с низкого уровня на линии SDA на высокий при высоком уровне на SCL.

В таблице 4.3.5 отображен список внутренних управляющих регистров микросхемы PCA9538.

Запись данных в регистр по протоколу I²C:



Чтение данных из регистра по протоколу I²C:



Условные обозначения:

S	Условие «СТАРТ»	Sr	Повторное условие «СТАРТ»
Wr	Флаг передачи данных от ЦП (разряд = 0)	Rd	Флаг передачи данных от регистра (разряд = 1)
A	Подтверждение приема		Направление передачи – от ЦП к регистру
P	Условие «СТОП»		Направление передачи – от регистра к ЦП

Рисунок 4.3.11 – Формат обмена данными между ЦП и регистром I²C

Таблица 4.3.5 – Управляющие регистры микросхемы PCA9538

Разряд		Байт команды	Регистр	Протокол	Значение по умолчанию
B1	B0				
0	0	0x00	Input port	Чтение	XXXX XXXX
0	1	0x01	Output port	Чтение/запись	1111 1111
1	0	0x02	Polarity Inversion	Чтение/запись	0000 0000
1	1	0x03	Configuration	Чтение/запись	1111 1111

Регистр ввода (Input port, 0x00) отражает поступающие логические уровни на ножках ввода-вывода P0-P7 последовательного регистра, независимо от того, на вход или на выход сконфигурированы эти ножки. Он доступен только для чтения. Перед началом чтения, необходимо послать байт команды 0x00 устройству по шине I²C, чтобы указать, что далее будет обращение к регистру ввода.

Регистр вывода (Output port, 0x01) задает выходные уровни напряжения на выводах, направление которых определено на вывод регистром конфигурации. Выводы, которые настроены на ввод, не изменяют своего состояния.

Регистр инверсии полярности (Polarity Inversion, 0x02) инвертирует значения сигналов на выводах, настроенных на ввод. Если разряд в этом регистре установлен в 1, соответствующий входной сигнал проинвертирован, если сброшен – сигнал не изменит свою полярность.

Регистр конфигурации (Configuration, 0x03) задает направление выводов P0-P7. Если разряд в этом регистре установлен в 1, то направление соответствующего вывода настраивается на ввод, а если сброшен в 0 – на вывод.

Разряды P4...P6 регистра необходимо настроить на режим вывода, так как они будут стробировать столбцы клавиатуры, а оставшиеся разряды P0...P3 – на ввод, они будут реагировать на нажатие клавиши в столбце, выбранном разрядами P4...P7. (рисунок 4.3.12)

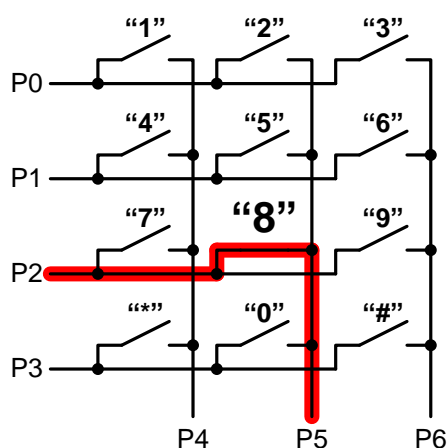


Рисунок 4.3.12 – Пример опроса клавиатуры

Программный опрос клавиатуры выполняется следующим образом: в цикле последовательно на один из разрядов P4...P6 подается сигнал низкого уровня, затем опрашиваются разряды P0...P3 на предмет появления сигнала низкого уровня, что будет означать нажатие кнопки в выбранном столбце и строке. На рисунке 4.3.13 изображен пример опроса в момент нажатия кнопки, подан низкий уровень на разряд P5, и при опросе разрядов P0...P3, в разряде P2 обнаружен низкий уровень, что означает нажатие кнопки во втором столбце третьей строки, что соответствует клавише «8».

Принципиальная схема подключения клавиатуры через регистр PCA9538 показана на рисунке 4.3.14, размещение клавиатуры на плате стенда показано на рисунке 4.3.15.

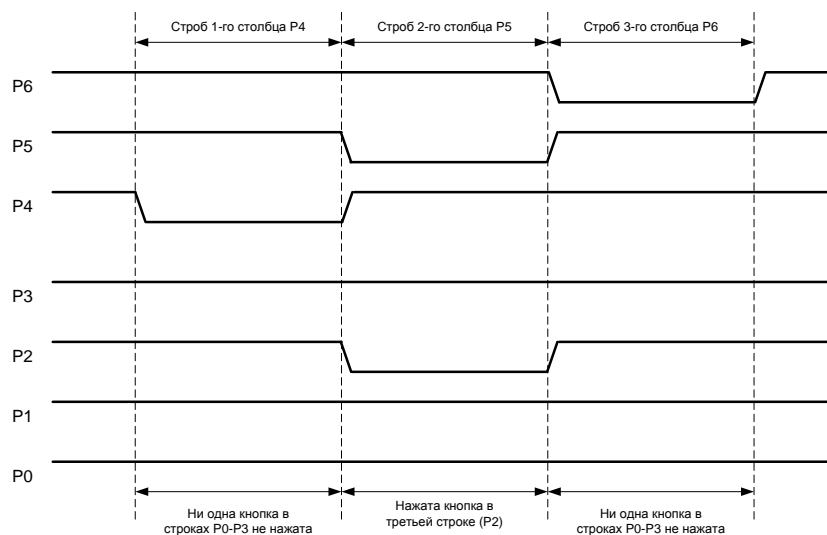


Рисунок 4.3.13 – Временные диаграммы опроса клавиатуры

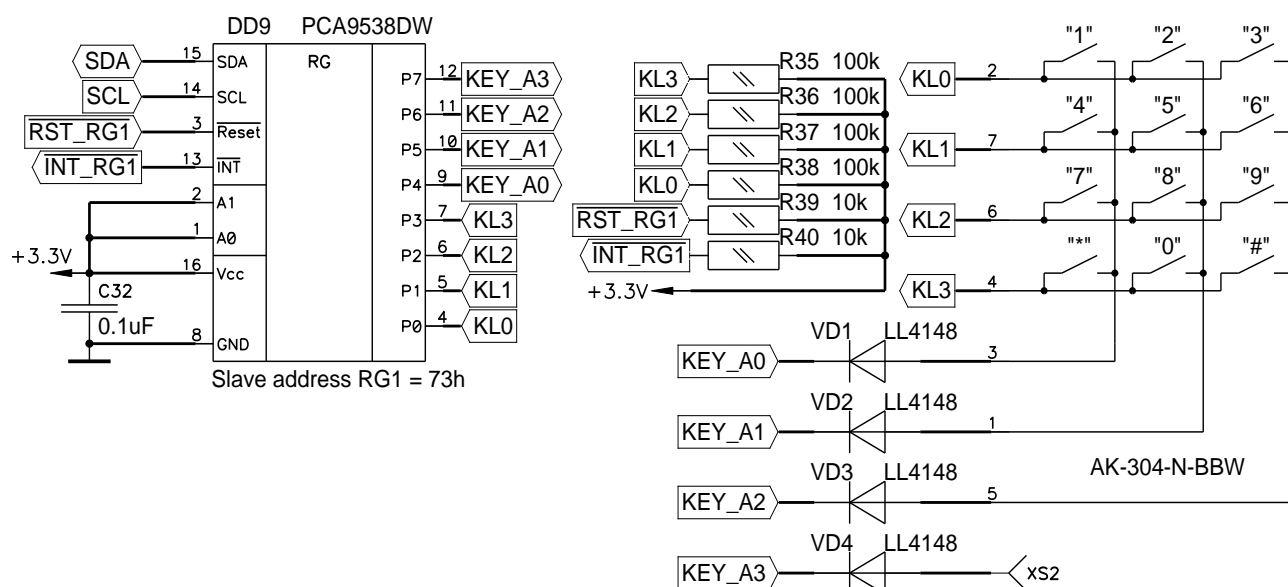


Рисунок 4.3.14 – Принципиальная схема подключения клавиатуры

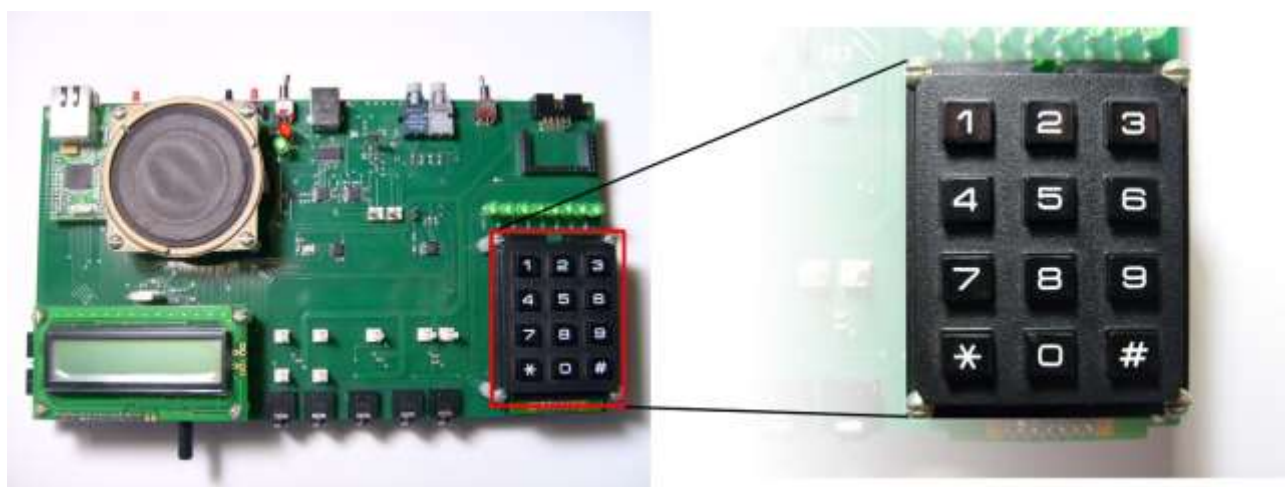


Рисунок 4.3.15 – Размещение клавиатуры на плате

Описание сигналов регистра PCA9538 (DD9) и клавиатуры (AK-304-N-BBW) описаны в таблице 4.3.6.

Таблица 4.3.6 – Описание сигналов и назначение выводов регистра управления клавиатурой PCA9538 (DD9)

Вывод DD9	Название	Подключение к цепи / [номер вывода]	Примечание
1	A0	+3,3V	Задание адреса устройства: адрес регистра – 73h
2	A1	+3,3V	
3	Reset	RST_RG1 / 28	Сигнал сброса регистра
4	P0	AK-304-N-BBW / 2	Строка KL0
5	P1	AK-304-N-BBW / 7	Строка KL1
6	P2	AK-304-N-BBW / 6	Строка KL2
7	P3	AK-304-N-BBW / 4	Строка KL3
8	GND	GND	Подключение к шине 0 В
9	P4	AK-304-N-BBW / 3	Столбец KEY_A0
10	P5	AK-304-N-BBW / 1	Столбец KEY_A1
11	P6	AK-304-N-BBW / 5	Столбец KEY_A2
12	P7	XS2	Не используется
13	INT	INT_RG1 / 19	Сигнал запроса прерывания
14	SCL	SCL / 31	Сигнал тактирования
15	SDA	SDA / 29	Линия данных
16	Vcc	+3,3V	Напряжение питания

Подключение светодиодного индикатора

Индикатор из 8 светодиодов подключен через последовательный регистр PCA9538 к шине I²C. Адресные линии регистра PCA9538, управляющего светодиодами, A0 и A1 подключены к общей шине, поэтому адрес регистра при обращении к нему по шине I²C – 70h.

При записи в последовательный регистр 8-разрядного числа, загораются или гаснут светодиоды, соответствующие номеру разряда записанного числа (если в разряде 0 – светодиод гаснет, если 1 – светится).

Принципиальная схема подключения светодиодного индикатора изображена на рисунке 4.3.16, а размещение на плате – рисунок 4.3.17.

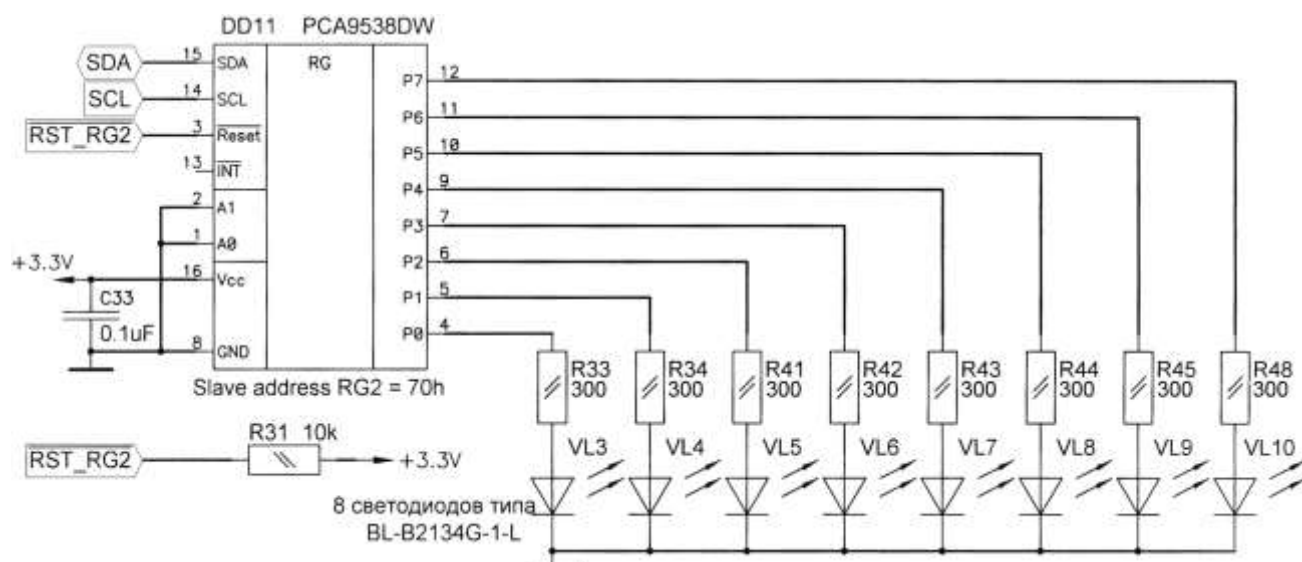


Рисунок 4.3.16 – Принципиальная схема подключения светодиодного индикатора

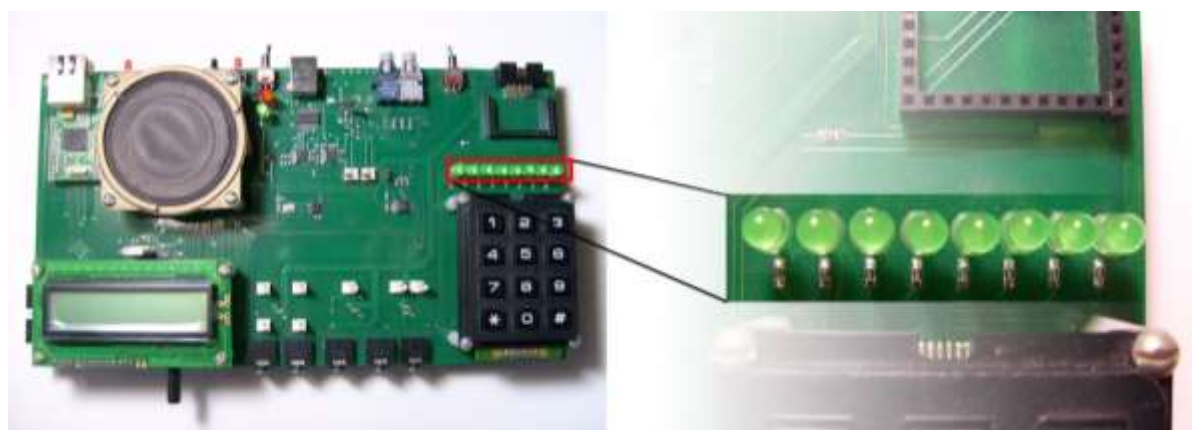


Рисунок 4.3.17 – Размещение светодиодного индикатора на плате

Описание сигналов регистра PCA9538 (DD11), через который подключены светодиоды, представлены в таблице 4.3.7.

Таблица 4.3.7 – Описание сигналов и назначение выводов регистра управления светодиодами PCA9538 (DD11)

Вывод DD11	Название	Подключение к цепи / [номер вывода]	Примечание
1	2	3	4
1	A0	GND	Задание адреса устройства: адрес регистра на шине I ² C – 70h
2	A1	GND	
3	Reset	RST_RG2 / 30	Сигнал сброса регистра

Продолжение таблицы 4.3.7

1	2	3	4
4	P0	VL3	Светодиоды
5	P1	VL4	
6	P2	VL5	
7	P3	VL6	
8	GND	GND	Подключение к шине 0 В
9	P4	VL7	Светодиоды
10	P5	VL8	
11	P6	VL9	
12	P7	VL10	
13	$\overline{\text{INT}}$	–	Не используется
14	SCL	SCL / 31	Сигнал тактирования
15	SDA	SDA / 29	Линия данных
16	Vcc	+3,3V	Напряжение питания

4.3.3 Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс-контроль знаний по принципам функционирования модулей USART в режиме I2C, входящих в состав микроконтроллера MSP430, а также по протоколу обмена данными по интерфейсу I2C. При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствии с индивидуальным заданием (таблица 3.8).

Задание. Разработать в среде программирования IAR Embedded Workbench программу на языке C, которая выполняет опрос клавиатуры лабораторного стенда и выводит информацию о нажатых клавишах с помощью блока светодиодов. Варианты индивидуальных заданий представлены в таблице 4.3.8.

Для решения задачи можно использовать следующие функции:

```
//-----
// инициализация системы тактирования
void Init_System_Clock()
{
    BCSCCTL1 &= ~XT2OFF;    // включение осциллятора XT2
    BCSCCTL2 |= SELM1+SELS;  // установка внешнего модуля тактирования
                             // MCLK = XT2, SMCLK = XT2
    do                       // ожидание запуска кварца
    {
        IFG1 &= ~OFIFG;      // Clear OSCFault flag
        for (char i = 0xFF; i > 0; i--); // Time for flag to set
    }
}
```

```

    while ((IFG1 & OFIFG));
}
//-----
// инициализация модуля I2C
void Init_I2C()
{
    P3SEL |= 0x0A;          // Выбор альтернативной функции для линий порта P3
                           // в режиме I2C SDA->P3.1, SCL->P3.3
    U0CTL |= I2C + SYNC;    // Выбрать режим I2C для USART0
    U0CTL &= ~I2CEN;        // Выключить модуль I2C
    // Конфигурация модуля I2C
    I2CTCTL=I2CSSEL_2;     // SMCLK
    I2CSCLH = 0x26;        // High period of SCL
    I2CSCLL = 0x26;        // Low period of SCL
    U0CTL |= I2CEN;        // Включить модуль I2C
    // формирование строба сброса I2C-регистров PCA9538 - RST_RG1->P3.1 и
RST_RG2->P3.2
    P3DIR |= 0x05;          // переключаем эти ножки порта на вывод,
    P3SEL &= ~0x05;         // выбираем функцию ввода-вывода для них
    P3OUT &= ~0x05;         // и формируем строб сброса на 1 мс
    wait_1ms(1);
    P3OUT |= 0x05;
}
//-----
// отправка данных по протоколу I2C
void Send_I2C(unsigned char* buffer,unsigned int num, unsigned char
address)
{
    while (I2CBUSY & I2CDCTL);          // проверка готовности модуля I2C
    BufTptr=buffer;
    I2CSA = address;                    // установка адреса приемника
    I2CNDAT =num;                       // количество передаваемых байт
    I2CIE = TXRDYIE+ALIE;               // разрешение прерываний по
окончанию передачи байта и по потере арбитража
    U0CTL |= MST;                       // режим Master
    I2CTCTL |= I2CSTT + I2CSTP + I2CTRХ; // инициализировать передачу
    while ((I2CTCTL & I2CSTP) == 0x02); // ожидание условия СТОП
}
//-----
// прием данных по протоколу I2C
void Receive_I2C(unsigned char* buffer,unsigned int num, unsigned char
address)
{
    while (I2CBUSY & I2CDCTL);          // проверка готовности модуля I2C
    BufRptr=buffer;
    I2CSA=address;
    I2CTCTL&=~I2CTRХ;                   // режим приема
    I2CNDAT=num;
    I2CIE=RXRDYIE;                      // разрешение прерывания по
окончанию приема байта
    U0CTL |= MST;
    I2CTCTL |= I2CSTT + I2CSTP;          // инициализировать прием

```

```

    while ((I2CTCTL & I2CSTP) == 0x02);    // ожидание условия СТОП
}
//-----
// обработчик прерываний модуля I2C
void USART0TX_func()
{
    switch(I2CIV)
    {
        case 0: break;                // нет прерывания
        case 2: break;                // потеря арбитража
        case 4: break;                // нет подтверждения
        case 6: break;                // прерывание собственного адреса
        case 8: break;                // регистр доступен для чтения
        case 10:                       // окончание приема байта
            *BufRptr++=I2CDRB;
            break;
        case 12:                       // окончание передачи байта
            I2CDRB=*BufTptr++;
            break;
        case 14: break;                // общий вызов
        case 16: break;                // обнаружено условие СТАРТ
        default : break;
    }
}
//-----
// вектор прерываний для модуля I2C
#pragma vector=USART0TX_VECTOR
__interrupt void I2C_ISR()
{
    USART0TX_func();
}
//-----

```

Порядок выполнения задания:

- включить лабораторный макет.
- запустить компилятор IAR Embedded Workbench.
- создать пустой проект.
- создать файл ресурса для кода программы и подключить его к проекту.
- ввести код исходного модуля программы обмена данными между микроконтроллером MSP430F1611 с регистрами PCA9538 по интерфейсу I2C соответствие с индивидуальным заданием, приведенным в таблице 4.3.8.
- выполнить компиляцию исходного модуля программы и устранить ошибки, полученные на данном этапе.
- настроить параметры программатора.
- создать загрузочный модуль программы и выполнить программирование микроконтроллера.

Проверить работоспособность загруженной в микроконтроллер

программы и показать результаты работы преподавателю.

В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы.

Таблица 4.3.8 – Варианты индивидуальных заданий

№ п.п.	Задание
1	2
1	Разработать программу, фиксирующую нажатия клавиш 1, 2 и * матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши #. Частота тактовых импульсов на линии SCL – 10 кГц.
2	Разработать программу, фиксирующую нажатия клавиш 5, 7 и 9 матричной клавиатуры включением светодиодов 5, 6 и 7 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши *. Частота тактовых импульсов на линии SCL – 20 кГц.
3	Разработать программу, фиксирующую нажатия клавиш 4, 7 и 8 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при одновременном нажатии клавиш * и #. Частота тактовых импульсов на линии SCL – 30 кГц.
4	Разработать программу, фиксирующую нажатия клавиш 1, 2 и 3 матричной клавиатуры включением светодиодов 4, 5 и 6 соответственно. Выход из цикла опроса осуществляется при одновременном нажатии клавиш * и #. Частота тактовых импульсов на линии SCL – 40 кГц.
5	Разработать программу, фиксирующую нажатия клавиш 2, 5 и 9 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при одновременном нажатии клавиш * и #. Частота тактовых импульсов на линии SCL – 50 кГц.
6	Разработать программу, фиксирующую нажатия клавиш 4, 8 и 9 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши *. Частота тактовых импульсов на линии SCL – 60 кГц.
7	Разработать программу, фиксирующую нажатия клавиш 3, 7 и 0 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши #. Частота тактовых импульсов на линии SCL – 70 кГц.
8	Разработать программу, фиксирующую нажатия клавиш 5, 6 и 7 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при одновременном нажатии клавиш * и #. Частота тактовых импульсов на линии SCL – 80 кГц.

Продолжение таблицы 4.3.8

1	2
9	Разработать программу, фиксирующую нажатия клавиш 1, 6 и 8 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши *. Частота тактовых импульсов на линии SCL – 90 кГц.
10	Разработать программу, фиксирующую нажатия клавиш 9, 0 и 1 матричной клавиатуры включением светодиодов 1, 2 и 3 соответственно. Выход из цикла опроса осуществляется при нажатии клавиши #. Частота тактовых импульсов на линии SCL – 100 кГц.

4.3.4 Содержание отчета

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный модуль разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности использования встроенных в микроконтроллер модуля USART0 при реализации обмена данными по последовательному протоколу I2C.

4.3.5 Контрольные вопросы и задания

1. Приведите алгоритм инициализации модуля USART микроконтроллера MSP430F1611 для работы в режиме I2C.
2. Поясните принципы обмена данными по интерфейсу I2C.
3. В каком регистре хранится адрес ведомого?
4. Каким образом разрешается конфликтная ситуация, возникающая при одновременной передаче данных по шине I2C от нескольких ведущих?
5. Какие регистры используются для настройки параметров передачи данных с помощью встроенного в микроконтроллер MSP430F1611 блока USART, работающего в режиме I2C?
6. Какие сигналы прерываний могут генерироваться блоком USART в режиме I2C?
7. Поясните формат кадра при обмене данными по интерфейсу I2C.

4.4 ИЗУЧЕНИЕ ПРИНЦИПОВ ОБРАБОТКИ ПЕРЕРЫВАНИЙ НА ПРИМЕРЕ УПРАВЛЕНИЯ ВСТРОЕННЫМИ В МИКРОКОНТРОЛЛЕР ТАЙМЕРАМИ-СЧЕТЧИКАМИ И КОМПАРАТОРОМ

Цель работы: изучить принципы разработки процедур обработки прерываний в микроконтроллере MSP430F1xxx, ознакомиться с принципами функционирования встроенных в микроконтроллер 16 – разрядных таймеров-счетчиков и компаратора для измерения сопротивления резистивного датчика.

4.4.1 Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе [1 – 4] и конспект лекций, изучить принципы разработки процедур обработки прерываний в микроконтроллере MSP430F1xxx, ознакомиться с возможностями функционирования встроенных в микроконтроллер 16 - разрядных таймеров-счетчиков и компаратора, изучить алгоритмы формирования временных задержек с помощью прерываний от таймеров-счетчиков.

4.4.1.1 Система прерываний в микроконтроллере MSP430F1xxx.

Приоритеты прерываний показаны на рисунке 4.4.1. Приоритеты определяются порядком расположения модулей в соединяющей их цепи. Чем ближе модуль к ЦПУ/NMIRS, тем выше его приоритет.

Прерывания делятся на три типа:

- системное (системный сброс);
- немаскируемое (NMI);
- маскируемое.

Немаскируемые прерывания.

Немаскируемые прерывания NMI не маскируются общим битом разрешения прерываний (GIE), но могут управляться индивидуальными битами включения прерывания (ACCVIE, NMIE, OFIE). Когда происходит немаскируемое прерывание NMI, все биты разрешения NMI-прерываний автоматически сбрасываются. Выполнение программы продолжается с адреса, содержащегося в векторе немаскируемого прерывания (0FFFCh). Программное обеспечение пользователя должно установить необходимые биты NMI-прерывания, чтобы оно было разрешено вновь. Блок-схема источников NMI-

прерываний показана на рисунке 4.4.2.

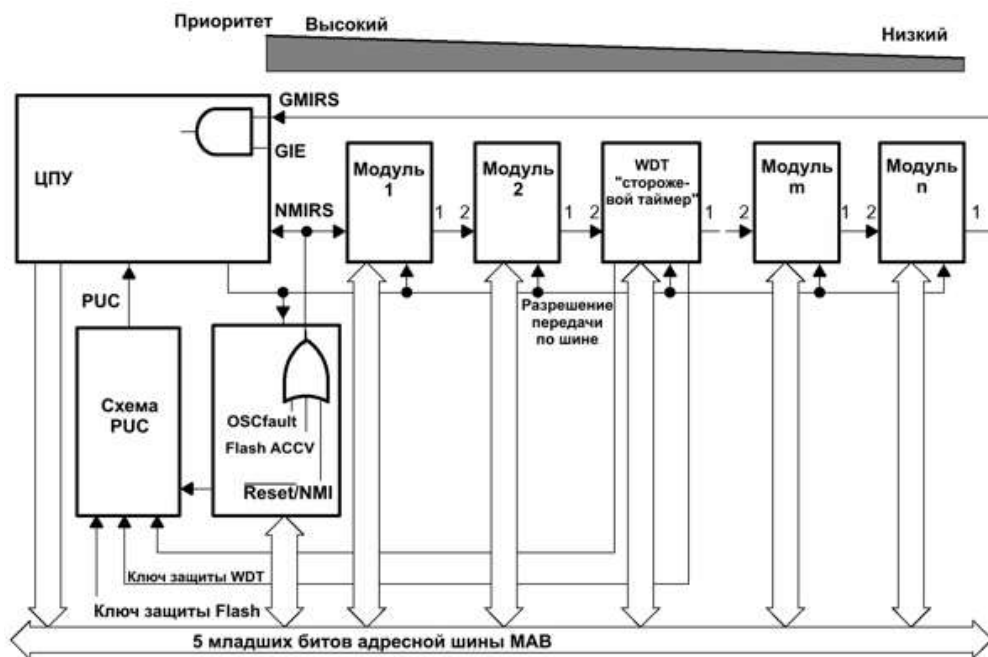


Рисунок 4.4.1 – Приоритеты прерываний

Немаскируемое прерывание NMI может быть вызвано тремя событиями:

- появление фронта сигнала на выводе RST/NMI;
- появление неисправности осциллятора;
- нарушение доступа к флэш-памяти.

При включении микроконтроллера вывод RST/NMI конфигурируется как вывод сброса. Его функциональное назначение определяется в регистре управления сторожевым таймером WDTCTL. Если вывод RST/NMI запрограммирован на функцию сброса, ЦПУ будет находиться в состоянии сброса до тех пор, пока на этом выводе присутствует сигнал низкого уровня. После смены уровня на этом входе на лог.«1», ЦПУ начинает выполнять программу с команды, адрес которой хранится в векторе сброса (0FFFFeh).

Если вывод RST/NMI сконфигурирован программой пользователя как вход вызова немаскируемого прерывания, фронт сигнала, выбранного битом NMIES, вызовет NMI-прерывание, если установлен бит NMIE. Также будет установлен флаг NMIFG.

Неисправность осциллятора

Сигнал неисправности осциллятора позволяет предотвратить ошибки, связанные с неправильным функционированием осциллятора. Установкой бита OFIE можно разрешить генерацию NMI-прерывания при неисправности

осциллятора. С помощью флага OFIFG процедура обработки NMI-прерывания может проверить, было ли NMI-прерывание вызвано неисправностью осциллятора.

Сигнал неисправности осциллятора может быть вызван PUC-сигналом, поскольку он переводит осциллятор LFXT1 из режима HF в режим LF. Сигнал PUC также отключает осциллятор XT2.

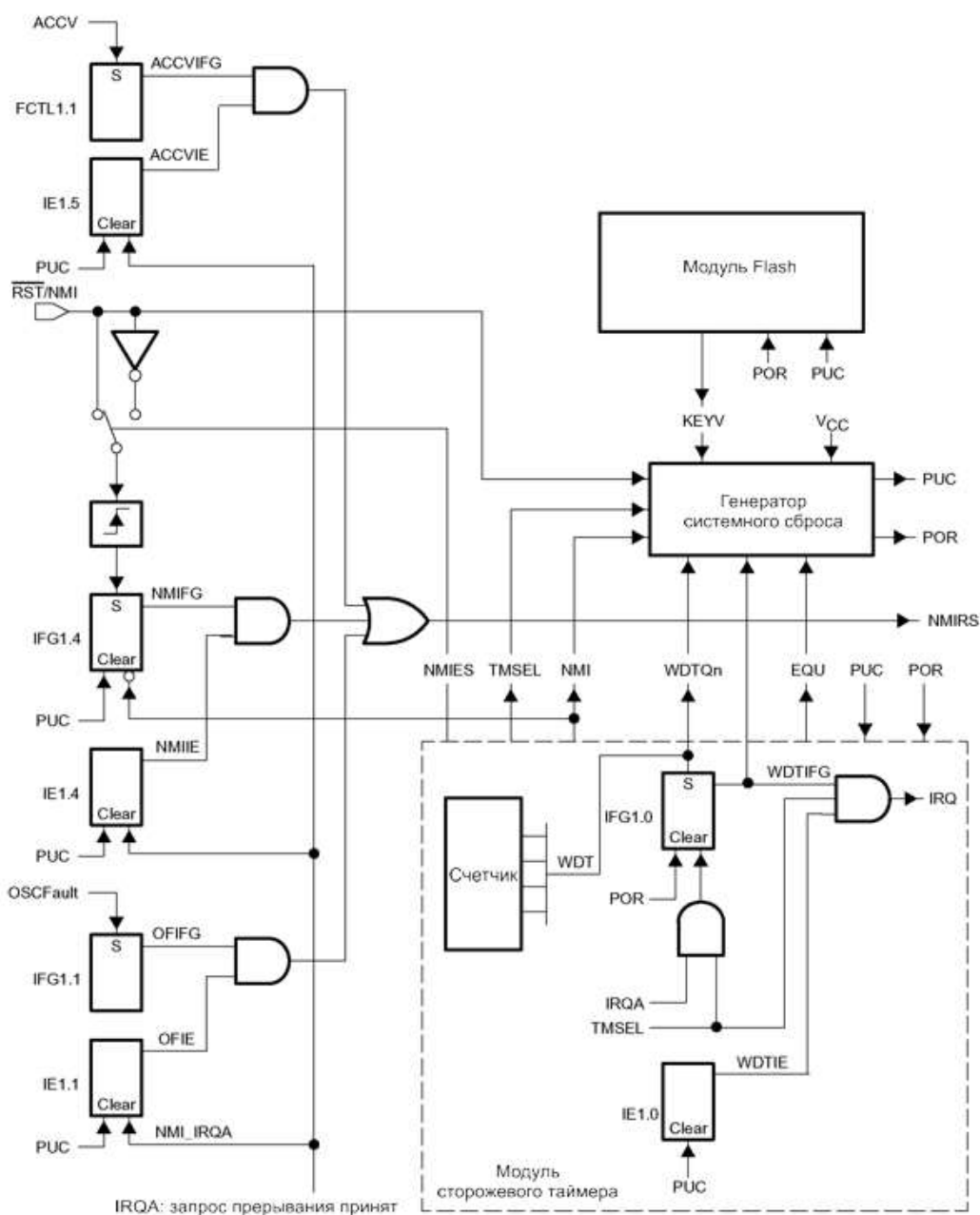


Рисунок 4.4.2 – Источники немаскируемого прерывания

Нарушение доступа к Flash-памяти

Флаг ACCVIFG устанавливается, когда происходит нарушение доступа к

Flash-памяти. Генерация NMI-прерывания при нарушении доступа к Flash-памяти происходит при установленном бите ACCVIE. Проверкой флага ACCVIFG в процедуре обработки NMI-прерывания можно определить, было ли вызвано прерывание нарушением доступа к Flash-памяти.

Маскируемые прерывания

Маскируемые прерывания вызываются периферийными устройствами, имеющими возможность вызова прерываний, включая ситуацию переполнения сторожевого таймера в активном режиме. С помощью индивидуальных битов разрешения прерывания можно отключать источники прерываний как по отдельности, так и все сразу с использованием общего бита разрешения всех прерываний (GIE) в регистре статуса (SR).

Обработка прерывания

Если периферийное устройство запросило прерывание и включены биты общего разрешения прерываний GIE и индивидуальный бит разрешения прерывания от этого устройства, будет вызвана процедура обработки прерывания. Для вызова немаскируемого (NMI) прерывания достаточно установки только индивидуального бита разрешения прерывания.

Получение прерывания

Время задержки вызова прерывания составляет 6 машинных циклов, с момента приема запроса на прерывание и до начала выполнения первой команды процедуры обработки прерывания. Логика обработки запроса прерывания имеет следующую последовательность:

- любая текущая команда выполняется до конца;
- содержимое программного счетчика PC, указывающего на следующую команду, помещается в стек;
- содержимое регистра статуса SR помещается в стек;
- если поступило несколько прерываний во время выполнения последней команды, обрабатывается прерывание с наивысшим приоритетом, остальные ожидают обслуживания;
- автоматически сбрасывается флаг одного источника прерывания. Флаги запроса остальных прерываний остаются установленными в ожидании обслуживания программным обеспечением.

Регистр SR очищается, за исключением бита SCG0, остающегося неизменным. Процессор переходит из режима пониженного потребления в активный режим;

Содержимое вектора прерывания загружается в PC и начинается выполнение процедуры обработки прерывания с загруженного адреса.

Возврат из прерывания

Подпрограмма обработки прерывания заканчивается командой RETI (возврат из подпрограммы обработки прерывания)

Для возврата из прерывания необходимо 5 машинных циклов.

Восстанавливается из стека содержимое регистра SR. Становятся актуальными все предыдущие установки GIE, CPUOFF и пр., в не зависимости от установок, использовавшихся в процедуре обработке прерывания.

Восстанавливается из стека содержимое программного счетчика PC и начинается выполнение программы с того места, где она была прервана.

Векторы прерываний

Векторы прерываний и стартовые адреса расположены в адресном диапазоне с 0FFFFh по 0FFE0h. Вектор программируется пользователем с помощью указания 16-разрядного стартового адреса соответствующей процедуры обработки прерывания.

4.4.1.2 Принципы функционирования аппаратных таймеров-счетчиков, входящих в состав микроконтроллера MSP430F1xxx

В состав микроконтроллеров MSP430 входят 16-разрядные таймеры А и В. Принципы функционирования таймеров рассмотрим на примере таймера А. Таймер А – это 16-разрядный таймер/счетчик с тремя регистрами захвата/сравнения (рисунок 4.2.3). Таймер А может обеспечить множество захватов/сравнений, выходов ШИМ и выдержку временных интервалов. Таймер А также имеет обширные возможности прерывания. Прерывания могут быть сгенерированы от счетчика при переполнении и от каждого из регистров захвата/сравнения.

16-разрядный таймер-счетчик

Регистр 16-разрядного таймера/счетчика TAR, инкрементируется или декрементируется (в зависимости от режима работы) с каждым нарастающим фронтом тактового сигнала. TAR может быть программно прочитан и записан. Кроме того, таймер может генерировать прерывание при переполнении. TAR можно очистить установкой бита TACLR. Установка TACLR также очищает делитель тактовой частоты и направление счета для режима вверх/вниз.

Выбор источника тактирования и делитель

Тактирование таймера TACLK может производиться от ACLK, SMCLK или от внешнего источника через TACLK или INCLK. Источник тактовых импульсов выбирается с помощью битов TASSELx. Выбранный источник

тактирования может быть подключен к таймеру напрямую или через делитель на 2, 4 или 8 с помощью битов IDx.

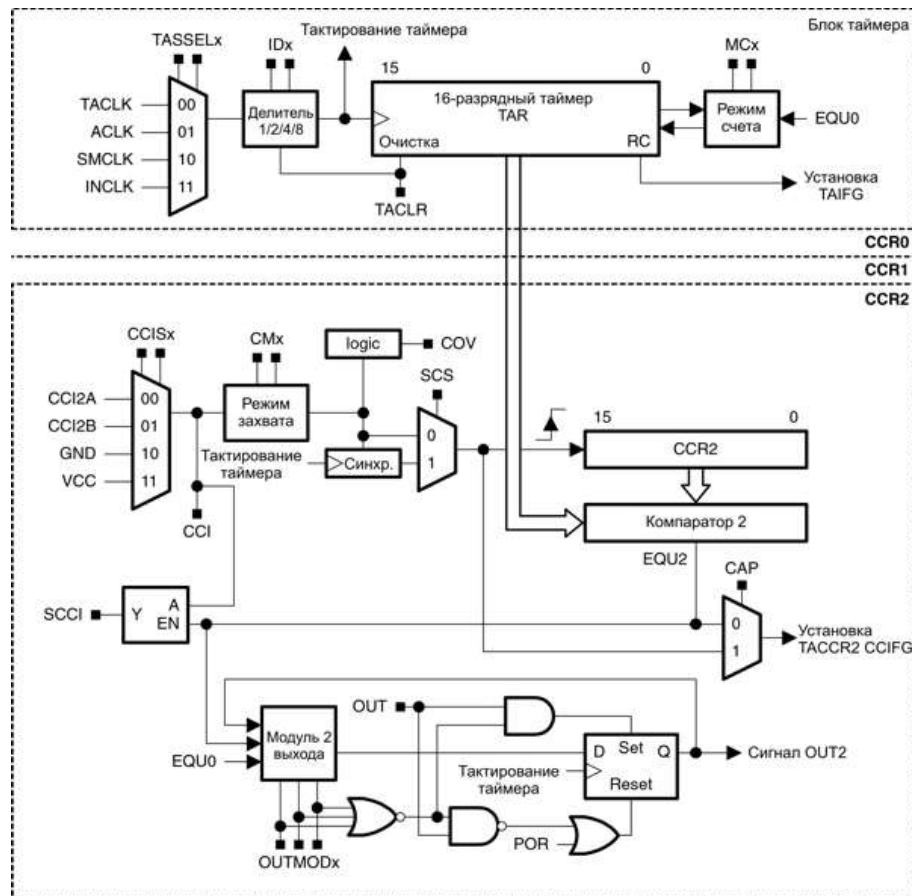


Рисунок 4.2.3 – Структура таймера А

Запуск таймера

Таймер может быть запущен или перезапущен следующими способами:

Таймер считает, когда $MCx > 0$ и источник тактовых импульсов активен ;

Когда таймер в режиме «вверх» или «вверх/вниз», он может быть остановлен путем записи 0 в TACCR0. Затем таймер может быть перезапущен путем записи ненулевого значения в TACCR0. В этом случае таймер начинает инкрементироваться от нуля.

Управление режимом таймера

Таймер имеет четыре режима работы, описанных в таблице 4.4.1: «стоп», «вверх», «непрерывный» и «вверх/вниз». Режимы работы выбираются с помощью битов MCx.

Режим «Вверх»

Режим «вверх» используется, если период таймера должен отличаться от количества отсчетов 0FFFFh. Таймер многократно считает вверх до значения,

содержащегося в регистре сравнения TACCR0, который задает период, как показано на рисунке 4.4.4. Количество отсчетов таймера в периоде равно TACCR0+1. Когда значение таймера становится равно содержимому TACCR0, таймер перезапускается, начиная счет от нуля. Если режим «вверх» выбран, когда значение таймера больше содержимого TACCR0, таймер немедленно перезапускается, начиная считать от нуля.

Таблица 4.4.1 – Режимы работы таймера A

МСх	Режим	Описание
00	Стоп	Останов таймера
01	Вверх	Таймер многократно считает от нуля до значения в TACCR0
10	Непрерывный	Таймер многократно считает от нуля до значения 0FFFFh
11	Вверх/вниз	Таймер многократно считает от нуля вверх до значения в TACCR0 и назад до нуля.

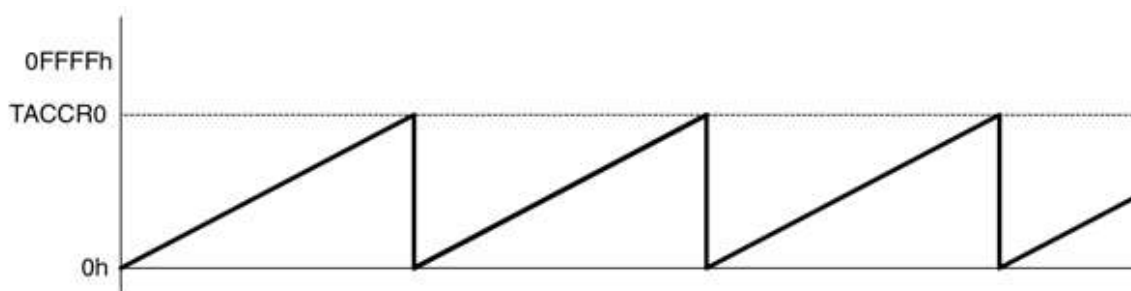


Рисунок 4.4.4 – Режим «Вверх»

Флаг прерывания TACCR0 CCIFG устанавливается, когда значение таймера равно содержимому TACCR0. Флаг прерывания TAIFG устанавливается, когда таймер считает от TACCR0 к нулю. На рисунке 4.4.5 показан цикл установки флагов.

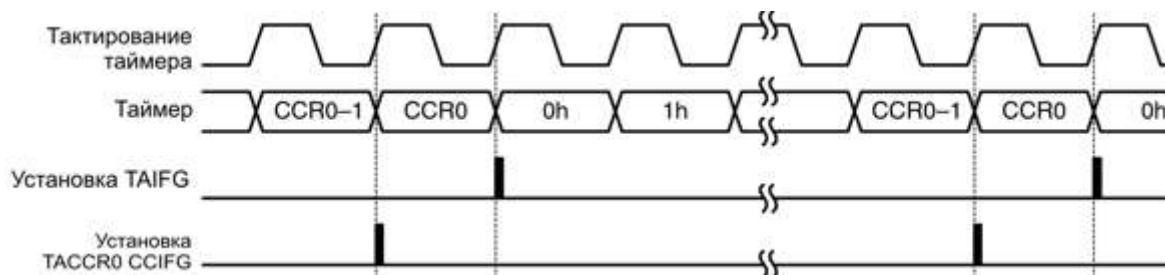


Рисунок 4.4.5 – Установка флагов в режиме «Вверх»

Непрерывный режим

В непрерывном режиме таймер многократно считает вверх до 0FFFFh и перезапускается от нуля, как показано на рисунке 4.4.6. Регистр захвата/сравнения TACCR0 работает подобно другим регистрам захвата/сравнения.

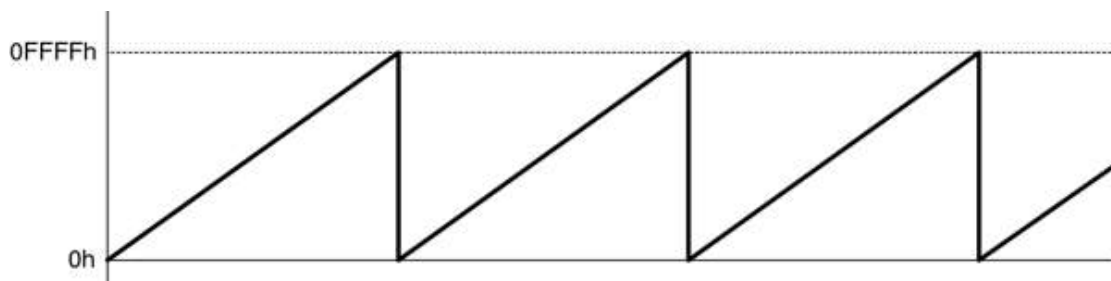


Рисунок 4.4.6 – Непрерывный режим

Флаг прерывания TAIFG устанавливается, когда таймер считает от 0FFFFh к нулю. На рисунке 4.4.7 показан цикл установки флага.

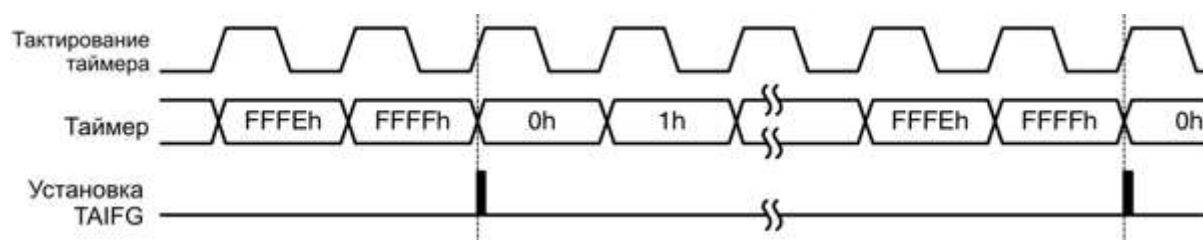


Рисунок 4.4.7 – Установка флага в непрерывном режиме

Режим «вверх/вниз»

Режим «вверх/вниз» используется, если период таймера должен отличаться от значения 0FFFFh и необходима генерация симметричных импульсов. Таймер непрерывно считает вверх до значения, находящегося в регистре сравнения TACCR0 и назад к нулю, как показано на рисунке 4.4.8. Период составляет удвоенную величину значения в TACCR0.

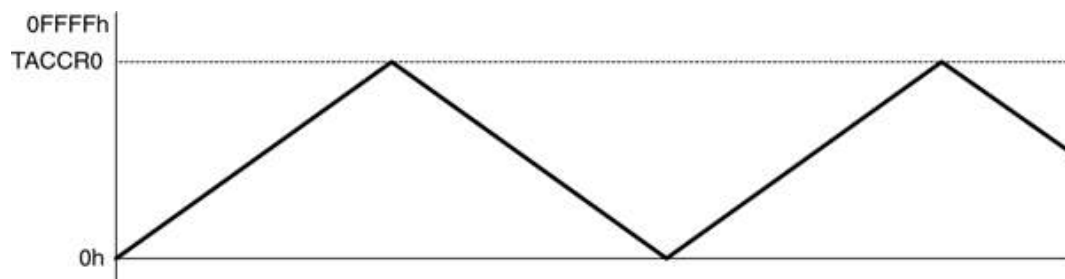


Рисунок 4.4.8 – Режим «вверх/вниз»

Направление счета запоминается. Это позволяет таймеру останавливаться и запускаться в том же направлении отсчета, которое было до останова. Если это нежелательно, для очистки направления нужно установить бит TACLR. Бит TACLR также очищает значения в TAR и в делителе TACLK.

В режиме «вверх/вниз» флаг прерывания TACCR0 CCIFG и флаг прерывания TAIFG устанавливаются только однажды во время периода, разделяясь 1/2 периода таймера. Флаг прерывания TACCR0 CCIFG устанавливается, когда таймер считает от TACCR0-1 к TACCR0, а флаг TAIFG устанавливается, когда таймер завершает счет вниз от 0001h к 0000h. На рисунке 4.4.9 показан цикл установки флагов.

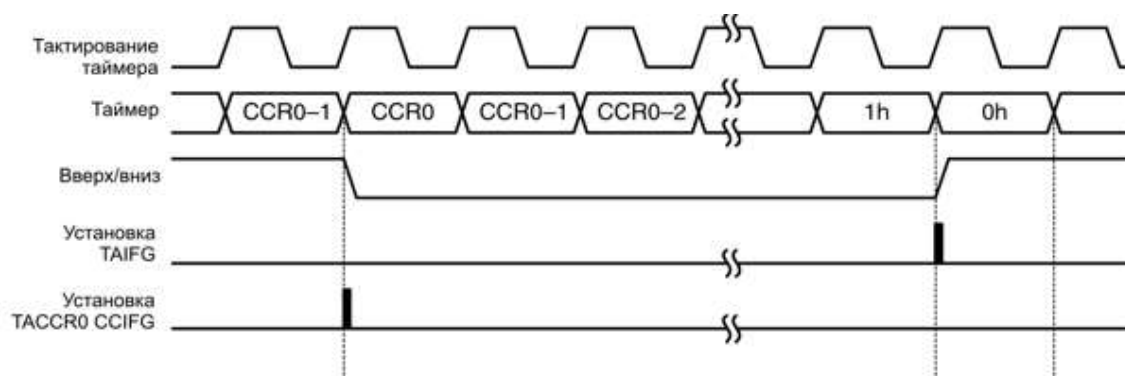


Рисунок 4.4.9 – Установка флагов в режиме вверх/вниз

Блоки захвата/сравнения

В таймере A представлено три одинаковых блока захвата/сравнения TACCRx. Любой из блоков может быть использован для фиксации (захвата) данных таймера или для генерации временных интервалов.

Режим захвата

Режим захвата выбирается, когда CAP=1. Режим захвата используется для регистрации временных событий. Это может потребоваться для быстрых вычислений или для измерения времени. Входы захвата CCIxA и CCIxB подключаются к внешним выводам или к внутренним сигналам и выбираются с помощью битов CCISx. Биты CMx определяют, как будет происходить захват: по фронту входного сигнала, по его спаду или в обоих случаях. Захват происходит по выбранному фронту входного сигнала. Если захват произошел, тогда:

- значение таймера копируется в регистр TACCRx;
- устанавливается флаг прерывания CCIFG.

Уровень входного сигнала может быть прочитан в любое время через бит

CCI. К входам CCIxA и CCIxB в устройствах семейства MSP430x1xx могут подключаться различные сигналы.

Сигнал захвата может быть асинхронен тактовой частоте таймера и вызывать состояние гонки сигналов. Установка бита SCS будет синхронизировать захват со следующим тактовым импульсом таймера. Рекомендуется устанавливать бит SCS для синхронизации сигнала захвата с тактовыми импульсами таймера.

Инициирование захвата программным обеспечением

Захваты могут быть инициированы программно. Биты CMx могут быть установлены для выполнения захвата по обоим фронтам. В этом случае программное обеспечение устанавливает CCI=1 и переключает бит CCISO для переключения сигнала захвата между VCC и GND, иницируя захват каждый раз, когда CCISO изменяет состояние:

```
MOV #CAP+SCS+CCIS1+CM_3, &TACCTLx      ;Настройка TACCTLx
XOR #CCISO, &TACCTLx                    ;TACCTLx = TAR
```

Режим сравнения

Режим сравнения выбирается, когда CAP=0. Режим сравнения используется для генерации выходных ШИМ-сигналов или прерываний через конкретные временные интервалы. Когда TAR досчитывает до значения в TACCRx, происходит следующее:

- устанавливается флаг прерывания CCIFG;
- внутренний сигнал EQU=1;
- EQUx воздействует на выход согласно режиму вывода;
- входной сигнал CCI фиксируется в SCCI.

Прерывания Таймера А

С 16-разрядным модулем таймера А связаны два вектора прерываний:

- вектор прерывания TACCR0 для флага TACCR0 CCIFG;
- вектор прерывания TAIV для всех других флагов CCIFG и TAIFG.

В режиме захвата любой флаг CCIFG устанавливается, когда значение таймера зафиксировано в соответствующем регистре TACCRx. В режиме сравнения устанавливается любой флаг CCIFG, если TAR досчитал до соответствующего значения TACCRx. Программное обеспечение может также устанавливать или очищать любой флаг CCIFG. Все флаги CCIFG запрашивают прерывание, когда установлены их соответствующие биты CCIE и бит GIE.

Прерывание TACCR0

Флаг TACCR0 CCIFG обладает наивысшим приоритетом прерывания Таймера A и имеет специализированный вектор прерывания, как показано на рисунке 4.4.10. Флаг TACCR0 CCIFG автоматически сбрасывается, когда обслуживается запрос на прерывание TACCR0.

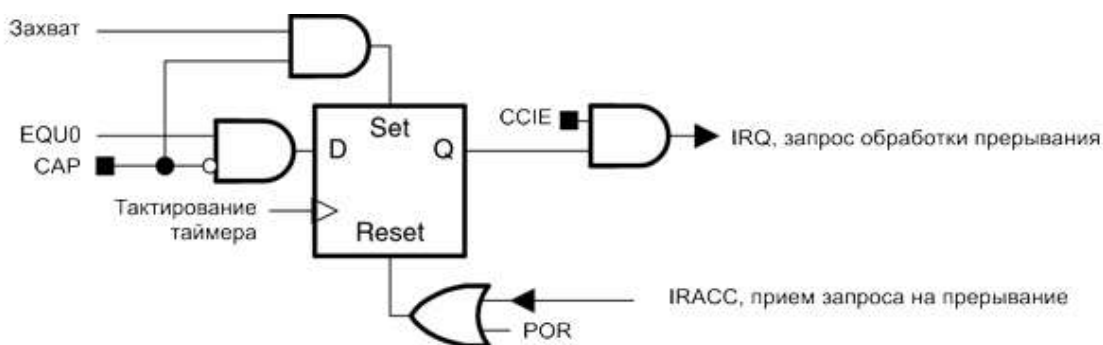


Рисунок 4.4.10 – Флаг прерывания захвата/сравнения TACCR0

Генератор вектора прерывания TAIV

Флаги TACCR1 CCIFG, TACCR2 CCIFG и TAIFG распределены по приоритетам и объединены в источник одного вектора прерывания. Регистр вектора прерывания TAIV используется для определения, какой флаг запросил прерывание.

Разрешенное прерывание с наивысшим приоритетом генерирует число в регистре TAIV. Можно оценить это число или добавить его к программному счетчику для автоматического входа в соответствующую программную процедуру. Запрещенные прерывания Таймера A не воздействуют на значение TAIV.

Любое обращение – чтение или запись регистра TAIV – автоматически сбрасывает флаг наивысшего ожидающего прерывания. Если установлен другой флаг прерывания, будет немедленно сгенерировано другое прерывание после обработки начального прерывания. Например, если флаги TACCR1 и TACCR2 CCIFG установлены, когда процедура обработки прерывания обращается к регистру TAIV, флаг TACCR1 CCIFG автоматически сбрасывается. После выполнения команды RETI процедуры обработки прерывания, флаг TACCR2 CCIFG генерирует другое прерывание.

Форматы управляющих регистров Таймера A и их описание приведены в [1] стр. 194-199.

Компаратор А

Модуль компаратора А поддерживает высокоточные аналого-цифровые преобразования напряжения, контроль напряжения питания и мониторинг внешних аналоговых сигналов. Блок-схема компаратора А показана на рисунке 4.4.11.

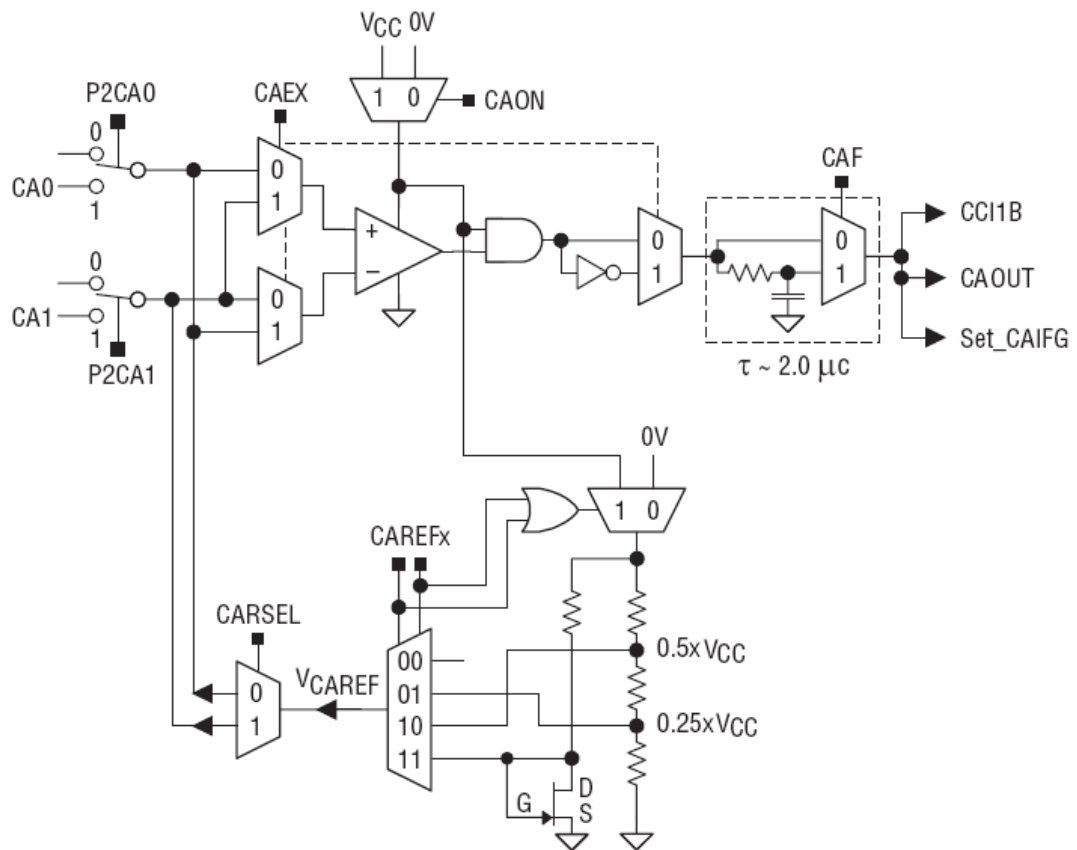


Рисунок 4.4.11 – Схема компаратора А

Компаратор сравнивает аналоговые напряжения на входах «+» и «-». Если вход «+» более положителен, чем вход «-», на выходе компаратора CAOUT появляется сигнал высокого уровня. Компаратор может быть включен или выключен с помощью бита CAON. Когда компаратор выключен, на выходе CAOUT всегда сигнал низкого уровня.

Входные аналоговые переключатели

Аналоговые входные переключатели подключают или отключают два входа компаратора от соответствующих выводов порта с помощью битов P2CAx. Оба входа компаратора могут управляться индивидуально. Биты P2CAx позволяют:

- подключать внешние сигналы к входам «+» и «-» компаратора;
- подключать внутреннее опорное напряжения к соответствующему

выходному выводу порта.

Когда компаратор включен, его входы должны быть подключены к источнику сигнала, источнику питания или к общему выводу. В противном случае плавающие уровни напряжения могут вызвать неожиданные прерывания и повышенное потребление тока.

Бит CAEX управляет входным мультиплексором, определяющим, какие входные сигналы будут подключены к входам «+» и «-» компаратора.

Выходной фильтр

Выход компаратора может использоваться как с внутренней фильтрацией, так и без неё. Когда управляющий бит CAF установлен, выход фильтруется с помощью интегрирующего RC-фильтра.

Выход любого компаратора беспорядочно генерирует, если разность потенциалов на его входах мала. Внутренние и внешние паразитные эффекты, перекрестная связь между сигнальными линиями, линиями питания и другими частями системы оказывают влияние на эту генерацию, как показано на рисунке 4.4.12. Колебания на выходе компаратора снижают точность и разрешающую способность результата сравнения. Использование выходного фильтра может уменьшить ошибки, связанные с генерацией компаратора.

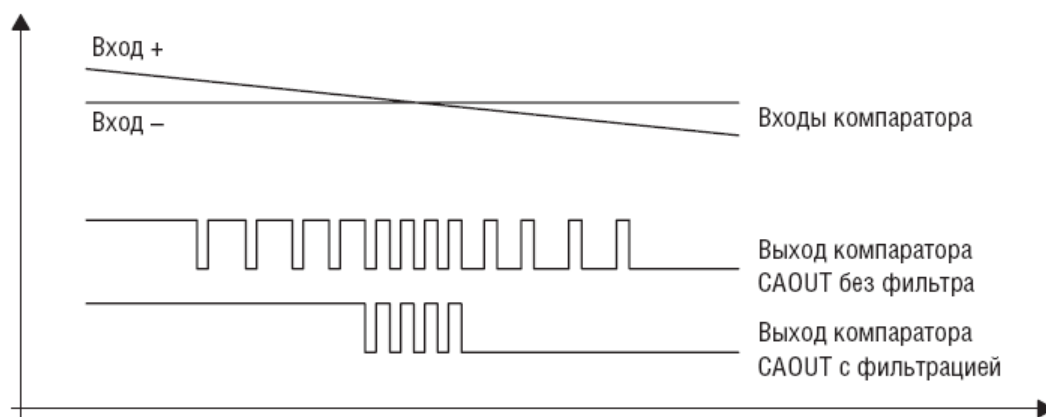


Рисунок 4.4.12 – Действие RC-фильтра на выходе компаратора

Генератор опорного напряжения

Генератор опорного напряжения используется для генерации напряжения VCAREF, которое может быть приложено к любому из входов компаратора. Бит CAREFx управляет выходом генератора напряжения. Бит CARSEL определяет вывод компаратора, к которому будет приложено напряжение VCAREF. Если внешние сигналы приложены к обоим входам компаратора, внутренний опорный генератор необходимо выключить, чтобы уменьшить

величину потребляемого тока. Генератор опорного напряжения может вырабатывать напряжение, пропорционально уменьшенное относительно напряжения питания V_{CC} или фиксированное пороговое напряжение транзистора около 0,55 В.

Регистр отключения порта CAPD

Функции ввода и вывода компаратора мультиплексированы с соответствующими ножками порта ввода/вывода, которые являются цифровыми КМОП-схемами. Когда аналоговые сигналы подключаются к цифровым КМОП-элементам, может появиться паразитный ток между V_{CC} и общим выводом. Этот паразитный ток появляется в случае, если величина входного напряжения находится около переходного уровня ячейки. Отключение буфера вывода порта устраняет паразитный ток и приводит к снижению общего потребления тока.

Когда биты CAPD x установлены, соответствующий входной буфер P2 отключается, как показано на рисунке 4.4.13. Когда величина уровня потребления тока критична, любой вывод P2, подключенный к аналоговым сигналам, должен быть отключен с помощью соответствующего бита CAPD x .

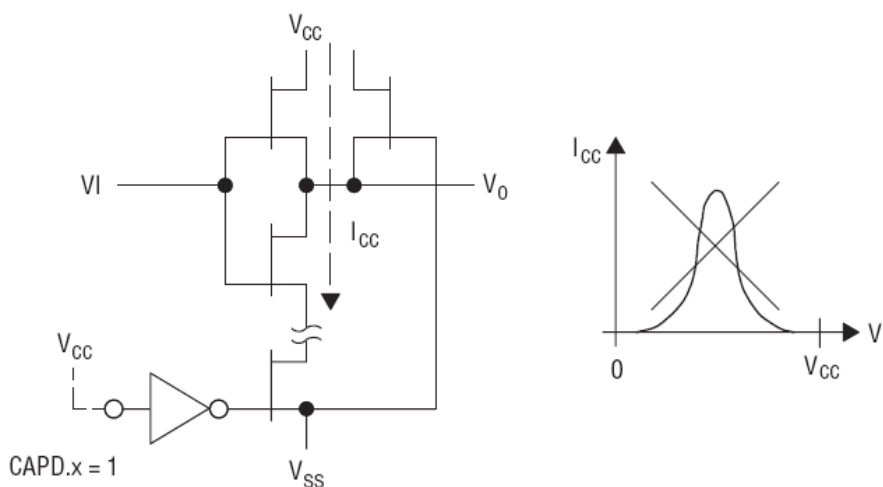


Рисунок 4.4.13 – Переходная характеристика и рассеивание мощности в инверторе/буфере КМОП

Прерывания компаратора А

С компаратором А связан один флаг прерывания и один вектор прерывания, как показано на рисунке 4.4.14. Флаг прерывания CAIFG устанавливается по любому фронту (нарастающему или спадающему) сигнала на выходе компаратора, что определяется битом CAIES. Если установлены оба

бита CAIE и GIE, флаг CAIFG генерирует запрос прерывания. Флаг CAIFG автоматически сбрасывается, когда обрабатывается запрос прерывания или может быть сброшен программно.

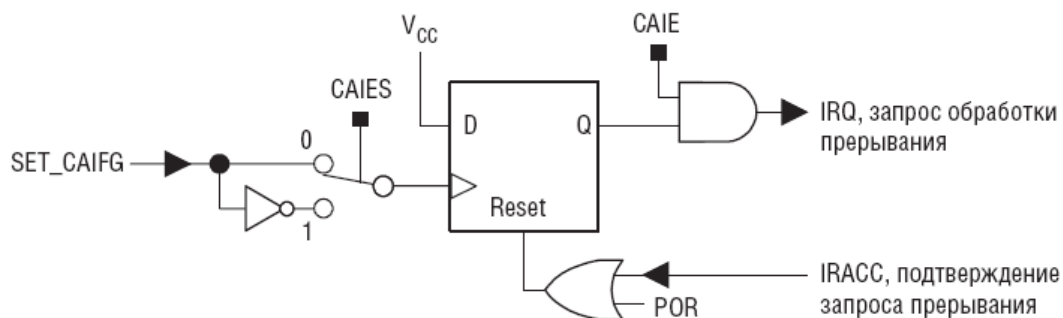


Рисунок 4.4.14 – Система прерывания компаратора А

Форматы управляющих регистров Компаратора и их описание приведены в [1] стр. 302-305.

4.4.2 Описание лабораторной установки

Задания выполняются на лабораторном стенде на базе 16-ти разрядного микроконтроллера MSP430.

Внутренний компаратор А микроконтроллера MSP430 можно использовать для высокоточного измерения резистивных элементов с помощью аналого-цифрового преобразования с одиночным интегрированием. К примеру, величина неизвестного резистора может быть измерена путем сравнения времен разряда конденсатора, подключаемого сначала к опорному резистору, а затем к измеряемому резистору. Сопротивление опорного резистора R_{ref} сравнивается с R_x (рисунок 4.2.15).

На этом принципе основана работа резистивного датчика, принципиальная схема подключения которого показана на рисунке 4.4.16.

Для вычисления сопротивления R_x , используются ресурсы MSP430:

- две цифровых ножки ввода/вывода для заряда и разряда конденсатора;
- ножка ввода/вывода устанавливается в высокий уровень (VCC) для заряда конденсатора и сбрасывается для разряда;
- когда ножка ввода/вывода не используется, она переключается на вход в «третье» состояние установкой соответствующего разряда CAPD4 или CAPTD5;

- один выход заряжает и разряжает конденсатор через Rref;
- один выход разряжает конденсатор через Rx;
- вход «+» компаратора CA0 подключается к положительному выводу конденсатора;
- вход «-» компаратора CA0 подключается к опорному уровню $0,25 \times V_{CC}$ (путем установки в регистре CACTL1 разрядов CAREF в значение 01);
- для минимизации шумов переключения должен использоваться выходной фильтр путем установки разряда CAF регистра CACTL2;
- выход CAOUT подключен к CCI1B таймера A, выполняющего захват времени разряда конденсатора.

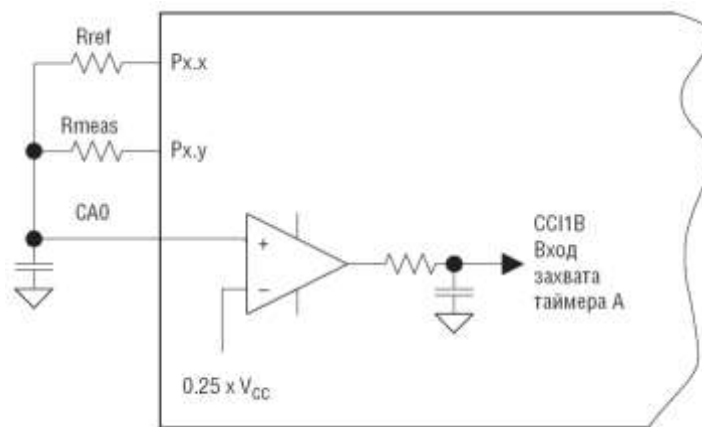


Рисунок 4.4.15 – Система измерения сопротивления резистивного датчика

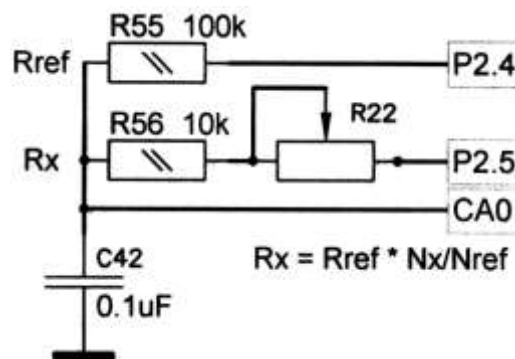


Рисунок 4.4.16 – Принципиальная схема резистивного датчика

Измерение сопротивления основано на принципе преобразования соотношения измерений. Отношение двух величин времен разряда конденсатора рассчитывается так, как показано на рисунке 4.4.17.

Величина сопротивления Rx рассчитывается по формуле:

$$R_x = R_{ref} * N_x / N_{ref}, \quad (4.4.1)$$

где R_x – искомое значение, Ом;

R_{ref} – сопротивление опорного резистора, Ом ($R_{ref} = R_{55} = 100000$ Ом);

N_x – значение накопленное в фазе измерения сопротивления датчика (t_{meas}) в счетчике таймера А, работающего в режиме захвата,

N_{ref} – значение накопленное в фазе измерения опорного сопротивления (t_{ref}) в счетчике таймера А, работающего в режиме захвата,

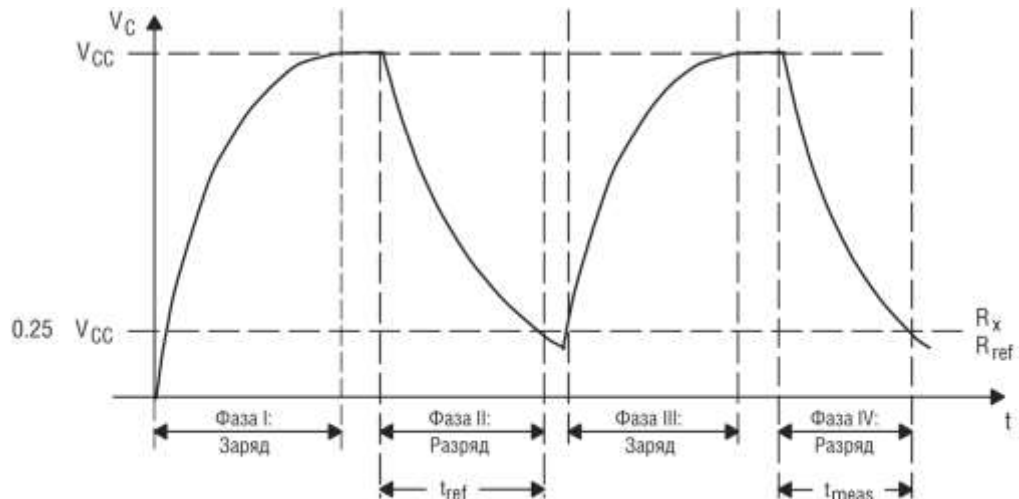


Рисунок 4.4.17 – Временные диаграммы измерения сопротивления

Зная сопротивление R_x , можно вычислить искомое сопротивление подстроечного резистора R_{22} по формуле:

$$R_{22} = R_x - R_{56} = R_x - 10000. \quad (4.4.2)$$

Размещение на плате подстроечного резистора показано на рисунке 4.4.18.

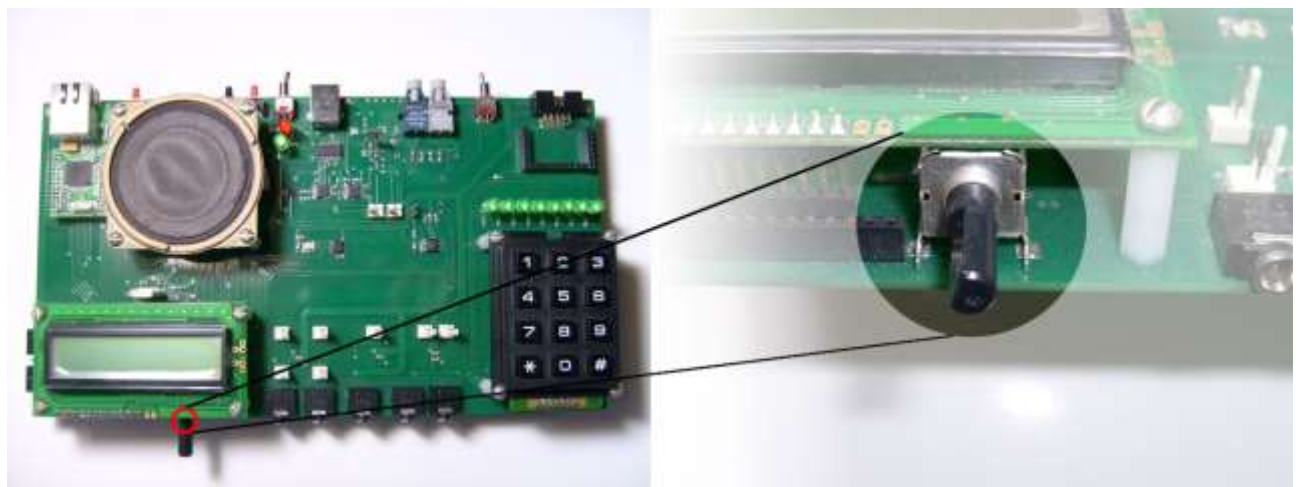


Рисунок 4.4.18 – Размещение подстроечного резистора на плате

4.4.3 Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс-контроль знаний по принципам функционирования таймеров/счетчиков и компаратора, входящих в состав микроконтроллера MSP430, а также по способам задания и измерения временных интервалов.

Задание. Разработать в среде программирования IAR Embedded Workbench программу на языке C для микроконтроллера MSP430, которая обеспечивает измерение сопротивления переменного резистора и выводит рассчитанное значение на ЖКИ. Для решения задачи необходимо использовать встроенный компаратор и таймер А в режиме захвата.

Для измерения величины сопротивления неизвестного резистора необходимо разработать функции `R22_get_resistance()`, `res_measure()` и обработчик прерываний от таймера, схемы алгоритмов которых изображены на рисунке 4.4.19.

В заголовочном `h`-файле необходимо ввести определения для ножек порта P2, к которым подключены соответствующие резистивные элементы: R_x – BIT5, R_{ref} – BIT4. Функция `R22_get_resistance()` конфигурирует ножки R_x и R_{ref} как порты ввода-вывода, а затем вызывает функцию измерения времени разряда конденсатора `res_measure()`, передавая ей в качестве аргумента сначала вывод R_{ref} , а затем R_x . Полученные соответствующие значения времени разряда конденсатора через опорный резистор $R_{ref} - N_{ref}$ и через подстроечный резистор $R_x - N_x$, используются для вычисления сопротивления резистора R22, по формулам (4.4.1) и (4.4.2).

Функция `res_measure(byte Rpin)` получает аргумент R_{pin} – номер ножки, через которую будет разряжаться конденсатор. Сначала от конденсатора отключается ножка R_x , путем установки направления порта на ввод («третье состояние»). Это делается для того, чтобы исключить искажение измерений из-за прохождения тока через нее в процессе разрядки конденсатора. Также отключаются входные буферы компаратора, кроме ножки R_{ref} – устанавливая все разряды кроме 4-го в 1. Ножка R_{ref} конфигурируется на выход, и на нее подается сигнал высокого уровня, для зарядки конденсатора. Конфигурируется таймер А, в регистре управления которого задаются параметры тактирования таймера от SMCLK, использования делителя входной тактовой частоты на 4, непрерывный режим счета и очистку счетчика. После этого контроллер переходит в режим пониженного энергопотребления LPM0, ожидая прерывание от таймера. Обработчик прерываний таймера инициирует выход контроллера из

LPM0, и продолжение выполнения программы.

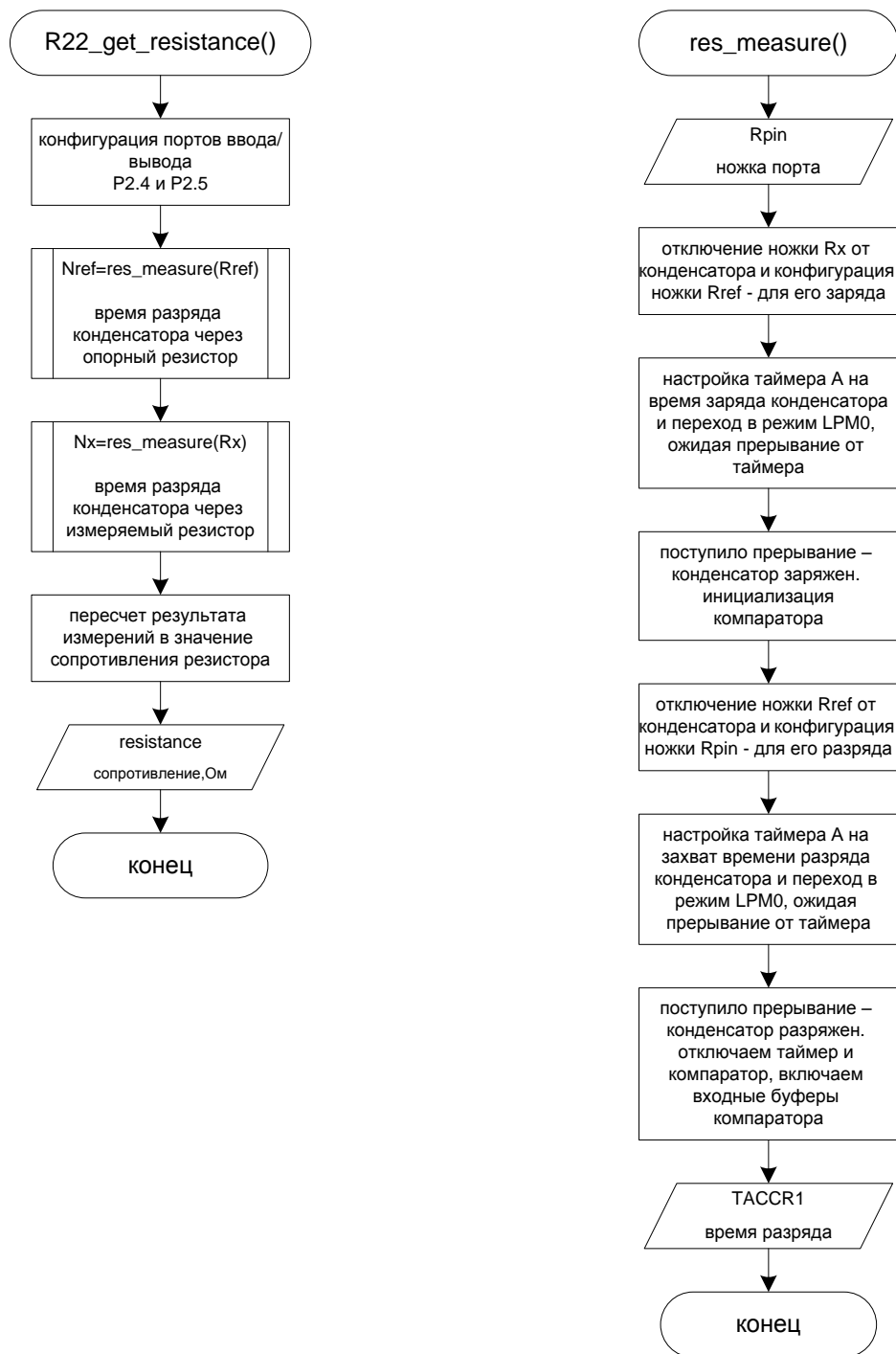


Рисунок 4.4.19 – Алгоритмы функций для работы с резистивным датчиком

Далее конфигурируется компаратор, установкой разрядов в регистре SACTL2 – P2CA0 (вход компаратора подключается к CA0) и CAF (включение выходного фильтра компаратора), а в регистре SACTL1 устанавливаются разряды CARSEL, CAREF_1 и CAON, что значит приложить к выводу «—»

компаратора опорное напряжение $0,25 \cdot V_{cc}$ и включить компаратор. Затем направление ножки Rref переключается на «ввод», что отключает ножку Rref от конденсатора.

После этого ножка Rx переключается на вывод с низким уровнем – конденсатор будет разряжаться через нее. Для измерения времени разряда конденсатора конфигурируется таймер A в режим захвата по заднему фронту входного сигнала CCI1B от компаратора A, сбрасывается счетчик таймера и разрешаются прерывания. Устанавливается режим пониженного энергопотребления LPM0, и ожидаются прерывания. Когда конденсатор разрядится – выход компаратора сменится с 1 на 0, и так как захват таймера установлен по заднему фронту, то произойдет прерывание таймера, а значение таймера (время разряда конденсатора) будет в регистре TACCCR1. Далее останавливается таймер, выключается компаратор и включаются входные буферы компаратора.

Порядок выполнения задания:

- включить лабораторный макет.
- запустить компилятор IAR Embedded Workbench.
- создать пустой проект.
- создать файл ресурса для кода программы и подключить его к проекту.
- ввести код исходного модуля программы измерения сопротивления переменного резистора.
- выполнить компиляцию исходного модуля программы и устранить ошибки, полученные на данном этапе.
- настроить параметры программатора.
- создать загрузочный модуль программы и выполнить программирование микроконтроллера.
- проверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.

В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы.

4.4.4 Содержание отчета

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный модуль разработанной программы;

– анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности использования встроенных таймеров микроконтроллера для формирования аппаратно-независимых временных интервалов.

4.4.5 Контрольные вопросы и задания

1. В чем преимущества обмена по прерываниям по сравнению с другими известными вам способами обмена информацией?
2. Что включает в себя понятия системы прерываний?
3. Поясните понятия вектора прерываний и таблицы векторов прерываний.
4. Какие действия выполняет микроконтроллер при переходе на процедуру обработки прерывания?
5. Поясните принципы формирования временных интервалов с помощью 16-разрядного таймера-счетчика.
6. В чем отличия между таймерами А и В?
7. В чем состоит принцип измерения сопротивления резистивного датчика при помощи компаратора?

4.5 ИЗУЧЕНИЕ ПРИНЦИПОВ РАБОТЫ СО ВСТРОЕННЫМ В МИКРОКОНТРОЛЛЕР АНАЛОГО-ЦИФРОВЫМ ПРЕОБРАЗОВАТЕЛЕМ НА ПРИМЕРЕ ИЗМЕРЕНИЯ ОТНОСИТЕЛЬНОЙ ВЛАЖНОСТИ ВОЗДУХА И ПОТРЕБЛЯЕМОГО СТЕНДОМ ТОКА

Цель работы: изучить принципы функционирования встроенного в микроконтроллер MSP430F1611 АЦП и методику измерения относительной влажности и потребляемого тока с помощью датчиков влажности и тока.

4.5.1 Указания по организации самостоятельной работы.

Перед работой необходимо проработать теоретический материал по литературе [1–4] и конспекту лекций, ознакомиться с основными возможностями и принципами функционирования АЦП, изучить принципы работы датчиков относительной влажности и тока.

Двенадцатиразрядный АЦП12

В состав микроконтроллеров MSP430x13x, MSP430x14x, MSP430x15x и MSP430x16x входит 12-разрядный аналого-цифровой преобразователь (АЦП).

Модуль АЦП12 обеспечивает быстрые 12-разрядные аналого-цифровые преобразования. Модуль имеет 12-разрядное ядро SAR, схему выборки, опорный генератор и буфер преобразования и управления объемом 16 слов. Буфер преобразования и управления позволяет получать и сохранять до 16 независимых выборок АЦП без вмешательства ЦПУ. Структурная схема АЦП12 показана на рисунке 4.5.1.

Ядро АЦП преобразует аналоговый входной сигнал в 12-разрядное цифровое представление и сохраняет результат в памяти преобразований. Ядро использует два программируемых/выбираемых уровня напряжения (V_{R+} и V_{R-}) для задания верхнего и нижнего пределов преобразования. Когда входной сигнал равен или выше V_{R+} на цифровом выходе (N_{ADC}) формируется значение 0FFFh, и ноль, когда входной сигнал равен или ниже V_{R-} . Входной канал и опорные уровни напряжения (V_{R+} и V_{R-}) задаются в памяти управления преобразованиями. Формула преобразования для результата АЦП выглядит следующим образом:

$$N_{ADC} = 4095 \times \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}} \quad (4.5.1)$$

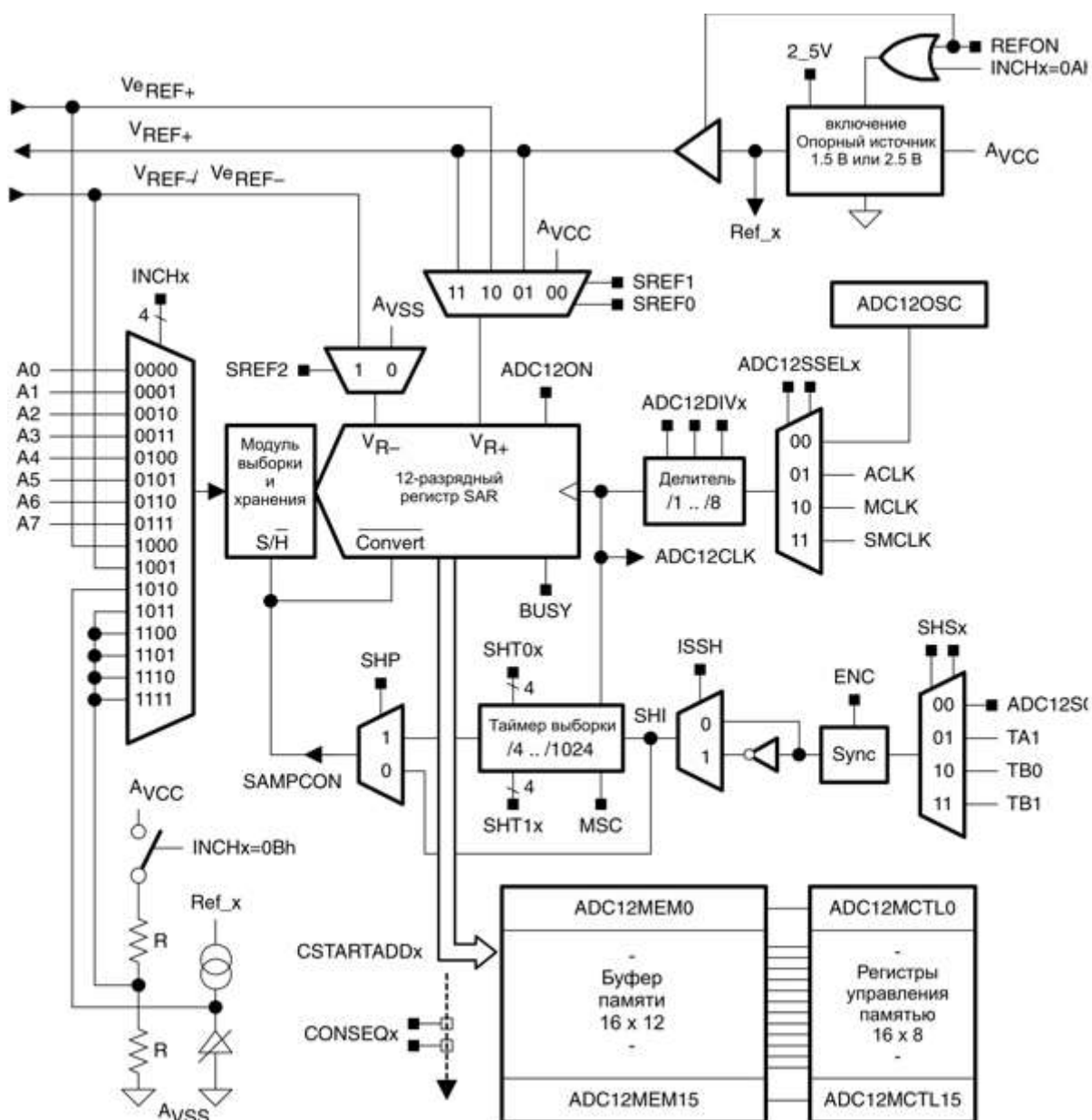


Рисунок 4.5.1 – Структурная схема АЦП12

Ядро АЦП12 конфигурируется двумя управляющими регистрами: ADC12CTL0 и ADC12CTL1. Ядро включается битом ADC12ON. За некоторыми исключениями биты управления АЦП12 могут быть модифицированы, только когда ENC=0. ENC должен быть установлен в 1 перед выполнением любого преобразования.

Выбор тактирования преобразования

ADC12CLK используется как для тактирования преобразования, так и для генерации периода выборки, когда выбран импульсный режим выборки. Для выбора источника тактирования ADC12 используются биты ADC12SELx, а частота выбранного источника может быть поделена на 1-8 с помощью битов

ADC12DIVx. Источниками сигнала тактирования ADC12CLK могут быть: SMCLK, MCLK, ACLK и внутренний осциллятор ADC12OSC.

Частота сигнала ADC12OSC составляет около 5 МГц, и зависит от конкретного устройства, напряжения питания и температуры.

Пользователь должен гарантировать, что выбранный источник тактирования для ADC12CLK останется активным до конца преобразования. Если тактирование будет остановлено во время преобразования, то операция не будет завершена и результат будет неверным.

Входы АЦП12 и мультиплексор

В качестве источников для АЦП могут использоваться восемь внешних и четыре внутренних аналоговых сигнала, коммутируемых аналоговым входным мультиплексором. Входной мультиплексор имеет тип break-before-make (разрыв перед включением), что уменьшает инжекцию шумов от канала к каналу, возникающую при переключении каналов, как показано на рисунке 4.5.2. Входной мультиплексор также является Т-переключателем, минимизирующим взаимосвязь между каналами. Невыбранные каналы изолированы от АЦП, а промежуточный узел подключен к аналоговой земле (AV_{SS}), поэтому паразитная емкость заземляется, что помогает устранять перекрестные помехи.

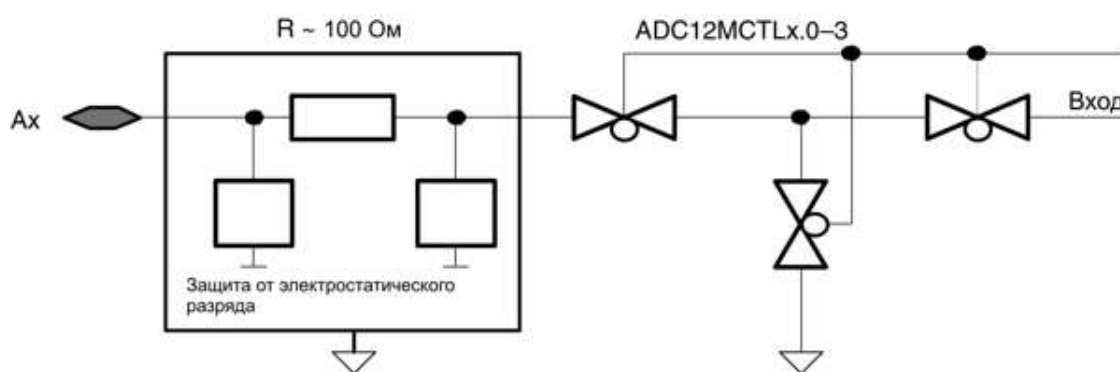


Рисунок 4.5.2 – Аналоговый мультиплексор

Выбор аналогового порта

Входы АЦП12 мультиплексированы с ножками порта P6, имеющими цифровые КМОП ячейки. Когда аналоговые сигналы прикладываются к цифровым КМОП-схемам, может течь паразитный ток от VCC к GND. Этот паразитный ток появляется, если величина входного напряжения находится около переходного уровня ячейки. Отключение буфера ножки порта устраняет протекание паразитного тока и вследствие этого уменьшает общий

потребляемый ток. Биты P6SELx дают возможность отключать входные буферы ножки порта.

```
; P6.0 и P6.1 конфигурируются как аналоговые входы  
BIS.B #3h,&P6SEL ; P6.1 и P6.0 – функция АЦП12  
BIS.B #3h,&P6DIR ; P6.1 и P6.0 переключены на ввод
```

Генератор опорного напряжения

Модуль АЦП12 содержит встроенный генератор опорного напряжения с двумя выбираемыми уровнями напряжения: 1,5 В и 2,5 В. Любое из этих опорных напряжений может быть использовано внутренне или внешне на выводе VREF+.

Чтобы включить внутренний опорный источник необходимо установить бит REFON=1. Если REF2_5V=1, то внутреннее опорное напряжение равно 2,5 В, а при REF2_5V=0 опорное напряжение равно 1,5 В.

Для правильной работы внутреннего генератора опорного напряжения необходимо использовать внешний конденсатор, подключенный между V_{REF+} и AV_{SS}. Рекомендуется использовать комбинацию из включенных параллельно конденсаторов на 10 мкФ и 0,1 мкФ. После включения в течение максимум 17 мс необходимо дать возможность генератору опорного напряжения зарядить конденсаторы.

Внешние опорные источники могут быть задействованы для V_{R+} и V_{R-} через выводы V_{eREF+} и V_{REF-/V_{eREF-}} соответственно.

Синхронизация выборки и преобразования

Аналого-цифровое преобразование инициируется по нарастающему фронту входного сигнала выборки SHI. Источник для SHI выбирается с помощью битов SHSx и может быть таким:

- бит ADC12SC;
- модуль вывода 1 таймера A;
- модуль вывода 0 таймера B;
- модуль вывода 1 таймера B.

Полярность источника сигнала SHI может быть инвертирована битом ISSH. Сигнал SAMPCON управляет периодом выборки и началом преобразования. Выборка активна, пока сигнал SAMPCON равен единице. По заднему фронту сигнала SAMPCON стартует аналого-цифровое преобразование, длительность которого составляет 13 циклов ADC12CLK.

Расширенный режим выборки

Расширенный режим выборки выбирается при $SHP=0$. Сигнал SHI напрямую управляет линией $SAMPCON$ и определяет длительность периода выборки t_{sample} . Когда $SAMPCON$ имеет высокий уровень, выборка активна. Переход сигнала $SAMPCON$ с высокого уровня на низкий запускает преобразование после синхронизации с $ADC12CLK$ (рисунок 4.5.3).

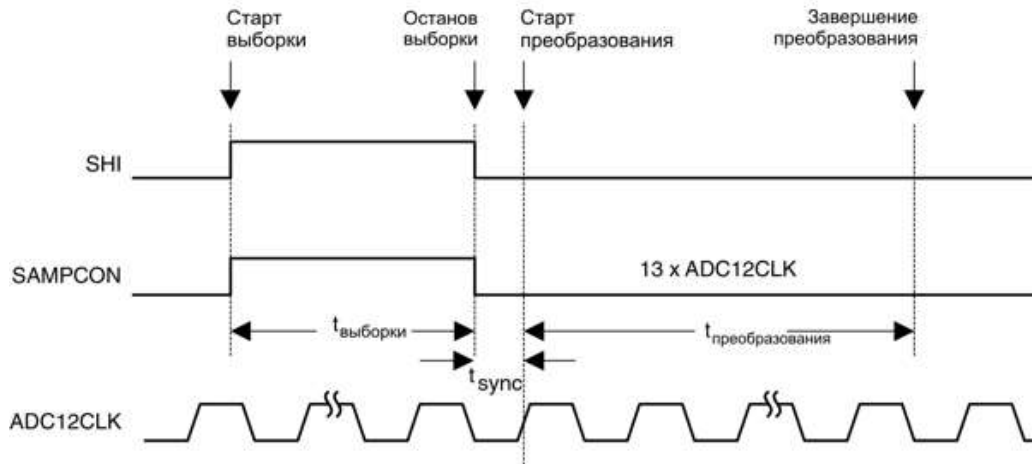


Рисунок 4.5.3 – Расширенный режим выборки

Импульсный режим выборки

Импульсный режим выборки выбирается при $SHP=0$. Сигнал SHI используется для запуска таймера выборки. Биты $SHT0x$ и $SHT1x$ в $ADC12CTL0$ управляют таймером выборки, который определяет период выборки t_{sample} . Сигнал $SAMPCON$ остается активным (уровень логической единицы) после синхронизации с $ADC12CLK$ в течение запрограммированного интервала t_{sample} . Общее время выборки составляет $t_{sample} + t_{sync}$ (рисунок 4.5.4).

Биты $SHTx$ задают время выборки кратным $4 \times ADC12CLK$. $SHT0x$ устанавливает время выборки для $ADC12MCTL0-7$, а $SHT1x$ устанавливает время выборки для $ADC12MCTL8-15$.

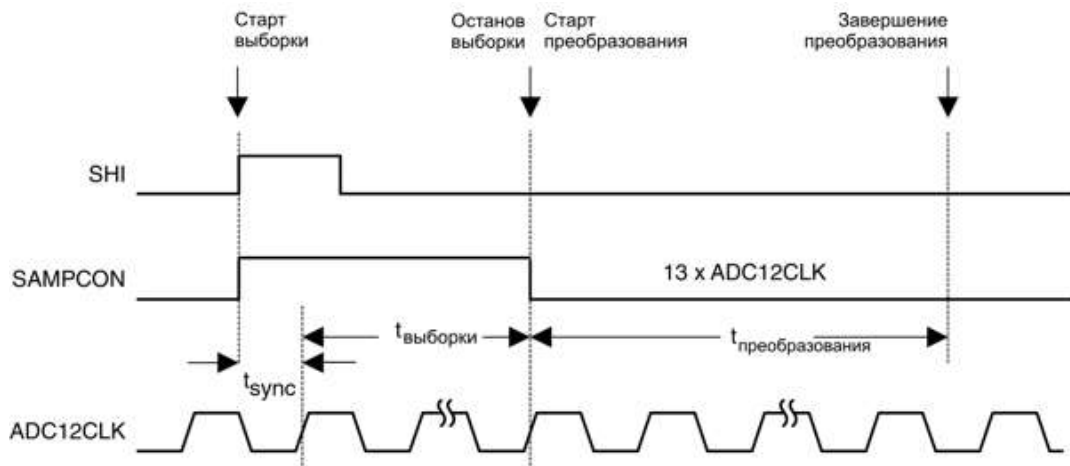


Рисунок 4.5.4 – Импульсный режим выборки

Определение длительности выборки

При $\text{SAMPCON}=0$ все входы Ax переводятся в высокоимпедансное состояние. При $\text{SAMPCON}=1$ выбранный вход Ax можно смоделировать в виде RC-фильтра нижних частот в течение периода квантования t_{sample} (рисунок 4.5.5). Внутреннее сопротивление R_I составляет около 2 кОм, а емкость конденсатора C_I не превышает 40 пФ. Конденсатор C_I должен быть заряжен напряжением V_C в пределах 1/2 младшего бита источника напряжения V_S для получения точного 12-разрядного преобразования.

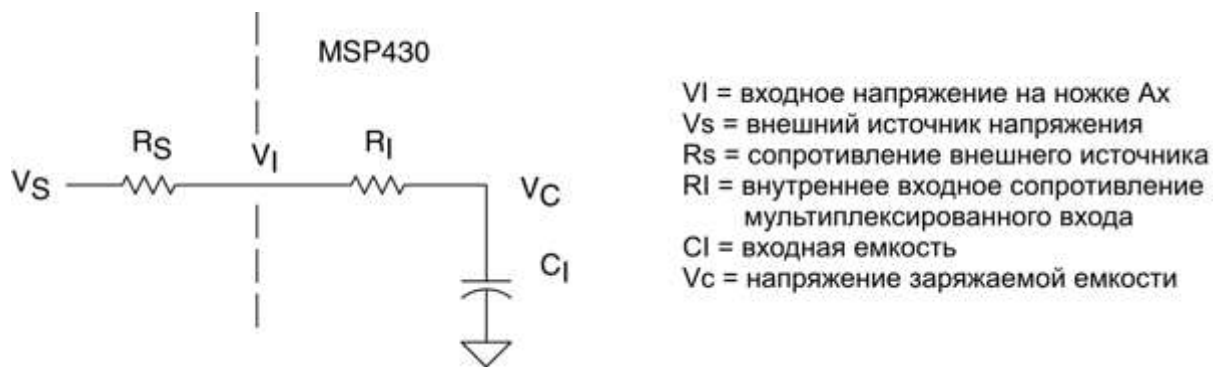


Рисунок 4.5.5 – Эквивалентная схема аналогового входа

Сопротивление источника R_S и R_I влияет на t_{sample} . Минимальное время выборки t_{sample} при 12-разрядном преобразовании определяется из соотношения

$$t_{\text{sample}} > (R_S + R_I) * \ln(2^{13}) * C_I$$

При подстановке значений R_I и C_I , указанных выше, неравенство приобретает следующий вид:

$$t_{\text{sample}} > (R_S + 2 \text{ кОм}) * 9.011 * 40 \text{ пФ}.$$

Например, если R_S равно 10 кОм, t_{sample} должно быть больше 4,33 мкс.

Память преобразований

Результаты преобразований сохраняются в 16-ти регистрах памяти преобразований ADC12MEMx . Каждый регистр ADC12MEMx конфигурируется соответствующим управляющим регистром ADC12MCTLx . Биты SREFx устанавливают опорное напряжение, а биты INCHx задают входной канал. Бит EOS определяет конец последовательности, если

используется последовательный режим преобразования. Если бит EOS в ADC12MCTL15 не установлен, то результаты преобразования последовательно сохраняются в регистрах с ADC12MEM15 по ADC12MEM0.

Биты CSTARTADDx определяют первый регистр ADC12MCTLx, используемый для преобразования. Если выбран одноканальный или повторный одноканальный режим преобразования, то биты CSTARTADDx указывают на единственный используемый регистр ADC12MCTLx.

Если выбран режим преобразования «последовательность каналов» или «повторяющаяся последовательность каналов», CSTARTADDx указывают на расположение первого регистра ADC12MCTLx, используемого в последовательности. Программно невидимый указатель автоматически инкрементируется после каждого преобразования. Последовательность продолжается пока не будет обнаружен установленный бит EOS в ADC12MCTLx.

При записи результата преобразования в выбранный регистр ADC12MEMx устанавливается соответствующий флаг в регистре ADC12IFGx.

Режимы преобразований АЦП12

АЦП12 имеет четыре режима работы, выбираемые битами CONSEQx так, как описано в таблице 4.5.1.

Таблица 4.5.1 – Перечень режимов работы АЦП

CONSEQx	Режим	Операция
00	Одноканальный одиночным преобразованием	Выполняется одно преобразование в одном канале.
01	Последовательность каналов	Выполняются однократные преобразования последовательности каналов.
10	Повторяющийся одноканальный	Выполняется повторяющееся преобразование в одном канале.
11	Повторяющаяся последовательность каналов	Выполняются повторяющиеся преобразования последовательности каналов.

Одноканальный режим с одиночным преобразованием

В одном канале однократно выполняется выборка и преобразование. Результат АЦП записывается в регистр ADC12MEMx, определенный битами CSTARTADDx. Если для запуска преобразования используется бит ADC12SC,

то очередное преобразование может быть запущено установкой бита ADC12SC. Если используется другой источник запуска, то бит ENC должен переключаться между каждым преобразованием.

Режим последовательности каналов

В режиме последовательности каналов однократно выполняется выборка и преобразование. Результат АЦП записывается в память преобразований, начиная с регистра ADCMEMx, номер которого задается битами CSTARTADDx. Последовательность останавливается после измерения в канале с установленным битом EOS. Если для запуска последовательности преобразований используется бит ADC12SC, то очередная последовательность преобразования может быть запущена установкой бита ADC12SC. Если используется другой источник запуска, то бит ENC должен переключаться между каждой последовательностью преобразования.

Повторяющийся одноканальный режим

В одном канале непрерывно выполняются выборка и преобразование. Результат АЦП записывается в ADC12MEMx, номер которого задается битами CSTARTADDx. Необходимо считывать результат после завершения текущего преобразования до завершения очередного преобразования.

Режим повторяющейся последовательности каналов

Непрерывно выполняются выборка и преобразование последовательности каналов. Результат АЦП записывается в память преобразований, начиная с регистра ADC12MEMx, номер определенного битами CSTARTADDx. Последовательность останавливается после измерения в канале с установленным битом EOS и стартует снова по следующему сигналу запуска.

Использование бита множественных выборок и преобразований

Для настройки АЦП на выполнение автоматических последовательных преобразований с максимальным быстродействием можно воспользоваться функцией множественных выборок и преобразований. Если бит MSC=1, CONSEQx>1 и используется таймер выборок, то первый фронт сигнала SHI запустит первое преобразование. Очередные преобразования запускаются автоматически после завершения предыдущего преобразования. Дополнительные фронты на SHI игнорируются, пока последовательность не будет закончена или пока бит ENC не будет переключен. Функция бита ENC не изменяется, пока используется бит MSC.

Останов преобразований

Прекращение активности АЦП12 зависит от режима работы. Рекомендуются следующие способы останова активного преобразования или

последовательности преобразований:

- сброс ENC в одноканальном режиме одиночного преобразования немедленно останавливает преобразование, при этом результат оказывается непредсказуемым (для получения правильного результата необходимо опрашивать бит занятости до сброса перед очисткой ENC);

- сброс ENC во время повторяющегося одноканального преобразования останавливает преобразователь в конце текущего преобразования;

- сброс ENC во время последовательного или повторно-последовательного режимов останавливает преобразователь в конце последовательности.

Любой режим преобразования может быть немедленно остановлен установкой $CONSEQx=0$ и сбросом бита ENC. Данные преобразования будут ненадежны.

Использование интегрированного температурного датчика

При использовании имеющегося на кристалле температурного датчика пользователь выбирает аналоговый входной канал $INCHx=1010_2$. Любая другая конфигурация рассматривается как выбор внешнего канала, включая выбор опорного источника, выбор памяти преобразований и т.д.

Типичная передаточная функция температурного датчика показана на рисунке 4.5.6. При работе с температурным датчиком период выборки должен быть больше 30 мкс.

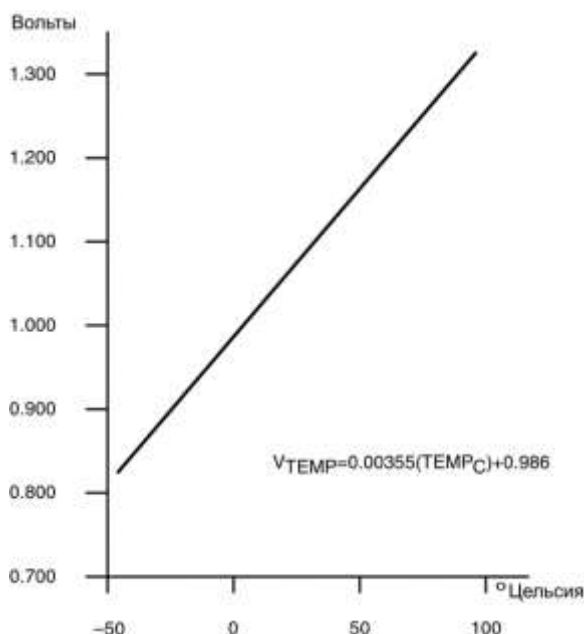


Рисунок 4.5.6 – Типичная передаточная функция температурного датчика

Прерывания АЦП12

АЦП12 имеет 18 источников прерывания:

- ADC12IFG0-ADC12IFG15;
- ADC12OV, переполнение ADC12MEMx;
- ADC12TOV, переполнение времени преобразования АЦП12.

Биты ADC12IFGx устанавливаются, когда в их соответствующие регистры памяти ADC12MEMx загружается результат преобразования. Если соответствующий бит ADC12IEx и бит GIE установлены, генерируется запрос прерывания. Состояние ADC12OV появляется, когда результат нового преобразования записывается в любой регистр ADC12MEMx до прочтения предыдущего результата. Состояние ADC12TOV генерируется, когда до завершения текущего преобразования затребована другая выборка-преобразование.

Вектор прерываний ADC12IV

Все источники прерываний АЦП12 разделены по приоритетам и являются источником одного вектора прерываний. Регистр вектора прерываний ADC12IV используется для определения, какой из разрешенных источников прерываний АЦП12 запрашивает прерывание.

Разрешенное прерывание АЦП12 с наивысшим приоритетом генерирует число в регистре ADC12IV. Это число может быть оценено или добавлено к программному счетчику для автоматического входа в соответствующую программную процедуру. Запрещенные прерывания АЦП12 не влияют на значение ADC12IV.

При любом типе доступа (чтение или запись) к регистру ADC12IV автоматически сбрасывается состояние ADC12OV или ADC12TOV, если любое из них было наивысшим ожидающим прерыванием. При доступе к ADC12IV флаги ADC12IFGx не сбрасываются. Биты ADC12IFGx сбрасываются автоматически, при доступе к соответствующим им регистрам ADC12MEMx, или же программно.

Форматы управляющих регистров АЦП12 приведены в [1] стр. 323-331.

4.5.2 Описание лабораторной установки

Задания выполняются на лабораторном стенде на базе 16-ти разрядного микроконтроллера MSP430F1611.

4.5.2.1 Подключение датчика влажности

Для измерения влажности используются датчики, основанные на

различных физических принципах и выполненные по различным технологиям. Можно выделить основные четыре типа датчиков: емкостные, резистивные, на основе оксида олова и на основе оксида алюминия (таблица 4.5.3). Рассмотрим кратко особенности каждого типа.

Из представленных четырех основных типов для измерения влажности, оптимальным по совокупности параметров является емкостной. Он обеспечивает широкий диапазон измерений, высокую надежность и низкую стоимость при использовании микроэлектронной технологии. Благодаря чему мы имеем миниатюрные габариты чувствительного элемента, возможность имплементации на кристалле специализированной интегральной схемы обработки сигнала. Технологичность и высокий выход годных кристаллов обеспечивают малую стоимость продукции данного типа. Итак, для измерения влажности емкостной метод является лучшим.

Таблица 4.5.3 – Отличительные особенности различных типов датчиков влажности

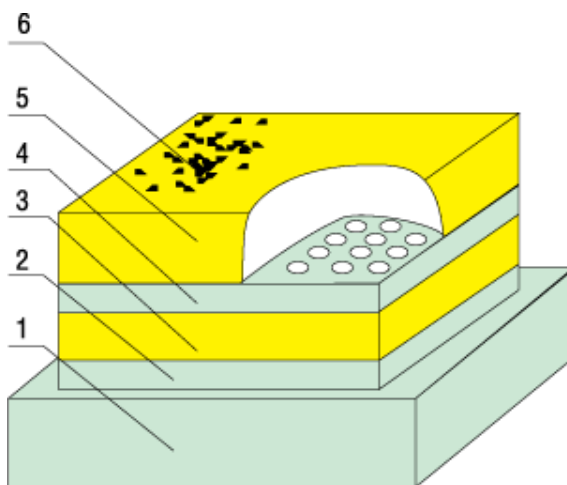
Тип датчика	Особенности
Емкостной	Высокая надежность, высокий выход годных кристаллов, низкая стоимость, широкий рабочий диапазон.
Резистивный	Самые дешевые, малая доля рынка.
На основе оксида олова	Плохая стабильность, плохая взаимозаменяемость
На основе оксида алюминия	Узкий диапазон измерения (малая влажность)

Рассмотрим теперь отличительные особенности датчиков влажности емкостного типа, которые предлагает компания Honeywell. Чувствительный элемент представляет собой многослойную структуру (рисунок 4.5.7).

На кремниевой подложке (1) напылена платиновая пленка (2), которая образует первый электрод конденсатора. Диэлектриком между обкладками служит термореактивный полимер (3), поверх которого выполнена вторая обкладка конденсатора – платиновая пленка с перфорацией (4), позволяющая влаге проникать к абсорбирующему слою (3) и изменять его относительную диэлектрическую проницаемость, а соответственно – и емкость конденсатора.

Верхним слоем является пленка термореактивного полимера (5), которая

служит защитой от пыли и грязи. Также эта конструкция делает возможной промывку датчиков, для этой цели рекомендуется использовать изопропиловый спирт.



1 – кремниевая подложка; 2, 4 – платиновый электрод;
3, 5 – термореактивный полимер; 6 – пыль, грязь, масло.

Рисунок 4.5.7 – Структура чувствительного элемента датчика влажности

Стоит также особо выделить диэлектрик – термореактивный полимер, который использует Honeywell в своих датчиках влажности. Емкостные датчики на основе термореактивного полимера имеют преимущества по сравнению с датчиками на основе термопластичного полимера: они долговечнее, более стойки к воздействиям окружающей среды, имеют высокую химическую стойкость и обладают широким рабочим температурным диапазоном.

В серии датчиков НН-4000 при образовании влаги на поверхности чувствительного элемента выходной сигнал устанавливается соответствующим низкому уровню (порядка 39 мВ), т.е. показывает отсутствие влажности. Без сомнения, это нововведение полезно, с помощью него есть возможность формировать сигнал ошибки, так как показания датчиков влажности в условиях конденсации влаги не являются точными.

В лабораторном стенде использован датчик влажности Honeywell НН-4000-003 (рисунок 4.5.8), технические характеристики которого приведены в таблице 4.5.4.



Рисунок 4.5.8 – Внешний вид датчика влажности Honeywell HIH-4000-003

Таблица 4.5.4 – Технические характеристики датчика HIH-4000-003

Параметр	Значение
Диапазон измерения, % RH	0...100
Повторяемость, $\pm\%$ RH	0,5
Напряжение питания, В	4,0...5,8
Ток потребления, мА	0,2
Рабочая температура, $^{\circ}\text{C}$	-40...85
Температура хранения, $^{\circ}\text{C}$	-50...125
Время отклика, с	15
Встроенный датчик температуры	нет
Калибровочный паспорт	есть

Все датчики влажности HIH-4000 комплектуются калибровочным паспортом. Этот документ отражает реальные значения выходных напряжений при эталонных значениях относительной влажности, крутизну характеристики преобразования, номера партии и пластины, соответствующие конкретному экземпляру. Таким образом, у каждого датчика своя передаточная характеристика. Образец калибровочного паспорта и типичной характеристики преобразования влажности в напряжение представлены на рисунке 4.5.9.

Для выполнения преобразования выходного напряжения датчика в процент влажности, служит формула (4.5.2), константы для которой берутся из паспорта конкретного датчика:

$$RH = \frac{(V_{out} - zero_offset)}{slope}, \quad (4.5.2)$$

где RH – относительная влажность, %;

V_{out} – выходное напряжение датчика, В;

$zero_offset$ – начальное смещение, равно $V_{out} @ 0\% RH$, В;

slope – угол наклона, В / % RH.

DATA PRINTOUT (EXAMPLE)	
MODEL	HHH-4000-003
CHANNEL	92
WAFER	030996M
MRP	337313
CALCULATED VALUES AT 5 V	
V _{out} @ 0% RH	0.958 V
V _{out} @ 75.3% RH	3.268 V
LINEAR OUTPUT FOR 3.5% RH	
ACCURACY @ 25 °C	
ZERO OFFSET	0.958 V
SLOPE	30.680 mV/%RH
SENSOR RH	(V _{out} -ZERO OFFSET)/SLOPE
	(V _{out} -0.958)/0.0307
RATIOMETRIC RESPONSE FOR 0 TO 100% RH	
V _{out}	V _{SUPPLY} (0.1900 TO 0.8040)

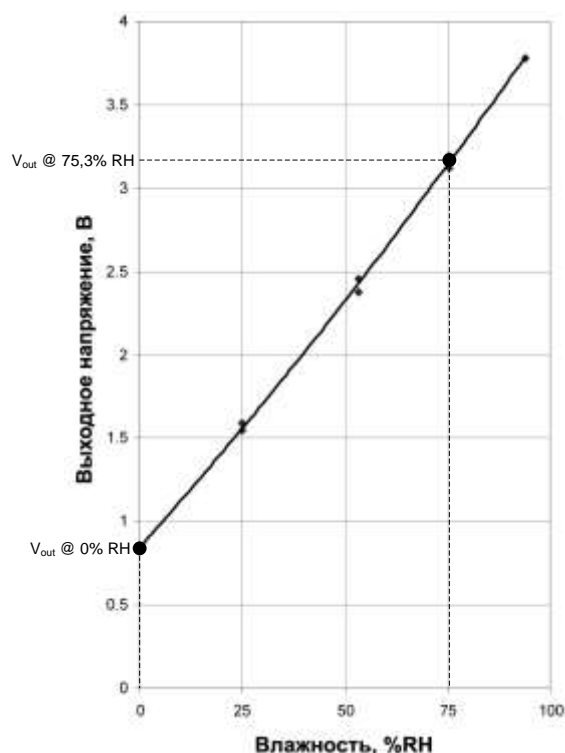


Рисунок 4.5.9 – Образец паспорта и характеристики преобразования датчика

Угол наклона *slope* определяет линейную зависимость выходного напряжения от относительной влажности, и вычисляется по формуле:

$$slope = \frac{V_{out} @ 75,3\% RH - V_{out} @ 0\% RH}{75,3\% - 0\%}, \quad (4.5.3)$$

где $V_{out} @ 75,3\% RH$ – выходное напряжение при влажности 75,3%, В;

$V_{out} @ 0\% RH$ – выходное напряжение при влажности 0%, В.

Датчик влажности 5-вольтовый, поэтому для согласования с АЦП используется резистивный делитель 10/11 (1 МОм/ 1,1 МОм), и повторитель на операционном усилителе. Выход операционного усилителя подключен к входу АЦП – ADC0_IN. В качестве опорного напряжения для модуля АЦП выбирается внешний источник $V_{R+}=V_{eREF+}=3,3В$, $V_{R-}=0В$. Формула преобразования аналогового входного сигнала V_{in} в цифровой N_{ADC} выглядит следующим образом:

$$N_{ADC} = 4095 \cdot \frac{V_{in} - V_{R-}}{V_{R+} - V_{R-}} = 4095 \cdot \frac{V_{in}}{3,3}. \quad (4.5.4)$$

Таким образом, учитывая делитель и опорное напряжение, формула пересчета результата преобразования АЦП в значение влажности имеет вид:

$$RH = \frac{((\frac{N_{ADC}}{4095} \cdot 3,3 \cdot 1,1) - zero_offset)}{slope}. \quad (4.5.5)$$

Принципиальная схема подключения показана на рисунке 4.5.10.

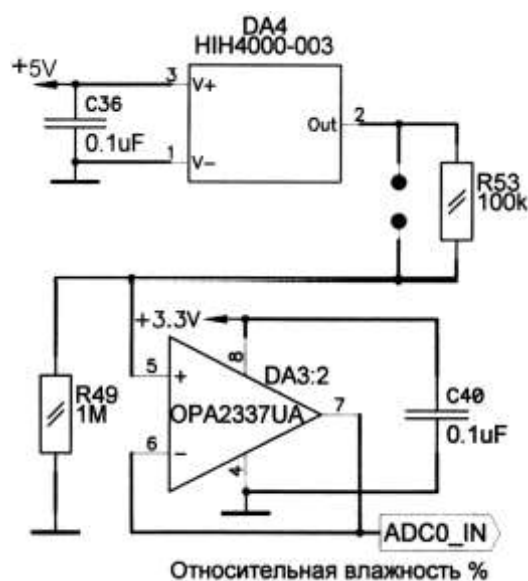


Рисунок 4.5.10 – Схема подключения датчика HIN4000-003

Сигналы и их подключение описаны в таблице 4.5.5.

Таблица 4.5.5 – Описание сигналов и назначение выводов датчика влажности HIN4000-003

Вывод HIN4000	Название	Подключение к цепи / [номер вывода]	Примечание
1	V-	GND	Подключение к шине 0 В
2	Out	DA3.2 / 5	Выход ОУ (DA3.2) подключен к входу АЦП ADC0_IN (MSP430, вывод 59)
3	V+	+5V	Напряжение питания

Размещение датчика на плате показано на рисунке 4.5.11.

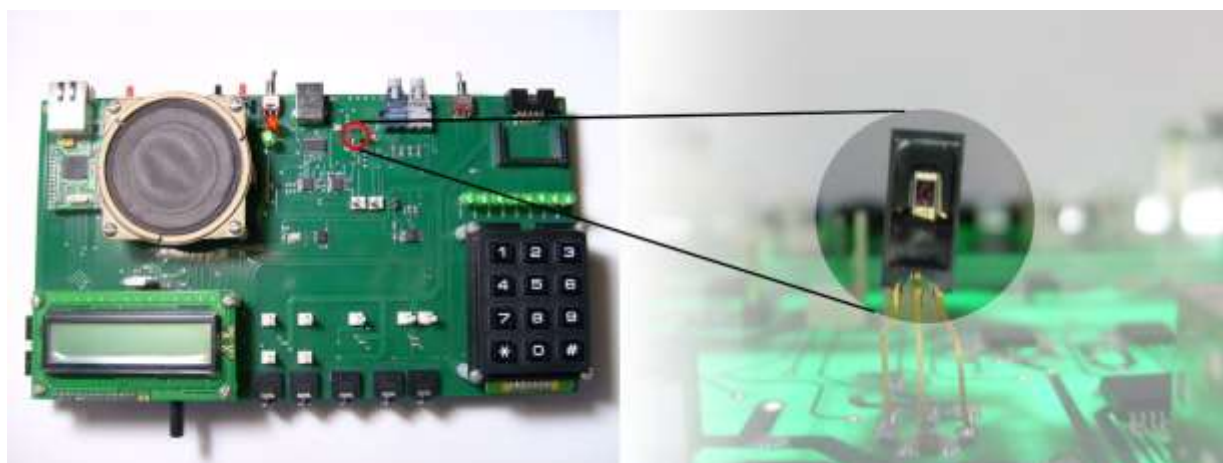


Рисунок 4.5.11 – Размещение датчика влажности на плате

4.5.2.2 Подключение датчика тока

В лабораторном стенде использован датчик тока INA139 с токовым выходом. Его характеристики приведены в таблице 4.5.6. На рисунке 4.5.12 показана схема с применением INA139, в которой кроме токового шунта требуется единственный внешний компонент – резистор R_L .

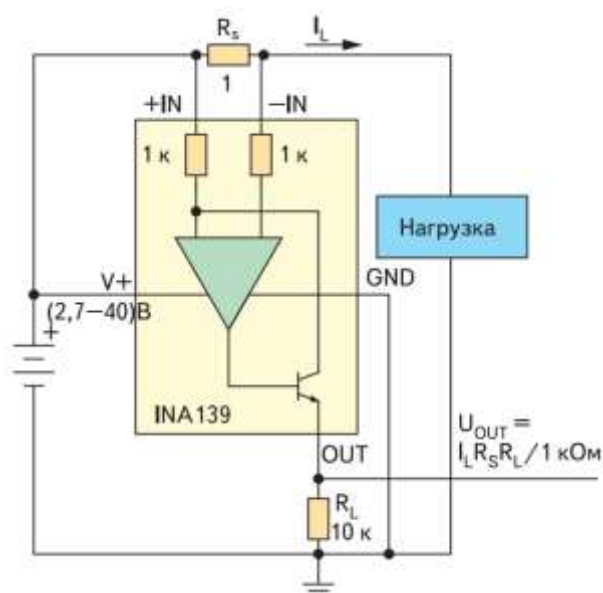


Рисунок 4.5.12 – Датчик тока в положительном полюсе с токовым выходом

Таблица 4.5.6 – Технические характеристики датчика тока INA139

Параметр	Значение
Диапазон входного напряжения, мВ	100...500
Синфазное входное напряжение, В	2,7...40
Коэффициент передачи, мА/В	1
Погрешность коэффициента передачи при +25°C, %, макс	1
Полоса пропускания -3 дБ, кГц	440
Напряжение питания, В	2,7...40
Ток потребления, мА, макс	0,125
Рабочая температура, °C	-40...85
Корпус	SOT23-5

В лабораторном стенде датчик тока INA139 измеряет ток потребления системы. Принципиальная схема подключения показана на рисунке 4.5.13.

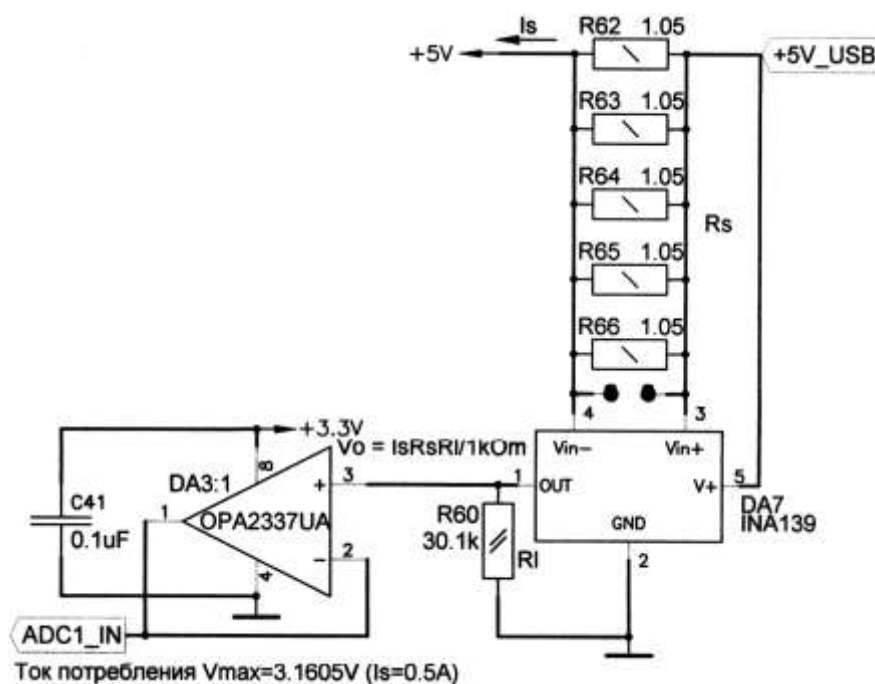


Рисунок 4.5.13 – Принципиальная схема подключения датчика INA139

Выходное напряжение датчика выражается формулой:

$$V_{out} = \frac{I_s \cdot R_s \cdot R_l}{1000}, \quad (4.5.6)$$

где I_s – ток потребления системы, А;

R_s – измерительное сопротивление, Ом ($R_s = 0,21$ Ом);

R_l – сопротивление нагрузки, Ом ($R_l = 30100$ Ом).

Подставив значения, получим формулу пересчета выходного напряжения датчика в ток потребления системы I_s , А:

$$I_s = \frac{V_{out}}{30,1 \cdot 0,21}. \quad (4.5.7)$$

Для согласования выхода датчика тока с входом АЦП используется повторитель на операционном усилителе. Выход операционного усилителя подключен к входу АЦП – ADC1_IN. В качестве опорного напряжения для модуля АЦП выбирается внешний источник $V_{R+}=V_{eREF+}=3,3$ В, $V_{R-}=0$ В. Для преобразования аналогового входного сигнала V_{in} в цифровой N_{ADC} служит формула 5.1. Таким образом, формула пересчета результата преобразования АЦП в значение тока потребления системы I_s имеет вид:

$$I_s = \frac{\frac{N_{ADC}}{4095} \cdot 3,3}{30,1 \cdot 0,21} = \frac{N_{ADC} \cdot 3,3}{4095 \cdot 30,1 \cdot 0,21}. \quad (4.5.8)$$

Размещение датчика на плате показано на рисунке 4.5.14.

Сигналы и их подключение описаны в таблице 4.5.7.

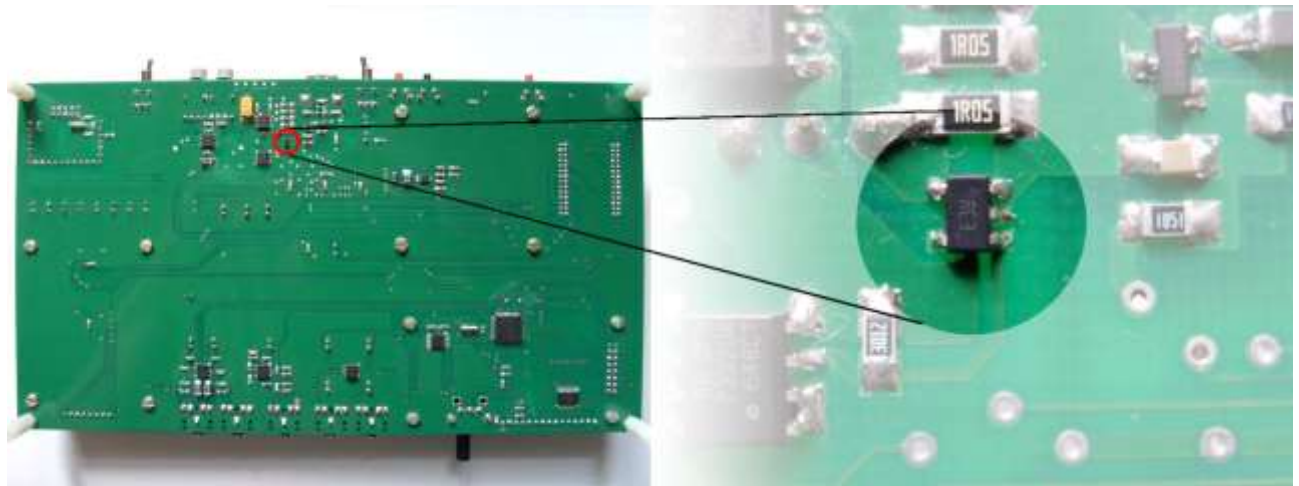


Рисунок 4.5.17 – Размещение датчика тока на плате

Таблица 4.5.7 – Описание сигналов и назначение выводов датчика INA139

Вывод INA139	Название	Подключение к цепи / [номер вывода]	Примечание
1	OUT	DA3.1 / 3	Выход ОУ (DA3.1) подключен к входу АЦП ADC1_IN (MSP430, вывод 60)
2	GND	GND	Подключение к шине 0 В
3	Vin+	R62..R66	Подключение к измерительному сопротивлению $R_s = 1,05 \text{ Ом} / 5 = 0,21 \text{ Ом} + 0,1\%$
4	Vin-	R62..R66	
5	V+	+5V_USB	Напряжение питания

4.5.3 Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс-контроль знаний по принципам функционирования модуля АЦП, входящего в состав микроконтроллера MSP430F1611. При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствие с индивидуальным заданием (таблица 5.8).

Задание: разработать в среде программирования IAR Embedded Workbench программу на языке C для измерения значений влажности и тока потребления в соответствие с параметрами режима работы, приведенными в таблице 4.5.8.

Порядок выполнения задания:

- включить лабораторный макет
- запустить компилятор IAR Embedded Workbench.
- создать пустой проект.
- создать файл ресурса для кода программы и подключить его к проекту.
- ввести код исходного модуля программы для считывания данных с модуля АЦП.
- выполнить компиляцию исходного модуля программы и устранить ошибки, полученные на данном этапе.
- настроить параметры программатора.
- создать загрузочный модуль программы и выполнить программирование микроконтроллера.

– проверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.

В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы.

Пример функций для работы с датчиками

```
//-----
const float HIH_zero_offset = 0.958; // параметр "начальное смещение"
датчика влажности, В
const float HIH_slope = 0.03068; // параметр "угол наклона датчика", В / %RH
const float HIH_ion = 3.3; // опорное напряжение, В
const float HIH_divisor = 1.1; // коэффициент резистивного делителя
const float INA_RS = 0.21; // измерительное сопротивление, Ом
const float INA_RL = 30.1; // сопротивление нагрузки, Ом
//-----

// Получить значение относительной влажности, %RH
float HIH_get_hum()
{
    P6SEL |= BIT0; // выбор для ножки P6.0 функции АЦП ADC0, к
    которому подключен датчик влажности
    ADC12CTL1 = SHP + CSTARTADD_0; // таймер выборки и стартовый адрес
    преобразования - ADC12MEM0
    // выбор опорного напряжения - Vr+ = VREF+ = 3.3В, Vr- = AVss = 0В
    // и входного канала ADC0 для ячейки памяти ADC12MEM0
    ADC12MCTL0 = SREF_3 + INCH_0;
    ADC12CTL0 = ADC12ON; // включение АЦП
    ADC12CTL0 |= ENC; // преобразование разрешено
    ADC12CTL0 |= ADC12SC; // запуск преобразования
    while ((ADC12IFG & BIT0) == 0); // ожидание результата преобразования
    // пересчет результата преобразования АЦП в значение влажности
    // с учетом делителя и опорного напряжения
    float rh = (((ADC12MEM0/4095.0) * HIH_ion * HIH_divisor) -
    HIH_zero_offset) / HIH_slope;
    ADC12CTL0 = 0; // выключение АЦП
    return rh;
}

//-----
// Получить значение тока потребления системы, А
float INA_get_curr()
{
    P6SEL |= BIT1; // выбор АЦП ADC1, к которому подключен датчик тока
    ADC12CTL1 = SHP + CSTARTADD_1; // таймер выборки и стартовый адрес
    преобразования - ADC12MEM1
    // выбор опорного напряжения - Vr+ = VREF+ = 3.3В, Vr- = AVss = 0В
    // и входного канала ADC1 для ячейки памяти ADC12MEM1
    ADC12MCTL1 = SREF_3 + INCH_1;
    ADC12CTL0 = ADC12ON; // включение АЦП
    ADC12CTL0 |= ENC; // преобразование разрешено
```

```

ADC12CTL0 |= ADC12SC;    // запуск преобразования
while ((ADC12IFG & BIT1)==0); // ожидание результата преобразования АЦП
ADC1

// пересчет результата преобразования АЦП в значение тока потребления
системы
// с учетом измерительного сопротивления и сопротивления нагрузки:
float curr = (ADC12MEM1*3.3) / (4095.0 * INA_RS * INA_RL);

ADC12CTL0 = 0;          // выключение АЦП
return curr;
}
//-----

```

Таблица 4.5.8 – Варианты индивидуальных заданий

№ п.п.	Задание
1	Разработать программу, выполняющую измерение относительной влажности в режиме одиночного преобразования (делитель частоты равен 2) и отображающую результат измерений на ЖКИ.
2	Разработать программу, выполняющую измерение относительной влажности в режиме непрерывного преобразования (делитель частоты равен 8) и отображающую результат измерений на ЖКИ.
3	Разработать программу, выполняющую измерение относительной влажности в режиме одиночного преобразования (делитель частоты равен 4) и отображающую результат измерений на ЖКИ.
4	Разработать программу, выполняющую измерение относительной влажности в режиме непрерывного преобразования (делитель частоты равен 5) и отображающую результат измерений на ЖКИ.
5	Разработать программу, выполняющую измерение потребляемого тока в режиме непрерывного преобразования (делитель частоты равен 6) и отображающую результат измерений на ЖКИ.
6	Разработать программу, выполняющую измерение потребляемого тока в режиме одиночного преобразования (делитель частоты равен 7) и отображающую результат измерений на ЖКИ.
7	Разработать программу, выполняющую измерение потребляемого тока в режиме непрерывного преобразования (делитель частоты равен 4) и отображающую результат измерений на ЖКИ.
8	Разработать программу, выполняющую измерение потребляемого тока в режиме непрерывного преобразования (делитель частоты равен 1) и отображающую результат измерений на ЖКИ.

4.5.4 Содержание отчета

В отчете необходимо привести следующее:

- характеристики лабораторной вычислительной системы;
- исходный модуль разработанной программы;
- анализ полученных результатов и краткие выводы по работе, в которых необходимо отразить особенности использования встроенного в микроконтроллер модуля АЦП для измерения аналоговых сигналов.

4.5.5 Контрольные вопросы и задания

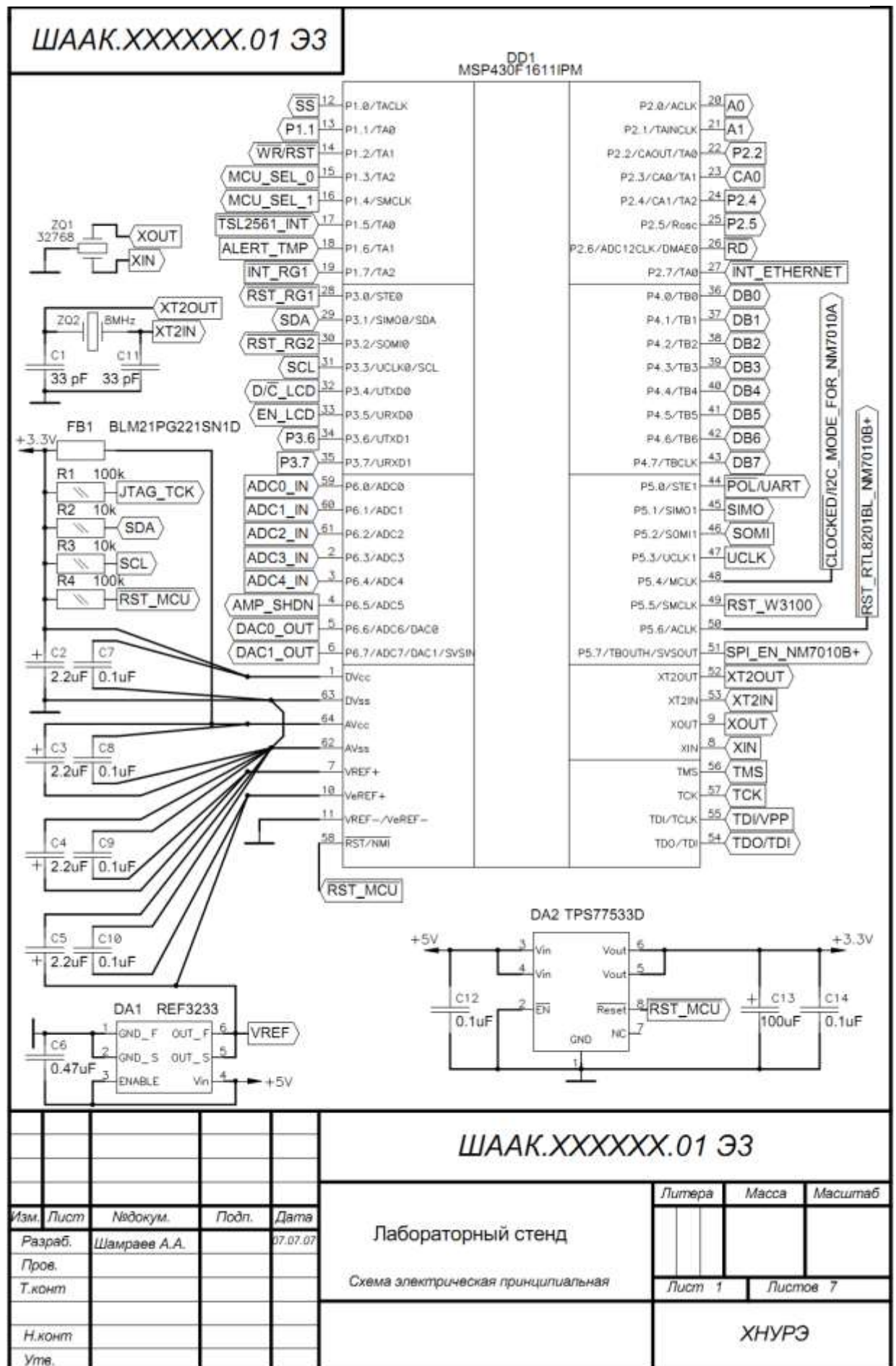
1. Поясните принцип работы встроенного в микроконтроллер MSP430F1611 12-разрядного АЦП.
2. Перечислите основные управляющие регистры АЦП, встроенного в микроконтроллер MSP430F1611, и поясните их функции.
3. Поясните принцип измерения температуры с помощью интегрированного датчика температуры.
4. Какие типы датчиков влажности вы знаете?
5. Каким образом осуществляется согласование уровней напряжения на выходе датчика влажности и входе АЦП?
6. Приведите основные характеристики датчика тока INA139.
7. Запишите формулы расчета значений относительной влажности и потребляемого тока по результатам АЦП преобразований.

ПЕРЕЧЕНЬ ССЫЛОК

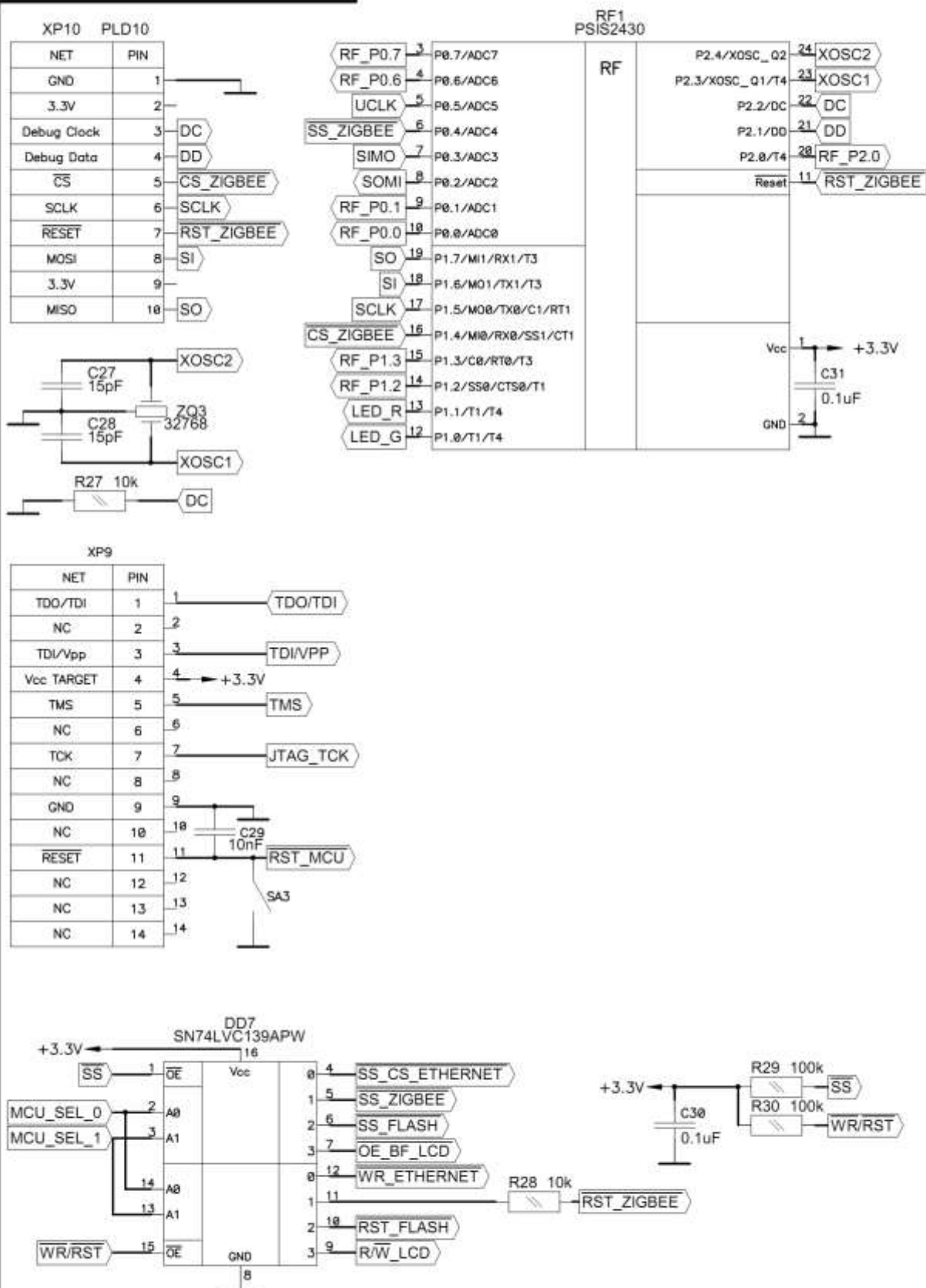
1. Семейство микроконтроллеров MSP430x1xx. Руководство пользователя: Пер. с англ.-М.: Серия «Библиотека Компэла». ЗАО «Компэл», 2004.-368с.
2. Семейство микроконтроллеров MSP430. Рекомендации по применению: Пер. с англ.-М.: Серия «Библиотека Компэла». ЗАО «Компэл», 2005.-544с.
3. Семейство микроконтроллеров MSP430x4xx. Руководство пользователя: Пер. с англ.-М.: Серия «Библиотека Компэла». ЗАО «Компэл», 2005.-416с.
4. «MSP 430 IAR C/C++ Compiler Reference Guide» файл EW430_COMPILERREFERENCE.PDF.
5. Бойт К. Цифровая электроника [Текст] / К. Бойт. – М.: Техносфера, 2007. – 472с.
6. Титце У. Полупроводниковая схемотехника: в 2т. – Т1 [Текст] / У. Титце, К. Шенк. – М.: Додэка-XXI, 2008. – 832с.
7. Титце У. Полупроводниковая схемотехника: в 2т. – Т2 [Текст] / У. Титце, К. Шенк. – М.: Додэка-XXI, 2008. – 942с.
8. Джексон Р.Г. Новейшие датчики [Текст] / Р.Г. Джексон. – М.: Техносфера, 2007. – 384с.
9. Хоровиц П. Искусство схемотехники [Текст] / П. Хоровиц, У. Хилл. – М.: Мир, 2001. – 704с.
10. Джонс М.Х. Электроника – практический курс [Текст] / М.Х. Джонс. – М.: Постмаркет, 1999. – 528с.
11. Шпак Ю.А. Программирование на языке С для AVR и PIC микроконтроллеров [Текст] / Ю.А. Шпак. – К.: МК-Прес, 2006. – 400с.

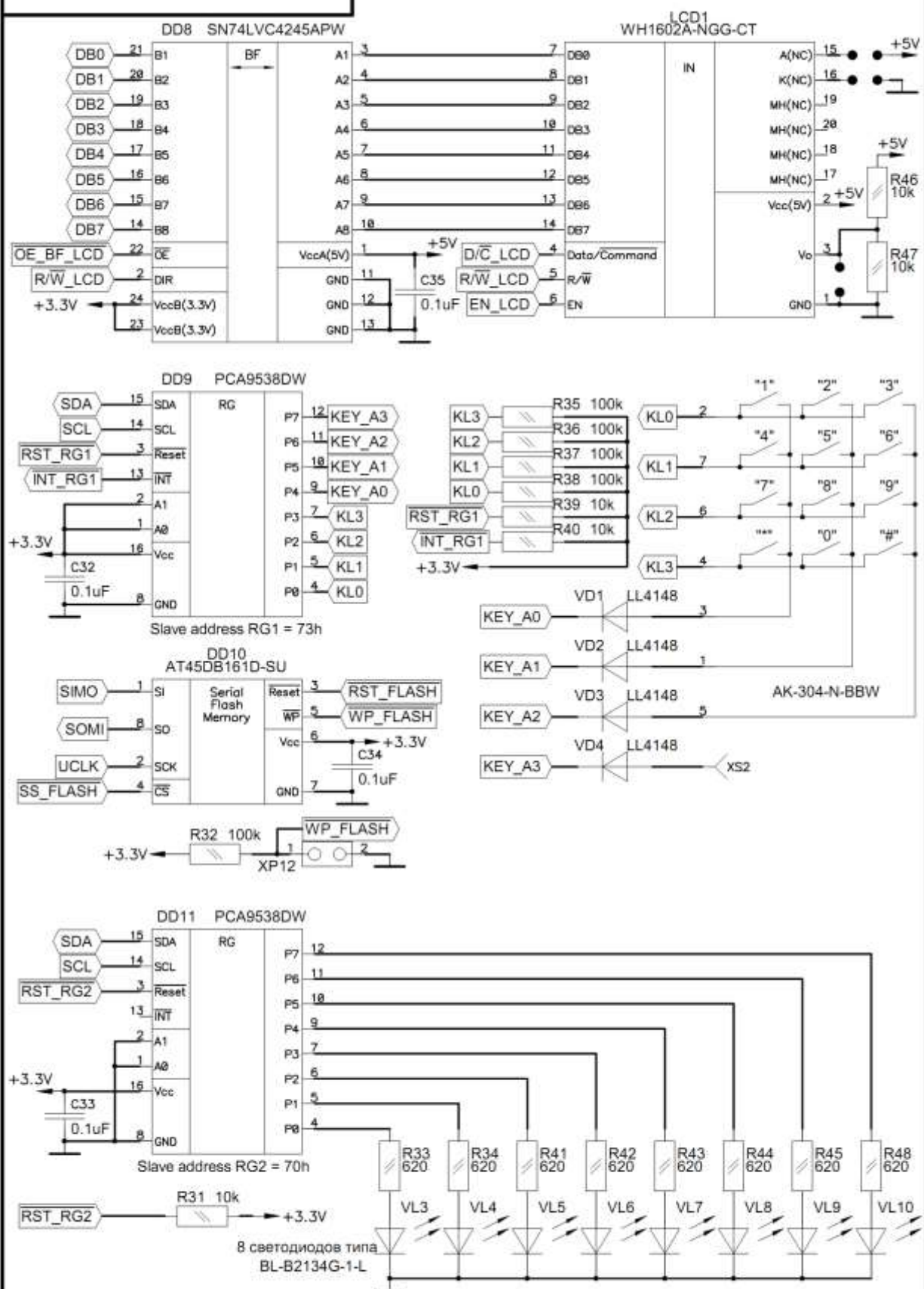
ПРИЛОЖЕНИЕ А

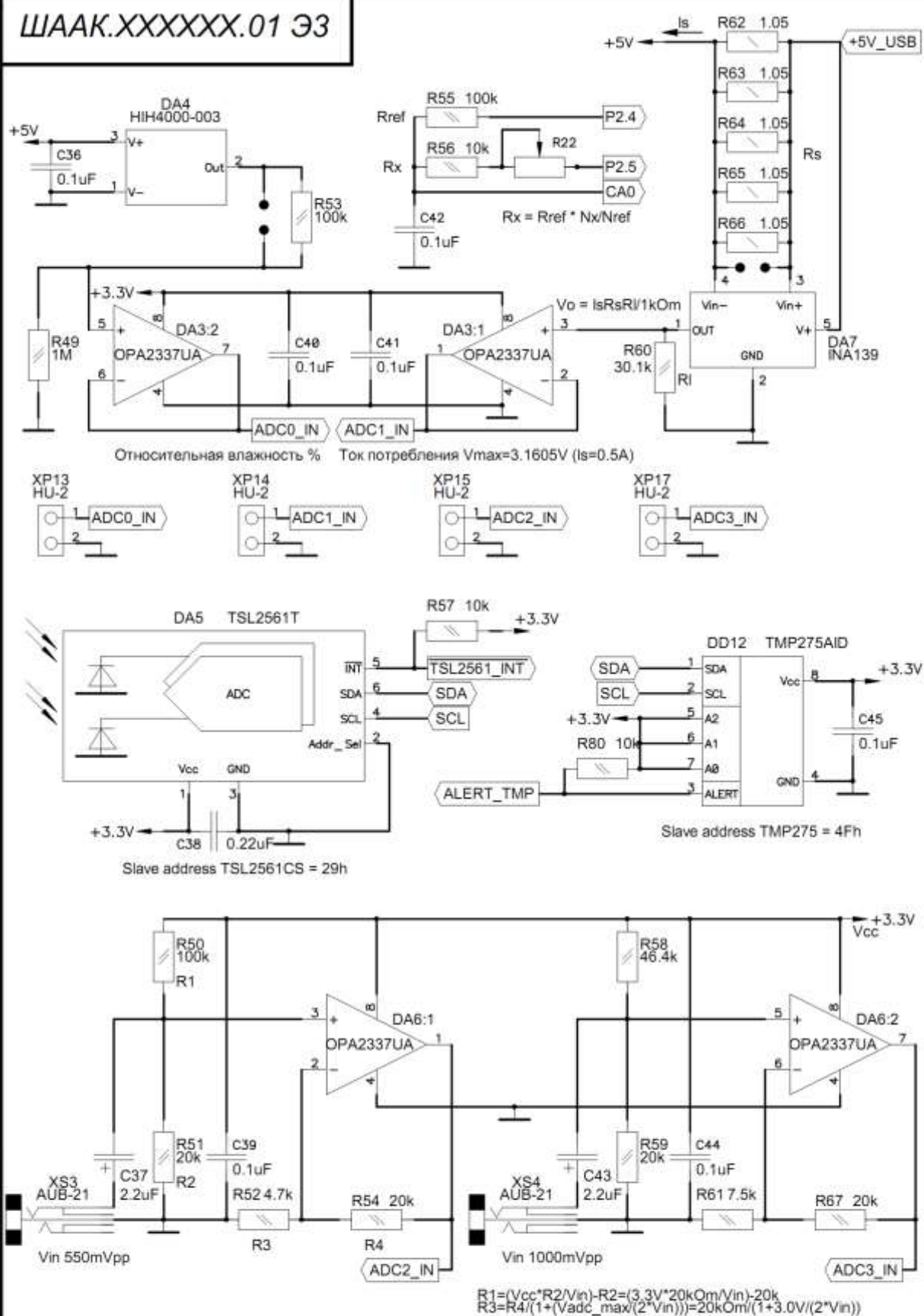
А.1 Лабораторный стенд. Схема электрическая принципиальная

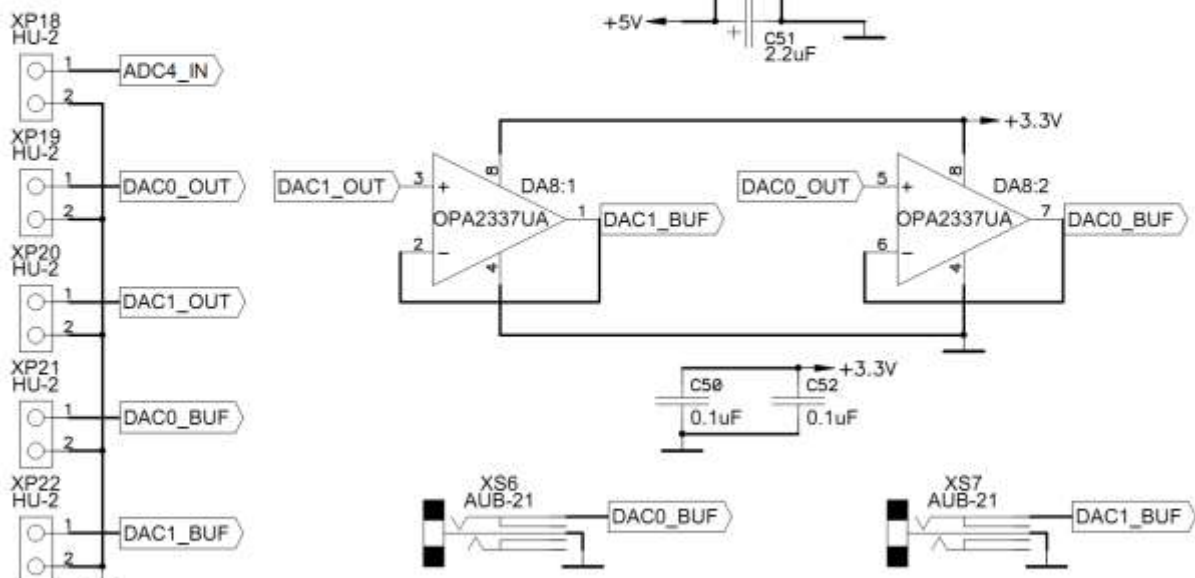
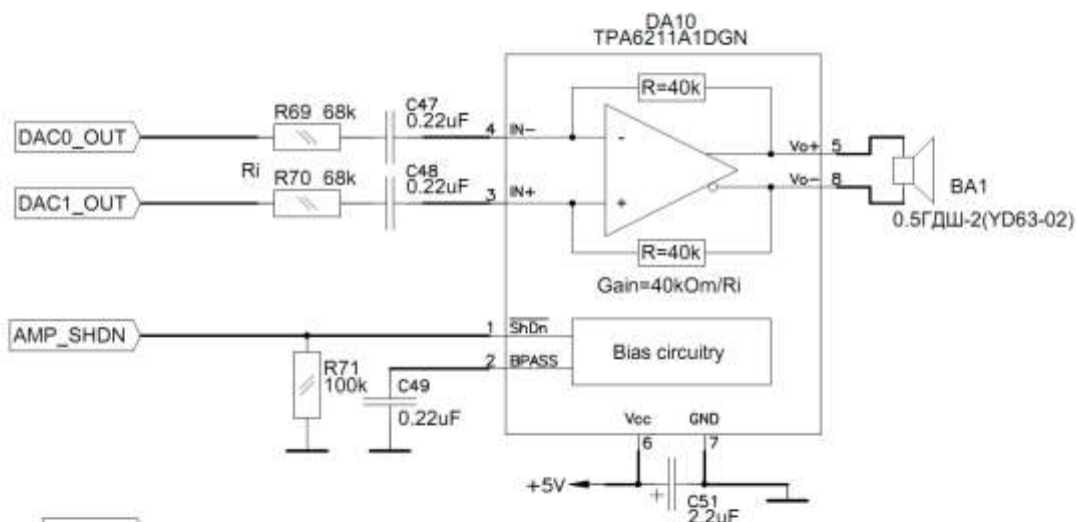
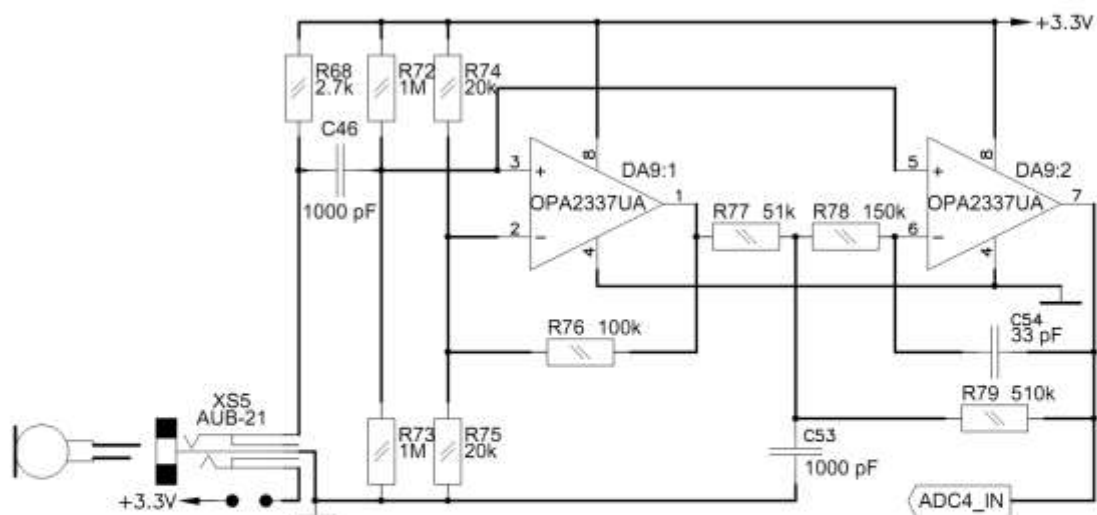












Изм.	Лист	Надокум.	Подп.	Дата

ПРИЛОЖЕНИЕ Б

Б.1 Параметры датчиков НН-4000-003

Таблица Б.1 – Параметры датчиков

Номер датчика	$V_{out} _{75,3\%RH}$, В	$V_{out} _{0\%RH}$, В	zero_offset, В
83	2,917	0,708	0,708
85	2,913	0,7129	0,7129
100	2,922	0,7056	0,7056
102	2,954	0,7129	0,7129
128	3,067	0,7368	0,7368
133	2,957	0,7129	0,7129
135	2,922	0,7031	0,7031
136	2,937	0,7056	0,7056
270	2,942	0,791	0,791
271	2,903	0,7617	0,7617
371	3,101	0,781	0,781

