БЕЛОРУССКИЙ НАЦИОНАЛЬНЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
Факультет информационных технологий и робототехники
Кафедра «Электропривод и автоматизация промышленных установок и технологических комплексов»

Согласовано
Заведующий кафедрой

_____С. А. Павлюковец

«_____» _____2023 г.

Согласовано
Декан факультета

_____А. М. Авсиевич

«_____» _____2023 г.

ЭЛЕКТРОННЫЙ УЧЕБНО-МЕТОДИЧЕСКИЙ КОМПЛЕКС ПО УЧЕБНОЙ ДИСЦИПЛИНЕ МИКРОКОНТРОЛЛЕРЫ В АВТОМАТИЗАЦИИ

для специальности 7-06-0713-04 «Автоматизация»

Составитель: О.Ф. Опейко

Рассмотрено и утверждено
На заседании совета факультета
информационных технологий и робототехники

“_03_” 05 2023 г., протокол № 8

Минск ◊ БНТУ ◊ 2024 г.

Перечень материалов

1. Теоретический раздел.
2. Практический раздел
3. Раздел контроля знаний
4. Вспомогательный раздел.

Пояснительная записка

Целью электронного учебно-методического комплекса (ЭУМК) является формирование знаний, умений и навыков использования микроконтроллеров в автоматизации.

Разработанный ЭУМК способствует формированию высококвалифицированных специалистов в области автоматизации, обладающих системным мышлением и умением применять в автоматизации новые технические средства и методы, базируясь на теоретических знаниях.

Учебный материал соответствует учебной рабочей программе дисциплины «Микроконтроллеры в автоматизации» для магистрантов, обучающихся по специальности 7-06-0713-04 «Автоматизация».

Особенностью структурирования и подачи учебного материала является наличие теоретического раздела (курса лекций), практического раздела (инструкций к выполнению лабораторных работ, заданий для практических занятий), и, во вспомогательном разделе, учебной рабочей программы дисциплины. Курс лекций содержит принципы построения центрального процессорного устройства (ядра) и принципы преобразования информации периферийными устройствами, а также взаимодействия ядра и периферии. Курс лекций содержит основы программирования микроконтроллера на языке С. Курс лекций сопровождается примерами применения микроконтроллеров в автоматизации. В конце разделов есть контрольные вопросы для самопроверки. Лабораторные работы и практические занятия ориентированы на изучение принципов функционирования устройств ввода и вывода информации, приема и передачи информации и на применение микроконтроллеров в задачах автоматизации.

Рекомендации по организации работы с ЭУМК. Электронный учебно-методический комплекс предназначен для магистрантов очной и заочной формы получения высшего образования, преподавателей дисциплины «Микроконтроллеры в автоматизации» для проведения лекционных, лабораторных, практических занятий, а также для самостоятельной работы магистрантов. ЭУМК может быть использован студентами при выполнении курсовых работ и в дипломном проектировании.

ОГЛАВЛЕНИЕ

1 ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ 6
1. 1. Архитектура и принципы функционирования микроконтроллеров 6
1.1.1. Основные понятия и определения 6
1.1.2. История создания микроконтроллеров 8
1.1.3. Назначение, области применения и технико-экономическая эффективность микроконтроллеров в автоматизации 9
1.1.4. Структура и принцип действия микроконтроллера 10
1.1.5 Система команд MSP430 11
1.2. Алгоритмы и программирование микроконтроллеров. 17
1.2.1. Алгоритмы реального времени 17
1.2.2 Интегрированная среда разработки проектов автоматизации 19
1.2.3. Применение языка *C* для программирования микроконтроллеров 19
1.3.1. Программируемый таймер и его применение 36
1.3.2. Аналого- цифровое и цифро-аналоговое преобразования. 38
1.3.3. Последовательный интерфейс в микроконтроллерах 42
1.4. Развитие архитектуры микроконтроллеров 47
1.4.1. Направления развития архитектуры 47
1.4.2. Микроконтроллеры архитектуры *ARM* 50
1.4.3. Микроконтроллеры управления электроприводами 52
1.5. Применение микроконтроллеров в автоматизации 54
1.5.1. Цифровые фильтры, их передаточные функции, алгоритмы и программы расчета выходной величины 55
1.5.2. Синтез цифрового управления 58
1.5.3. Методы расчета параметров ПИД- регуляторов. Алгоритмы и программы расчета выходных сигналов ПИД регуляторов 64
1.6. Алгоритмы интеллектуального управления 68
1.7. Применение микроконтроллеров для логического управления и обеспечения безопасности 74
2. ПРАКТИЧЕСКИЙ РАЗДЕЛ 84
Лабораторная работа №1. Архитектура 16 разрядного микроконтроллера MSP430 85
1.1. Цель работы 85
1.2. Архитектура микроконтроллера 85
1.2. Организация памяти и внутренние регистры процессора 89
1.4. Система команд и способы адресации 90
1.5. Среда разработки проекта 92
1.6. Порядок выполнения работы 92
1.7. Варианты заданий 92
1.8. Содержание отчета 93
1.9. Контрольные вопросы 93
2 Лабораторная работа №2. Создание проекта автоматизации в интегрированной среде Code Composer Studio™ (CCS) для микроконтроллеров MSP430 95
2.1. Цель работы. 95
2.2. Устройство eZ430RF2500. 95
2.3. Использование программного обеспечения Code Composer Studio™ 95
2.5. Порядок выполнения работы 97
2.6. Содержание отчета 97
2.7. Варианты задач 97
2.8. Контрольные вопросы 97

3 Лабораторная работа №3. Алгоритмы и программы автоматизации для микроконтроллеров 98
3.1. Цель работы. 98
3.2. Разработка алгоритма 98
3.3. Разработка программы 98
3.4 Порядок выполнения работы 99
3.5. Содержание отчета 99
3.6 Варианты заданий 99
3.7. Контрольные вопросы 100
4.Лабораторная работа №4. Разработка алгоритмов и программ с использованием программируемого таймера 100
4.1. Цель работы 100
4.2. Разработка алгоритма 100
4.3. Таймер TA микроконтроллера MSP430 101
4.4. Построение программы с использованием программируемого таймера 105
4.5. Порядок выполнения работы 108
4.6. Варианты заданий 108
4.7. Содержание отчета 109
4.8. Контрольные вопросы 109
5.Лабораторная работа №5. Разработка алгоритмов и программ микроконтроллера MSP430 с использованием аналого-цифрового преобразователя ADC10 110
5.1. Цель работы 110
5.2. Аналого-цифровой преобразователь ADC10 микроконтроллера MSP430 110
5.4. Порядок выполнения работы 115
5.5. Варианты заданий 116
5.6 Содержание отчета 116
5.7. Контрольные вопросы 116
6. Лабораторная работа №6. Алгоритмы и программы расчета выходных величин регуляторов, интегрирующих и дифференцирующих звеньев 117
6.1. Цель работы 117
6.3. Порядок выполнения работы 118
6.4. Варианты заданий 119
6.5 Содержание отчета 119
6.6. Контрольные вопросы 119
7. Лабораторная работа №7. Использование последовательного интерфейса для приема и передачи данных 121
7.1. Цель работы 121
7.2. Последовательный интерфейс микроконтроллера MSP430F2274 121
7.3. Пример программы 124
7.4. Порядок выполнения работы 125
7.5 .Варианты заданий 125
7.6 Содержание отчета 125
7.7. Контрольные вопросы 125
8.Лабораторная работа №8. Изучение архитектуры микроконтроллера TMS320F28335 для частотного управления электродвигателями 126
8.1. Цель работы 126
8.2. Функциональная схема микроконтроллера 126
Рисунок 8.1 – Функциональная схема микроконтроллера 128
8.3. Порядок выполнения работы 129
8.4. Варианты заданий 129

8.5. Содержание отчета 130
8.6 Контрольные вопросы 130
2.5. Задания для практических занятий 134
3 РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ 135
4 ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ 137
Список литературы 145

1 ТЕОРЕТИЧЕСКИЙ РАЗДЕЛ

1. 1. Архитектура и принципы функционирования микроконтроллеров

1.1.1. Основные понятия и определения

Процессором (микропроцессором) называется устройство преобразования информации, представленной в двоичном коде. Процессор работоспособен только в составе *вычислительного устройства* (рисунок 1.1). Это означает, что для работы процессора необходимы источник питания и генератор тактовых импульсов, оперативное запоминающее устройство (ОЗУ, RAM) и постоянное запоминающее устройство (ПЗУ, ROM) и устройства ввода и вывода информации.

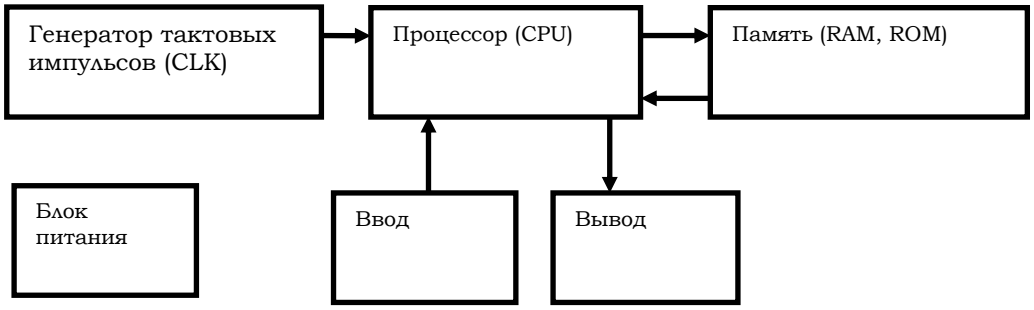


Рисунок 1.1 – Вычислительное устройство

Микроконтроллером называется вычислительное устройство в одной интегральной микросхеме (в одном кристалле, чипе), предназначенное для *управления*.

Цифровой сигнальный процессор (*Digital Signal Processor*) — это вычислительное устройство, которое выполняет преобразование сигналов, представленных в двоичном коде и предназначен для применения, в частности, в технике связи.

Микроконтроллер и цифровой сигнальный процессор предназначены для выполнения близких, а то и совпадающих задач преобразования сигналов, в частности, в системе управления.

Первые микроконтроллеры и цифровые сигнальные процессоры (Digital Signal Processors) появились в 1980-е годы.

Микроконтроллер может выполнять функцию как логического управления, так и функцию устройства управления в замкнутой системе с обратными связями на основе *программной реализации* функций управления.

Программная реализация функций управления имеет значительные преимущества.

1. Гибкость управления.
2. Возможность тиражирования программ управления.
3. Оптимальное и адаптивное управление системой.
4. Применение методов искусственного интеллекта

В 1953 функционировали 100 ЭВМ, а в 1958- 64 появилась ЭВМ PDP-1 (120 000 $) фирмы DEC. Изобретение в 1958 интегральной микросхемы и в 1970-е годы – микропроцессоров положило начало вычислительным средствам массового применения. В 1980-90 г, фирма INTEL начала производить микропроцессоры и микроконтроллеры.

Начиная от 1980-х годов и до настоящего времени все процессоры выполняются в виде однокристальных *микропроцессоров* или как составная часть однокристального *микроконтроллера либо сигнального процессора (Digital Signal Processor, DSP)* как его *ядро,* то есть *центральное процессорное устройство* (ЦПУ). На рисунке 1.2 показана функциональная схема ЦПУ.

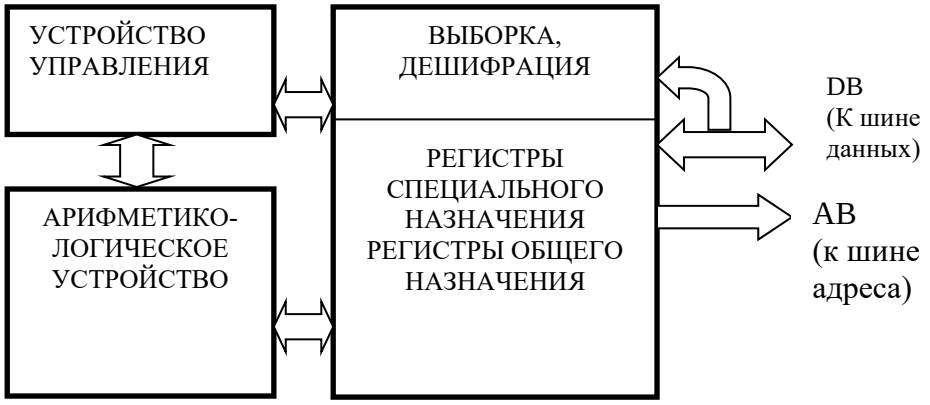


Рисунок 1.2 – Центральное процессорное устройство (ЦПУ).

Принцип действия ЦПУ заключается в выполнении *командного цикла. Командным циклом* называется интервал времени, за который выполняется одна команда. Выполнение командного цикла показано на рисунке 1.3 и начинается с выборки команды из памяти команд по шине данных.

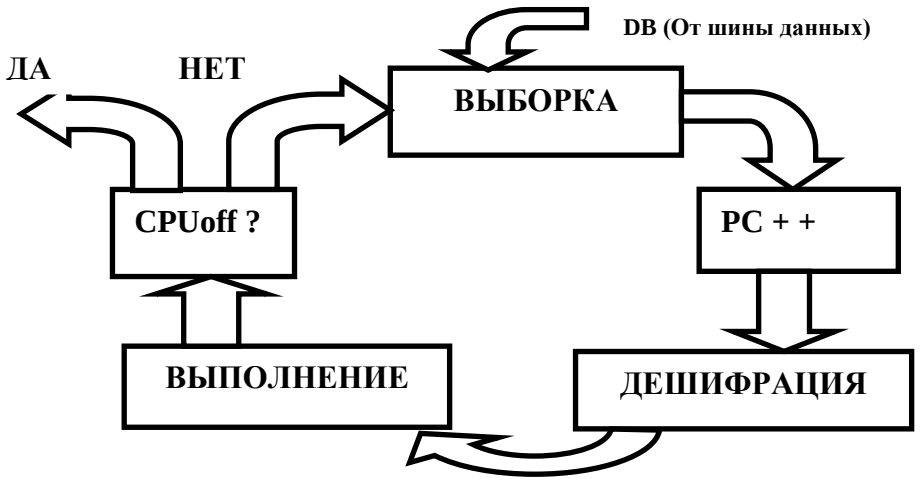


Рисунок 1.3 – Командный цикл

После сброса или подключения электроэнергии регистр R0, который является программным счетчиком PC, установлен на начальный адрес памяти программ, например, PC = 0x8000. Поэтому первый командный цикл начинается с выборки первой команды программы, расположенной по адресу 0x8000. После этого программный счетчик инкрементируется. Далее происходит дешифрация команды, а затем – выполнение. Если не установлен бит **CPUoff,** то выполняется следующий командный цикл.

Архитектурой микроконтроллера называется его структура в совокупности с *системой команд процессора*.

Структура микроконтроллера состоит из *ядра и периферии*. Ядром является центральное процессорное устройство (ЦПУ, CPU – Central Processing Unit). Таким образом, все устройства микроконтроллера, кроме ЦПУ, составляют периферию. Примеры архитектур микроконтроллеров рассмотрены в литературе [1] – [6].

1.1.2.История создания микроконтроллеров

Предпосылки создания и развития микроконтроллеров сложились к середине двадцатого века. В 1937 году разработан принцип построения машины Тьюринга для преобразования информации, представленной в двоичном коде. В 1937 году Джон Атанасов в США разработал ЭВМ (электронную вычислительную машину).

В 1945 Нейман (Von Neumann) разработал архитектуру вычислителя – машины Неймана, которая стала основой современных процессоров. Изобретение транзистора в 1947 году в *Bell Laboratory*, США привело к созданию в дальнейшем, в 1950-е годы, транзисторных вычислительных машин взамен ламповых.

Норберт Винер написал в 1947 году книгу «Кибернетика или управление и связь в животном и машине», что явилось началом развития кибернетики, науки об управлении и информационных процессах. Техническая кибернетика, как наука о синтезе управления в технических системах, опирается на вычислительную технику. Так, в настоящее время одним из основных средств управления являются *микроконтроллеры*.

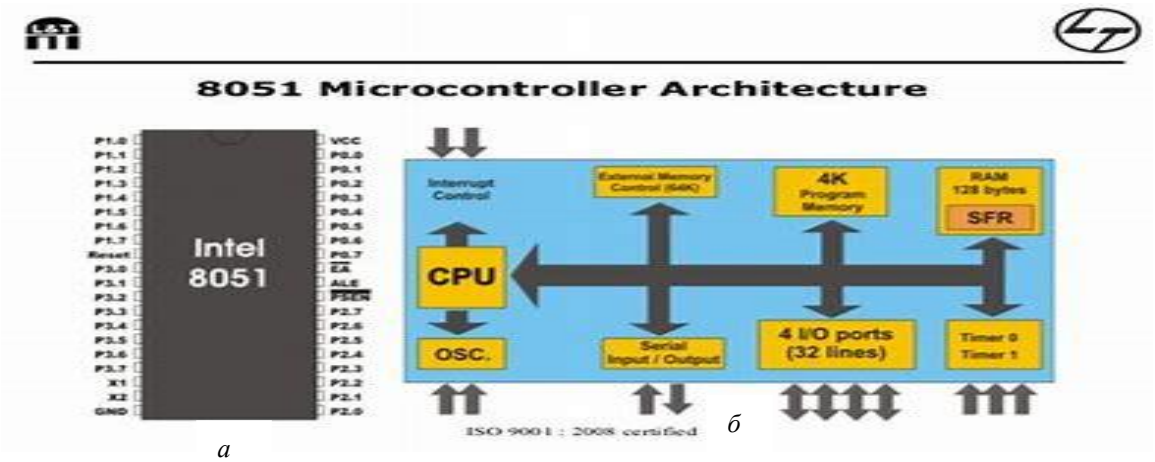
В 1953 функционировали 100 ЭВМ, а в 1958- 64 появилась ЭВМ PDP-1 фирмы DEC. Изобретение в 1958 интегральной микросхемы и в 1970-е годы – микропроцессоров положило начало вычислительным средствам массового применения. В 1980-90 г, фирма INTEL начала производить микропроцессоры и микроконтроллеры.

На рисунке 1.4 показан внешний вид микросхемы и структура 8-разрядного микроконтроллера *INTEL* 8051. Микроконтроллер является законченным вычислительным устройством в одной микросхеме (в одном кристалле, в одном чипе). Здесь *OSC* – источник тактового сигнала *External Memory Controller* – контроллер внешней памяти, который позволяет подключить к микросхеме внешнюю память; 4K program Memory – память программ 4*K; RAM* 128 bytes - оперативная память данных 128 байт; *SFR* – регистры специальных функций, которые необходимы для настройки режимов работы микросхемы и для хранения служебной информации; *Serial Input/output* - ввод и вывод через последовательные порты, 4 *I/O ports* (32 *lines*) - 4 порта ввода-вывода по одному байту каждый, всего 32 бита, *Timer* 0, *Timer* 1 – два таймера.

Микроконтроллер имеет *Гарвардскую архитектуру,* это означает, что память программ и память данных имеют отдельные адресные пространства, причем программный счетчик *PC* хранит текущий адрес команды программы, а указатель на данные (DP, data pointer) хранит адрес данных.

Архитектура микроконтроллера INTEL 8051 нашла применение и развитие во многих типах микроконтроллеров. Например, фирма ATMEL в

микроконтроллерах AVR применяет ядро INTEL 8051 в сочетании с расширенной периферией (рисунок 1.5).



а *б*

Рисунок 1.4 – Общий вид (а) и структура (б) 8-разрядного микроконтроллера INTEL 8051.

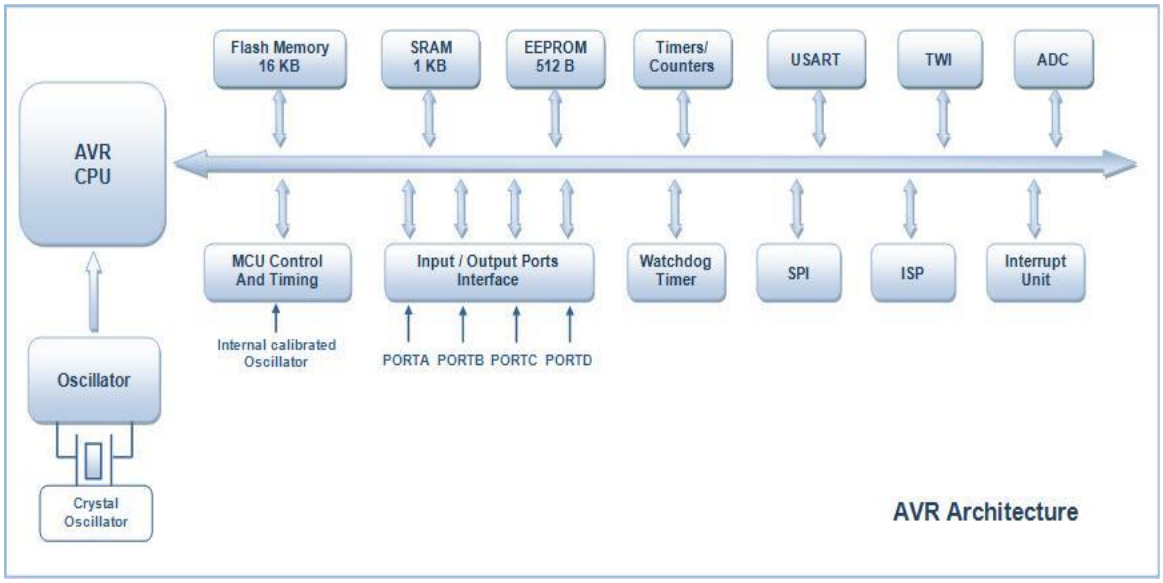


Рисунок 1.5 – Структура 8-разрядного микроконтроллера *AVR* фирмы *ATMEL*.

1.1.3. Назначение, области применения и технико-экономическая эффективность микроконтроллеров в автоматизации

В системах автоматизации на разных уровнях иерархии применяются средства вычислительной техники различных уровней. Это промышленные компьютеры верхнего уровня для управления производственным процессом в целом, программируемые логические контроллеры (ПЛК), каждый из которых управляет определенной производственной задачей.

Микроконтроллеры в основном предназначены для *встроенных* применений. Это определяет широкую применимость микроконтроллеров в промышленности, транспортных средствах, средствах связи, в бытовой технике и для биомедицинских целей. Микроконтроллеры, в частности, применяются внутри ПЛК, а также в составе датчиков технологических величин и для сбора информации о технологическом процессе.

Важнейшим направлением в применении микроконтроллеров является энергосбережение и ресурсосбережение, достигаемое за счет рационального логического управления (включения и отключения приводов рабочих механизмов). Обычно суммарная стоимость микроконтроллера, дополнительных датчиков, изготовления печатной платы и программирования микроконтроллера для подобных задач не велика, а экономия за счет рационального управления может быть значительной.

Микроконтроллеры, будучи составной частью системы автоматизации, и, в отличие от ПЛК, имея широкие возможности программирования, способны повысить уровень автоматизации до интеллектуального управления.

1.1.4. Структура и принцип действия микроконтроллера

Далее рассматривается структура и принцип действия 16 разрядного микроконтроллера *MSP*430*F*2274. Микроконтроллер, как показано на рисунках 1.1, 1.4, 1.5 содержит центральный процессор *(ядро*) и другие устройства, которые относятся к *периферии*: это запоминающие устройства, а также устройства ввода- вывода данных. Ядро соединено с периферийными устройствами *магистралью*. Микроконтроллер, как устройство управления, имеет развитую периферию для обеспечения ввода сигналов обратных связей и задающих сигналов и вывода сигналов управления. На рисунке 1.6 представлены внутренние регистры процессора. Всего микроконтроллер имеет 16 регистров *R*0, *R*1, …, *R*15 по 16 разрядов в каждом. К каждому регистру можно обращаться в программах на ассемблере по имени (регистровая адресация).

Рисунок 1.6 –Регистры ядра.

В ядре есть специальные регистры (регистры специальных функций,). Это программный счетчик ***PC* (*R*0),** указатель стека ***SP* (*R*1),** регистр состояния ***SR* (*R*2)** и генератор констант ***CG*1, *CG*2 (*R*2, *R*3).** Программный счетчик ***PC* (*R*0)** предназначен для генерации адреса команд. После включения питания и после сброса ***PC*** содержит значение 0x8000 начального адреса области хранения программ. Выполняя командные циклы, ***CPU*** инкрементирует программный счетчик, значение которого передается на шину адреса для чтения следующей команды. Указатель стека ***SP* (*R*1)** предназначен для организации стековой *адресации* в заданной области оперативной памяти данных. Регистр состояния

SR* (*R2*)** предназначен для хранения *флагов* ***C, Z, N, V состояния программы после выполнения логической либо арифметической операции. Флаги ***C, Z, N*** и ***V*** переноса, нуля, отрицательности и переполнения устанавливаются по результату работы арифметико-логического устройства (*ALU*). Регистр состояния представлен на рисунке 1.7. Кроме флагов, регистр состояния содержит биты управления: общее разрешение прерываний **GIE** (*General Interrupt Enable*), бит отключения процессора **CPUOFF** и биты управления осциллятором. На рисунке 7 показан регистр состояния ***SR,*** флаги выделены.

15,,,,,,9	8	7	6	5	4	3	2	1	0
-	**V**	SCG1	SCG0	OSC OFF	CPU OFF	GIE	**N**	**Z**	**C**

Рисунок1.7 – Регистр состояния ***SR***

1.1.5 Система команд MSP430

Микроконтроллер MSP430F22xx имеет CPU архитектуры RISC с сокращенным набором команд. Классическая архитектура, предложенная фон Нейманом для процессоров, характеризуется полным набором команд (CISC). Структура команды показана на рисунке 1.8 и является общепринятой для всех вычислителей, состоит из кода операции КО и адресной части А.

КО	А

Рисунок 1.8 - Структура команды

Адресная часть зависит от кода операции и способа адресации. Есть команды без адресной части, например *NOP* (нет операции). Обычно в микроконтроллерах применяется *непосредственная, регистровая, прямая и косвенная адресация.* В зависимости от кода операции команды бывают с одним либо двумя операндами в адресной части, в некоторых микроконтроллеров с тремя. Система команд микроконтроллера называется *ортогональной*, если в любой из команд для каждого операнда можно применить любой способ адресации. Например, система команд микроконтроллера MSP430 представлена в таблице 1.1 и является высоко ортогональной [7].

Система команд MSP430F22xx является сокращенной и насчитывает 27 команд ассемблера, в том числе команды с 2 и с одним операндом.

Регистровая адресация означает, что в команде на месте операнда указано имя регистра (*R*0…*R*15).

Непосредственная адресация означает, что в команде на месте операнда указано его числовое значение (предпочтительно в 16-й системе счисления).

Прямая адресация означает, что в команде на месте операнда указан его адрес (0200*h*…03*FFh*) из области памяти данных.

Косвенная адресация означает, что в команде на месте операнда указан регистр-хранитель адреса (указатель на адрес) операнда.

Каждая команда может быть применена к байтам (расширение .*b*) или 16-разрядным словам (расширение. w или без расширения).

Все команды перехода выполняются за 2 такта. Все команды с регистровой адресацией выполняются за 1 такт. Другие виды адресации требуют большего числа тактов.

Система команд содержит операции пересылки, арифметические и логические операции, а также операции сдвига, условных и безусловных переходов и обращения к подпрограммам. Мнемокод ассемблера для каждой команды соответствует коду операции, и далее записывается адресная часть.

Логические операции выполняются над байтами и 16-разрядными словами над одноименными битами в соответствии с таблицей истинности для каждой операции. Таблица 1.1 истинности представлены для операций И (конъюнкция ∩), ИЛИ (дизъюнкция U) и исключающее ИЛИ (XOR, ∀) соответственно.

Таблица 1.1 – Конъюнкция, дизъюнкция и исключающее ИЛИ (*XOR*)

И (конъюнкция)			ИЛИ (дизъюнкция)			исключающее ИЛИ (*XOR*)		
X_1	X_2	$X=X_1 \cap X_2$	X_1	X_2	$X=X_1 U X_2$	X_1	X_2	$X=X_1 \forall X_2$
0	0	0	0	0	0	0	0	0
0	1	0	0	1	1	0	1	1
1	0	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	0

Логические операции применяются для тестирования и маскирования байтов и слов данных. *Маскированием* называется выделение в слове данных определенных битов. Для тестирования байтов и слов данных применяются команды *BIT s, d* и *CMP s, d*, которые не влияют на содержимое *s* и *d*, но приводят к изменению флагов в регистре *SR (R2*). Команда *BIT s, d* выполняет конъюнкцию, как показано в таблице 1.2, и влияет на флаги *N, Z, C*.

В языке С приняты следующие знаки логических операций. Знак & означает конъюнкцию (логическую операцию И), знак | означает дизъюнкцию (логическую операцию ИЛИ), знак ~ означает отрицание, а знак ^ - исключающее ИЛИ.

Команды условных и безусловных переходов (см таблицу 1.1) позволяют переходить внутри программной единицы к выполнению команды, расположенной по указанному в команде адресу. Нарушается естественный порядок выполнения команд. Это позволяет реализовать ветвления и циклы алгоритма. В команде перехода адрес указывается символическим именем метки, которая также указывается перед операцией, к которой следует перейти. В качестве условия перехода в командах применяют состояния флагов регистра состояния (в *MSP*430 это *R2(SR)*).

Применение подпрограмм в микроконтроллерах. Подпрограммы используются для выполнения операций многократного применения, что дает ряд преимуществ: экономия памяти программ, структурирование программного обеспечения и рационализация труда программиста. Для

организации подпрограмм применяется *стековая адресация* и специальные команды, использующие стековую адресацию:

```
PUSH   d1          ;   запись из d1 в стек,   SP - 2
PUSH   d2          ;   запись из d2 в стек,   SP - 2
…
CALL  #adrS,       ;   PC в стек,   SP – 2,  #adrS → PC
…
POP   d2           ; из стека в  d2, SP + 2
POP   d1           ; из стека в  d1, SP + 2
…
#adrS  ;                    начало подпрогр.
              PUSH   d3          ;запись из d3 в стек
              PUSH   d4          ;запись из d4 в стек
……
               POP   d4           ; из стека в  d4
               POP   d3           ; из стека в  d3
RETURN                            ; старый адр → PC
```

Для стековой адресации в памяти данных микроконтроллера отводится специальная стековая область памяти, на которую указывает указатель стека ***SP.*** Указатель стека хранит адрес вершины стека, а именно адрес последней записи. По команде *PUSH d1* обращения к стеку происходит запись из регистра *d1* в стековую область памяти по адресу, на который в данный момент указывает ***SP.*** Одновременно указатель стека декрементируется на два, поскольку регистр *d1* 16-ти разрядный, следовательно, занимает два адреса. По команде *POP d1* обращения к стеку происходит чтение из стека в регистр *d*1 по адресу стека, на который в данный момент указывает ***SP.*** Одновременно указатель стека инкрементируется на два.

Пример 1.1. Обеспечить мигание светодиодов на выходе параллельного порта *P*1 с заданным периодом. Светодиоды, красный и зеленый, подсоединены к битам *P*1.0 и *P*1.1 порта соответственно. Применить подпрограмму для организации выдержки времени.

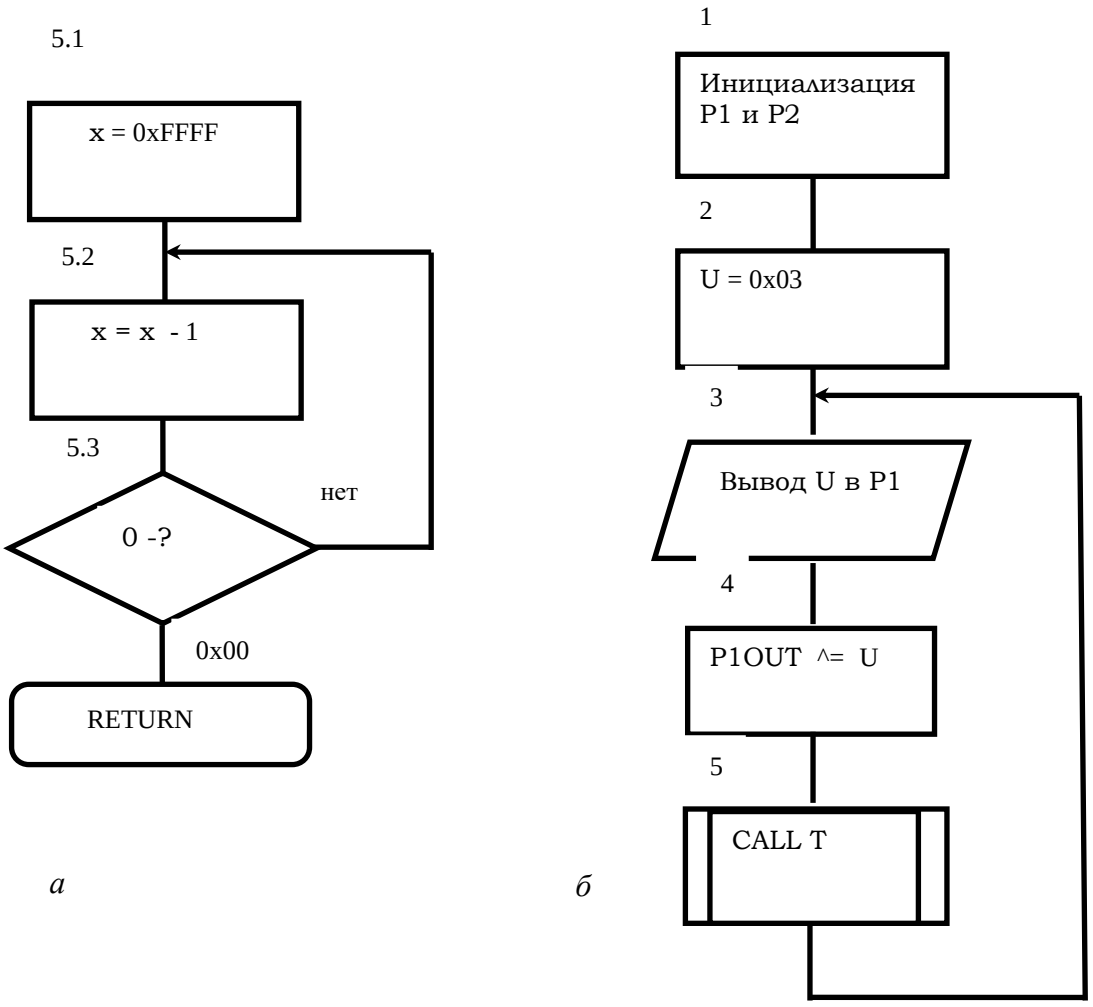
Алгоритм должен в инициализационной части обеспечить настройку порта *P*1 на вывод и начальное условие в младших битах порта *P*1. Если начальное условие 0x00 либо 0x03, оба светодиода будут подключаться одновременно. Если начальное условие 0x01 либо 0x02, светодиоды будут подключаться в противофазе.

Алгоритм должен в циклической части обеспечить переключение светодиодов и интервал времени, соответствующий заданному периоду t. Для организации интервала времени возможно использование подпрограммы. На рисунке 1.9, а представлен алгоритм подпрограммы формирования интервала времени. На рисунке 1.9, б показан алгоритм главной программы.

Принцип программной организации интервала времени T заключается в задании целой переменной *x* значения, пропорционального интервалу времени T. Далее, в некотором 16-ти разрядном регистре микроконтроллера, где хранится величина *x,* должно циклически выполняться декрементирование этой величины, как показано в алгоритме подпрограммы на рисунке 1.9, а. Когда достигается нулевое значение, выполняется выход из подпрограммы. Для расчета величины *x* используется значение тактовой частоты f_T и, кроме того, количество тактов N_Σ, за которое выполняется цикл в подпрограмме. Время выполнения одного цикла равно N_Σ/f_T. Время T выполнения *x* циклов равно $T = xN_\Sigma/f_T$, следовательно

$$x = T * f_T / N_\Sigma, \qquad (1.1)$$

Из выражения видно, что чем выше тактовая частота, тем большее значение переменной x необходимо для получения требуемого интервала времени.



а *б*

Рисунок 1.9 – Алгоритм подпрограммы (а) и главной программы (б)

Суммарное количество тактов N_Σ, за которое выполняется один цикл, зависит от программной реализации цикла, и это видно из текста программы, написанной на ассемблере. Далее приводятся тексты программ на языке ассемблера и на языке С.

Пример подпрограммы выдержки времени, построенной на языке *ассемблера* в соответствии с алгоритмом на рисунке 1.9, приводится далее.

```
            .cdecls C,LIST,<msp430x21x1.h>          ; Адреса периф. устройств
            .text
RESET       mov.w   #300h,SP                        ; инициализация стека
StopWDT     mov.w   #WDTPW+WDTHOLD,&WDTCTL          ; остановка таймера
                    WDT
                                                    ; разряд P1.0 – на вывод;
            bis.b   #001h,&P1DIR
```

```
M       xor.b   #001h,&P1OUT          ; переключение P1.0
        call    #wait                 ; запаздывание
        jmp     M                     ; повторение цикла
Wait    mov.w   #50000,R15            ; подпрограмма
L1      dec.w   R15                   ; запаздывания 0,2 с при
        jnz     L1                    ; 1МГц
        ret
;-------------                        ; Interrupt Vectors
            .sect.reset"              ; MSP430 RESET Vector
            .short  RESET
            .end
```

Переменная *x* записана в регистр R15. Цикл, создающий интервал времени T, имеет вид

```
L1      dec.w   R15         ; R15 - 1
        jnz     L1          ; если интервал не истек, L1
        ret                 ; возврат
```

Команда декрементирования *dec.w R*15 отсутствует в системе команд (таблица 1.1) микроконтроллера MSP430 и эмулируется командой *sub.w* #01*h, R*15, выполняемой за два такта [4]. Команда ***jnz*** *L*1 выполняется всегда за два такта. Поэтому в данном случае суммарное количество тактов N_Σ, за которое выполняется один цикл, равно N_Σ = 4. Представленная выше программа состоит из двух программных единиц: главной программы и подпрограммы.

Главная программа, в соответствии с алгоритмом, начинается инициализационной частью, в которой инициализируется стек (определяется адрес вершины стека в оперативной памяти данных), выполняется остановка таймера *WDT*, инициализируется на вывод младший бит порта *P*1 и определяется начальное значение 01*h* в регистре *P1OUT* порта *P*1. Циклическая часть начинается меткой *M*. В циклической части команда «исключающее ИЛИ» xor. *b* #001h, &*P1OUT* организует переключение бита P1.0, затем выполняется обращение *call #wait* к подпрограмме интервала времени. Далее цикл повторяется до бесконечности.

Подпрограмма, в соответствии с алгоритмом, начинается с метки wait, записи в регистр *R*15 значения x = 50000, пропорционального требуемому интервалу времени. Далее с метки *L*1 начинается цикл декрементирования *R*15, который повторяется, пока *R*15 не ноль (команда *jnz L*1). Затем происходит возврат (команда *ret*) из подпрограммы.

Интервал времени, в соответствии с (1.1), равен

$$T = N_\Sigma\, x/\, f_T = 4*50\,000\, /10^6 = 0{,}2 \text{ с} .$$

Директория проекта m0m0 на языке *C* содержит два исполняемых файла: главную программу *main.c* и функцию *wait.c,* как показано далее.

m0m0[Active - Debug] – имя проекта, который активен;

>Binaries - двоичный код, образуемый компилятором *Build*
>Incudes - включаемые в проект файлы библиотек:
>Debug - используется во время выполнения
>TargetConfigs -конфигурация: тип устройства и интерфейс;
>Ink_msp430f2274.cmd – командный файл;
>main.c - исходный текст главной программы;
>wait.c - исходный текст функции.

Тексты программ wait.c и main.c на языке *C*, построенные в соответствии с алгоритмом на рисунке 1.9 имеют вид:

```
#include <msp430.h> // addr. periph.
 void wait ( )              //  title
{
unsigned int i ;     //  0x00 ...0xFFFF
  P1OUT ^= 0x02 ;     //    xor.b
 for (i = 0; i <50000; i++); // cycl  0.2 s
  }
```

```
#include <msp430.h> // addr. periph.
void main ( void )          //  title
{
void wait ( );         // func. prototype
   P1DIR = 0xFF;  //   for output
WDTCTL = WDTPW | WDTHOLD;//stop wdt
   P1OUT = 0x01;    //init. value
      for (  ;  ;  )         // infinity cycle
   {
     P1OUT ^= 0x01;    //    xor.b
       wait (  );             //   call   vait
        }
}
```

Здесь программа *wait.c* является функцией языка *C* и служит для организации интервала времени 0.2 с. Выделены ключевые слова языка *C*. Файл *disassembly* на языке ассемблера**,** формируемый в процессе компиляции из исходного текста двух программ, доступен в интегрированной среде разработки.

Текст на языке ассемблера в файле *disassembly*, полученный при компиляции с *C* как правило больше, чем составленный вручную программистом текст программы на ассемблере. В последнем случае программа короче и выполняется быстрее. Тем не менее решающее преимущество языка *C* это ясность исходного текста программы для понимания и для дальнейшего усовершенствования.

В связи с этим программное обеспечение проектов автоматизации на основе микроконтроллеров в большинстве случаев выполняется на языках высокого уровня. Предпочтение отдается языкам *C*, *C*++, как первоначально ориентированным на возможности микропроцессоров и созданных для их программирования.

Контрольные вопросы и задачи

1 Принцип действия процессора.
2 Как выполняется командный цикл?
3 Какие способы адресации применяются в командах?
4 Как выполняются логические операции над байтами и словами?
5 Каковы функции специальных регистров процессора?
6 Для чего нужны флаги и где они хранятся?
7 Как отключить процессор?
8 Какие требования предъявляются к алгоритму для микроконтроллера?
9 Что должен содержать алгоритм в инициализационной части?

10 Что должен содержать алгоритм в циклической части?
11 Как в алгоритме обозначают условные переходы?
12 Как в алгоритме обозначают циклы и ветвления?
13 Как в алгоритме обозначают безусловные переходы?
14 Составить алгоритм формирования интервала времени.
15 Как организовать подпрограмму и обращение к ней?
16 Как используется стековая адресация в организации подпрограмм?
17 Что происходит в программном счетчике *PC* и указателе стека при обращении к подпрограмме и возврате?
18 Что происходит в программном счетчике *PC* и указателе стека при возврате из подпрограммы?
19 Когда используется прямая адресация?
20 Где используется косвенная адресация?
21 Как организовать прямую адресацию в языке *C*
22 Как организовать косвенную адресацию в языке *C*?
10 Сравните исходный текст на ассемблере и на языке *C*, построенные по алгоритму на рисунке 1.9.
11 В регистры ядра записываются значения: mov.b #03Eh, R5; mov.b #08Ah, R6; Что будет в регистре R6 после выполнения следующих логических операций?

```
1)      and.b R5,   R6   ;   R5 ∩ R6→R6     =   0Ah
bis.b       R5,   R6   ;   R5 U R6→R6     =   BEh
xor.b       R5,   R6   ;   R5 ⩛ R6→R6     =   04h

2)      bis.b       R5,   R6   ;   R5 U R6→R6
 xor.b       R5,   R6   ;   R5 ∀ R6→R6
 and.b       R5,   R6   ;   R5 ∩ R6→R6

3)      xor.b       R5,   R6   ;   R5 ⩛ R6→R6
and.b       R5,   R6   ;   R5 ∩ R6→R6
        bis.b       R5,   R6   ;   R5 U R6→R6
```

1.2. Алгоритмы и программирование микроконтроллеров.

1.2.1. Алгоритмы реального времени

Программное обеспечение для микроконтроллеров служит для систем автоматизации и функционирует в *реальном времени.* В таких системах решающим фактором является быстродействие. Поэтому архитектура любого микроконтроллера, включая систему команд, ориентирована на минимизацию времени выполнения команд. Микроконтроллеры в настоящее время выполняются по архитектуре *RISC,* а значит имеют сокращенную систему команд и трехступенчатый конвейер. Это способствует повышению быстродействия микроконтроллера за счет, во-первых, упрощения алгоритма работы его устройства управления, которое формирует командный цикл, и, во-вторых, параллельному выполнению команд за счет конвейера.

Система команд микроконтроллера MSP430 представлена в таблице 1 и ориентирована на применение языков высокого уровня *C* и *C*++ [8] –[14]. В проектах автоматизации можно применять программы как на ассемблере [7], так и на языке C и C++.

Преимущество программ на ассемблере для автоматизации заключается в быстродействии. Компиляция программ с языков высокого уровня C и C++

начинается формированием программных модулей на ассемблере. Как правило эти программы на ассемблере значительно превосходят по объему программы, которые составил бы для данной задачи пользователь.

Однако наглядность и переносимость программ на языке С и С++ является решающим преимуществом при создании больших проектов автоматизации. Так, программное обеспечение Motor Control для векторного управления электроприводами переменного тока обычно имеет исходный текст на языке С и С++.

Алгоритмы и программы автоматизации, как программы реального времени, должны содержать *инициализационную и циклическую части.* На рисунке 2.1 показаны примеры алгоритмов с инициализационной и циклической частями.

Инициализационная часть программы содержит инициализацию периферийных устройств для выполнения функций автоматизации и начальные условия. Так, для использования параллельных портов они настраиваются на ввод и вывод через заданные биты. Для использования любого интерфейса этот интерфейс настраивается на определенный режим.

Циклическая часть программы должна повторяться неограниченно, пока устройство в работе. Есть по крайней мере два способа организации циклической части. Это программная и программно-аппаратная организация (режим прерываний). В случае программной реализации цикл формируется в алгоритме. В случае использования прерываний цикличность возникает за счет периодических запросов прерываний от периферийных устройств.

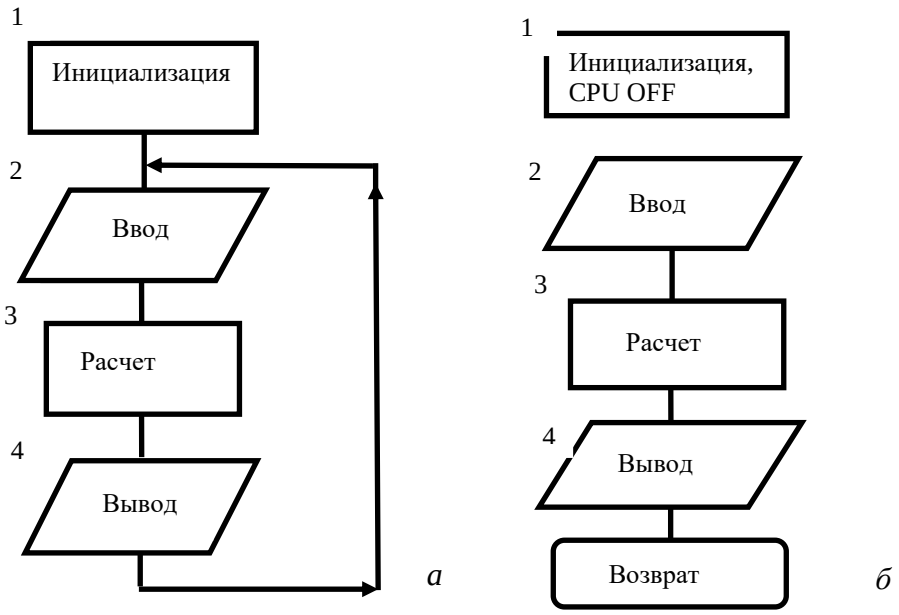


Рисунок 2.1 – Алгоритмы автоматизации, *а* – программная организация ввода и вывода, программно-аппаратная *(б*)

На рисунке 2.1, а показан алгоритм *программной* реализации циклической части. В этом случае инициализация периферийных устройств и расчет начальных условий (блок 1) выполняется однократно, а блоки 2, 3, 4 ввода информации, расчета и вывода выполняются, циклически повторяясь неограниченное число раз. Способ имеет недостаток: входные устройства

опрашиваются и в том случае, если их состояние не изменяется, затем многократно выполняется один и тот же расчет и выводится неизменный результат, на что непродуктивно расходуются вычислительные ресурсы. От этого недостатка свободен аппаратный способ организации цикличности.

На рисунке 2.1, б показан алгоритм *программно-аппаратного* способа реализации циклической части программы. В этом случае инициализация периферийных устройств и расчет начальных условий (блок 1) выполняются однократно, причем хотя бы одно из периферийных устройств инициализируется на режим прерываний. После этого CPU отключается, то есть переходит в режим низкого потребления электроэнергии (сна). Алгоритм подпрограммы прерывания выполняется по прерываниям от периферийного устройства, тогда и только тогда, когда состояние входа изменившись, вызвало прерывание. Каждый раз по запросу прерывания CPU выходит из режима низкого потребления для выполнения подпрограммы прерывания.

1.2.2 Интегрированная среда разработки проектов автоматизации

Интегрированная среда разработки (ИСР, *IDE – Integrated Development Environment*) проектов автоматизации для микроконтроллеров содержит программные средства разработки, необходимые для создания и отладки программного обеспечения автоматизации, его записи в память микроконтроллера. ИСР должна быть инсталлирована на персональном компьютере (ПК).

Обычно ИСР выпускается производителем микроконтроллеров для определенной серии производимых микроконтроллеров. Известна ИСР *Keil,* которая ориентирована на широкий ряд микроконтроллеров различных производителей. В любом случае работа в интегрированной среде начинается с создания *проекта автоматизации*, в котором указывается имя проекта, тип микросхемы и язык программирования в проекте. В результате создается проект, который содержит папки и файлы с информацией о микросхеме и об интерфейсе с ПК. В проекте должен содержаться хотя бы один файл исходного текста программы автоматизации на языке ассемблера либо на языке высокого уровня, который пишется пользователем в окне редактора. После компиляции (*BUILD*) и устранения ошибок проект загружается в память микроконтроллера (*DEBUG*). Далее возможно выполнение программы в непрерывном либо пошаговом режиме. Обычно ИСР обеспечивает вывод на экран ПК содержимого регистров ядра и периферийных устройств, в том числе и запоминающих устройств. Это позволяет проанализировать выполнение программы и ее эффективность.

1.2.3. Применение языка *C* для программирования микроконтроллеров

Язык *C/C++* находит широкое применение в современном программировании. Язык С, созданный в 1980-е годы, первоначально ориентирован для программирования микроконтроллеров. Бьярн Страуструп в 1988-89 годах создал объектно-ориентированный язык *C++*. Язык *C++* полностью унаследовал и расширил возможности языка *C*.

На языке *C/C++* [8]-[15] написано около семидесяти процентов программного обеспечения во всем мире после опубликования в 1978 году

книги [12], которая переиздавалась [13]. Стандарт машинно-независимого языка *С* принят ANSI в1988 году [14] и неоднократно дорабатывался [15].

Язык С, используемый для программирования микроконтроллеров и для системного программирования, усовершенствован в 1985-91 годах Бьярном Страуструпом [10], создавшим объектно-ориентированный язык С++. Язык С++ полностью унаследовал и расширил возможности языка С. В отличие от С в языке С++ есть возможность создания *классов*, то есть новых типов переменных, определяемых пользователем.

Создание программ на языке С/С++ для микроконтроллеров значительно повышает (по сравнению с ассемблером) производительность программирования, наглядность и эффективность программного обеспечения реального времени для управления электроприводами и автоматикой.

Язык программирования обычно содержит *алфавит, словарь и грамматику (синтаксис).* Алфавит *С* включает латинский алфавит, цифры, знаки арифметических и логических операций, три вида скобок, знаки препинания, знаки комментариев и другие знаки.

Словарь языка С содержит *ключевые слова*, которые приводятся в справочной литературе, и выделены цветом при их записи в окне редактора исходного текста программы.

Синтаксис требует соблюдения в тексте программы правил написания директив и операторов языка.

Текст программы на языке С содержит *директивы препроцессора, операторы и комментарии.*

Перед компиляцией программы на С/С++ работает *препроцессор* (служебная программа предварительной обработки исходного текста). При этом выполняется включение в программу указанных в директивах файлов, и другие действия директив препроцессора. На следующем этапе компилятор преобразует исходный текст программы с расширением **.c** в файл на языке ассемблера с расширением **.asm** для данного микроконтроллера. Далее формируется объектный модуль с расширением **.obj** и из него - исполняемый файл в абсолютных адресах с расширением **.exe.**

Директива препроцессора может быть включена в исходный текст программы в любом месте и действует до конца программы или до отмены. Некоторые из директив препроцессора показаны в таблице 2.1.

Таблица 2.1 Директивы препроцессора

Директива	Действие директивы	Пример
#include “*имя файла* **“** или **#include <***имя файла* **>**	Включение файла-	**#include <** msp430x21x1.h**>**
#define “имя макроса**“ ‘**фрагмент кода**‘** **#undef “**имя макроса**“**	Создание нового макроопределения Отмена	**#define “**Z**“** ‘sqrt(R * R + X * X)**‘** **#undef “**Z**“**
#if константное выражение **#ifdef** идентификатор **#ifndef** идентификатор **#else** **#endif**	Условная компиляция	**#if a** **#ifdef** xx **#ifndef** xx **#else** **#endif**
#line *целая константа* **“** *имя файла* **“** **#error *сообщение об ошибке”***	Вставить из указанного файла строку с номером, заданным целой	**#line 3 “**AZ.c **“**

	константой сообщение об ошибке	**#error zero_ div**
#pragma	Управление специфическими возможностями компилятора	**#pragma CODE_SECTION (Motor_Control, "secureRamFuncs");** /*Размещение программы **Motor_Control** в защищенной области оперативной памяти*/

С помощью директивы **#include "** *имя файла* **"** можно включить в создаваемую программу имеющиеся программы, в том числе библиотечные.

С помощью директивы **#define "***имя макроса***" '***фрагмент кода***'** создается макроопределение. После этой директивы везде в тексте программы, где встречается *имя макроса,* оно при компиляции заменяется на *фрагмент кода.* Директива действует до конца файла или до директивы **#undef"** *имя макроса***",** если таковая имеется.

Директивы **#if** *константное выражение,* **#ifdef** *идентификатор*, **#ifndef** *идентификатор,* **#else, #endif** условной компиляции позволяют выполнить компиляцию только при соблюдении определенного в директиве условия.

Директивы **#asm, #endasm** позволяют вставить в текст программы на языке С фрагмент, написанный на ассемблере. Это позволяет использовать известные преимущества языка ассемблера, например возможность абсолютной прямой адресации, сокращение объема памяти, занимаемой программным фрагментом.

Использование директив препроцессора может зависеть от среды разработки программ и от особенностей микроконтроллера. Например, директива **#pragma** может использоваться для указания векторов прерывания, которые свойственны определенному типу микроконтроллеров.

Структура программы на С/С++ следующая. Текст программы состоит из директив препроцессора, заглавия, операторов программы и комментариев. После заглавия следуют операторы программной единицы, обрамленные фигурными скобками. Перед первым выполняемым оператором в каждой программной единице (программе, подпрограмме-функции) должны быть описания всех применяемых в этой программе переменных. Далее следуют выполняемые операторы программы. Каждый оператор должен заканчиваться точкой с запятой. Комментарии, выделяемые двумя косыми чертами (//), должны располагаться в одной строке. Если комментарий занимает несколько строк, его должны обрамлять символы /*, */. В проекте автоматизации может содержаться много программ, однако только одна является главной и имеет имя **main**, которое является точкой входа. Остальные программы вызываются из главной, либо являются подпрограммами прерываний. Главная программа может иметь вид

```
#include < msp430x21x1.h>
void main (void)

{ P1DIR = 0xFF ;
    P1OUT = 0x03;
}
```

```
/* включает заголовочный файл с адресами периферийных устройств */
        /* main - главная программа, void - отсутствие передаваемых данных*/
        // инициализация порта P1 на вывод
        // вывод единиц в 2 младших бита
```

В результате выполнения данной программы в два младшие бита порта P1OUT выводятся единицы. В данной программе отсутствуют переменные, но используются две константы: 0xFF и 0x03.

Если в программе на языке С необходимо использование переменных, они должны быть описаны перед первым выполняемым оператором с указанием их *типов*. В результате в памяти резервируется область для хранения переменных. Так, тип переменной **char** применяется для символьных переменных, а также для однобайтовых значений. Переменная типа **integer** занимает одно 16-разрядное слово. Это означает для переменной со знаком возможен диапазон значений (-32 768, … …, 0, …., 32 767), для переменной без знака (**unsigned**) возможен диапазон от нуля до 65535. Тип переменной **float** означает ее представление с плавающей точкой, а именно, в двойном слове из 32 бит хранится мантисса и порядок в формате ±0.MMM e±PPP.

Обычно в программах для микроконтроллеров используются типы, указанные в таблице 2.2.

Таблица 2.2. Типы переменных

Тип	Диапазон значений	Пример применения
char	0x00, ….,.0xFF	**char** a, b, c **;**
integer **int**	0xFFFF….0x0000….0x7FFF -32 768,… …,.0,……..32 767	**int** x, y, z;
unsigned int	0x0, …..0xFFFF 0,…,65535	**unsigned int** u, v**;**
float	-10^{38},…,-10^{-38},…,0.0,…$+10^{-38}$,…$+10^{38}$	**float** w, r, p**;**

В следующем примере вычисляется сумма и применяются переменные различных типов.

```
#include “ msp430x21x1.h“ // адреса периферийных устройств
void main (void)       //главная программа
{
char i, k = 4;//целые переменные, 8 разрядов, со знаком
int a, b;   // целые переменные, 16 разрядов, со знаком
float X = 0, Y = 0; // переменные с плавающей точкой
b = 3;
a = 4;
X = a / b; //далее выполняется вычисление Y :
for (i = 1; i < k; i++)    // цикл повторится 3 раза
Y = Y + i  *  X;      }   // сумма 0 + X + 2 * X + 3 * X
```

Переменные бывают *глобальные и локальные.* Локальные переменные действуют только внутри программной единицы, где объявлены, а глобальные – в главной программе и используемых ею подпрограммах, то есть в пределах разрабатываемого проекта. Область видимости любой переменной зависит от того, в какой программной единице она объявлена.

Выражением называется последовательность операндов, круглых скобок и знаков операций, задающих вычисление. Круглые скобки задают порядок

действий. Если нет скобок, порядок действий определяется приоритетами операций в соответствии, преимущественно, с правилами алгебры.

Произвольное математическое выражение превращается в оператор, если его закончить точкой с запятой. Арифметические операции описывают арифметические действия в соответствии с таблицей 2.3.

Таблица 2.3 Арифметические операции

Символ	Действие	Примеры

-	замена знака, вычитание	*X = -y; //X обратно y*
		a= b – c;
		a = a – c; // или
		a -= c; // из a вычитается c

++	инкремент	*a* **++** *; // a = a* + 1
- -	декремент	*a - - ; // a = a* - 1

+	суммирование	*a = b + c ;*
		a = a + b; //или
		a += b; //суммирование с b

*****	умножение	*a = b * c ;*
		*a = a * c ; //или*
		*a *= c ;// умножение на c*

/	деление	*a = b / c ;*
		a = a / c ; //или
		a /= c ;// деление на c

Битовые операции выполняются над одноименными битами 8 и 16 разрядных слов. Операции сравнения выполняют проверку на равенство, неравенство и результат сравнения, имея тип **int,** принимает значения 1 (истинно, TRUE) или 0 (ложно, FALSE). Примеры операций сравнения:

C = (a == x); - равно? *C = (a > x); больше?*
C = (a ~= x); не равно? *C = (a <= x); меньше или равно?*
C = (a < x); меньше? *C = (a >= x); больше или равно?*

Логические и битовые операции описывают логические действия в соответствии с таблицей 2.4.

Таблица 2.4 Логические и битовые операции

Символ	Действие	Примеры
Битовые операции		
~	*- инверсия (для одного аргумента)*	*X =~y; //X – побитовая инверсия y*

&	*- конъюнкции*	*a = b & c; //*
		a = a & b ; // или
		a &= b ; //

|	*дизъюнкция*	*a = b | c; //*
		a = a **|** *b ; //* или
		a **|**= *b ; //*

^	*исключающее “или”*	*a = b ^ c; //* *a = a* **^** *b ; //* или

		a ^= b ; //
Логические операции		
!	*инверсия*	*a =! (b == c);* *a = !c;//*
&&	*конъюнкция*	*a = (b > c) && (c == d);*
||	*дизъюнкция*	*a = (b > c)* **||** *(c == d);*

Этапы компиляции программы на С следующие.

1 Исходный текст программы на языке С содержит директивы препроцессора, с выполнения которых и начинается компиляция.

2 Преобразование исходного текста программных модулей в программы на языке ассемблера с расширениями **.asm** и преобразование их в двоичный код.

3 Формируются объектные файлы программных модулей с расширениями **.obj**.

4 Далее выполняется связывание программных модулей (**link**).

5 Формируется исполняемый файл с расширением **.exe,** который пригоден для записи в запоминающее устройство (обычно ПЗУ, FLASH) микроконтроллера.

Использование директив препроцессора может зависеть от среды разработки программ и от особенностей микроконтроллера.

Имена переменных составляются из букв латинского алфавита, цифр и знака подчеркивания, всего в имени может быть до 32 символов. Имена не должны совпадать с ключевыми словами языка С (ключевые слова выделены цветом).

Переменные бывают *глобальные и локальные.* Локальные переменные действуют только внутри программной единицы, где объявлены. Глобальные переменные действуют в главной программе и используемых ею подпрограммах, то есть в пределах разрабатываемого проекта. Область видимости любой переменной зависит от того, в какой программной единице она объявлена.

Выражением называется последовательность операндов, круглых скобок и знаков операций, задающих вычисление. Круглые скобки задают порядок действий. Если нет скобок, порядок действий определяется приоритетами операций. Например, *a = 3 * (e + c)* - это выражение. Произвольное выражение превращается в *оператор*, если его закончить точкой с запятой, например *a = 3 * (e + c);* - это оператор.

Пример применения логических операций в тексте программы:

```
#include "msp430x22x4.h"
void main ( void )
{ char a = 15, c = 5, b = 0xED;
P1DIR = 0x0F;     // настройка на вывод
if (a == 0x0F ) && ( a & (b + c ) ) > 0 ) P1OUT = a & (b + c );// операция И.
 else P1OUT = a; //
}
```

В этом примере в порт P1 выводится либо величина *a & (b + c),* если она имеет ненулевые биты в младшей тетраде, либо величина *a.*

Пример применения битовых операций в тексте программы:

```
#include "msp430x22x4.h"
void main ( void )
{ char a, c = 5, b = 0xED;
P1DIR |= 0x0F;     // побитовая логическая операция ИЛИ над байтами
a = 0x0F & ( b + c ); // побитовая логическая операция И над 2-мя байтами
P1OUT |= a;     /*операция ИЛИ над байтом позволит сохранить в других битах в порту ранее выведенную информацию */.
}
```

В этом примере в порт P1 выводится величина *a* = 0x0F *& (b + c).*

Операторы. Если арифметическое или логическое выражение закончить знаком (;), получится оператор. Простейший оператор имеет вид:(;) – это пустой оператор. Оператор условия имеет разновидности

If (условие**)** оператор;

If (условие**)** оператор1 ; **else** оператор2;

If (условие1**)** оператор1 ; **elseif (**условие2**)** оператор2;

Условие в скобках принимает одно из двух значений, 1 – TRUE, 0 – FALSE целого топа. В зависимости от этого следующий за условием оператор выполняется (если 1) либо не выполняется (если 0). Вторая и третья разновидности имеют продолжение в виде альтернативы.

Пример 1. Составить алгоритм и программу включения контактора путем вывода 1 в бит выходного порта при наличии во входном порту сигнала ПУСК и отсутствии сигнала СТОП. Алгоритм предлагается составить самостоятельно.

Программа может иметь следующий вид.

```
#include "msp430x22x4.h"
void main ( void )
{
char a, b;
P1DIR = 0xFF;     // настройка на вывод
P2DIR = 0xFC;     // настройка на ввод  2 младших битов
b = 0x00;
L:  P1OUT = b ;
a = P2IN;
if (a == 0x01) b = 0x01;// ПУСК.
if (a == 0x00)   ;// состояние выхода прежнее
if (a >= 0x02) b =0x00;// состояние выхода 0
goto L;
}
```

В этой программе для организации цикла применяется оператор перехода **goto** L, что является нежелательным. Возможно упрощение программы, если применить другие формы оператора *if* и организовать циклы.

Операторы цикла. В языке С три оператора цикла. Оператор **while** имет вид

while (условие**)** оператор;

Если условие выполнено, его значение 1, и оператор выполняется. Иначе оператор не выполняется. Оператор **do while** имет вид

do оператор **while (**условие**)** ;

Оператор выполняется, пока условие выполнено, то есть его значение 1, иначе оператор не выполняется. Программа предыдущего примера может иметь вид.

```
#include "msp430x22x4.h"
void main ( void )
{
char a, b;
P1DIR = 0xFF;     // настройка на вывод
P2DIR = 0xFC;     // настройка на ввод  2 младших битов
b = 0x00;
while (1) {
P1OUT = b ;
a = P2IN;
if (a == 0x01) b = 0x01;// ПУСК.
if (a == 0x00)   ;// состояние выхода прежнее
if (a >= 0x02) b =0x00;// состояние выхода 0
}
}
```

Здесь в составе **while** место оператора занимает группа операторов в фигурных скобках, что допускает язык С. Программа предыдущего примера может иметь вид.

```
#include "msp430x22x4.h"
void main ( void )
{
char a, b;
P1DIR = 0xFF;     // настройка на вывод
P2DIR = 0xFC;     // настройка на ввод  2 младших битов
b = 0x00;
do {
P1OUT = b ;
a = P2IN;
if (a == 0x01) b = 0x01;// ПУСК.
if (a == 0x00)   ;// состояние выхода прежнее
if (a >= 0x02) b =0x00;// состояние выхода 0
}
while (1) ;   }
```

Пример 2.1. Составить программу формирования интервала времени. Программа может использовать оператор **while:**

```
#include "msp430x22x4.h"
void main ( void )
{
int k, b;
P2DIR = 0xFF;     // настройка на вывод  2 младших битов
b = 0x01;
while (1) {       // начало цикла
```

```
k = 50000;    // интервал времени
while (k > 0) k-- ; //  k = k - 1;  вложенный цикл отсчета времени
P1OUT ^= b ;  //   переключение на выходе
}}
```

Программа содержит внешний цикл, повторяемый бесконечное количество раз, и внутренний цикл отсчета времени, повторяемый 50 000 раз. После каждого интервала времени предусмотрено переключение младшего бита выходного порта операцией ^ исключающее ИЛИ.

Программа может использовать оператор **do while:**

```
#include "msp430x22x4.h"
void main ( void )
{
char k, b;
P2DIR = 0xFF;     // настройка на вывод  2 младших битов
b = 0x01;
k = 50000;        // интервал времени

do  {                    // начало цикла
k-- ;
while (k > 0)  ;
    }
P1OUT ^= b;   // переключение на выходе
 }
```

Оператор **for** наиболее универсален для организации циклов и имет вид:

for (выражение1**;** выражение2**;** выражение3**)** оператор;

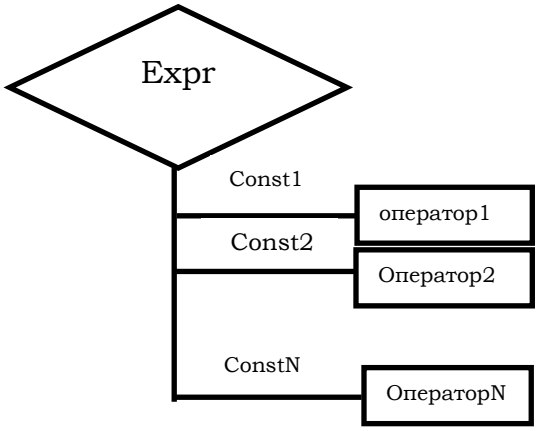
Здесь выражение 1 — это начальное условие, выражение 2 - условие повторения цикла, выражение 3 - расчет. Оператор повторяется в каждом цикле. Программа примера 2.1, где формируется интервал времени, принимает вид

```
#include "msp430x22x4.h"
void main ( void )
{
char k, b;
P2DIR = 0xFF;     // настройка на вывод  2 младших битов
b = 0x01;
  // интервал времени
for (  ;  ;  )      // внешний цикл, повторяется бесконечно
{
for ( k = 50000;k >0; k-- )  ; //  вложенный цикл отсчета времени
P1OUT ^= b ;  //   переключение на выходе
}
}
```

Здесь для обозначения бесконечно повторяемого цикла три поля в скобках оператора **for** остаются пустыми.

Оператор переключения. Оператор переключения предназначен для ветвления на две и более ветвей алгоритма, как показано на рисунке 2.2, где Expr – целочисленное переключающее выражение, которое может принимать значения Const1, Const2, … , ConstN константных выражений. После каждого ключевого слова **case** указывается очередное значение константного выражения, и после двоеточия – оператор.

Оператор переключения **switch** имеет вид




```
switch (Expr)
{ case Const1:  оператор1;
  case Const2:  оператор2;
…
default :           операторM;
 …

  case ConstN:  операторN;
}
```

Рисунок 2.2 – Алгоритм ветвления

В соответствии с синтаксисом языка С везде, где записывается оператор, возможна также группа операторов в фигурных скобках

Пример 1 может быть выполнен с применением оператора переключения следующим образом.

```
#include "msp430x22x4.h"
void main ( void )
{
char a, b;
P1DIR = 0xFF;     // настройка на вывод
P2DIR = 0xFC;     // настройка на ввод  2 младших битов
b = 0x00;
for (       ;     ;     )  // основной цикл
{ P1OUT = b ;
a = P2IN;
switch (a )   // переключение
{
case  0x01:  b = 0x01;// ПУСК.
case  0x00:        ;// // состояние выхода прежнее
defaulte    b = 0x00;// состояние выхода 0
}
}
}
```

Здесь инициализационная часть не изменилась, но циклическая часть организована оператором **for(; ;),** и анализ входного сигнала выполняется в операторе переключения.

Указатели и ссылки. В языке С косвенная адресация выполняется применением *указателей*. Значение указателя равно адресу переменной. При описании указателя указывают тип переменной, на которую он указывает. Например

int* *p*; // описание указателя *p* на целую переменную

Для получения адреса переменной к ней применяют операцию раскрытия адреса (*&*). Чтобы обратиться к переменной, используя указатель *p*, применяют (*). Например,

```
int *p, a ;   // указатель p на целую переменную и целая переменная a

p = &a;  // после этого указатель указывает на a;
*p++ ;    // инкрементирование  a; это равнозначно a++ ;
p  += 2 ; // здесь инкрементируется на 2 адрес переменной;
```

Ссылка, как и указатель, равна адресу, но используется как имя уже описанного объекта для использования с функциями. Например,

```
int  b, a ;   //
int& refb = b, refa = a ;   // ссылки на целые переменные b, a
refb += refa; // эквивалентно b += a;
```

Структуры данных языка С. Массивы. В языке С наряду с простыми переменными можно применять *массивы, структуры и объединения.* Массив хранит данные одного типа, называемые *элементами* массива. Массив может иметь одну или две размерности (вектор или матрица). Структура и объединение данных хранят данные разных типов, и состоят из полей. Поля могут содержать как простые переменные, так и массивы, структуры объединения. Поэтому структуры и объединения позволяют хранить сложные системы взаимосвязанных данных, относящихся к однотипным объектам.

Описание массива может иметь следующий вид: *тип имя[размер].* Например, описания массивов разных типов и размеров могут иметь вид

```
char  Y[4]={0x05, 0x01, 0x0F, 0x0C } ;  // массив, 8 разрядов с заданным начальным значением
int   Y[4]={525, 38, 3, 4 } ;  // массив целого типа, 16 разрядов с заданным начальным значением
int   A[4][5]= {{0,1,2,3,4}, {6,7,8,9,10}, {0,1,3,4,5}, {16,0,2,3,4}} ; // массив 4 x 5 целого типа
float32  Y[4] ;          // массив из 4 элементов с плавающей точкой, 32 разряда
```

Элементы массивов нумеруются, начиная от нуля, причем для обращения к элементу массива следует указать его индексы, например ***A*[0][1] + = 1;** - это элемент первой строки второго столбца матрицы инкрементируется. Следует учитывать, что имя массива является указателем на адрес его первого элемента. Поэтому есть способ доступа к любому элементу массива, используя указатель. Далее рассматриваются примеры программ с использованием массивов.

Пример 2.2. Составить алгоритм и программу формирования сигнала задания скорости с ограничением на ускорение при разгоне и торможении

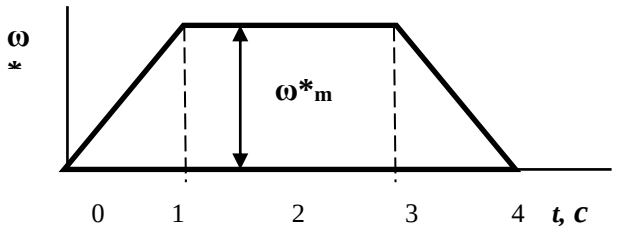


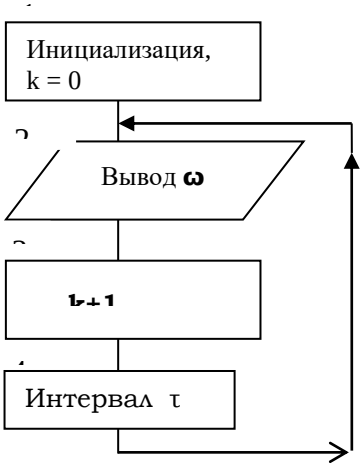
Рисунок 2.3 – Сигнал задания скорости

(рисунок 2.3).

Два варианта алгоритма и программы представлены на рисунках 2.4 и 2.5. Вариант, показанный на рисунке 2.4 имеет простой алгоритм, но

используется большой массив с повторяющимися значениями. Здесь k = 0, 1, …, n. Величина $\mathbf{n = 2*\omega^*_m + t_0/\tau}$ есть количество шагов на интервале разгона, установившегося движения и торможения, $\mathbf{t_0 = 2\ c}$ – время установившегося движения, n равно размеру массива w значений скорости, $\mathbf{\omega^*_m=4}$. На каждом шаге сигнала задания следует обеспечить формирование интервала времени $\mathbf{\tau = t_p/\omega^*_m}$.

Вариант, показанный на рисунке 2.5, использует массив значений скорости на этапе разгона. При разгоне элементы массива выводятся по возрастанию индекса *k*. На участке постоянства скорости циклически выводится наибольшее значение из массива, а на участке торможения элементы массива выводятся по убыванию индекса.




```
#include <msp430.h> // адреса периф. устройств
void main (void) / *главная программа */
{
int i, k, wm= 4, n; //целые, n= 2 * wm + 8=16
int w[17] = {0,1,2,3,4,4,4,4,4,4,4,4,4,4,3,2,1,0};
w[0] = 0x00;
P1DIR  = 0xFF ; //    настройка на вывод

//    cycle
```

Рисунок 2.4 – Алгоритм и программа

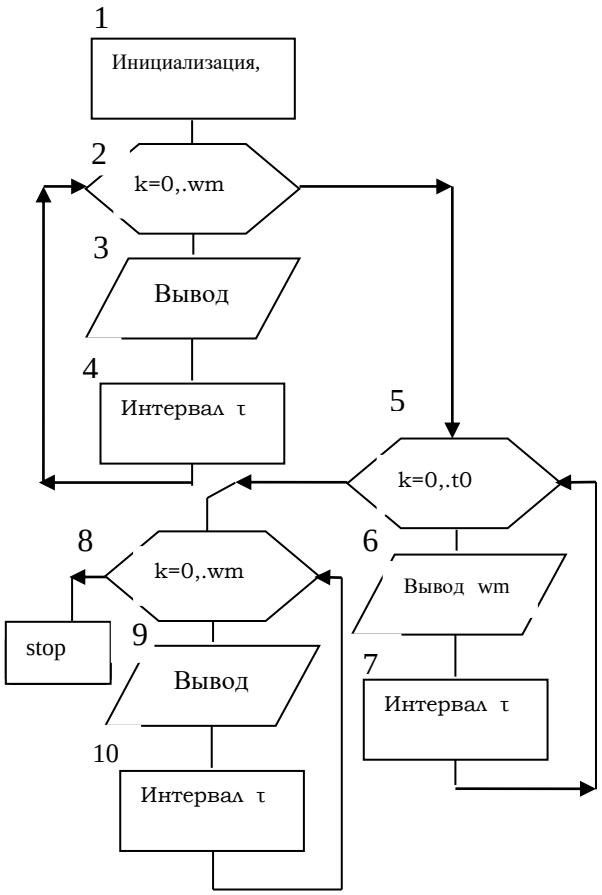
Структуры и объединения. Структуры и объединения предназначены для систематизации данных. Обычно каждая структура хранит данные, относящиеся к одному объекту, но каждое ее поле содержит данные одного определенного типа. Поля структуры размещаются в оперативной памяти в порядке их описания в структуре. Каждое поле может содержать простые переменные и массивы определенного типа, либо структуры и объединения. Описание структуры имеет вид

```
struct имя
{
поля
};
```

Объединения отличаются от структур только способом хранения в памяти: каждое поле объединения начинается в памяти от одного адреса, и, следовательно, поля накладываются. Это означает, что в каждый момент времени выделенная область памяти хранит одно определенное поле. Описание объединения имеет вид

```
union имя
{
поля
```

```
};
```




```
#include " msp430x21x1.h" // адреса периф. устройств

void main (void) /*главная программа */
{
int i, k, wm= 4, t0 = 10;   //целые , t0 = t0/τ
char ω[wm+1] ={0, 1, 2, 3, 4}; ω[k] = 0x00;
P1DIR  = 0xFF ;      //   настройка на вывод
for( k  = 0; k <= wm ; k++)// цикл  wm  раз
  {                                  // начало цикла
    P1OUT =  ω[k] ;  // вывод
for  (i  = 0; i < 50000; k++)  //интервал τ
  }                                   // конец цикла
for(k  = 0; k <= t0; k++)//  t0  раз, уст.реж
  {                                  // начало цикла
    P1OUT =  w[wm] ;  // вывод ωт
for  (i  = 0; i < 50000; k++)  //интервал τ
  }                                   // конец цикла
for(k  = ωm; k > =0; k- -)// цикл  wm раз
  {                                  // начало цикла
    P1OUT =  ω[k] ;  // вывод
for  (i  = 0; i < 50000; k++)  //интервал τ
  }                                   // конец цикла
}
```

Рисунок 2.5 – Алгоритм и программа задания скорости

Описание оканчивается точкой с запятой после фигурной скобки.

Пример описания структуры данных асинхронного электродвигателя приводится далее.

```
struct AD
{
char pp;              // пар полюсов
int  Pn, Un, nn, In;  //мощность, напряжение, частота вращения, ток
int  , Mn; // момент
float32  kpd, cosf; // КПД и коэфф. мощности,
};
AD AIRM132M2, AIR180S2, AIR180M2;
```

После описания структуры может быть создано множество структур такого вида. В данном случае список структур представлен после описания структуры. Каждая из структур списка предназначена для хранения данных электродвигателя определенного типа.

Доступ к полям структуры возможен указанием имени структуры и имени поля в расширении, например

```
AIRM132M2.pp = 1;                AIRM132M2.Pn = 11;
AIRM132M2.Un = 400;              AIRM132M2.In = 21;
AIRM132M2.nn = 2900;               AIRM132M2.kpd = 0.89;
```

```
AIRM132M2.cosf = 0,89;
```

Применение структур и объединений оказалось необходимым для развития объектно ориентированного программирование в языке С++. Развитием структур являются классы, используемые в языке С++.

Функции в С и С++. Любая программа на языке С должна содержать единственную функцию с именем **main** (главная программа), которая обеспечивает точку входа в откомпилированную программу. Остальные функции, если они есть, запускаются из главной программы.

Функции являются внешними объектами (класс хранения в памяти **extern**). Доступность функции обеспечивается ее описанием в модуле до первого вызова.

Описание функции содержит заголовок и тело функции. В заголовке указывают тип результата функции и имя, в круглых скобках список параметров с описанием их типов. Это формальные параметры. При обращении к функции они заменяются фактическими. По умолчанию тип результата функции и ее аргументов (формальных параметров) целый (**int**). Например, описание функции возведения переменной в куб может быть таким:

```
int xcub (int x)
{
return x * x * x ;
}
```

Оператор **return** в теле функции необязателен в том и только в том случае, если функция имеет тип **void.**

Прототипом функции называется описание обращений к функции. Например, прототип

```
extern int xcub (int x)
```

делает доступной функцию, описание которой находится в другом модуле. После этого возможен вызов функции в операторах программы. Вызов функции означает присутствие в операторе программы в качестве операнда имени функции со списком фактических параметров (аргументов функции) в круглых скобках. Например,

```
int a = 25, b = 7, y;

 y = a + b +xcub ( a ); //  оператор содержит обращение к функции
```

Если функция имеет тип **void,** а ее аргументы описаны, как глобальные, то списки формальных и фактических параметров могут быть пустыми. Для описания глобальных переменных может быть создан специальный файл с расширением **.h**, например **GL.h**. Тогда необходима директива **#include < GL.h** >, которая подключает этот файл к главной программе.

Контрольные вопросы

1 Как использовать указатель для доступа к элементам массива?
2 Как задать начальные значения элементам массива?
3 Составить алгоритм и программу умножения матрицы на вектор.

4 Как использовать массив для формирования сигнала задания скорости?

5 Чем отличается структура данных от массива?

6 В чем отличие структуры данных и объединения?

7 Как получить доступ к полям структуры для записи и чтения?

8 Как и для чего применяются директивы препроцессора?

9 В каком месте программы располагаются объявления и описания переменных?

10 Как описать переменные различных типов?

11 Каковы требования к именам переменных?

12 Какие части должна содержать программа на языке C/C++ ?

13 Построить пример применения условного оператора.

14 Построить пример применения оператора цикла.

15 Построить пример применения оператора переключения.

16 Сравните исходный текст на ассемблере и результат компиляции с языка *C* на ассемблер.

17 Какими способами организации циклов располагает язык *C*?

18 Как организовать выполнение логических операций в *C*?

19 Как организовать условный оператор?

20 Каков синтаксис оператора переключения?

21 Как объявить массив в *C*?

22 Составить алгоритм и программу вычисления элементов массива.

23 Составьте программы на языке *C* по алгоритму на рисунке 1.9 разными способами, с использованием и без использования функции.

24 Составьте программы на языке *C* по алгоритму на рисунке 1.9, организуя функцию разными способами.

25 Составьте программы на языке C по алгоритму на рисунке 1.9 для больших интервалов времени (x > 0xFFFF), с использованием и без использования функций.

1. 3. Организация обмена информацией (интерфейс)

Магистральный принцип организации обмена информацией в микроконтроллерах. В микроконтроллерах, как и в вычислительной технике в целом, применяется магистральный принцип обмена информацией. Альтернатива магистральному принципу, соединение каждого устройства с каждым, имеет преимущество в быстродействии, однако приводит к большому объему соединительных проводников.

Магистраль состоит из *шины адреса AB*, *шины данных DB, и шины управления CB* (рисунок 3.1). Магистраль соединяет *CPU* с оперативным (*RAM*) и постоянным (*FROM*) запоминающими устройствами, аналого-цифровым преобразователем (*ADC*), параллельными портами *P1,…,P4* ввода и вывода, сторожевым таймером *WDT* и таймерами *Timer A, Timer B* общего применения, а также с устройством последовательных интерфейсов *SPI, I2C*, *UART*.

Каждая шина состоит из совокупности m проводников по количеству разрядов шины. В микроконтроллерах обычно m =16 или m = 32, в 8-разрядных микроконтроллерах для шины данных m = 8. *Шинный формирователь* обеспечивает 3-стабильное состояние шины: 0, 1 и *отключено* (*высокоимпедансное* состояние).

Обмен информацией по магистрали предполагает не более одного передающего устройства в каждый момент времени. Как правило, в процессе выполнения командного цикла (рисунок 1.3) обмен происходит под управлением CPU. Это *запись, чтение* (*write, read*) процессором запоминающих и других периферийных устройств (рисунок 3.1).

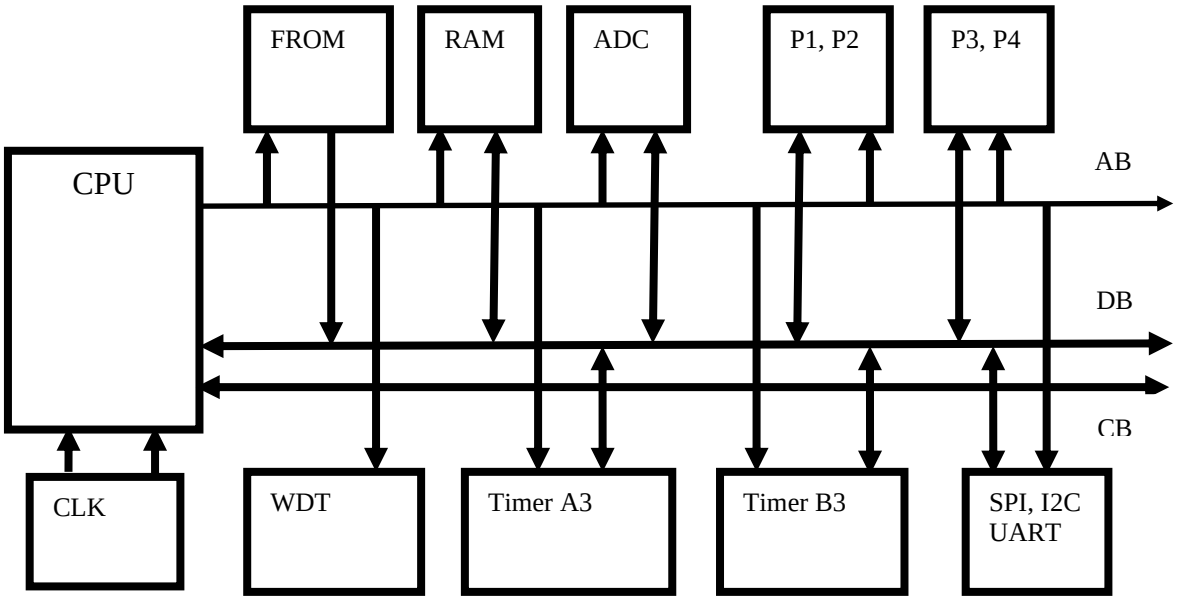


Рисунок 3.1 – Микроконтроллер с обменом информацией по магистрали

Программная, аппаратно-программная и аппаратная организация обмена информацией. Обмен информацией между устройствами может быть организован программным, аппаратно-программным и аппаратным

способами. Как правило, программная организация любых функций имеет преимущество в гибкости. Аппаратные средства обеспечивают быстродействие.

Программная организация обмен информацией между устройствами означает, что ввод и вывод информации выполняются как команды в программе. Например, на языке ассемблера ввод и вывод двоичного сигнала, не превышающего 1 байт, имеет вид

```
mov.b   &P2IN, R7   ;  ввод сигнала
mov.b   R7, &P1OUT  ;  вывод сигнала.
```

Программный ввода и вывод не эффективен, когда опрос периферийных устройств выполняется циклически многократно, а входные данные неизменны. Программно-аппаратный ввод и вывод выгодно отличается тем, что опрос входных сигналов организуется лишь при их изменении.

Программно-аппаратный ввод-вывод выполняется в режиме *прерываний*. Режимом *прерываний* называют режим, при котором процессор прерывает выполнение программы для обслуживания периферийного устройства (обычно для ввода или вывода). Каждое периферийное устройство имеет свой *вектор прерывания* (адрес начала подпрограммы прерывания для обслуживания этого периферийного устройства). Каждое периферийное устройство запрашивает прерывание, когда готово к вводу-выводу. Например, таймер запрашивает прерывание, когда сформирован интервал времени; аналого-цифровой преобразователь запрашивает прерывание по окончании преобразования; параллельный порт запрашивает прерывание, когда входная величина изменилась.

Управление прерываниями в микроконтроллере предполагает *фиксированный либо циклический* приоритет периферийных устройств. Фиксированный приоритет обычно определяется местом подключения устройства к магистрали, а циклический изменяется по мере обслуживания устройств таким образом, что последнее обслуженное устройство приобретает низший приоритет. Управление прерываниями начинается контролем приоритетов. Далее, по окончании текущего командного цикла, процессор дает разрешение на прерывание, принимает по шине данных вектор прерывания от периферийного устройства и переходит к подпрограмме прерываний.

Если во время выполнения подпрограммы прерываний поступает запрос на прерывание от устройства с более высоким приоритетом, организуется переход к подпрограмме этого устройства.

Аппаратная организация обмена информацией выполняется *прямым доступом к памяти* (ПДП, *Direct Memory Access, DMA*), минуя процессор. Периферийное устройство генерирует запрос на захват магистрали. Если этот запрос не единственный, контроллер ПДП (DMA) выполняет арбитраж. В это время процессор выполняет программу, и, как только магистраль освобождается, подтверждает захват от периферийного устройства с высшим приоритетом и отключается. После этого формируется начальный адрес и размер массива, и массив передается из памяти в периферийное устройство или в обратном направлении.

Во многих микроконтроллерах имеется отдельная магистраль для прямого доступа к памяти нескольких периферийных устройств, что повышает производительность микроконтроллера. Аппаратная организация

обмена ускоряет ввод и вывод значительных объемов информации и позволяет освободить процессор от операций ввода и вывода.

Параллельный и последовательный интерфейс. Внутри микроконтроллера обмен информацией между ядром и периферией выполняется через магистраль, где слова и байты передаются в параллельном представлении (каждый бит передается по отдельному проводу).

Параллельный интерфейс представлен параллельными портами ввода-вывода, которые применяются для обмена информацией между устройствами. Обычно параллельный порт содержит буферные регистры для ввода, вывода данных и регистры для управления вводом-выводом. В основном параллельные порты применяются для ввода и вывода битовых сигналов логического управления оборудованием.

Параллельный интерфейс имеет преимущество в быстродействии, поскольку для ввода вывода требуется от 1 до 5 тактовых периодов. Недостатки параллельного интерфейса вызваны необходимостью m проводников для передачи сигнала в параллельном виде, где m- количество двоичных разрядов. Расход проводов, индуктивность и емкость электрической цепи возрастают с ростом m, а устойчивость к помехам ухудшается.

Для организации связи между вычислительными устройствами на значительные расстояния используется последовательный интерфейс, который обеспечивает обмен данными в последовательном представлении. Слово данных передается по одному проводнику за m тактовых периодов. Это обеспечивает малую индуктивность и емкость электрической цепи, устойчивость к помехам.

1.3.1. Программируемый таймер и его применение

Программируемый таймер предназначен для формирования интервалов времени, а также выполнения зависимых от времени функций. Таймер программируется на выполнение определенных функций путем записи в регистры управления *управляющих слов*. Программируемые таймеры используются как самостоятельные устройства в виде отдельных микросхем, а также входят в состав микроконтроллеров. Обычно в составе микроконтроллера от двух до шести и более программируемых таймеров.

Известны два способа формирования интервалов времени: программный и аппаратный (с помощью таймера). Мерой времени для таймера является период тактовой частоты. Поэтому таймер позволяет более точно формировать интервалы времени, чем это возможно программным способом, где интервал времени формируется с точностью до длительности повторяющегося цикла, содержащего несколько команд.

Принцип действия таймера заключается в следующем. В регистры управления записывают управляющие слова, чтобы задать режим работы таймера. В регистр сравнения записывают рассчитанное значение, пропорциональное нужному интервалу времени. После этого таймер готов к применению. По сигналу разрешения счета содержимое регистра счета инкрементируется либо декрементируется при каждом тактовом импульсе. На рисунке 3.2 показаны возможные режимы работы таймера. Счет идет до заданного либо до нулевого значения. Обычно в конце счета таймер

генерирует запрос на прерывание, что позволяет в подпрограмме прерывания использовать сформированный интервал времени.

Важнейшими режимами, в которых используются программируемые таймеры, являются режимы ввода и вывода *событий.* Событием для микроконтроллера может быть:

- появление уровня логической единицы на внешнем выводе;
- появление уровня логического нуля на внешнем выводе;
- появление нарастающего фронта на внешнем выводе;
- появление спадающего фронта на внешнем выводе;
- появление любого фронта на внешнем выводе;

Захватом называют режим *ввода внешнего события* через заданный вход микроконтроллера, сопровождаемый записью содержимого регистра-счетчика таймера. Таким образом, время появления события фиксируется. Так, если на вход микроконтроллера поступают импульсы, то режим захвата позволяет измерить их длительность.

В электроприводах режим захвата позволяет вводить сигналы инкрементального датчика положения (энкодера) и рассчитывать скорость, что необходимо для обратных связей по положению и скорости.

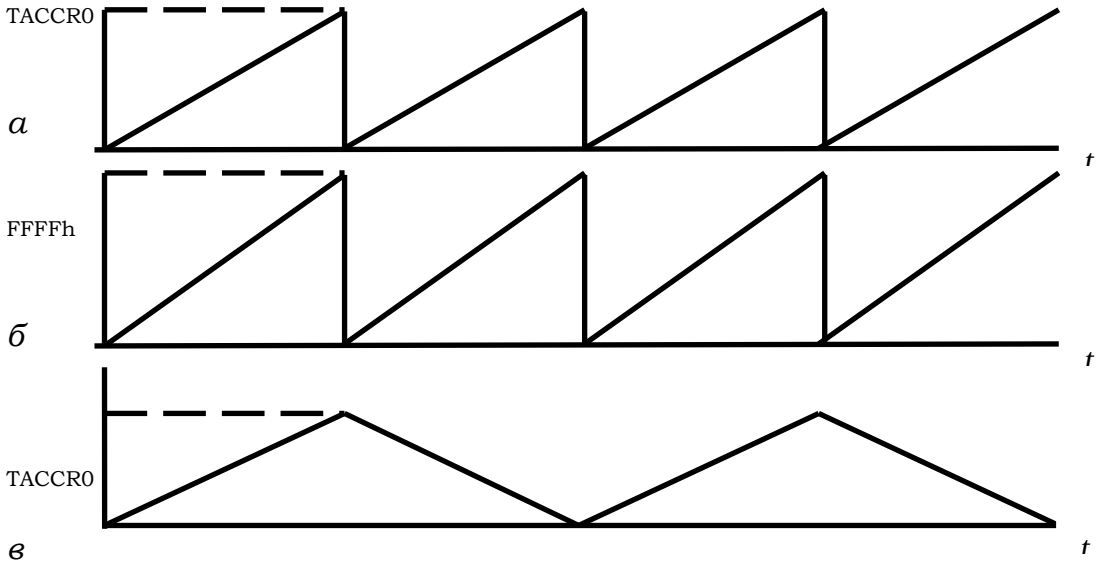


Рисунок 3.2 – Режимы работы таймера, *а* –прямой счет, *б* – непрерывный счет, *в* – реверсивный счет.

Режимом *сравнения* называют режим вывода события на заданном выходе микроконтроллера в заданный момент времени. Для этого значение регистра-счетчика таймера сравнивается с содержимым регистра модуля захвата-сравнения. Когда их значения совпадают, выводится событие. Режим сравнения является *режимом вывода события.* В частности, режим сравнения позволяет сформировать импульсы определенной длительности на выходе микроконтроллера.

Пример исходного текста программы *msp*430x22x4_*ta*_01.*asm* на языке ассемблера показан далее. Программа предназначена для включения и отключения светодиода с заданным периодом по сигналу прерывания, который генерирует таймер *TA*, что указано в комментарии перед текстом программы.

Программа содержит инициализационную часть, где задается адрес вершины стека, остановлен сторожевой таймер и настроен на вывод один бит порта *P*1. Далее разрешено прерывание модуля захвата-сравнения *TACCTL*0. В регистр *TACCR*0 записывается значение, пропорциональное интервалу времени, который необходимо сформировать. Выполняется инициализация таймера *TA*. Выполняется отключение *CPU* и разрешение прерывания. В результате происходит прерывание при каждом переполнении таймера. Поскольку прерывание выводит процессор из состояния останова, периодически выполняется подпрограмма прерывания. В подпрограмме прерывания организовано переключение младшего бита порта *P*1. Этим формируется попеременное включение и отключение светодиода операцией *xor* (исключающее ИЛИ).

```
#include "msp430x22x4.h"
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  P1DIR = 0xFF;                          // P1.0 output
  TACCTL0 = CCIE;                          // TACCR0 interrupt enabled
  TACCR0 = 50000;
  TACTL = TASSEL_2 + MC_2;                  // SMCLK, contmode
  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0 w/ interrupt enabled
 }
// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  P1OUT ^= 0x01;                          // Toggle P1.0
  TACCR0 += 50000;                          // Add Offset to TACCR0
  }
```

В тексте программы значения настроек #TASSEL_2, MC_2 и другие использованы для выбора режимов таймера в соответствии с таблицами из справочника [4].

1.3.2. Аналого- цифровое и цифро-аналоговое преобразования.

Принцип действия цифро-аналоговых преобразователей *(ЦАП, DAC, Digital Analog Converter).* Цифро-аналоговый преобразователь предназначен для преобразования двоичного кода в пропорциональный сигнал электрического напряжения. Цифро-аналоговый преобразователь показан на рисунке 3.3. Двоичный код хранится в параллельном виде в регистре *RG.* К выходным битам регистра подключены входы операционного усилителя *A* через резисторы R0…Rm-1, где m – количество разрядов двоичного кода. В точке В реализуется суммирование токов через резисторы.

Аналого-цифровой преобразователь (АЦП, ADC**)** предназначен для преобразования изменяющейся во времени аналоговой величины на входе АЦП в двоичный код. В результате преобразования формируется 10-,12- или 16- –разрядные двоичные значения, пропорциональные входной величине. Запуск аналого-цифрового преобразования может быть программным или от таймера. АЦП поразрядного уравновешивания показан на рисунке 3.4 показана функциональная схема поразрядного уравновешивания.

Входной аналоговый сигнал u^* сравнивается с выходным сигналом *u цифро-аналогового преобразователя* ЦАП (DAC) компаратором А. Если $u^* - u >$

0, на выходе компаратора формируется +1, и счетный регистр инкрементируется.

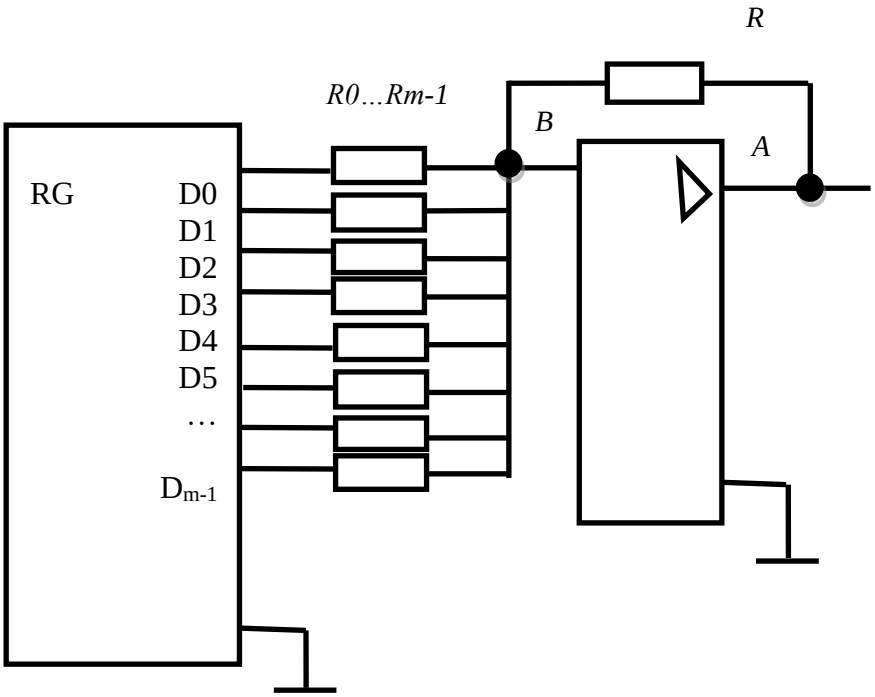


Рисунок 3.3 – Цифро-аналоговый преобразователь

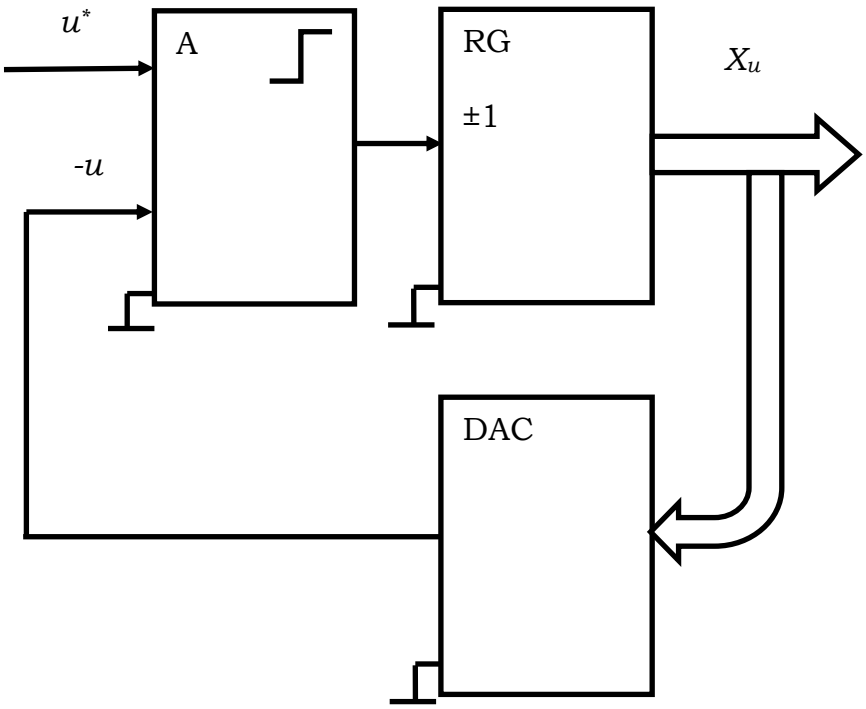


Рисунок 3.4 – АЦП поразрядного уравновешивания

Если $u^* - u > 0$, на выходе компаратора формируется -1, и счетный регистр декрементируется. Если $u^* - u = 0$, на выходе компаратора 0, и содержимое регистра не изменяется, что означает окончание преобразования входной аналоговой величины u^* в двоичный код X_u. Поразрядное уравновешивание просто реализуется, однако имеет очевидный недостаток: чем больше входная аналоговая величина, тем больше время преобразования. Время преобразования значительно меньше при использовании метода последовательного приближения.

В АЦП последовательного приближения (рисунок 3.5) используется алгоритм последовательного приближения.

1 На выходе формируется начальное условие $X_u = 0$. При появлении на входе u^* рассчитывается значение $e_1 = \text{sign}(u^* - u)$ на первом шаге, $k = 1$.

2 Если e = +1, то $X_u = X_m$, где $X_m = 2^{m-1}$, где m – количество разрядов АЦП. В результате на выходе ЦАП формируется значение, наибольшее возможное. 3 Далее повторяется цикл:

$k = k + 1$ и, если $e_k = 0$, преобразование считается оконченным.

Если e = -1, то $X_{uk} = X_{uk-1} - 2^{m-k}$,

Если e = +1, то $X_{uk} = X_{mk-1} + 2^{m-k}$,

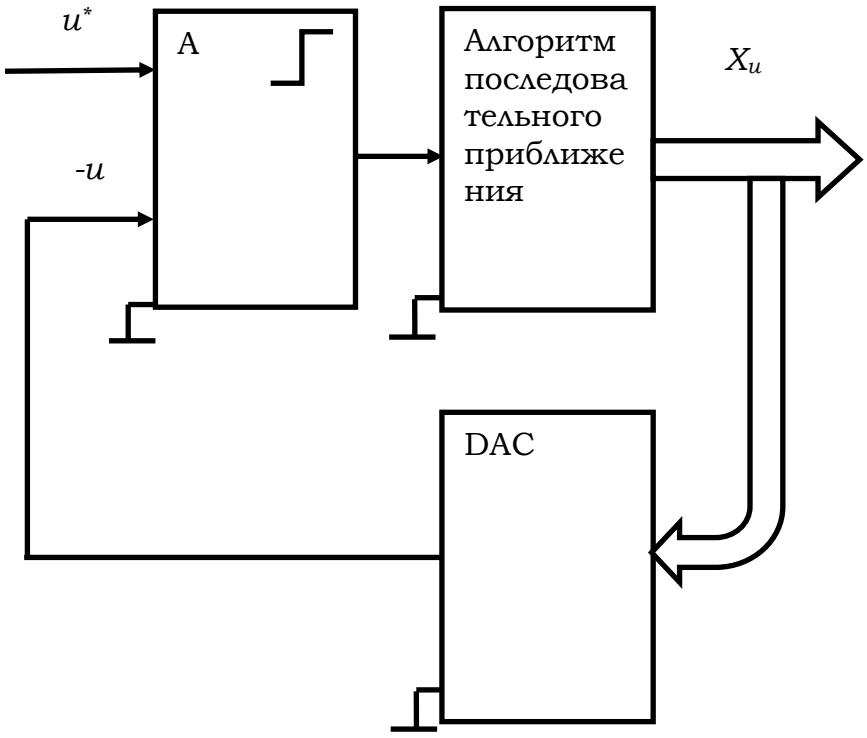
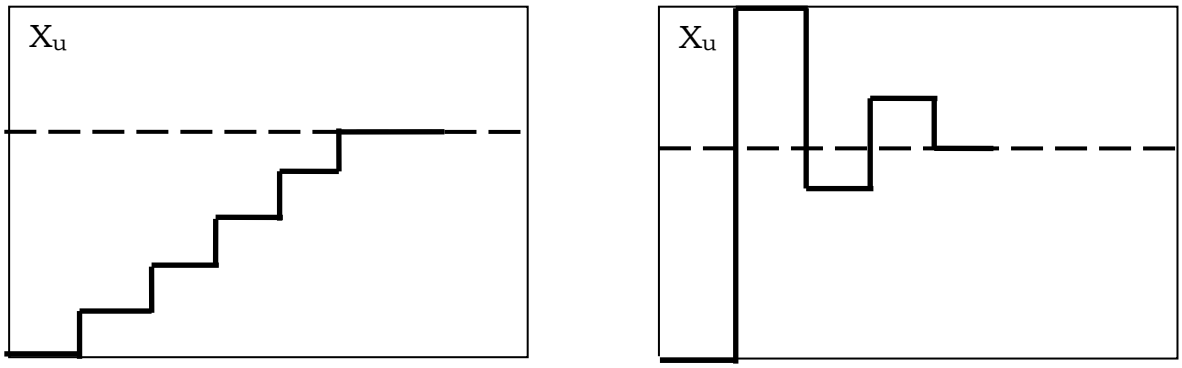


Рисунок 3.5 – АЦП последовательного приближения

Примеры того, как изменяется результат преобразования во время цикла, показаны на рисунке 3.6, из графиков видно, что при поразрядном уравновешивании преобразование происходит за количество шагов, равное преобразуемой величине в двоичном коде. Метод последовательного приближения позволяет значительно сократить время преобразования больших значений.



а б

Рисунок 3.6 – Процесс преобразования методом поразрядного уравновешивания (а) и последовательного приближения (б)

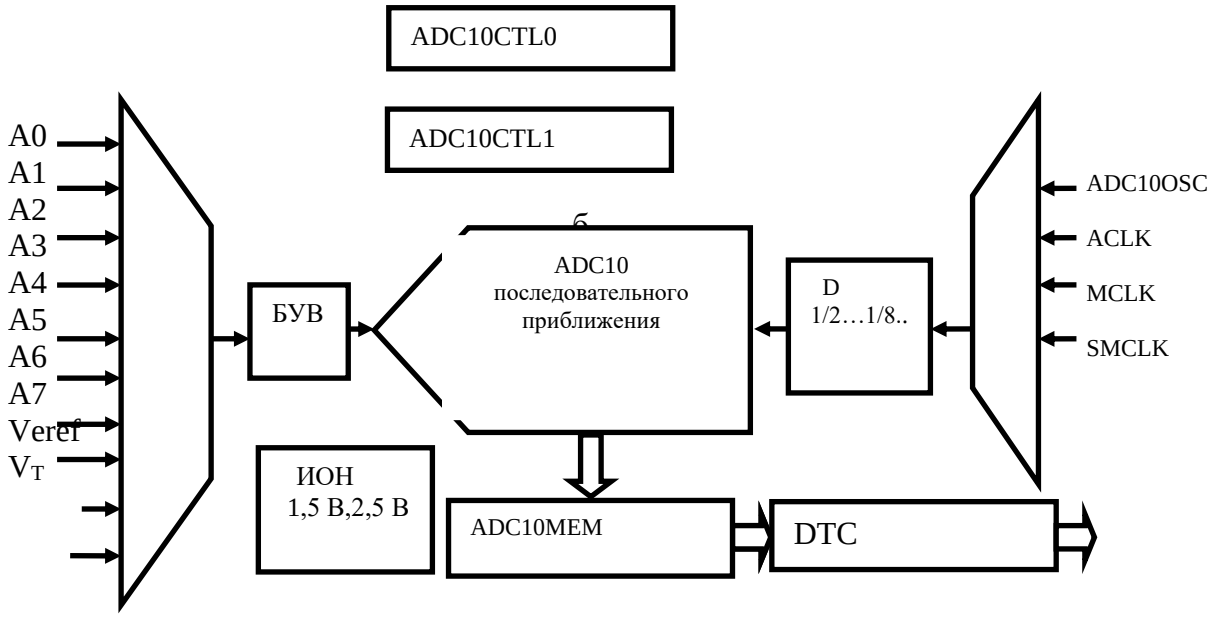


Рисунок 3.7 – Функциональная схема **ADC10.**

Запуск аналого-цифрового преобразования производится по нарастающему фронту сигнала выборки SHI. Источником сигнала SHI, который задаётся битами SHSx, может быть: бит ADC10SC; один из модулей вывода Таймера А. Биты SHTx определяют период выборки *ts*, который может быть равен **4, 8, 16 или 64 тактам** ADC10CLK. Минимальное время выборки для выполнения 10-битного преобразования можно определить по формуле:

$$tsample > (R_S + R_I)\cdot\ln(2^{11})\cdot C_I.$$

Для выполнения аналого-цифрового преобразования необходим опорный сигнал. Микросхема содержит встроенный генератор опорного сигнала, позволяющий сформировать два значения напряжения. Включение генератора опорного сигнала выполняется установкой бита **REFON**=1. Если бит

Ref2_5V=1, внутреннее опорное напряжение равно 2,5 V, а при **Ref2_5V=0** внутреннее опорное напряжение равно 1,5 V.

Для инициализации аналого-цифрового преобразователя **ADC10** необходимо в управляющие регистры записать информацию о выбираемых режимах работы. Если выборка и преобразование запускаются от таймера, необходимо так же инициализировать таймер, записывая в его управляющие регистры информацию о режиме работы.

1.3.3. Последовательный интерфейс в микроконтроллерах

Микроконтроллеры содержат модули универсального последовательного коммуникационного интерфейса (например, в микроконтроллерах *MSP*430 *USCI***,** *Universal serial communication interface*), настраиваемые на различные режимы передачи данных [4]. Это режим *UART* (*Universal asynchronous receiver-transmitter*) универсального асинхронного приёмопередатчика, режим *SPI* (*Serial programmable interface*) последовательного программируемого интерфейса микроконтроллеров и режим двухпроводного интерфейса *I2C***,** предложенного фирмой Philips**.**

Режим *UART* универсального асинхронного приёмопередатчика совместим со стандартным последовательным интерфейсом *RS*232 персональных компьютеров. Режим *UART* универсального асинхронного приёмопередатчика организует обмен информации символами, обрамляемыми старт-битом и стоп-битом (старт-стопный ввод-вывод). Символы объединяются в блоки. Обычно первым символом блока является адрес, далее передаются данные.

Например, в режиме *UART* асинхронного приёмопередатчика модули USCI_Ax микроконтроллеров *MSP*430 позволяют подключать внешние устройства к выводам UCAxRXD приемника и UCAxTXD передатчика. Режим *UART* имеет следующие особенности: 7- или 8-битные данные с контролем чётности, нечётности или без контроля; изменяемый порядок передачи и приёма битов; встроенная поддержка коммуникационных протоколов *idle-line* (неактивная линия) и *address-bit* (адресный бит) для многопроцессорных систем; обнаружение приёмником фронта старт-бита для автоматического выхода из режимов пониженного энергопотребления LPMx; программируемая скорость обмена; флаги обнаружения и игнорирования ошибок; флаг обнаружения адреса; независимые прерывания передачи и приёма. Блок-схема модуля *USCI_Ax*, сконфигурированного для работы в режиме *UART*, приведена на рисунке 3.8.

В режиме *UART* модуль *USCI* передаёт и принимает данные с заданной скоростью асинхронно по отношению к другому устройству. Синхронизация приёма/передачи каждого символа осуществляется на основе выбранной скорости обмена модуля. Блоки передачи и приёма используют одно и то же значение скорости обмена.

Для случаев, когда обмен осуществляется между тремя и более устройствами, модулем *USCI* поддерживаются два формата многопроцессорного обмена: *idle-line* (неактивная линия) и *address-bit* (адресный бит).

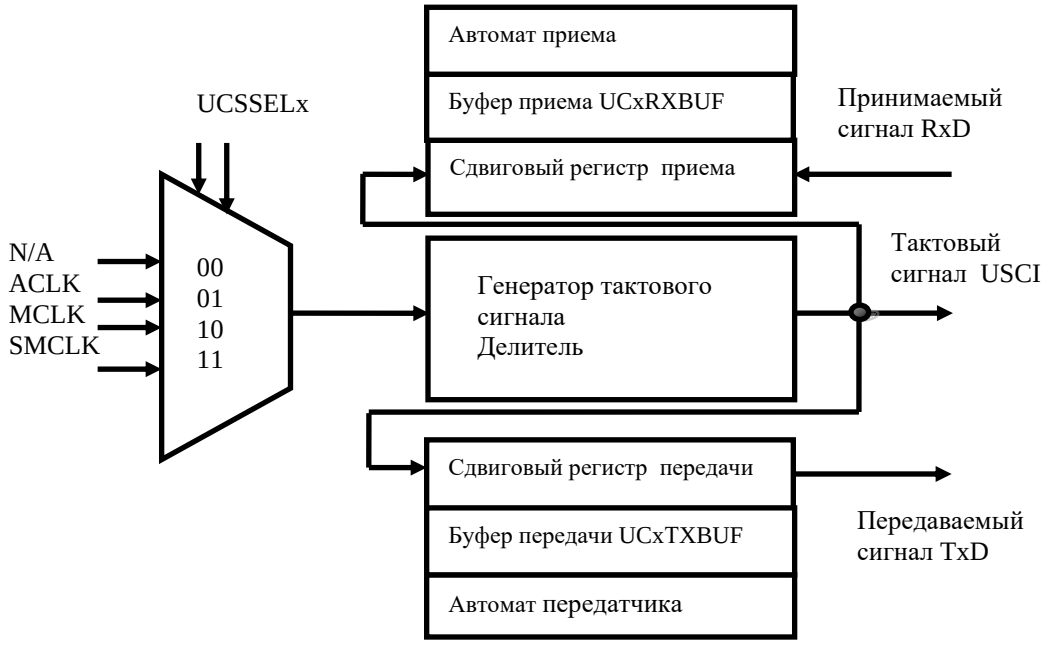


Рисунок 3.8. Функциональная схема последовательного интерфейса *USCI.*

Формат многопроцессорного обмена *idle-line* (неактивная линия) выбирается при *UCMODEx* = 01. При использовании этого формата блоки данных разделяются периодом неактивности на линиях передачи или приёма, как показано на рисунке 3.9.

Символ, передаваемый *UART*, содержит старт-бит *ST*, биты адреса *A*0 *A*15, бит чётности *PA*, один или два стоп-бита *SP*, период неактивности длительностью менее 10 бит, и далее очередной символ блока: старт-бит *ST*, семь или восемь битов данных *D*0…*D*7, бит чётности *PD*, а также один или два стоп-бита SP,и так далее. Периоды неактивности длиной 10 бит или более разделяют блоки символов.

Рисунок 3.9. Формат обмена в режиме *idle-line.*

Формат многопроцессорного обмена *address-bit* выбирается при *UCMODEx* = 10. При использовании этого формата передаваемые символы содержат дополнительный бит адреса A, как показано на рисунке. 3.10. Первый символ передаваемого блока содержит установленный бит адреса A=1,

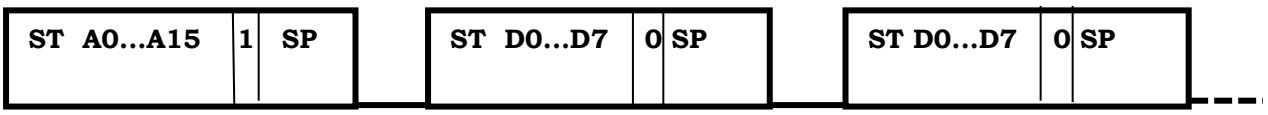


Рис. 3.10 - Формат многопроцессорного обмена *address-bit.*

указывающий, что данный символ является адресом.

При приёме символа с установленным битом адреса и перемещении его в регистр UCAxRXBUF устанавливается бит UCADDR. Отсутствие активности между передаваемыми символами не учитывается.

Режим *SPI* **с**инхронного трехпроводного интерфейса применяется для соединения двух микроконтроллеров на параллельную работу в режиме ведущего и ведомого (*Master-Slave*). Синхронизация достигается тактовой частотой ведущего устройства, которая передается к ведомому устройству. Еще два проводника необходимы для приема и передачи данных, как показано на рисунке 3.11.

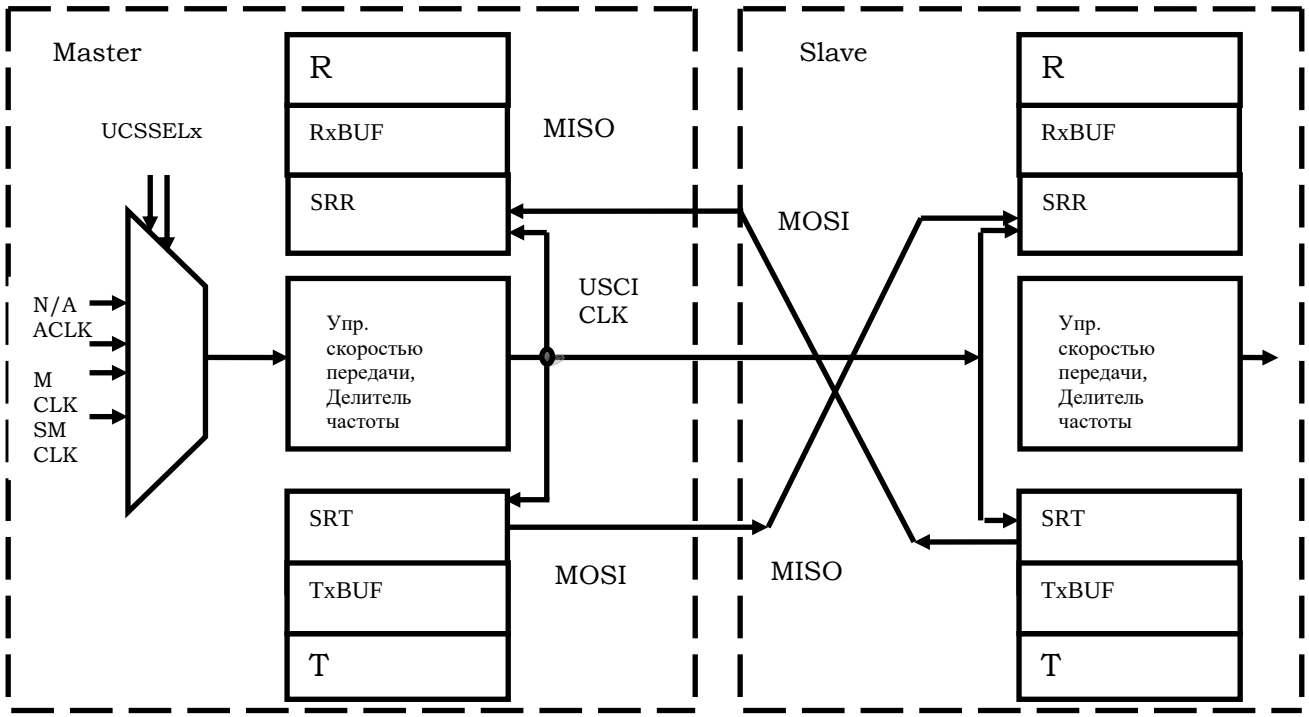


Рисунок 3.11 - Функциональная схема параллельной работы двух микроконтроллеров, соединенных *SPI***.**

Как видно из схемы, выход *MOSI* (*Master OUT – Slave IN)* передатчика ведущего устройства соединен со входом *MOSI* приемника ведомого. Вход *MISO* (*Master IN – Slave OUT приемника)* ведущего устройства соединен с выходом *MISO* передатчика ведомого устройства. Таким образом, приемник и передатчик в этом режиме соединены одноименными внешними выводами.

В режиме *I2C* модуль *USCI* обеспечивает взаимодействие микроконтроллеров и различных *I2C*-совместимых устройств, подключённых к 2- проводной последовательной шине *I2C.* Внешние устройства, подключённые к шине, передают и (или) принимают в последовательном виде данные с помощью 2-проводного интерфейса *I2C*.

Режим *I2C* соответствует спецификации версии 2.1 *Philips Semiconductor.* Возможны 7 и 10-битный режимы адресации; режимы ведущий-передатчик и ведущий-приёмник. Обнаружение ведомым-приёмником состояния СТАРТ служит для автоматического выхода из режимов

Пример использования шины *I2C* показан на рисунке 3.12. Каждое устройство на шине имеет уникальный адрес и может выступать в качестве

либо передатчика, либо приёмника. Ведущее устройство инициирует процесс пересылки данных и формирует тактовый сигнал *SCL*. Любое устройство, адресованное ведущим, рассматривается как ведомое. Обмен по шине **I2C** осуществляется с использованием двух линий: линии данных и адреса (*SDA*) и линии синхронизации (*SCL*). Обе линии являются двунаправленными и должны быть подключены к источнику постоянного *напряжения через подтягивающие резисторы R1, R2.*

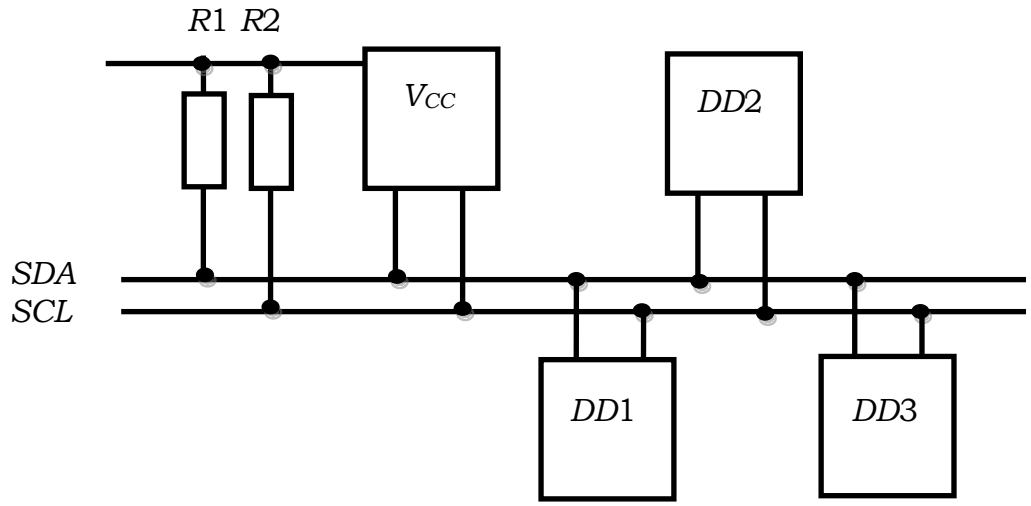


Рисунок 3.12. Соединение устройств с помощью шины *I2C.*

Для каждого передаваемого по шине *I2C* бита данных ведущий генерирует один тактовый импульс. В режиме *I2C* обмен осуществляется побайтно (рисунок 3.13). Первый байт, передаваемый после состояния СТАРТ, содержит 7-битный адрес ведомого и бит направления передачи *R/W*. При *R/W* = 0 ведущий передаёт данные ведомому. При *R/W* = 1 ведущий принимает данные от ведомого. После каждого байта приёмник в период 9-го импульса *SCL* передаёт бит квитирования *ACK*. Состояния СТАРТ и СТОП формируются ведущим устройством. Состоянию СТАРТ соответствует изменение уровня на линии *SDA* с высокого на низкий при высоком уровне на линии *SCL*. Состоянию СТОП соответствует изменение уровня на линии *SDA* с низкого на высокий при высоком уровне на линии *SCL*. Флаг занятости *UCBBUSY* устанавливается после формирования состояния СТАРТ и сбрасывается после формирования состояния СТОП.

Сигнал на линии *SDA* должен быть неизменным в течение всего времени, пока на линии *SCL* присутствует высокий уровень. Уровень сигнала на линии *SDA* может изменяться только при низком уровне на линии SCL, в противном случае произойдет формирование состояния СТАРТ или СТОП

Рисунок 3.13 – Передача данных по шине *I2C*

Преимущество интерфейса *I2C* в том, что используется всего два провода, и возможен обмена информацией со многими устройствами. Однако в сравнении с *SPI* для передачи информации требуется значительно большее

время, поскольку по одному проводу передается и адрес, и данные в двух направлениях поочередно, а одновременные прием и передача, как в *SPI*, невозможна.

Модуль последовательного интерфейса, работая в любом из описанных режимов, имеет два вектора прерывания, *для передачи и для приема*.

Запрос прерывания *приема* генерируется в момент готовности принятых данных в буфере *RxBUF* приема. В подпрограмме прерывания приема данные из буфера *RxBUF* могут быть скопированы и использованы по назначению.

Например, в микроконтроллере *MSP*430 флаг прерывания *UCAxRXIFG* устанавливается каждый раз при приёме символа и загрузке его в регистр *UCAxRXBUF*. При установленных битах *GIE* и *UCAxRXIE* генерируется запрос прерывания. Флаг *UCAxRXIFG* автоматически сбрасывается при чтении регистра *UCAxRXBUF*.

Запрос прерывания *передачи* генерируется в момент окончания передачи и готовности буфера *TxBUF* передачи к записи новых данных. В подпрограмме прерывания передачи новые данные могут быть записаны в буфер *TxBUF*.

Например, в микроконтроллере MSP430 флаг прерывания *UCAxTXIFG*, устанавливаемый передатчиком, означает готовность регистра *UCAxTXBUF* к загрузке нового символа. Если установлены биты GIE и *UCAxTXIE*, то при установке флага *UCAxTXIFG* генерируется запрос прерывания. Флаг *UCAxTXIFG* автоматически сбрасывается при записи в регистр *UCAxTXBUF*.

Контрольные вопросы и задачи

1 Что такое магистраль?
2 Каковы области применения параллельного и последовательного интерфейсов?
3 Как работает вычислительное устройство в режиме прерывания?
4 Для чего нужен прямой доступ к памяти?
5 Каков принцип действия программируемого таймера?
6 Когда таймер запрашивает прерывание?
7 Каков принцип действия цифро-аналогового преобразователя?
8 Каков принцип действия аналого- цифрового преобразователя ()?
9 Когда АЦП запрашивает прерывание?
10 Каков принцип действия устройства последовательного интерфейса?
11 Когда запрашивает прерывание приемник и передатчик?
12 Каковы области применения последовательного интерфейса?
13 Составить алгоритм и программу ввода данных через ADC и их вывода через интерфейс UART.
14 Организовать формирование ШИМ таймером.

1.4. Развитие архитектуры микроконтроллеров

1.4.1. Направления развития архитектуры

Микроконтроллеры и микропроцессоры совершенствуются, начиная от 1980-х годов, двумя путями: за счет развития полупроводниковых технологий и развития архитектуры. Уровень развития полупроводниковых технологий измеряют двумя показателями. *Проектная норма* равна размеру одного транзистора в микросхеме. С*тепень интеграции* равна количеству элементов в одном кристалле (одной микросхеме). В 1980-х годах проектная норма была 4 мкм, а степень интеграции достигала 10^6. Например, восьмиразрядный микроконтроллер Intel 8051, созданный в 1980- х годах, содержит $128*10^3$ транзисторов. В настоящее время проектная норма измеряется нанометрами. Начиная от 1980-х годов, выявилось отставание полупроводниковых технологий от потребностей вычислительных систем, что потребовало развития архитектуры. Классической является архитектура CISC (Complete Instruction Set Computer) процессоров с полным набором команд. Примеры микроконтроллеров с классической архитектурой процессора представлены на рисунках 4, 5.

Архитектура RISC (Reduced Instruction Set Computer) с сокращенным набором команд является дальнейшим развитием классической архитектуры и начала разрабатываться в 1980-х годах с целью повышения производительности.

Сокращенный набор команд формируется по условию его функциональной полноты из наиболее используемых команд. Это позволяет упростить дешифрацию команд и алгоритм командного цикла, этим упростить процессор и ускорить его работу. Внутренняя структура процессора RISC характеризуется большим количеством внутренних регистров (16 и более), наличием многоступенчатого конвейера и кэш-памяти. Основная часть команд должна выполняться с внутренними регистрами ядра, и есть команды, обеспечивающие доступ к памяти и другим периферийным устройствам. Обычно совершенствование архитектуры идет в направлении интенсификации использования аппаратных ресурсов микросхемы за счет параллелизма.

Параллелизм на уровне команд введен для повышения коэффициента использования аппаратных средств процессора в командном цикле (рисунок 2) и достигается применением конвейерного выполнения команд. В самом деле, в классической архитектуре на этапе выборки команды задействована магистраль и регистр команд, а остальные устройства (PC, дешифратор, ALU) простаивают. И далее, на каждом этапе командного цикла (рисунок 2) есть не используемые устройства.

Для преодоления этого недостатка процессор должен содержать специальное аппаратное средство, а именно *многоступенчатый конвейер*. Так обеспечивается параллельное выполнение одновременно нескольких команд. Принцип действия трехступенчатого конвейера показан на рисунке 4.1.

ВЫБОРКА	Команда1	Команда2	Команда3	Команда4	…
ДЕШИФРАЦИЯ		Команда1	Команда2	Команда3	Команда4
ВЫПОЛНЕНИЕ			Команда1	Команда2	Команда3

Рисунок 4.1 - Принцип действия конвейера

В первой фазе происходит выборка команды1 из памяти программ в регистр команд по магистрали, инкрементируется программный счетчик *PC*. Во второй фазе выполняется дешифрация, и в третьей – выполнение. После выборки команды1 магистраль освобождается, что позволяет выполнять выборку команды2 одновременно с дешифрацией команды1, и дешифрацию команды2 в фазе выполнения команды1, причем в этой же фазе происходит выборка команды3.

Итак, трехступенчатый конвейер организует одновременное выполнение трех команд в трех фазах командного цикла. Каждая фаза требует, как минимум, одного тактового периода. В то время, как классическая архитектура допускает минимальное время выполнения команды в три такта, то при наличии трехступенчатого конвейера, три команды выполняются за три такта. Таким образом, среднее время выполнения одной команды составляет как минимум один тактовый период, но это при регистровой адресации. Другие виды адресации так правило требуют большего числа тактов выполнения команды, однако наличие 3-х ступеней конвейера в три раза сокращает время выполнения команд.

Следует отметить, что на первых этапах развития архитектуры в 1980-х годах предпочтение отдавалось так называемым *суперскалярным (с* многоступенчатым, 5 и более ступеней конвейером*)* и *суперконвейерным* (с несколькими конвейерами) архитектурам. Очевидно, такая архитектура позволяет во много раз сократить среднее время выполнения одной команды. Вместе с тем конвейерное выполнение команд имеет недостаток: эффективность резко снижается при наличии в алгоритме ветвлений, обращений к подпрограммам и при работе в режиме прерываний, поскольку конвейер приходится каждый раз сбрасывать. На рисунке 4.1 видна работа конвейера после сброса: в первых двух фазах выполняются одна и две команды вместо трех. Очевидно, чем больше ступеней в конвейере, тем значительнее снижение эффективности при сбросах. Поэтому в последнее время признаны оптимальными 3-х ступенчатые конвейеры в сочетании со средствами эффективного выполнения ветвлений и переходов к подпрограммам.

Кэш-память (Cash memory) предназначена для буферного хранения информации (команд, данных) в процессе выполнения программы и имеет малое время доступа. Требуемый объем кэш-памяти значительно меньший, чем основной памяти. Основная оперативная память RAM обычно выполняется как динамическая, DRAM, где физической единицей хранения информации является емкость. Такая память имеет низкую стоимость, однако сравнительно большое время доступа из-за необходимости регенерации после чтения информации. Кэш память выполняется как статическая (S RAM), где

физической единицей хранения информации является триггер, что обеспечивает малое время доступа.

Кэш-память хранит используемый в текущем периоде фрагмент информации, скопированный из основной памяти. Принцип действия вычислительного устройства с кэш-памятью показан на рисунке 4.2.

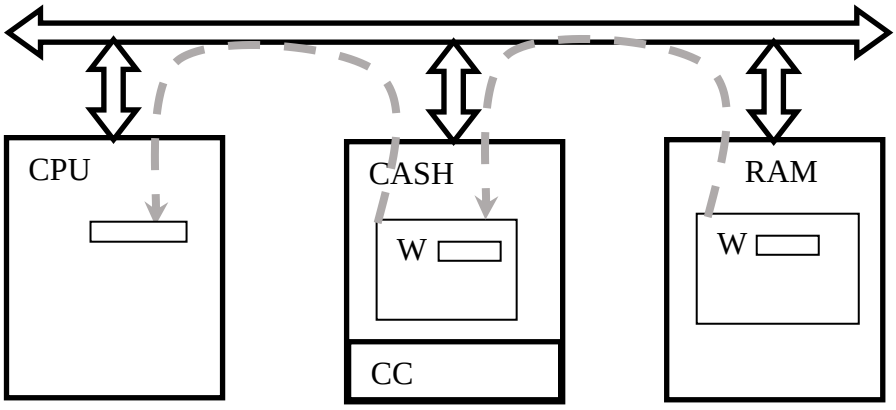


Рисунок 4.2 - Принцип действия вычислительного устройства с кэш-памятью

В фазе выборки микроконтроллер обращается не в основную память, а в кэш, и, если находит требуемое слово W данных, то экономится время выборки. Если же требуемое слово данных не оказывается в кэш, происходит повторное обращение, но уже в основную память. Однако из основной памяти извлекается в кэш не адресуемое слово W, а содержащий его фрагмент (рисунок 4.2). Это повышает вероятность найти нужную информацию в кэш-памяти при следующих обращениях. Кэш память снабжена контроллером кэш CC, который управляет записью и чтением информации, а также учитывает модифицированную в кэш информацию для записи в основную память.

Таким образом, RISC архитектура отличается от классической CISC архитектуры сокращенным набором команд, большим количеством внутренних регистров (16 и более). Быстродействие повышается также использованием многоступенчатого конвейера и кэш-памяти.

Программирование для устройств с RISC архитектурой, содержащих конвейер, желательно выполнять с минимальным количеством ветвлений в алгоритме. Например, вычисление $u = f_1$ если $x < a$, и $u = f_2$ если $x = a$ или $x > a$ обычно выполняется по двум ветвям в алгоритме. Однако в данном случае можно избежать ветвления, если использовать логические выражения

$$u = f_1 * (x < a) + f_2 * (x <= a).$$

В данном случае при выполнении логического условия в скобках выражение в скобках принимает значение *TRUE*, равное единице типа *integer,* а при невыполнении условия - значение *FALSE* = 0. Таким образом, величина u будет равна либо f_1, либо f_2 в зависимости от условий.

Параллелизм на уровне алгоритмов возможен в многопроцессорных (многоядерных) устройствах. Это обычно 2-х ядерные, 4-х ядерные микроконтроллеры, а также устройства, содержащие процессорную матрицу.

В многоядерных (многопроцессорных) устройствах каждое из ядер (каждый процессор) служит для решения определенного типа задач. Например, в двухъядерном микроконтроллере TMS320M28xxx один из процессоров выполняет расчеты по программам, а другой процессор обслуживает периферийные устройства (устройства ввода и вывода информации).

Микроконтроллер может содержать *процессорную матрицу M.* Структура микроконтроллера с процессорной матрицей представлена на рисунке 4.3.

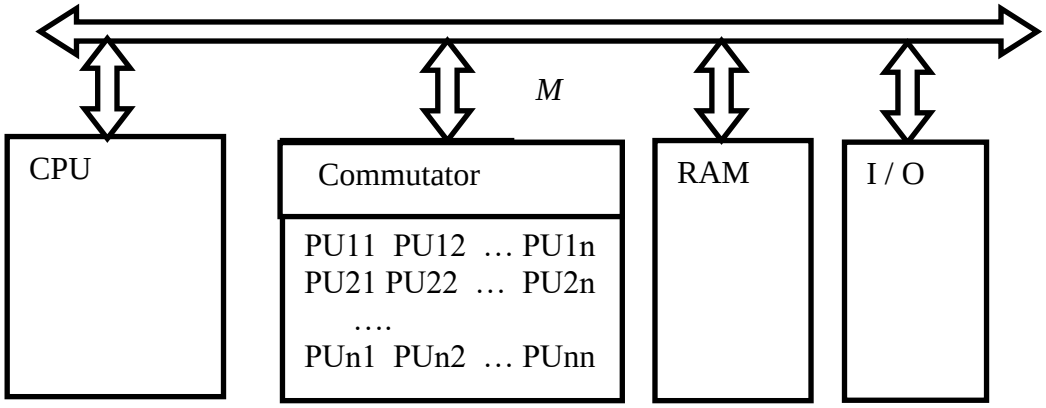


Рисунок 4.3 - Структура вычислительного устройства с процессорной матрицей

Процессорная матрица размером *mxn* содержит *mxn* процессоров, каждый из которых специализирован для выполнения определенного алгоритма. Это позволяет в *mxn* раз сократить время выполнения операций с матрицами и векторами. Процессорная матрица работает в вычислительном устройстве под управлением центрального процессора CPU, с которым связана через магистраль и коммутатор.

1.4.2. Микроконтроллеры архитектуры *ARM*.

Архитектура *ARM* микроконтроллеров предложена в 1990-е годы британской фирмой *ARM Ltd* (Кембридж, Великобритания, www.arm.com) по проектированию микропроцессоров, графических процессоров и нейропроцессоров [16].

Преимущества архитектуры *ARM* заключаются в большем быстродействии, достигаемом рационализацией конвейера, сокращением времени доступа к памяти и усовершенствованной структурой команды. В микроконтроллерах *ARM* применяется конвейер команд с наиболее рациональным количеством ступеней, равным 3. Эффективность конвейера снижается при наличии в алгоритме ветвлений и обращений к подпрограммам. Поэтому в ядре ARM предусмотрены меры повышения

эффективности. Имеются специальные регистры адреса возврата из подпрограмм, что способствует быстродействию. Для улучшения эффективности конвейера структура команды усовершенствована, как показано на рисунке 4.4. Биты *NZCV* являются префиксом, дополняющим классическую структуру команды из кода операции КО и адресной части А.

Рисунок 4.4 – Структура команды процессоров *ARM*

Становится возможным условное выполнение команды, что позволяет избежать ветвления в алгоритме. Если в команде установлен в 1 один из битов *N, Z, C, V,* это означает, что команда должна выполняться только при этом условии. Для этого выполняется сравнение битов *N, Z, C, V* команды и регистра состояния ядра. Таким образом, если при выполнении предыдущих команд установлен соответствующий флаг, то команда должна быть выполнена, иначе команда не выполняется и происходит переход к следующей команде. Например, префикс EQ. означает, что для выполнения команды флаг Z должен быть установлен, поэтому в случае команд

```
SUB #01, R7
EQMOV R8, R9
```

установка флага Z после выполнения первой команды приведет к выполнению второй команды, иначе вторая команда не выполнится.

В ядре *ARM* предусмотрена *арифметика с насыщением*, что позволяет учитывать ограничения на переменные системы при расчете сигнала управления. Все операции учитывают ограничение результата заданным нижним и верхним значениями.

Например, если значения ограничены интервалом (−100, 100), получатся следующие результаты: 50 + 30 → 80; однако 50 + 73 → 100. (не 123). В случае нескольких операций: (50 + 53) − (55 + 70) → 0. (*не* −22, так как 100 − 100 → 0). При умножении: 9 × 13 → 100. (*не* 117); 97 × 10 → 100. (*не* 970.) 30 × (7 − 31) → 100. (*не* 120). Как видно из примеров, ассоциативность и дистрибутивность не справедливы для арифметики с насыщением.

На рисунке 4.5 представлена схема функциональная микроконтроллера архитектуры *ARM.* Структура содержит устройства последовательного интерфейса USART, SPI, USB, а также CAN и Ethernet, и устройство ввода ADC аналого- цифрового преобразования сигнала. Ядро ARM7 соединено через контроллер памяти (Memory controller) с периферийными устройствами: внутренней памятью SRAM, Flash и ROM и с устройствами ввода-вывода USART, SPI, USB, CAN и Ethernet. Для связи с устройствами ввода-вывода используется шина APB Bus ускоренной передачи данных через Peripherial Bridge. На основании архитектуры *ARM* разработана группа 32-битных RISC процессоров *ARM Cortex-M,* лицензированная Arm Holdings, которые широко применяются в микроконтроллерах и других микросхемах. Это Cortex-M0, Cortex-M0+, Cortex-M1, Cortex-M3, Cortex-M4, и другие. Многие процессоры, например Cortex-M4, содержат сопроцессор FPU для данных с плавающей точкой.

Процессоры *ARM* применяются в устройствах реального времени, в том числе во встроенных микроконтроллерах, телефонах, смартфонах, компьютерах и компьютерных сетях и везде, где необходимы высокопроизводительные устройства с низким энергопотреблением.

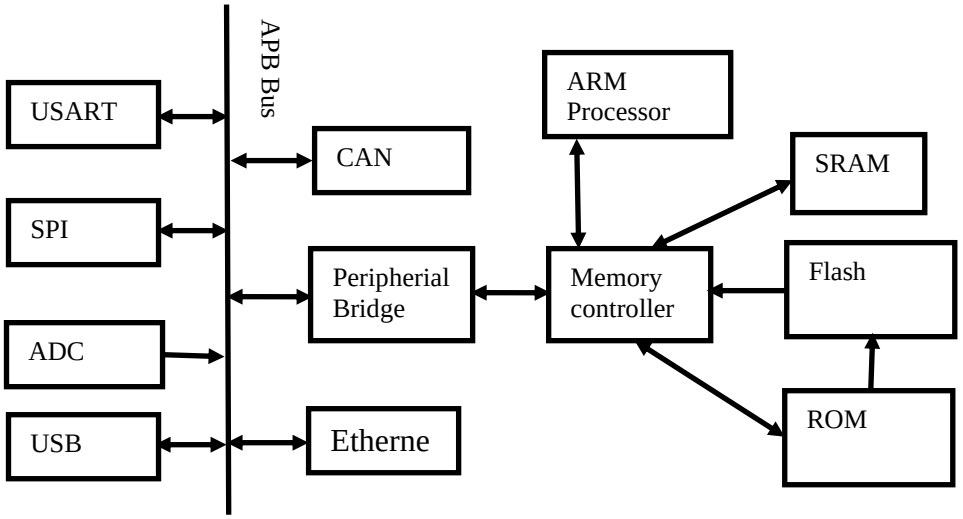


Рисунок 4.5 – Функциональная схема микроконтроллера архитектуры ARM

1.4.3. Микроконтроллеры управления электроприводами

Системы управления электроприводами во многих случаях требуют управления моментом (усилием), скоростью и положением рабочего органа. Для этого электродвигатель получает напряжение от полупроводникового управляемого преобразователя электрической энергии, что делает электропривод регулируемым. За последние десятилетия регулируемые электроприводы переменного тока получают все большее применение, что способствует ресурсо- и энергосбережению. Для управления электроприводами применяются специализированные микроконтроллеры [17] – [19], значительная сложность которых обусловлена высокими требованиями к системе управления электроприводом.

Микроконтроллер управления электроприводом должен удовлетворять следующим требованиям.

Слово данных требуется не менее 32 разрядов с возможностью операций с плавающей точкой, чтобы обеспечить точное формирование сигнала управления.

Производительность микроконтроллера требуется для обеспечения быстрой реакции системы на возмущения и достигается не только высокой тактовой частотой, но и усовершенствованной архитектурой RISC, в том числе системой команд, специализированной для управления.

Периферийные устройства микроконтроллера должны обеспечивать ввод аналоговых сигналов обратных связей, сигналов обратной связей по положению от инкрементального датчика положения, ввод логических битовых сигналов. Должны быть возможности для вывода сигналов управления полупроводниковыми силовыми ключами с широтно импульсной модуляцией (ШИМ) различных видов.

Необходим обмен информацией по последовательному интерфейсу.

Во многих случаях микроконтроллеры управления электроприводами имеют стандартные промышленные последовательные интерфейсы или интерфейсы, совместимые с ними.

Система команд микроконтроллера должна быть пригодна для применения языков программирования высокого уровня.

Специализированные микроконтроллеры [17] – [19] управления электроприводами обычно имеют архитектуру RISC и содержат элементы архитектуры ARM, что способствует производительности.

Контрольные вопросы

1. Каковы особенности архитектуры *RISC*?
2. Что такое конвейер?
3. Какова функция кэш-памяти?
4. Что достигается сокращением системы команд?
5. Как достигается параллелизм вычислений?
6. Каковы преимущества архитектуры *ARM*?
7. Как в алгоритме избежать ветвлений?
8. Какова структура команды микроконтроллеров архитектуры *ARM*?
9. Каковы требования к микроконтроллерам управления электроприводами?
10. Каковы требования к устройствам ввода микроконтроллеров управления электроприводами?
11. Каковы требования к устройствам вывода микроконтроллеров управления электроприводами?
12. Как формируется ШИМ для управления силовыми ключами преобразователя электрической энергии?
13. Как обеспечивается точность в микроконтроллерах управления электроприводами?
14. Как обеспечивается быстродействие в микроконтроллерах управления электроприводами?
15. Каковы функции микроконтроллера в системе управления электроприводом?

1.5. Применение микроконтроллеров в автоматизации

Системы управления по выполняемым функциям делят на два класса: системы логического управления и замкнутые системы с обратными связями и регуляторами [20]-[22]. Разработка алгоритма выполняется одновременно с разработкой функциональной схемы системы управления.

Функциональная схема системы управления с микроконтроллером разрабатывается, когда определена цель управления, назначение и требования к системе. Определяются все функционально необходимые элементы системы. Сюда входят электродвигатели необходимого типа, силовые преобразователи, если необходимо регулирование скорости, датчики, реле, путевые и конечные выключатели, и другие аппараты, микросхемы. Формируются связи между элементами системы. Затем выполняется выбор периферийных устройств микроконтроллера и режимов их работы.

Наибольших ресурсов микроконтроллера требует векторное управление электродвигателями переменного тока, синхронными и асинхронными, в том числе с применением датчиков скорости и без датчиков скорости. Для этого ведущими производителями микросхем, в том числе Infineon, Analog Devices, STM, Texas Instruments и другими выпускаются специализированные микроконтроллеры для управления электроприводами и полупроводниковыми преобразователями электрической энергии. В системе векторного управления электроприводами микроконтроллер должен выполнять следующие функции:

- ввод сигналов задания и сигналов датчиков обратных связей;
- преобразования Кларка (от 3-х фазной к 2-х фазной системе) и Парка (от стационарной к синхронно вращающейся системе координат) для сигналов тока фаз статора электродвигателя;
- расчет сигналов управления на выходе регуляторов;
- обратные преобразования Парка и Кларка для сигналов управления;
- широтно-импульсная модуляция;
- вывод состояния ключей преобразователя на драйвер силовых ключей;
- тестирование и диагностика.

Алгоритм и программа должны иметь *инициализационную и циклическую* части.

Инициализационная часть обычно содержит настройку периферийных устройств микроконтроллера, в том числе интерфейсов для ввода и вывода сигналов, которые предполагает использовать разработчик, а также описание типов переменных и их начальные значения. Инициализационная часть выполняется один раз при начальном запуске системы.

Циклическая часть должна предусматривать периодический ввод информации с периферийных устройств и формирование сигналов управления на основании этой информации, вывод значений сигналов управления для воздействия на объект. Циклическая часть повторяется периодически неограниченное количество раз во время функционирования системы, и прекращает выполняться с отключением системы от источника электроэнергии.

Программная реализация управления требует цикличности программы. Периодический опрос состояния периферийных устройств (сигналов от коммутационных аппаратов, реле, датчиков) и периодическое формирование сигналов управления обеспечивает работу системы в реальном времени.

Каждый цикл должен предусматривать чтение входной информации, расчет значения сигнала управления и его вывод на объект управления.

Программно-аппаратный обмен информацией в процессе управления выполняется с использованием прерываний от периферийных устройств, что позволяет экономить ресурсы ядра, так как отпадает необходимость опроса периферийных устройств.

Аппаратный обмен информацией в процессе управления выполняется с использованием прямого доступа к памяти (ПДП, DMA) от периферийных устройств. Это позволяет передавать большие массивы информации между памятью и устройствами ввода-вывода, минуя процессор, что значительно сокращает время ввода-вывода. С другой стороны, процессор освобождается для выполнения других задач.

Содержание программного обеспечения циклической и инициализационной части замкнутой системы определяется в зависимости от структуры и вида регуляторов, которые необходимо реализовать. Поэтому алгоритм формирования сигнала управления разрабатывается после решения задачи синтеза замкнутой системы.

Задача синтеза замкнутой системы с цифровым управлением может быть решена точным, основанным на дискретной модели объекта управления, либо приближенным методом, основанным на непрерывной модели объекта. Дискретный метод приходится применять, когда интервал времени, в течение которого выполняется циклическая часть программы, сравним со временем переходного процесса в синтезированной системе. Тогда дискретные передаточные функции (ДПФ) регуляторов синтезируются на основании ДПФ объекта. Чем меньше время выполнения циклической части, тем меньше запаздывание в контуре регулирования, и тогда динамика цифровой системы приближается к динамике непрерывной системы. Приближенные методы допустимы, если время переходного процесса синтезированной системы во много раз превышает время выполнения циклической части программы.

В связи с этим, при проектировании управления целесообразно вначале определить допустимое время выполнения циклической части программы на основании требуемого быстродействия. С учетом требуемой точности управления это позволяет оценить пригодность конкретного типа микроконтроллера для выполнения данной задачи управления. Если время выполнения циклической части достаточно мало, можно применять методы синтеза для непрерывных систем, а иначе следует синтезировать систему как дискретную.

1.5.1. Цифровые фильтры, их передаточные функции, алгоритмы и программы расчета выходной величины

С целью дискретизации сигнала управления, следует перейти к z-преобразованию, или к системе разностных уравнений для объекта и регулятора. Для этого можно использовать приближенное выражение, связывающее переменную $\boldsymbol{z}$ и переменную $\boldsymbol{p}$ преобразования Лапласа.

$$z = e^{pT} \approx 1 + Tp + T^2 p^2 / 2! + T^3 p^3 / 3! + \ldots \approx 1 + Tp. \qquad (5.1)$$

В этом случае звено запаздывания z^{-1} приближенно заменяется апериодическим звеном

$$z^{-1} \approx \frac{1}{Tp+1}. \qquad (5.2)$$

Цифровое дифференцирование и интегрирование. Идеальное непрерывное дифференцирующее звено со входной величиной $u_1(t)$ и выходной величиной $u(t)$ описывается дифференциальным уравнением $du_1 / dt = u$ и имеет передаточную функцию p. Такое звено физически не реализуемо ввиду невозможности знать будущее изменение дифференцируемого сигнала $u_1(t)$. Для дискретизации производная заменяется отношением конечных разностей $u = \Delta u_1 / \Delta t = (u_{1k} - u_{1k-1}) / T$. Переходя к z-изображению по правилу

$$u(z) \Rightarrow u_k \quad z^{-i}u(z) \Rightarrow u_{k-i}\ , \qquad (5.3)$$

получим

$$(u_1(z) - z^{-1}u_1(z)) / T = (1 - z^{-1})u_1(z).$$

$$u(z) = z^{-1}u(z) + Tu_1(z).$$

Поэтому дискретные передаточные функции дифференцирующего и интегрирующего звеньев имеют вид

$$p \approx \frac{1 - z^{-1}}{T}, \quad \frac{1}{p} \approx \frac{Tz^{-1}}{1 - z^{-1}}.$$

На рисунках 5.1, 5.2 показана реакция дифференцирующего и интегрирующего звеньев на единичное ступенчатое воздействие u_1.

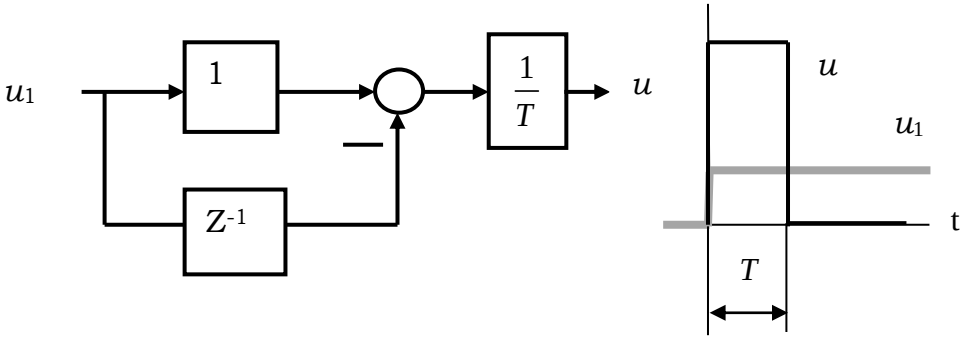


Рисунок 5.1 – Дифференцирующее звено и его реакция на единичное ступенчатое воздействие

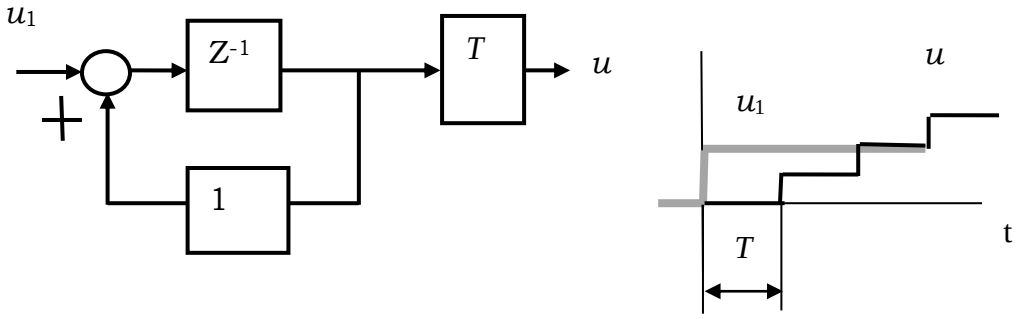


Рисунок 5.2 – Интегрирующее звено и его реакция на единичное ступенчатое воздействие

Переход от непрерывных к дискретным передаточным функциям можно выполнить директивой c2d программного обеспечения Control System Toolbox в пакете программ MATLAB. В качестве примера определим ДПФ для непрерывной передаточной функции

$$W(p) = \frac{b_1 p + b_0}{a_2 p^2 + a_1 p + a_0}.$$

Для определенности примем значения параметров: $b_1 = 0.01$; $b_0 = 1$; $a_2 = 0.02$; $a_1 = 0.1$; $a_0 = 1$ и интервал дискретности $T = 0.001c$ цикла расчета сигнала управления. После загрузки MATLAB следует открыть m-файл, в котором записать

```
      % параметры
b1=0.01;b0=1;
a2=0.02;a1=0.1;a0=1;
 T=0.001;
num =[b1 b0];    %  числитель
den=[a2 a1 a0]; %  знаменатель
w=tf(num,den)   %  передаточная функция (ПФ)
wd=c2d(w,T)      %  дискретная ПФ   (ДПФ)
```

Выполнение программы дает в командном окне результат:

```
Transfer function:
    0.01 s + 1
--------------------
0.02 s^2 + 0.1 s + 1
 Transfer function:
0.0005237 z - 0.0004738
-----------------------
 z^2 - 1.995 z + 0.995
 Sampling time: 0.001
```

Здесь представлены выражения непрерывной и соответствующей ей дискретной передаточной функции звена, а также величина интервала дискретности.

1.5.2. Синтез цифрового управления

В многоконтурной системе цифрового управления по условиям устойчивости и качества наибольшее быстродействие требуется и присуще внутреннему контуру. Каждый внешний контур обычно должен иметь время реакции на ступенчатое воздействие, в несколько раз большее, чем охватываемый им внутренний контур. Известен метод последовательной оптимизации (подчиненного управления) для многоконтурной, но непрерывной системы [20].

Для многоконтурной системы цифрового управления электроприводом технологического оборудования предпочтительными являются методы синтеза [20] - [23], которые учитывают дискретность цифрового устройства.

Методы [24], [25] основаны на построении масштабов времени для контуров управления, для каждого контура свой масштаб.

Выбор числа разрядов слова данных по требуемой точности системы управления. Количество разрядов шины данных в микроконтроллерах принимает значения 8, 16, 32 и, возможно, может быть большим. Современные микроконтроллеры ориентированы на применение языков высокого уровня, обычно языка C/C++. Это означает, что вычисления с применением 16 и 32-разрядных данных, а также с плавающей точкой, возможны независимо от разрядности шины данных устройства, однако для 8 и 16 разрядных устройств приводится в действие программная реализация 32– разрядных операций и операций с плавающей точкой. Тогда время обработки информации значительно увеличивается. Поэтому для высокого быстродействия, например, для управления электроприводами, предпочтительны специализированные микроконтроллеры с 32- разрядной шиной данных, имеющие сопроцессоры вычислений с плавающей точкой (*Float Point Unit, FPU*) либо сопроцессоры (*Control Low Accelerator, CLA*) ускорения вычисления сигнала управления.

Определение требуемого быстродействия микроконтроллера. В устройстве цифрового управления, а именно, в микроконтроллере, масштаб времени измеряется тактовыми периодами. Сигнал управления формируется за время $T_C = t_{IN} + t_C + t_{OUT}$, где t_{IN} - время ввода сигналов, t_C - время расчета сигнала управления, t_{OUT} – время вывода сигнала управления. Все интервалы времени кратны тактовому периоду.

Следовательно, для внутреннего контура, где требуется наибольшее быстродействие, следует формировать управление за наименьшее возможное количество тактов. В электроприводе желательна синхронизация циклов расчета T_C с периодами широтно-импульсной модуляции (ШИМ) преобразователя электрической энергии. Учитывая, что, в преобразователях обычно применяется частота ШИМ в пределах 1 ÷ 10 *kHz,* время цикла расчетов $T_C = 10^{-4} \div 10^{-3}$ *s*. Для электроприводов с частотным управлением этому требованию могут удовлетворять микроконтроллеры с тактовой частотой 60 ÷ 660 *MHz, с FPU* либо с *CLA.*

Для электропривода внутренним является контур управления током. Возможна аппаратная, аппаратно-программная и программная реализация регуляторов тока.

Синтез дискретных регуляторов [23] – [25]. Системы векторного управления электроприводами как правило являются многоконтурными, и в

каждом контуре управления применяются ПИ- регуляторы. Далее рассматривается синтез регуляторов тока и скорости асинхронного электродвигателя (АД), представленного в синхронно вращающейся системе координат (*d, q*) на рисунке 5.3.

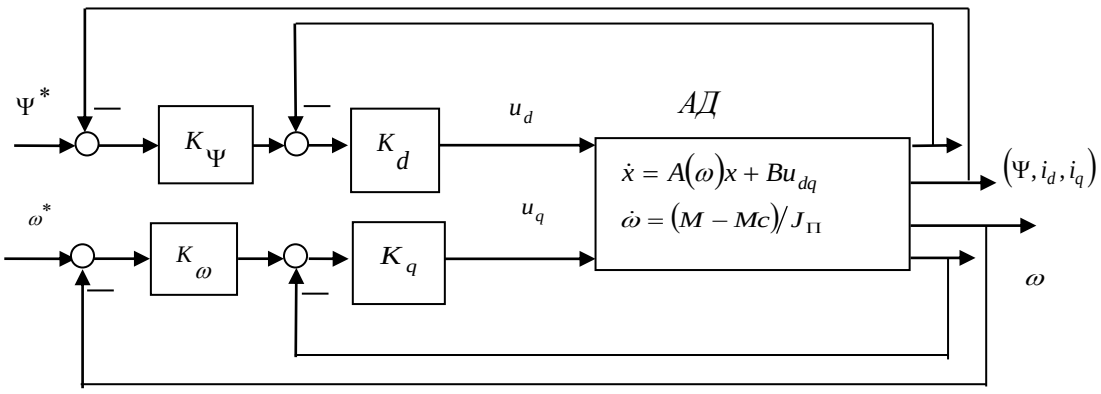


Рисунок 5.3 – Структура системы управления

На входах каналов потокосцепления и скорости действуют сигналы задания Ψ*, ω* потокосцепления и скорости, которые сравниваются с фактическими значениями потокосцепления Ψ ротора и скорости ω.

Ошибки $e_\Psi = \Psi^* - \Psi$, $e_\omega = \omega^* - \omega$ регулирования поступают на входы регуляторов K_d, K_q K_Ψ, K_ω потокосцепления Ψ и скорости Система содержит регуляторы K_d, K_q составляющих i_d, i_q тока по осям (*d, q*). Объект управления, асинхронный электродвигатель (АД) в структуре на рисунке 5.3 описывается в синхронно вращающейся системе координат уравнениями

$$\dot{x} = A(\omega)x + Bu_{dq},\ \dot{\omega} = (M - M_c)/J.$$

Здесь приняты обозначения: $x = (x_1, x_2, x_3)^T = (\Psi, i_d, i_q)^T$, $u_{dq} = (u_d, u_q)^T$ – вектор сигнала управления, $M = k_M \Psi i_q$ – электромагнитный момент $k_M = 1.5 k_2 p_П$, $p_П$ – число пар полюсов, M_c – момент сил сопротивления, J – приведенный к валу электродвигателя суммарный момент инерции.

Матрицы A и B имеют вид

$$A(\omega) = \begin{bmatrix} -a & aL_{12} & 0 \\ ak_2k_e & -a_e & p_П\omega \\ k_2k_e p_П\omega & -p_П\omega & -a_e \end{bmatrix}, \quad B = \begin{bmatrix} 0 & 0 \\ k_e & 0 \\ 0 & k_e \end{bmatrix}.$$

Параметры $a = 1/T_2$, $a_e = 1/T_e$, $k_e = 1/L_e$, и другие в матрицах определяются выражениями, представленными в таблице 5.1.

Таблица 5.1 Параметры объекта управления

R_1 R_2 L_1 L_2 L_{12} $p_П$	сопротивление фазной обмотки статора, *Ohm* сопротивление фазной обмотки ротора, *Ohm* индуктивность фазной обмотки статора, *Hn* индуктивность фазной обмотки ротора, *Hn* взаимная индуктивность, *Hn* число пар полюсов
$k_1 = L_{12}/L_1$, $k_2 = L_{12}/L_2$	коэффициенты взаимосвязи статора и ротора
$T_1 = L_1/ R_1$, $T_2 = L_2/ R_2$	постоянные времени статора и ротора, *s*
$R_e = R_1 + k_2^2 R_2$	эквивалентное сопротивление, *Ohm*
$L_e = L_1 - k_2 L_{12}$ $k_e = 1/L_e$	эквивалентная индуктивность, *Hn* и обратная ей величина
$T_e = L_e/ R_e$	постоянная времени эквивалентная, *s*
$k_M = 1.5\ k_2\ p_П$	Коэффициент в выражении момента

Расчетная структура контура тока на рисунке 5.4 построена в предположении, что сигнал задания тока $i^*_{dq} = (i^*_d,\ i^*_q)^T$ можно считать постоянной величиной, и, следовательно, динамика внешних контуров не окажет влияния на динамику контура тока. Ошибка $e_{dq} = i^*_{dq} - i_{dq}$ регулирования тока поступает на вход импульсного звена *I*, учитывающего квантование по времени.

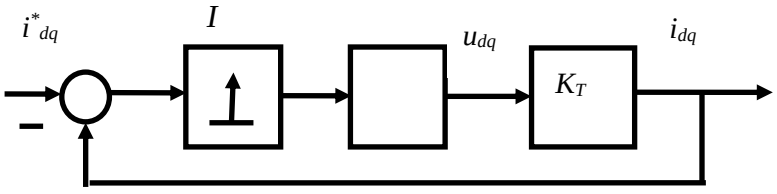


Рисунок 5.4 – Структура контура тока

Сигнал управления u_{dq} синтезируется одним из известных методов линейного синтеза управления, в результате получается регулятор тока, имеющий передаточную функцию K_{dq}. Рассматривается режим постоянства потокосцепления, поэтому электромагнитный момент пропорционален i_q . Величина тока i_q, формируя ускорение, позволяет управлять скоростью, а качество системы в значительной степени зависит от быстродействия контура тока. Сигнал управления формируется микроконтроллером программным способом в виде дискретного сигнала с периодом T_C. Цикл T_C может быть меньше, равен или больше, чем период широтно-импульсной модуляции (ШИМ) T_S. Далее предполагается, что $T_C = T_S$ и цикл расчета управления

синхронизирован с ШИМ. Передаточная функция (ПФ) электромагнитного звена двигателя и соответствующая дискретная передаточная функция (ДПФ) имеют вид

$$K_T(s) = \frac{1}{R_e(T_e s + 1)}, \quad K_T(z) = \frac{1 - d_e}{R_e(z - d_e)}.$$

Здесь d_e =exp$(-T_C/T_e)$. Используя ДПФ $K_T(z)$, можно определить ДПФ пропорционально- интегрирующего (ПИ) регулятора $K_{dq}(z)$ тока

$$K_{dq}(z) = b_1 + b_0 T_C / (z-1) = (b_1 (1 - z^{-1}) + b_0 T_C z^{-1}) / (1 - z^{-1}).$$

Здесь b_1, b_0 - искомые параметры регулятора. Ошибка $e_{dq} = i_{dq}^* - i_{dq}$ регулирования тока поступает на вход регулятора тока, на выходе которого формируется сигнал управления, имеющий z-изображение

$$u_{dq}(z) = K_{dq}(z)\, e_{dq}(z) = e_{dq}(z)\, (b_1(1-z^{-1}) + b_0 T_C z^{-1}) / (1-z^{-1}),$$

после умножения на $(1-z^{-1})$,

$$(1-z^{-1})\, u_{dq}(z) = K_{dq}(z)\, e_{dq}(z) = (b_1(1-z^{-1}) + b_0 T_C z^{-1})\, e_{dq}(z),$$

В результате получается z-изображение сигнала управления на текущем шаге с номером k

$$u_{dq}(z) = z^{-1} u_{dq}(z) + b_1 e_{dq}(z) + (b_0 T_C - b_1) z^{-1} e_{dq}(z). \qquad (5.4)$$

Если произвольный сигнал $u(k)$ имеет z-изображение $u(z)$, то z-изображение $z^{-i}u(z)$ имеет оригинал $u(k-i)$, то есть $z^{-i}u(z) \rightarrow u(k-i)$. Выражение для оригинала сигнала управления на текущем интервале с номером k

$$u_{dq}(k) = u_{dq}(k-1) + b_1 e_{dq}(k) + (b_0 T_C - b_1)\, e_{dq}(k-1). \qquad (5.5)$$

Последнее выражение является основой для алгоритма и программы расчета сигнала управления микроконтроллером. Параметры b_1, b_0 должны быть определены исходя из желаемых значений $z_{1,2}$ корней характеристического полинома на плоскости комплексной переменной Передаточная функция синтезированного контура тока, если обозначить $b_1' = b_1(1-d_e)/R_e$, $b_0' = b_0(1-d_e)T_C/R_e$, примет вид

$$W_C(z) = \frac{b_1'(z-1) + b_0'}{z^2 - z(1 + d_e - b_1') + b_0' - b_1' + d_e}$$

Характеристический полином замкнутого контура тока

$$N_C(z) = z^2 - z\cdot(1 + d_e - b_1') + b_0' - b_1' + de$$

должен иметь корни, принадлежащие внутренности единичного круга на комплексной плоскости. Значения желаемых корней $z_{1,2} = \sigma$ или близкие к ним

действительные либо комплексные значения $z_{1,2} = \sigma \pm j\upsilon$ обеспечивают процессы, близкие к апериодическим. Параметры b_0', b_1' рассчитываются по значениям корней z_1 и z_2. Из выражений $1 + d_e - b_1' = z_1 + z_2$, $b_0' - b_1' + d_e = z_1 z_2$ получаются параметры регулятора

$$b_1 = R_e (1 + d_e - z_1 - z_2)/ (1 - d_e),$$
$$b_0 T_C = (z_1 z_2 + 1 - z_1 - z_2) R_e /(1 - d_e) \qquad (5.6)$$

Для комплексных корней $z_{1,2} = \sigma \pm j\upsilon$ выражения принимают вид

$$b_0' = (1 - \sigma)^2 + \upsilon^2,$$

$$b_1 = R_e (1 + d_e - 2\sigma)/ (1 - d_e), \quad b_0 T_C = ((1 - \sigma)^2 + \upsilon^2) R_e /(1 - d_e).$$

В случае пропорционального регулятора тока $b_0' = 0$. Тогда ПФ контура тока примет вид $W_C = b_1'/(z - d_e + b_1')$. Коэффициент усиления b_1' регулятора тока определяется на основании $z_1 = \sigma$ выражением $b_1' = d_e - \sigma$. Здесь по условию устойчивости $|\sigma| < 1$, а по условию обеспечения качества динамических режимов в системе с цифровым управлением $\sigma \approx 0.75$. Для расчета усиления П- регулятора тока справедливо выражение $b_1 = R_e (d_e - \sigma)/(1 - d_e)$. В установившемся режиме контур тока создает усиление $W_{C\infty} = (d_e - \sigma) /(1 - \sigma) < 1$, меньшее единицы, что приводит к ошибке регулирования тока. Это обстоятельство делает П- регулятор тока малопригодным в условиях программной реализации регулятора тока.

Параметры (5.6) дискретного ПИ-регулятора приближаются к значениям для непрерывной системы, если отношение T_C/T_e стремиться к нулю в результате уменьшения интервала дискретности T_C.

Если значения параметров регулятора тока рассчитаны по выражениям (5.6) при заданных корнях $z_{1,2} = 0.75 \pm 0.05i$, то переходная функция близка к апериодической, и затухает за время $t_0 \approx 3T_C/ln(1 - \sigma) \approx 10{,}4T_C$, как показано на рисунке 5.4.

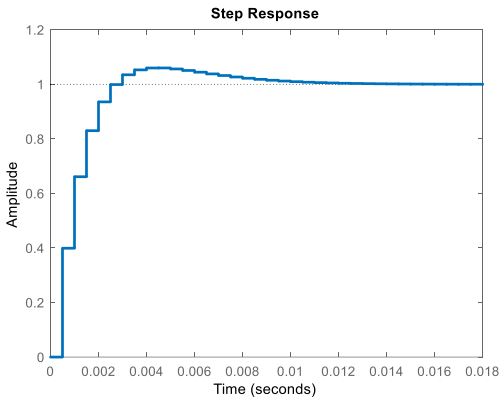


Рисунок 5.4 –Переходная функция контура тока с дискретным ПИ-регулятором

Контур скорости содержит ПИ-регулятор с ДПФ $K_\omega(z) = c_1 + c_0 T_C/(z-1)$, где c_1 , c_0 - параметры регулятора, которые подлежат определению.

Контур скорости имеет ДПФ $W_S(z) = K_\omega(z) W_C(z) b_p / (z - 1 + K_\omega(z) W_C(z) b_p)$, где $b_P = k_M \Psi/J$ - параметр объекта управления, который изменятся в широких пределах. Быстродействие контура тока (рисунок 5.4), позволяет при синтезе регулятора скорости приближенно ДПФ $W_C(z)$ принимать равной единице, $W_C(z) \approx 1$.

Редуцированный контур скорости имеет ДПФ $W_{S0}(z) = K_\omega(z) / (z - 1 + K_\omega(z) b_p)$ с характеристическим полиномом $N_{S0}(z) = (1-z)^2 + (c_1(1 - z) + c_0 T_C) b_P$. Задание малого положительного $\varepsilon < 0{,}5$ позволяет для характеристического полинома $N_{S0}(z)$ назначить желаемые корни,

$$z_{3,4} = \sigma_s \pm j\upsilon_s = 1 - \varepsilon(1-\sigma) \pm j\varepsilon\nu. \qquad (5.7)$$

которые обеспечивают значительно меньшее быстродействие контура скорости по сравнению с контуром тока.

Тогда для параметров c_1, c_0 ПИ-регулятора скорости получаются расчетные выражения, где b_{P0} - расчетное значение параметра объекта.

$$c_1 T_C = 2(1-\sigma_s)/ b_{P0},\ \mathbf{c}_0 T_C = (\sigma_s^2 + \upsilon_s^2 + 1 - 2\sigma_s) / b_{P0} = ((1-\sigma_s)^2 + \upsilon^2)/b_{P0}. \qquad (5.8)$$

Замена переменных $q = (z\text{-}1)$ приводит к преобразованию области устойчивости внутри единичного круга с центром в начале координат на плоскости комплексной переменной z в область устойчивости [7] в виде круга единичного радиуса с центром в точке (-1, j0) на плоскости комплексной переменной q. В малой окрестности начала координат плоскости s, когда $s < 0.2/T_C$ переменные q и s приближенно можно считать пропорциональными, $q \approx T_C s$, что позволит анализировать синтезированную систему методами, известными для непрерывных систем. Расчетные полиномы контуров тока и скорости принимают вид

$$N_C(q) = q^2 + 2(1-\sigma)q + \sigma^2 + \upsilon^2, \qquad (5.9)$$

$$N_{S0}(z) = q^2 + 2(1-\sigma_s)\, q + (1-\sigma_s)^2 + \upsilon_s^2. \qquad (5.10)$$

Значения корней связаны соотношениями

$$q_{1,2} = -(1-\sigma) \pm j\,\upsilon,\ q_{3,4} = \varepsilon q_{1,2} = -(1-\sigma_s) \pm j\,\upsilon_s, \qquad (5.11)$$

Учитывая (5.6), (5.8) можно получить ДПФ $W_C(q)$, $W_S(q)$ контура тока и скорости, выраженные через желаемые полюса и переменную q, в виде

$$W_C(q) = M_C(q) / N_C(q),\ W_S(q) = M_S(q) / N_S(q),$$

$$M_C(q) = (1+d_e - 2\sigma)\, q + (1-\sigma)^2 + \upsilon^2,\ N_C(q) = q^2 + 2(1-\sigma)q + (1-\sigma)^2 + \upsilon^2,$$

$$M_S(q) = (2(1-\sigma_s)q + (1-\sigma_s)^2 + \upsilon_s^2)\, M_C(q),$$

$$N_S(q) = q^2(q^2 - 2(1-\sigma)\, q + (1-\sigma)^2 + \mathrm{v}^2) + (2(1-\sigma_s)\, q + B\,(1-\sigma_s)^2 + \upsilon_s^2)\,(q(1+d_e - 2\sigma) + (1-\sigma)^2 + \mathrm{v}^2).$$

Обычно $T_2 > T_e > T_C$, и требуемое время реакции канала потокосцепления значительно превосходит T_C. Канал управления скоростью так же требует времени реакции, которое во много раз превосходит T_C. С учетом малого параметра ε, характеристический полином подсистемы управления скоростью принимает вид

$$N_S(q) = q^2(q^2 - 2(1-\sigma)\, q + (1-\sigma)^2 + \mathrm{v}^2) + (2(1-\sigma)\,\varepsilon\, q + B\,\varepsilon^2\,(1-\sigma)^2 + \upsilon^2)\,(q(1+d_e - 2\sigma) + (1-\sigma)^2 + \mathrm{v}^2).$$

Выражение показывает, что чем меньше ε, тем в большем диапазоне может увеличиваться усиление B объекта управления с сохранением устойчивости.

Следовательно, малая чувствительность в структуре электропривода (рисунок 5.3) к параметрическим возмущениям достигается расположением корней расчетных характеристических полиномов $N_C(q)$ контура тока и $N_{S0}(z)$ контура скорости в соответствии с (5.7).

Аналогичным образом рассчитываются параметры ПИ- регулятора в контуре потокосцепления (рисунок 5.3).

По такому же принципу возможен синтез регуляторов не только двухконтурных, но и многоконтурных систем. Это означает, что регулятор каждого внешнего контура синтезируется в предположении о безынерционности внутренней по отношению к нему подсистемы. В общем случае в контуре регулирования может применяться пропорционально-интегро- дифференцирующий (ПИД) регулятор.

1.5.3. Методы расчета параметров ПИД- регуляторов. Алгоритмы и программы расчета выходных сигналов ПИД регуляторов

Обычно ПИД регуляторы применяются в одноконтурных системах с управлением по выходу, то есть с обратной связью по выходной регулируемой величине. Примером являются электроприводы турбомеханизмов с обратной связью по напору. Синтез линейного робастного ПИД регулятора для нелинейного объекта возможен после линеаризации объекта и определения интервалов изменения параметров в передаточной функции (ПФ).

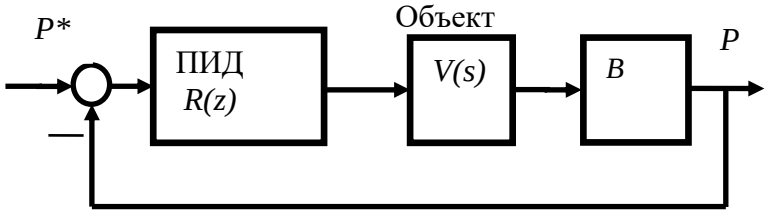


Рисунок 5.7 – Структура управления с ПИД регулятором

Необходимо определить параметры ПИД регулятора по требуемым показателям качества. Структура на рисунке 5. 7 содержит ПИД регулятор с ПФ $R(z)$, объект управления с ПФ $V(s)$ и датчик B безынерционный, причем $R(z)$ имеет вид

$$R(z) = M_R(z)/N_R(z) = k_I T_C z^{-1}/(1 - z^{-1}) + k_P + k_D(1 - z^{-1})/T_C, \qquad (5.12)$$

в то время, как объект описывается непрерывной ПФ $V(s) = M_P(s)/N_P(s)$. В выражении (5.12) следует перейти к комплексной переменной $q = (z - 1)/T_C$ учитывая, что $s \approx q$ при $s < 0{,}2T_C^{-1}$. Тогда из (5.12) получается

$$R(q) = M_R(q)/N_R(q) = k_I/q + k_P + k_D q/(T_C q + 1), \qquad (5.13)$$
$$M_R(q) = c_0 + c_1 q + c_2 q^2,\ N_R(q) = q(T_C q + 1).$$

Коэффициенты c_i, $(i=0,1,2)$ выражаются через коэффициенты k_I, k_P, k_D по формулам: $c_0 = k_I$, $c_1 = k_P + k_I T_C$, $c_2 = k_D + k_P T_C$. С учетом $s \approx q$ характеристический полином системы примет вид $N(s) = N_R(s)N_P(s) + M_R(s) M_P(s)$, где

$$N_R(s) = s(T_C s + 1),\ N_P(s) = s^2 + a_{P1}s + a_{P0},\ M_R(s) = c_0 + c_1 s + c_2 s^2,\ M_P(s) = b_P.$$

Параметры объекта принадлежат интервалам $b_P \subset [\underline{b_P}\ \overline{b_P}]$, $a_{Pi} \subset [\underline{a_{Pi}}\ \overline{a_{Pi}}]$, ($i$ = 0, 1, 2). С учетом обозначений $c_1' = c_1/c_2$, $c_0' = c_0/c_2$, характеристический полином примет вид

$$N(s) = s\,(T_C s + 1)\,(s^2 + a_{P1} s + a_{P0}) + (c_0' + c_1' s + s^2)\,c_2 b_P.$$

Полином $M_R(s)$ устойчив по условиям синтеза, а разность порядков двух полиномов, составляющих $N(s)$, в данном случае равна порядку n_P полинома объекта. При $n_P = 1$ условия М. В. Меерова устойчивости системы при бесконечном усилении $c_2 b_P = \infty$ выполняются. Здесь, поскольку $n_P = 2$, для устойчивости достаточно, чтобы выполнялось соотношение между коэффициентами полиномов, полного и вырожденного при $c_2 b_P = \infty$, в виде $T_C^{-1} + a_{P1} - c_1' > 0$.

При бесконечном усилении $c_2 b_P = \infty$ полином становится равен вырожденному $N_\infty(s)$, и, таким образом, свойства системы будут полностью определяться параметрами регулятора и не зависеть от параметров объекта. Следовательно, расчет параметров ПИД регулятора должен производится по желаемым корням $s_{1,2} = -\alpha$, полинома $N_\infty(s)$

$$c_1' = 2\alpha,\ c_0' = \alpha^2 \qquad (5.14)$$

При конечном, но достаточно большом усилении, используя малую величину $0 < \varepsilon < 0{,}5$, и частоту Ω_C среза разомкнутого контура, для расчета параметров ПИД регулятора можно применить выражения (5.14) при соблюдении ограничений

$$T_C^{-1}\varepsilon > \Omega_C,\ T_C^{-1}\varepsilon\, c_2 b_P\, |N_\infty(j\omega)| > |N_P(j\omega)\, N_R(j\omega)|. \qquad (5.15)$$

Свойства системы в основном определяются параметрами регулятора и мало зависят от параметров объекта. Способ позволяет определить все четыре параметра T_C, c_0, c_1, c_2 ПИД – регулятора на основании выражений (5.14), (5.15) и является приближенным.

Пример 5.1 Составить алгоритм и программу расчета сигнала управления в одноконтурной системе с ПИ – регулятором напора (рисунок 5.8).

Два входных аналоговых сигнала *P*, P* поступают на входы A0 и A1 аналого-цифрового преобразователя *ADC10* через внешние выводы P2.0 и P2.1 параллельного порта P2.

Микроконтроллер выполняет расчет ошибки ***e = P*- P*** регулирования и на основании ошибки регулирования вычисляет выходной сигнал *u* ПИ-регулятора. Выходная величина *u* ПИ- регулятора через последовательный интерфейс SPI передается на вход полупроводникового преобразователя электрической энергии UZ, который обеспечивает частотное управление электродвигателем M насоса. В начальный период времени после включения ошибка регулирования и, следовательно, сигнал *u* ПИ- регулятора имеют значительную величину. Поэтому преобразователь формирует напряжение и частоту, способствующие разгону насоса до скорости, при которой напор достигает заданного значения.

На напорном патрубке насоса имеется датчик давления *BP,* создающий

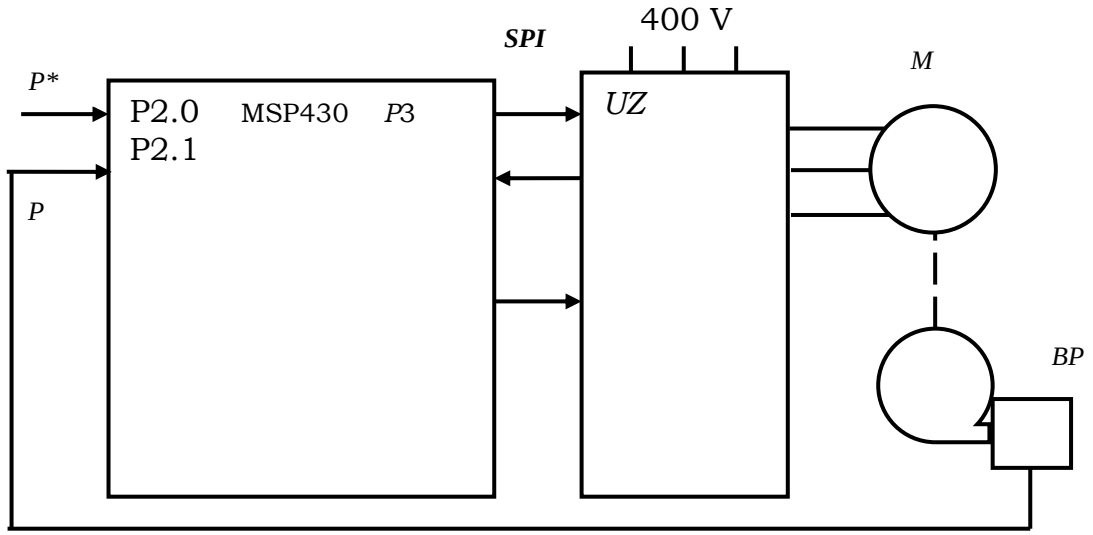


Рисунок 5.8 – Функциональная схема управления напором

сигнал, пропорциональный давлению, для использования в качестве обратной связи.

Микроконтроллер управления электроприводом насоса должен выполнять следующие функции: ввод двух аналоговых сигналов ***P*, P*** и их преобразование в двоичный код; расчет выходной величины ***u*** на выходе ПИ-регулятора, которая является взвешенной суммой сигнала ошибки ***e*** и интеграла ошибки с учетом ограничения; вывод полученного сигнала управления на вход объекта (преобразователя частоты электропривода насоса) с использованием последовательного интерфейса.

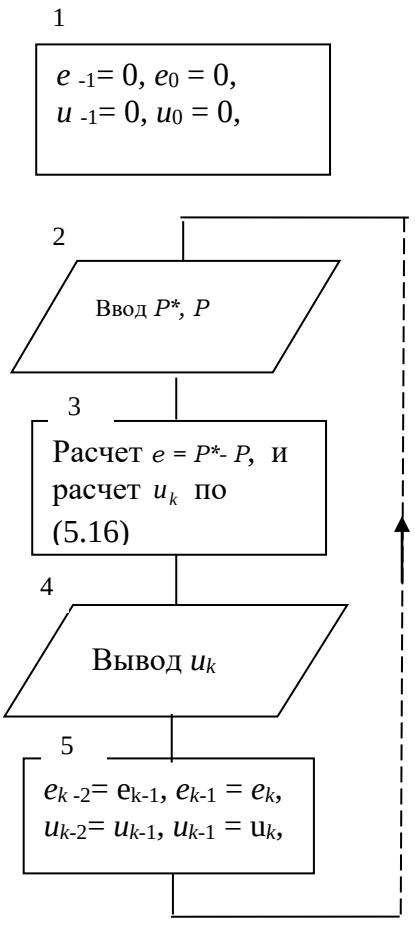


Рисунок 5.9 – Алгоритм расчета сигнала управления

Алгоритм представлен на рисунке 5.9. Инициализационная часть должна содержать инициализацию ADC10 на последовательный циклический режим ввода заданного и фактического значений давления, а также настройку последовательного интерфейса. Начальные условия должны бать нулевыми, ***e* = 0, *u*** = 0.

С учетом зависимости напора от расхода для турбомеханизма, передаточная функция линеаризованного объекта содержит параметр $c_P = 2M_H / (J\omega_H)$ который зависит от номинального момента M_H электродвигателя и момента инерции J, $v = \omega / \omega_H$ - относительная величина скорости, $a_M = k_e k_M / (R_e J)$ - обратная величина

электромеханической постоянной времени, $b_P = b_{P0}v$, где $b_{P0} = 2M_H a_M / (\eta Q_H k_e)$ зависит от производительности Q_H и КПД η турбомеханизма.

Для определения расчетного выражения сигнала управления следует определить z - изображение сигнала управления. По дискретной передаточной функции (ДПФ) ПИ- регулятора, которая имеет вид

$$R(z) = \frac{u(z)}{e(z)} = \frac{b_1 + (b_0 T_C - b_1) z^{-1}}{1 - z^{-1}},$$

Из операторного уравнения $(1\text{-}z^{-1})\ u(z) = (c_1 + (c_0 T_C - c_1) z^{-1})\ e(z)$ выражается z -изображение сигнала управления $u(z) = z^{-1}u(z) + (c_1 + (c_0 T_C - c_1) z^{-1})\ e(z)$. Значение u_k сигнала управления на текущем k–м шаге получается по правилу $u_k \leftarrow u(z)$, $u_{k-i} \leftarrow z^{-i}u(z)$, и имеет вид

$$u_k = u_{k-1} + b_1 e_k + (b_0 T_C - b_1) e_{k-1}.$$

Это рекурсивное выражение, по которому составляется алгоритм, показанный на рисунке 5.9. Алгоритм должен предусматривать в инициализационной части (блок 1) нулевые начальные условия. После ввода сигналов задания и обратной связи (блок 2) выполняется расчет ошибки регулирования и расчет по формуле (5.16) в блоке 3. Далее сигнал управления выводится на вход объекта управления. В блоке 5 формируются начальные условия для следующего цикла расчета.

Цикличность может формироваться программным либо программно-аппаратным способом (по прерываниям). Алгоритм, показанный на рисунке 5.9, может быть частью алгоритма главной программы либо реализовываться в виде функции. Алгоритм позволяет сформировать программный модуль (функцию) расчета выходной величины ПИ-регулятора, к которому периодически обращается главная программа.

Далее показаны описание функции ПИ- регулятора и обращение к ней в главной программе.

```
//*****************************************************************************
#include "msp430x22x4.h"
      int  PI( int e , e1, u, int c0 ,c1,Tb0 )  // описание функции
{ u += c1 * e + (Tb0 – c1) * e1;
return u ;
}
void main(void)
{   int  ee = 0x00, ee1= 0x00, uu = 0x00, pp=0, ppz=0, kC=0; сигналы
int bb0 = 8,bb1 = 4, Tcb0 = 1;              // параметры
int  PI( int e , e1, u, int b0 ,b1, Tb0);   // прототип функции
.....
 {
ee = ppz – pp;                               //  расчет ошибки регулирования
uu = PI( ee , ee1, u, bb0 ,bb1 );           // PI- controller
ee1 = ee;
}
```

Контрольные вопросы

1. Чем отличается дискретный регулятор от непрерывного?
2. Записать дискретные передаточные функции интегрирующего, дифференцирующего и пропорционального звеньев.
3. Записать дискретные передаточные функции П, ПИ и ПИД-регуляторов.
4. Как выполнить расчет параметров регуляторов?

1.6. Алгоритмы интеллектуального управления

Интеллектуальное управление предполагает применение в автоматике методов искусственного интеллекта, что придает системе свойства адаптируемости к изменениям условий функционирования, обучаемость и, в некоторых случаях, создает условия для самоорганизации.

Методы искусственного интеллекта все более применяются в автоматизации, что возможно на основе вычислительной техники трех уровней. Промышленные компьютеры применяются на верхнем уровне иерархии в системе автоматизации. Программируемые контроллеры и устройства числового программного управления выполняют задачи автоматизации определенным технологическим агрегатом. Микроконтроллеры предназначены для встроенных приложений. На каждом из уровней могут применяться методы искусственного интеллекта. Наиболее применяемы методы нечеткой логики, искусственные нейронные сети и генетические алгоритмы [1], [20], [26] - [29].

Контроллеры нечеткой логики основаны на теории размытых (нечетких) множеств и связанной с ними нечеткой логике [1], [27], [28]. Теория нечетких множеств позволяет ввести *лингвистические переменные*, для которых формируются правила нечеткой логики (нечетких логических выводов).

Вместо действительных значений вводится совокупность лингвистических значений, которые обычно характеризуют величину как нулевую, малую, среднюю и большую в зависимости от принадлежности определённому интервалу значений. Преобразование множества действительных значений в лингвистические переменные называют фаззификацией (fuzzification). Так как в системе управления все сигналы ограничены, получается конечное количество таких отрезков. В результате, хотя точность утрачивается, появляется возможность логического управления непрерывным процессом по выходу. Такое управление не нуждается в знании динамических свойств и параметров объекта управления, но опирается на логику его функционирования. Этим достигается удовлетворительное функционирование системы в условиях неопределенности свойств объекта.

На рисунке 6.1 показана структура системы с ПИД- контроллером нечеткой логики (КНЛ).

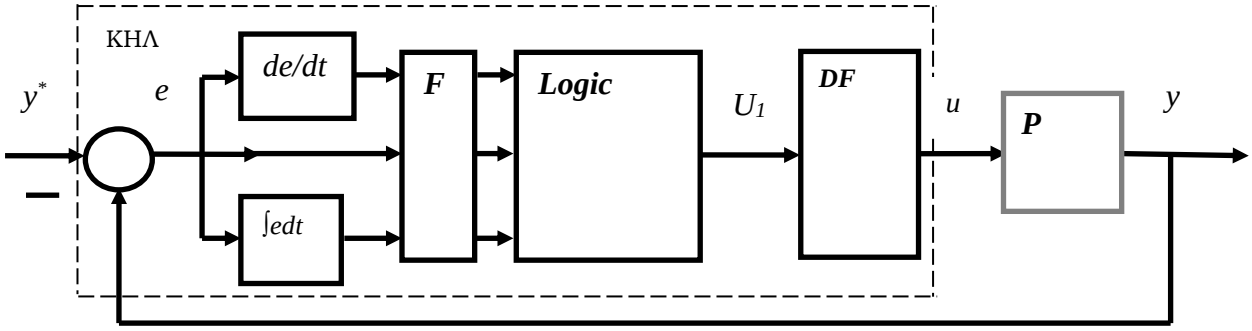


Рисунок 6.1 – Структура системы с ПИД- контроллером нечеткой логики

Сигнал y обратной связи сравнивается с сигналом y^* задания и формируется ошибка $e = y^* - y$ регулирования, которая вместе с ее производной и интегралом поступает на вход блока ***F*** фаззификации, где все величины преобразуются в лингвистические значения. В блоке ***Logic*** выполняются правила логического вывода над лингвистическими значениями переменных с целью формирования сигнала U_1 управления объектом ***P.*** Сигнал U_1 является лингвистической переменной, и в блоке ***DF*** ***(****Defuzzification)* преобразуется в плавно изменяющийся сигнал в виде действительной переменной. Таким образом, алгоритм нечеткого управления представлен следующими действиями.

0. Инициализация.
1. Ввод сигналов задания и обратной связи.
2. Расчет ошибки ошибка $e = y^* - y$ регулирования.
3. Расчет производной и интеграла ошибки регулирования.
4. Формирование лингвистических переменных в блоке ***F***.
5. Выполнение правил логического вывода для формирования сигнала управления U_1.
6. Переход от лингвистических значений U_1 к действительному сигналу u управления (сглаживание) блоке ***DF***.
7. Переход к 1.

Для моделирования системы с контроллером нечеткой логики для проверки работоспособности разработанных правил логического вывода можно использовать программный пакет MATLAB.

На основе алгоритма возможна разработка программного обеспечения для микроконтроллеров. Для управления инерционными объектами блок ***DF,*** где применяются вычисления в действительных переменных, не обязателен**.** В таких системах, поскольку управление формируется логическими, а не арифметическими операциями, возможно применение микроконтроллера с малым количеством разрядов в слове данных, что является преимуществом КНЛ.

Применение нечеткой логики перспективно для управления объектами со значительной неопределенностью, когда внешние и внутренние возмущения непредвиденным образом изменяют свойства объекта, и, в то же время, требуемая точность не высока. В этих условиях КНЛ обеспечит удовлетворительное функционирование системы в условиях возмущений.

Искусственные нейронные сети (ИНС) являются упрощенными моделями нейронных сетей в головном мозгу живых организмов [20], [26], [27]. ИНС является универсальным вычислительным устройством, однако структура

и принципы вычисления иные, чем в общепринятых вычислительных устройствах (компьютерах, микроконтроллерах и др.) Принципы преобразования информации в ИНС во многом определяются структурой искусственного нейрона (ИН, рисунок 6.2).

Искусственный нейрон является сумматором входных величин $x_1, \ldots, x_n$ с весовыми коэффициентами $w_1 \ldots, w_n$. *Обучением* называют настройку весовых коэффициентов для выполнения определенного класса задач, для этого используют алгоритм обучения (АО). Сигнал s на выходе сумматора подлежит ограничению, и для этого ИН содержит нелинейное звено с активационной функцией $f(s)$. Выходная величина ИН формируется активационной функцией $u = f(s)$.

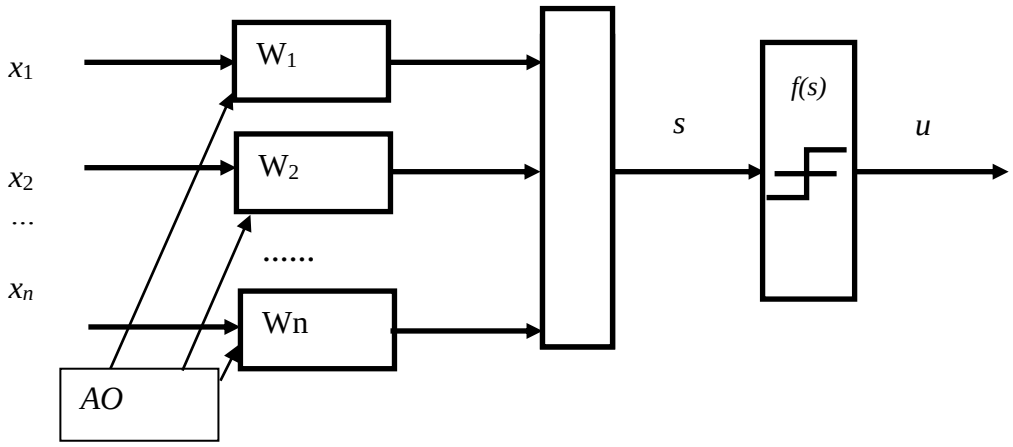


Рисунок 6.2 – Структура искусственного нейрона

Если на входе ИН действует вектор $\boldsymbol{x}$, $\boldsymbol{x}^T = (x_1, \ldots, x_n)$, то выходная величина сумматора определяется выражением

$$s = \boldsymbol{x}^T \mathbf{w} = w_1 x_1 + w_1 x_1 + \ldots + w_n x_n .$$

Пространство $\boldsymbol{x}$ делится поверхностью $s = 0$ на два полупространства. Если $s > 0$, то активационная функция формирует на выходе высокий уровень сигнала u, условно $u = 1$. Если же $s \le 0$, то активационная функция формирует на выходе низкий уровень, $u = 0$ либо $u = -1$, в зависимости от вида зависимости $f(s)$. Активационные функции, применяемые в ИНС, бывают различного вида. Это однополярные или двухполярные релейные функции (рисунок 6.2), кусочно-линейные функции с насыщением и гладкие функции с насыщением, однополярные и двухполярные.

Таким образом, искусственный нейрон выполняет классификацию входных векторов по признаку принадлежности верхнему или нижнему полупространству пространства $\boldsymbol{x}$. Изменением вектора $\mathbf{w}$ весовых коэффициентов (обучением) можно изменять положение в пространстве разделяющей поверхности $s = 0$. Следовательно, совокупность ИН позволяет установить принадлежность входного вектора выпуклому многограннику в пространстве $\boldsymbol{x}$, что необходимо для распознавания входного вектора.

С другой стороны, структура искусственного нейрона делает его универсальным вычислителем. В самом деле, весовой сумматор позволяет выполнять 4 арифметических действия при соответствующих настройках весов. Если активационная функция имеет вид релейной характеристики, то

ИН после обучения может выполнить логические операции, все, кроме исключающего ИЛИ. Группа из нескольких нейронов позволяет формировать произвольные нелинейные функциональные зависимости.

Подобно тому, как в мозгу нейроны соединены в 3-х или 4-х слойные нейронные сети, так и искусственные нейроны соединяются в искусственные нейронные сети, как показано на рисунке 6.3. Здесь $X = (x_1,\ldots,x_n)^T$ – входной вектор, $u = (u_1,\ldots,u_{m3})^T$ – выходной вектор ИНС. Первый слой является входным, второй – скрытым, а третий – выходным. В ИНС на каждый вход нейрона поступают все выходные сигналы от нейронов предыдущего слоя.

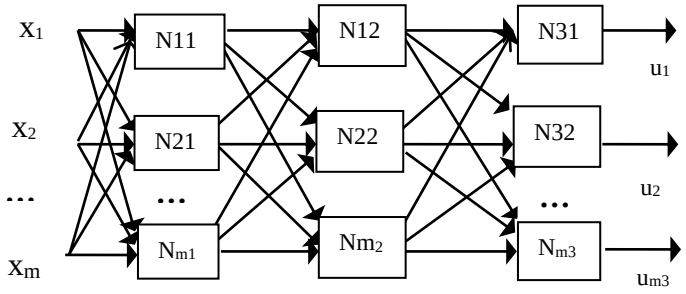


Рис.6.3 - Искусственная нейронная сеть.

ИНС путем обучения может быть настроена на решение определенного класса задач. Методы обучения ИНС рассматриваются в [26], [27]. Поскольку ИНС преобразует информацию не алгоритмически, а путем распознавания образов, ИНС наиболее эффективна для задач классификации и распознавания образов.

Генетические алгоритмы применяются в системах управления как эффективный способ организации движения к экстремуму (минимуму либо максимуму) критерия качества. Обычно рассматривают минимизацию критерия, а максимизация сводится к минимизации заменой знака. Генетические алгоритмы построены по аналогии с механизмом наследования в живой природе, где критерием является живучесть организмов каждого нового поколения.

Критерий качества $Q(\boldsymbol{x})$ (целевая функция) зависит от вектора $\boldsymbol{x}$ переменных, и необходимо найти значение вектора $\boldsymbol{x}$, доставляющее минимум критерию.

В области генетических алгоритмов придерживаются определенной терминологии. Один бит, принимающий значение 0 либо 1, эквивалентен гену. Хромосома состоит из генов, и аналогична байту или слову. Популяция – это множество хромосом. Таким образом, в памяти микроконтроллера каждая популяция представляется массивом данных.

Генетический оператор преобразует исходную популяцию в новое поколение. Генетический оператор последовательно выполняет селекцию, скрещивание и мутацию. Селекция представлена на рисунке 6.4 и заключается в выборе родителей по критерию *Q(x).*

Q(x)	*Хромосомы*
51	A
2	B
7	C
13	D
23	E
41	F

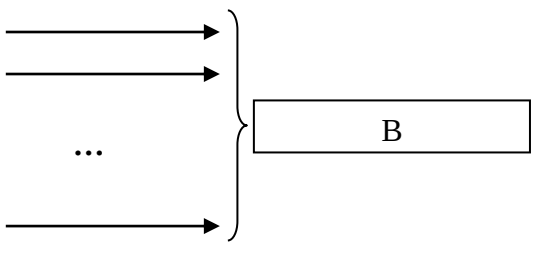


Рисунок 6.4 – Селекция по минимуму критерия *Q(x)*

После селекции следует скрещивание (рисунок 6.5). Имеется две хромосомы, *A*1 и *A*2, каждая из которых представлена байтом. Выбирается произвольная точка скрещивания, которая делит каждый байт на две части. В результате обмена содержимым двух байтов *A*1 и *A*2 вокруг точки скрещивания получаются два представителя A3 и A4 нового поколения. Так как каждая точка скрещивания дает двух представителей нового поколения, два байта A1 и A2 могут породить 16 представителей нового поколения.

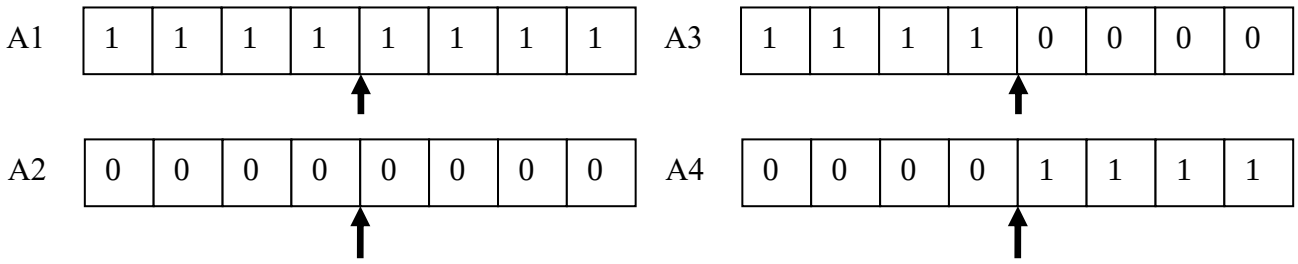


Рисунок 6.5 – Скрещивание A1 и A2, новое поколение - A3 и A4

Мутация заключается в случайном изменении некоторых битов в представителях нового поколения и выполняется изредка, не на каждом шаге алгоритма. Генетический алгоритм представлен на рисунке 6.6.

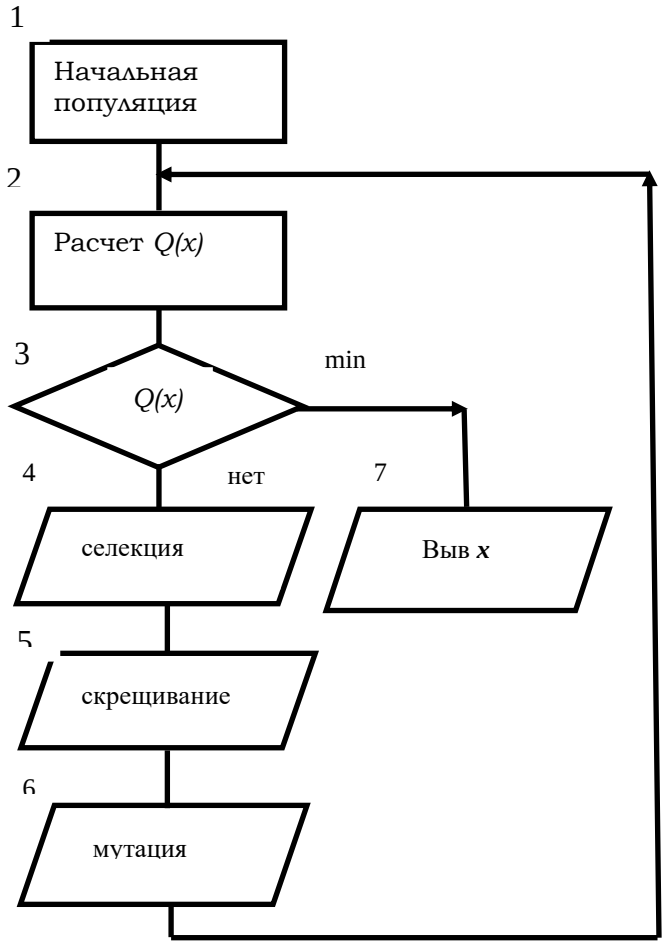


Рисунок 6.6 - Алгоритм генетический

В таблице 6.1 представлено сравнение генетических алгоритмов с традиционными методами поиска экстремума.

Таблица 6.1 Сравнение традиционных методов поиска экстремума и генетических алгоритмов

Традиционные методы	Генетические алгоритмы
Исходное значение вектора одно	Множество исходных векторов (исходная популяция)
Аналитические выражения для направления и шага поиска зависят от метода	Генетический оператор: селекция, скрещивание, мутация
Траектория поиска одна, и есть риск попадания в локальный, а не глобальный минимум	Поиск одновременно идет в множестве направлений, и не эффективные ветви отсеиваются селекцией

Из таблицы 6.1 видно, что генетический алгоритм имеет преимущества по сравнению с традиционными методами, так как позволяет вести поиск из разных исходных точек в нескольких направлениях. Кроме того, периодически применяемые мутации вносят элемент случайного поиска. Это во многих случаях позволяет сократить время поиска экстремума в несколько раз.

Недостаток генетического алгоритма заключается в целочисленном представлении аргумента x, что ограничивает точность метода. При количестве разрядов $m \geq 16$ относительная погрешность может быть достаточно малой, однако время цикла расчетов увеличивается за счет продолжительности селекции и скрещивания.

Контрольные вопросы

1 Каков принцип действия контроллера нечеткой логики?
2 Преимущества и недостатки контроллера нечеткой логики.
3 Чем полезно применение лингвистических переменных?
4 Где целесообразно применять контроллеры нечеткой логики?
5 Каков принцип действия искусственного нейрона?
6 Какие активационные функции применяются?
7 Как активационная функция преобразует сигнал?
8 С какой целью проводится обучение нейронной сети?
9 Как соединяются нейроны в сеть?
10 Где возможно применение ИНС в автоматике?
11 Какие задачи могут решать генетические алгоритмы?
12 Как построены генетические алгоритмы?

1.7. Применение микроконтроллеров для логического управления и обеспечения безопасности

Логическое управление технологическим оборудованием как правило основывается на алгоритмах, где учитываются условия безопасности и безаварийности. Далее рассмотрены примеры логического управления.

Пример 7.1 Реверсивное управление асинхронным электродвигателем *М*. Схема функциональная управления показана на рисунке 7.1.

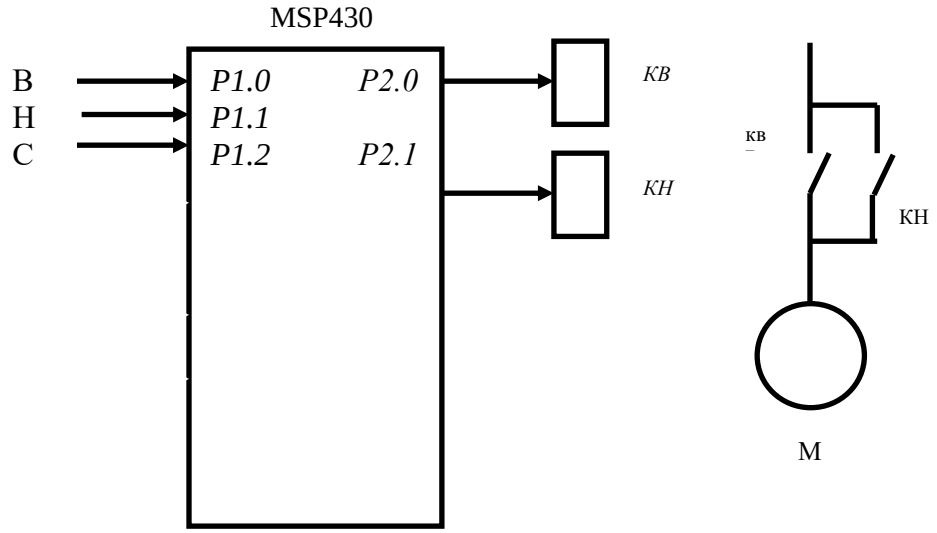


Рисунок 7.1 – Схема функциональная управления электроприводом

На рисунке 7.1 на параллельный порт Р1 микроконтроллера поступают сигналы «Вперёд», «Назад» и «Стоп» от кнопок пульта управления. На выходе следует сформировать сигналы управления контакторами КВ и КН для реверсивного управления асинхронным электродвигателем *М.*

Выходные сигналы микроконтроллера должны быть сформированы так, чтобы исключить одновременное включение контакторов «Вперёд» и «Назад», и этим исключить возможность короткого замыкания, а так же обеспечить отключение при наличии на входе сигнала «Стоп» либо недопустимой кодовой комбинации. Если сигналы «Вперёд», «Назад» и поступают на микроконтроллер от выключателей, то управление формируется в соответствии с логическими выражениями

$$КВ = (В \,\&\, (\sim Н) \,\&\, (\sim С)) ,$$
$$КН = (Н \,\&\, (\sim В) \,\&\, (\sim С)).$$

Таким образом, бит КВ должен формироваться при наличии во входном порту кодовой комбинации Р1 = 0х01 = 0000 0001, а бит КН – при Р1 = 0х02 = 0000 0010.

Если сигналы «Вперёд», «Назад» и «Стоп» поступают на микроконтроллер от кнопок с самовозвратом, то выходные сигналы микроконтроллера должны быть сформированы в соответствии с логическими выражениями

$$КВ = (В | КВ) \,\&\, (\sim Н) \,\&\, (\sim С)),$$

КН = (Н | КН) & (~В) & (~С)).

Здесь & означает конъюнкцию (логическую операцию И), знак | означает дизъюнкцию (логическую операцию ИЛИ), а знак ~ означает отрицание. Последние выражения учитывают необходимость запоминания в битах КВ, КН состояний контакторов.

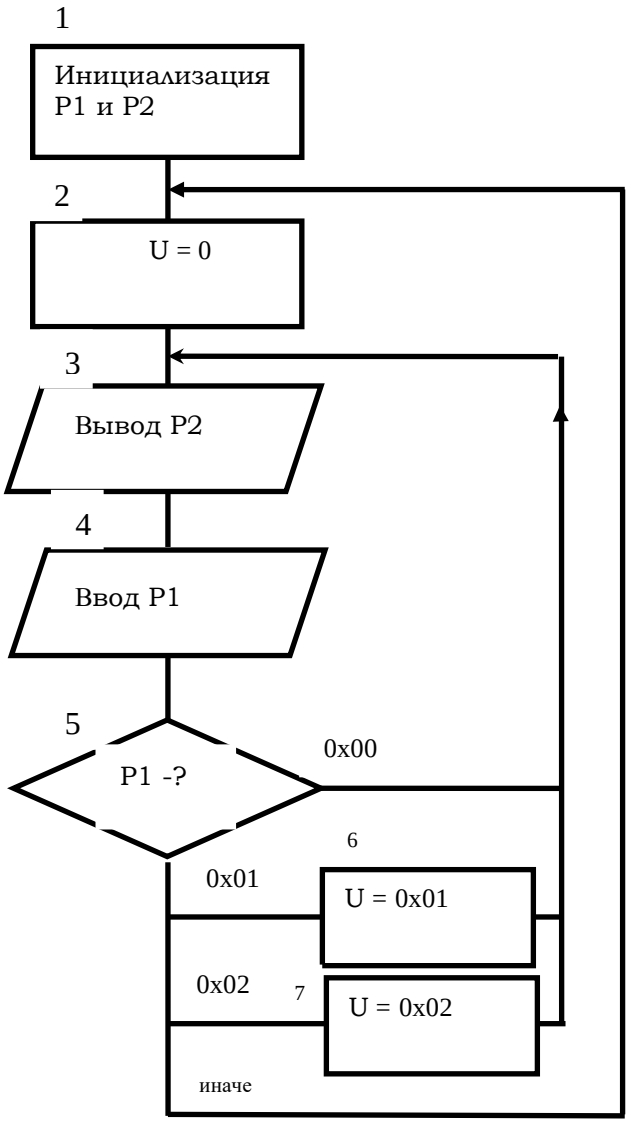


Рисунок 7.2 - Алгоритм управления

Алгоритм управления должен содержать инициализационную и циклическую части. В инициализационной части необходимо предусмотреть инициализацию периферийных устройств (в данном случае параллельных портов Р1 и Р2), определяющую режим их работы. Порт Р1 следует инициализировать для ввода сигналов через младшие три бита, а порт Р2 - для вывода битовых сигналов состояния контакторов КН и КВ. Также в инициализационной части формируются начальные условия (рисунок 7.2).

В циклической части следует предусмотреть ввод сигналов, формирование выходного сигнала в соответствии с логикой управления и вывод, которые могут следовать в любом порядке, поскольку повторяются циклически.

Алгоритм на рисунке 7.2 после инициализации формирует начальное условие U = 0x00, что необходимо в целях безопасности для гарантирования отключенного состояния контакторов после включения. Нулевое значение выводится через порт P2. После этого предусмотрен ввод входного байта через порт P1. Далее в блоке 5 анализируется входной байт, в зависимости от значения которого формируется сигнал управления и далее цикл повторяется.

Алгоритм (рисунок 7.2) формирует программную организацию ввода-вывода информации. По алгоритму можно составить программу, на языке С, в которой для организации циклической части применен оператор **for** (; ;) организации бесконечного цикла. Внутри этого цикла ветвление алгоритма обеспечивается оператором **switch** (P1IN) переключения в зависимости от содержимого в регистре P1IN.

```
        #include <msp430.h>  //        Адреса периф. устройств
        int main(void)       //        Главная программа,
        { int u = 0;         //        нач. условие
WDTCTL = WDTPW WDTHOLD;//               остановка WDT
P1DIR = 0x00;                          //инициализация
P2DIR = 0xFF;                          // для ввода и вывода
//циклическая часть
 for ( ; ; )
{P2OUT = u;                            //  вывод
        switch (P1IN )
{ case 0x00 :   ;                      //старое значение u
case 0x01 : u = 0x01;                  // КВ вкл
 case 0x02 : u = 0x02;                 // КН вкл
        default : u = 0x00;
        }
        }
```

Режим ручного управления необходим для настройки и наладки оборудования. Автоматический режим позволяет облегчить функции человека-оператора по управлению технологическим оборудованием. Следующий пример показывает, как можно организовать ручной и автоматический режимы работы.

Пример 7.2. Система реверсивного управления АД показана на рисунке 7.3. В автоматическом режиме должен выполняться цикл от кнопки П: движение вперед в течение времени t_1, остановка в течение времени t_1,

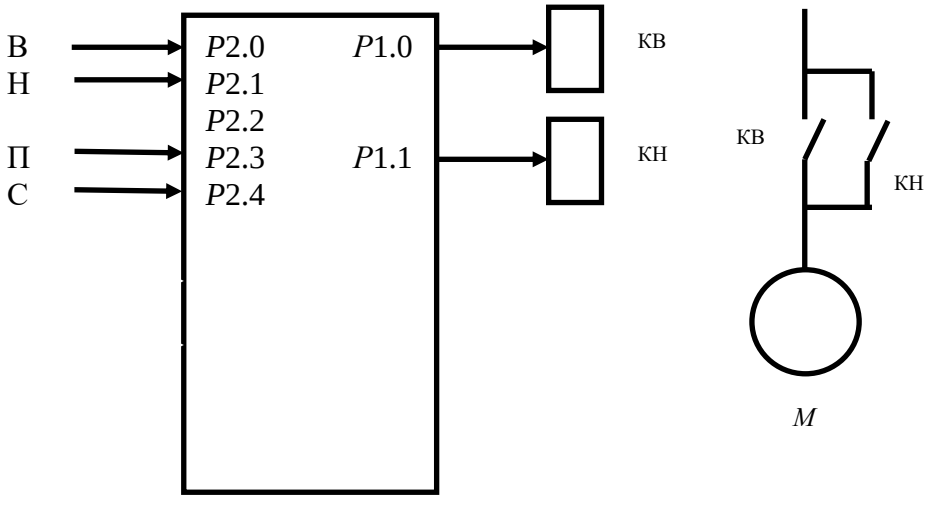


Рисунок 7.3 – Схема функциональная реверсивного управления электроприводом

движение назад в течение времени t_1, и вновь остановка на время t_1 и остановка от кнопки С. Предусмотреть ручное управление от кнопок вперед В, назад Н и стоп С (рисунок 7.4). Составить алгоритм и программу управления

Алгоритм основан на логических выражениях для сигнала управления в *ручном* режиме P1OUT = u,

```
if (P2IN == 0x01)    u = 0x01;          (7.1)
if (P2IN == 0x02)    u = 0x02;
```

Первое выражение формирует включение контактора КВ, а второе – КН. Эти выражения, однако, не означают, что выходной сигнал на контакторы можно автоматически приравнивать входному, но *u = P2IN* тогда и только тогда, когда на вход поданы сигналы «Вперед» либо «Назад», то есть если будет истинным (*P2IN* == 0x01) || (P2IN == 0x02). В остальных случаях, в соответствии с (7.1), на выходе должен формироваться ноль - *u* = 0.

В *автоматическом* режиме в выходной порт выводится значение *P1OUT = ua*, в течение выполняемого заданного цикла равное последовательно 0x01, 0x00,0x02 и 0x00 на последовательных интервалах времени длительностью t_1. Это выражается следующим образом

```
for( ; ;  ) {
t1 = 0xC350;
while (t1 > 0) {ua = 0x01; t1--; }
while (t1 < 0xC351) { ua = 0x00; t1++; }
while (t1 > 0) { ua = 0x02; t1--; }
while (t1 < 0xC351) {ua = 0x00; t1++; }
}
```

Алгоритм главной программы и подпрограммы прерывания от параллельного порта представлен на рисунке 7.4.

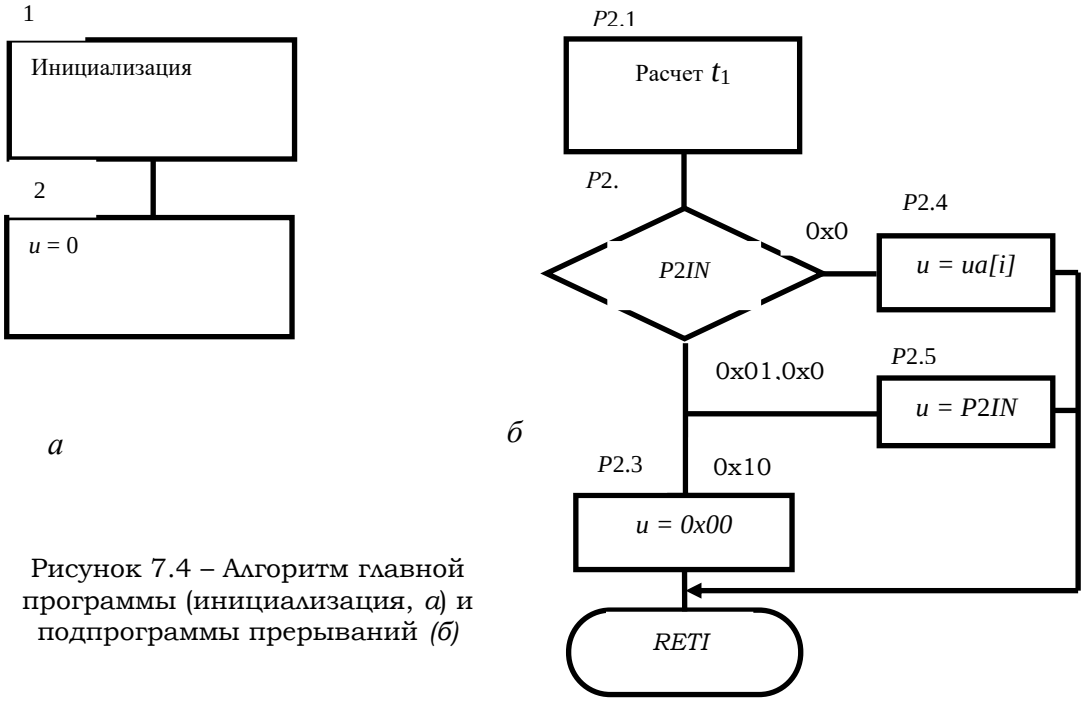


Рисунок 7.4 – Алгоритм главной программы (инициализация, *а*) и подпрограммы прерываний *(б)*

Далее представлен текст программы на языке С в двух вариантах. Первый вариант соответствует алгоритму на рисунке 7.4 и предусматривает формирование циклической части программы по прерываниям от параллельного порта P2, и циклическая часть выполняется в подпрограмме прерываний.

```
#include <msp430.h>
int ua[4] = {0x00, 0x01, 0x00, 0x02};
int K, i, u, t1;
int main(void)
{
        WDTCTL = WDTPW | WDTHOLD;  // stop watchdog timer
int ua[4] = {0x00, 0x01, 0x00, 0x02};
void  wait ();          //  the  function prototype
   WDTCTL = WDTPW | WDTHOLD;   // stop watchdog timer
   P1DIR = 0xFF;
   P2DIR = 0x10;                          //      input
   P2IE = 0x1F;                           //  interrupt enable
   P2IFG = 0x00;
   P1OUT = ua[0];       //   0x00
   __bis_SR_register (CPUOFF + GIE); //cpu off, general interrupts enable
}
#pragma vector=PORT2_VECTOR
__interrupt void Port_2(void)
{
t1 = 0xC350;
     switch ( P2IN)
 { case 0x00 : ;    //  the old output;
    break;
   case 0x01 : u = 0x01; //  KB = 1
   break ;
   case 0x02 : u = 0x02; //  KH = 1
   break;
   case 0x01 : {
     for(i = 0; i <=4; i++)   // program cycle, time delay;
         { u = ua[i];
         for (K = 0xC350; K >0; K--);  //
       P1OUT = u;}
           }
   default : u = 0x00;
 }
 P1OUT = u ;
   P2IFG &= ~0x10;    // P1.2 IFG cleared
}        //   end
```

Второй вариант предусматривает формирование циклической части программным методом, и циклическая часть выполняется в главной программе. В подпрограмме прерываний только сбрасывается флаг прерываний.

```
#include <msp430.h>
 int K,i ;  //    K = 0x0000 .... 0xFFFF
 int u;
 void main(void)                        //        title
{
int ua[4] = {0x00, 0x01, 0x00, 0x02};
        WDTCTL = WDTPW | WDTHOLD;   // stop WDT
   P1DIR = 0xFF;      //      все биты настроены на вывод
```

```
    P2DIR = 0x00;      //      настройка на ввод
    P2IE = 0x1F;           //  interrupt enable
    P2IFG = 0x00;
    P1OUT = ua[0];       // начальное условие 0x00
__bis_SR_register (CPUOFF + GIE);   //  отключение процессора
       for ( ; ; )              //    если прерывание от  P2
{    switch ( P2IN)
 { case 0x00 : ;    //  the old output;
   case 0x01 : u = 0x01; //  KB = 1
   case 0x02 : u = 0x02; //  KH = 1
   break;
   case 0x08 : {
     for(i = 0; i <=4; i++)   // program cycle, time delay;
         { u = ua[i];
         for (K = 0xC350; K >0 ;K--);  //
        P1OUT = u;}
            }
   default : u = 0x00;
 }
 P1OUT = u ;
 }
#pragma vector=PORT2_VECTOR
    __interrupt void Port_2(void)
    {
   P2IFG = 0x00; // P2  IFG cleared
}
```

Применение микроконтроллеров для тестирования и диагностики. Микроконтроллер можно применять для тестирования и диагностики оборудования. Далее рассматривается метод и алгоритм тестирования и диагностики АД.

Пример 7.3. Токи в фазных обмотках АД измеряются 3-мя датчиками. Требуется обеспечить диагностику обрыва обмотки в любой из фаз и защиту электродвигателя от перегрузки. Функциональная схема системы представлена на рисунке 7.5.

Функциональная схема содержит датчики *BA* тока в трех фазах АД. Сигналы датчиков поступают на три входа аналого-цифрового преобразователя *ADC*. Микроконтроллер, имея информацию о токе в фазах двигателя, должен сигнализировать обрыв фаз *A, B, C* и отключить двигатель при перегрузке с сигнализацией перегрузки светодиодами.

Управление содержит три задачи: реверсивное управление (выполняется аналогично примеру 7.1), выявление обрыва фазы и выявление состояния перегрузки.

Выявление обрыва фазы требует анализа значений i_A, i_B, i_C тока в каждой фазе на протяжении $k \geq 3$ последовательных интервалов расчета. Ток каждой фазы синусоидальный, и естественным образом проходит через нулевое значение два раза за период. Поэтому равное нулю значение тока не есть признак обрыва фазы. Значения тока за период следует сохранять в массиве i размером (3, m_m). Здесь 3 – количество фаз (0 – фаза *A*, 1 = фаза B, 2 фаза *C*), m_m - количество хранимых значений тока каждой фазы, m – номер шага расчета, (m = 0, 1, …, m_m-1). Для выявления обрыва фазы можно применить логические выражения для значения битов *A, B, C,* выводимых в порт *P1OUT* и равных 1 при обрыве фазы

$$A = (i(0,m)==0) \&\&(i(0,m-1)==0) \&\&(i(0,m-2)==0),$$

$$B = (i(1,m)==0) \&\&(i(1,m-1)==0) \&\&(i(1,m-2)==0), \quad (7.2)$$
$$C = (i(2,m)==0) \&\&(i(2,m-1)==0) \&\&(i(2,m-2)==0).$$

Выявление состояния перегрузки требует расчета потерь энергии W_1 в обмотках фаз статора за интервал времени T_P установившегося перегрева двигателя. Обычно $T_P > 600$ *с.* Мощность потерь в обмотке каждой фазы определяется выражениями $i_A^2 R_1$, $i_B^2 R_1$, $i_C^2 R_1$. Энергия потерь приводит к нагреву электродвигателя и определяется на m-м интервале T_C выражением

$$W_1(m) = W_1(m-1) + T_C(i^2(0, m)\, R_1 + i2(1, m)\, R1 + i^2(2, m)\, R_1). \quad (7.3)$$

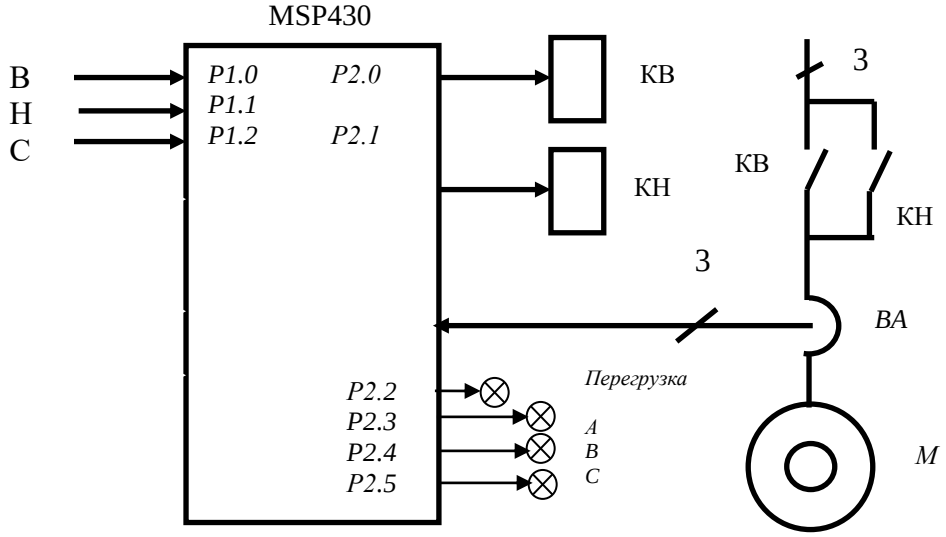


Рисунок 7.5 – Схема функциональная диагностики и защиты электропривода

За время T_P энергия потерь не должна превысить заданное значение W_{1P}. Алгоритм реверсивного управления, выявления обрыва фазы и выявления состояния перегрузки представлен на рисунке 7.6. В блоках 1 ÷ 7 выполняется логика реверсивного управления, как на рисунке 7.2. В блоке 8 выполняется выявление обрыва фазы по выражениям (7.2). В блоке 9 выполняется выявление перегрузки по выражениям (7.3). При необходимости при обрыве фазы или перегрузке в блоках 8 и 9 может быть предусмотрено отключение. Однако для некоторых технологических агрегатов достаточно сигнализации об этих неисправностях, чтобы оператор мог принять решение об отключении в зависимости от состояния технологического процесса. Например, для механизма подъема-опускания груза отключение допустимо лишь при окончании цикла движения.

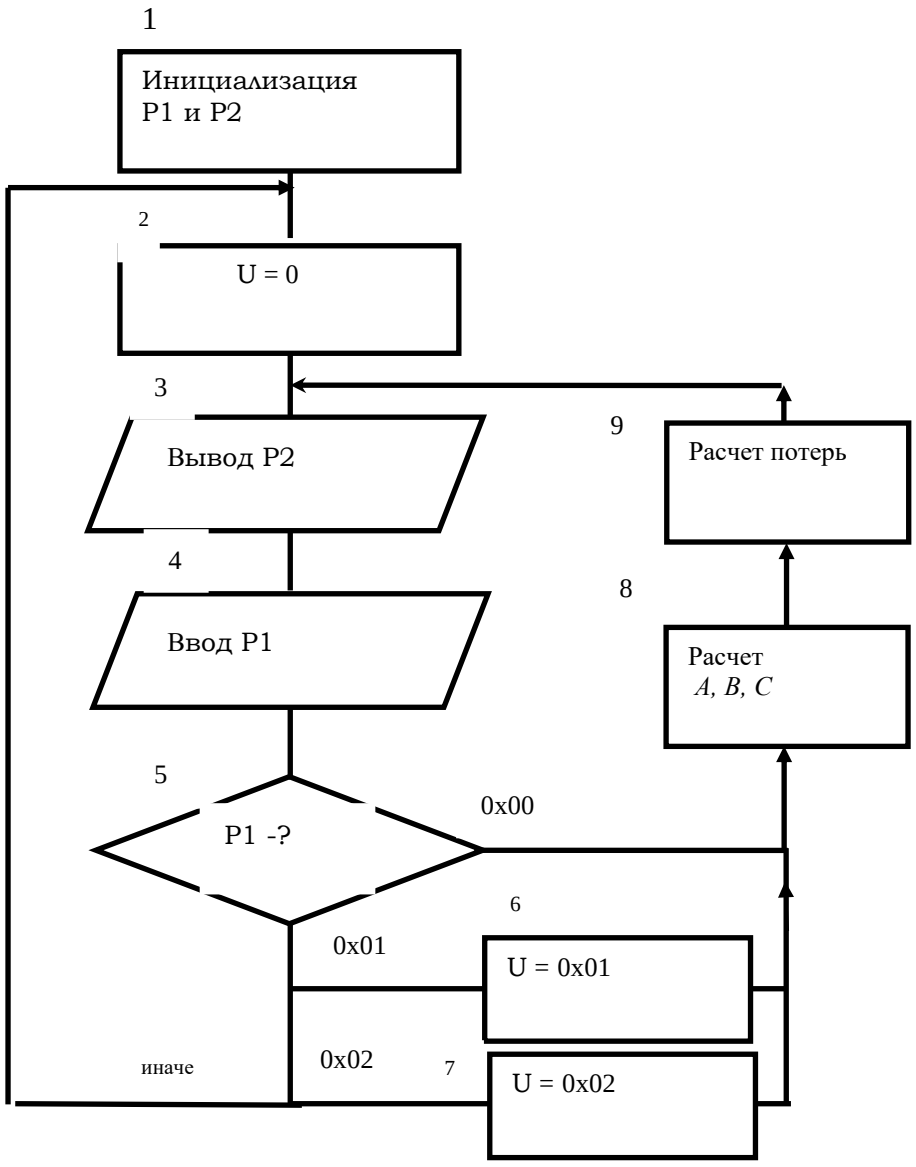


Рисунок 7.6 - Алгоритм управления

Применение микроконтроллеров для рационализации энергопотребления является перспективным ввиду низкой стоимости микроконтроллеров, дополнительных датчиков и реле, которые могут понадобиться для автоматизации и позволяют обеспечить значительную экономию за счет энерго- и ресурсосбережения.

Известен способ экономии электроэнергии за счет частотно управляемого электропривода. Для организации частотного управления необходимо применение дорогостоящего преобразователя электрической энергии. В то же время для некоторых общепромышленных механизмов, в том числе конвейеров, вентиляционных и насосных установок невысокой мощности, и других механизмов энергосбережение можно обеспечить за счет рациональной автоматизации на основе микроконтроллеров.

Функциональная схема электропривода с управлением от микроконтроллера представлена на рисунке 7.7.

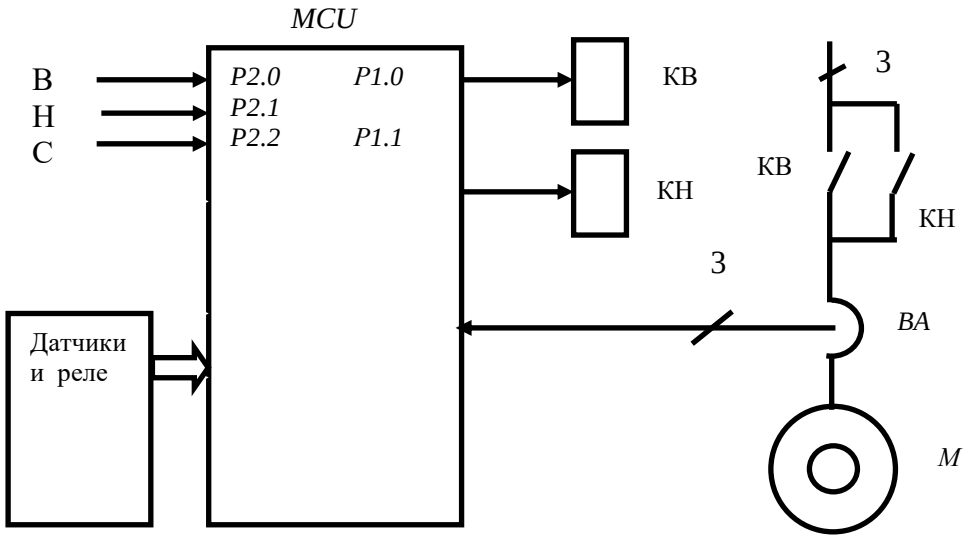


Рисунок 7.7 – Схема функциональная не регулируемого электропривода

На рисунке 7.7 с пульта управления на микроконтроллер *MCU* поступают сигналы В, Н, С. С датчиков и реле на микроконтроллер могут поступать сигналы, характеризующие состояние технологического объекта, через входы *ADC,* биты параллельных портов и другие средства интерфейса. В частности, сигналы тока статора электродвигателя во многих случаях применяются, чтобы оценить момент нагрузки на валу электродвигателя.

Алгоритм управления должен формировать включенное либо отключенное состояние электродвигателя в зависимости от состояния технологического объекта. В инициализационной части следует предусмотреть инициализацию устройств ввода и, для вывода, двух бит состояния контакторов. В циклической части должен быть организован ввод переменных сигналов и формирование логики управления на основании анализа этих сигналов.

В результате выполнения программы управления формируются сигналы включенного состояния контакторов, если необходима работа механизма, или отключение, если состояние технологического объекта продолжает оставаться допустимым и работа механизма не требуется.

Например, электропривод вентилятора должен оставаться включенным, пока концентрация вредных примесей в воздухе помещения превышает допустимый уровень, или температура не соответствует допустимой. Иначе, вентиляция может быть отключена.

Электропривод конвейера может оставаться включенным лишь при условии наличия нагрузки на нем.

Электропривод насоса может быть включен и работать до достижения заданного напора и далее отключаться до момента, когда датчик напора даст показание недопустимо низкого напора.

В некоторых случаях возможен упрощенный вариант логического управления оборудованием, не требующий реле и датчиков состояния технологического объекта: это управление оборудованием в функции времени.

Контрольные вопросы

1 Как организовать ввод на вход микроконтроллера аналоговых сигналов от датчиков?
2 Как организовать ввод на вход микроконтроллера сигналов от реле уровня, давления, температуры?
3 Как организовать ввод на вход микроконтроллера сигналов от фотодиодов?
4 Как организовать вывод сигнала с выхода микроконтроллера на вход технологического объекта?

2. ПРАКТИЧЕСКИЙ РАЗДЕЛ

Лабораторные работы по курсу

«Микропроцессоры в автоматизации»

для магистрантов специальности 7-06-07 13-04 «Автоматизация»

ЛАБОРАТОРНАЯ РАБОТА №1. АРХИТЕКТУРА 16 РАЗРЯДНОГО МИКРОКОНТРОЛЛЕРА MSP430

1.1. Цель работы

1.2. Архитектура микроконтроллера

Архитектурой вычислительного устройства называется его внутренняя структура (функциональная схема) в совокупности с системой команд. Изучение архитектуры является первым этапом, необходимым для использования микроконтроллера в системах автоматики и управления электроприводами.

Микроконтроллер MSP430 имеет ряд преимуществ по сравнению с другими микроконтроллерами. Это ультранизкое потребление мощности, производительность 16 MIPS (16 миллионов операций в секунду), 16 разрядный процессор с сокращенным набором команд (RISC CPU, Reduced Instruction Set Computer Central Processing Unit). Микроконтроллер MSP430 имеет полный набор преобразований сигнала в кристалле, □27 команд, выполняемых каждая за один цикл (при регистровой адресации), □7 способов адресации, применимых со всеми командами. Микроконтроллер допускает программирование на языке ассемблера или на *C, C++* с использованием интегрированной среды разработки CCS (Code Composer Studio).

Вычислительное устройство обычно состоит из процессора, устройств памяти и устройств для ввода и вывода информации. В состав вычислительного устройства входит блок питания и источник тактовых сигналов для согласования работы устройств по времени. Обработанная информация выводится с помощью устройства вывода (рисунок 1.1).

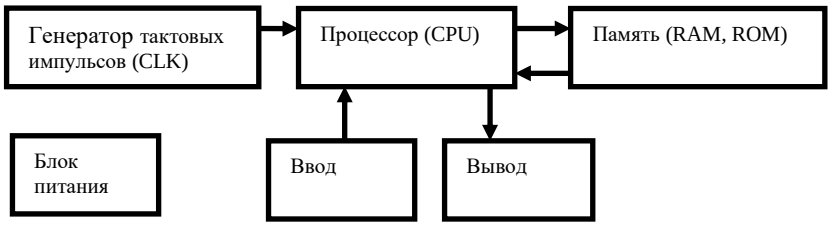


Рисунок 1.1 – Вычислительное устройство.

Процессором называется устройство для преобразования информации, представленной в двоичном коде. *Микропроцессором* называется процессор в виде одной интегральной микросхемы (на одном кристалле). Микропроцессоры бывают универсальные, они используются на системных платах персональных компьютеров, и специальные. Специальные микропроцессоры используются, например, для цифровой обработки сигналов (ЦОС, или, в английском сокращении DSP, Digital Signal Processor).

Микроконтроллером называется специальное микропроцессорное вычислительное устройство для управления, выполненное в виде одной микросхемы. Современные устройства управления выполняются на микроконтроллерах. Таким образом, микроконтроллер в одном кристалле содержит как процессор (ядро), так и периферийные устройства (устройства памяти и устройства ввода и вывода).

Микроконтроллеры семейства MSP430 производства Texas Instruments содержат 16-разрядное центральное процессорное устройство (ЦПУ), периферийные модули, генератор тактовых сигналов до 16 МГц.

Микроконтроллеры предназначены для встраиваемых систем, поэтому имеют весьма низкое энергопотребление.

Схемы внешних выводов микроконтроллеров **MSP430x22x4** показана на рисунке 1.2. Каждый из внешних выводов может выполнять несколько функций, указанных через косую черту. Назначение выводу той или иной функции выполняется программными средствами в процессе инициализации интерфейса.

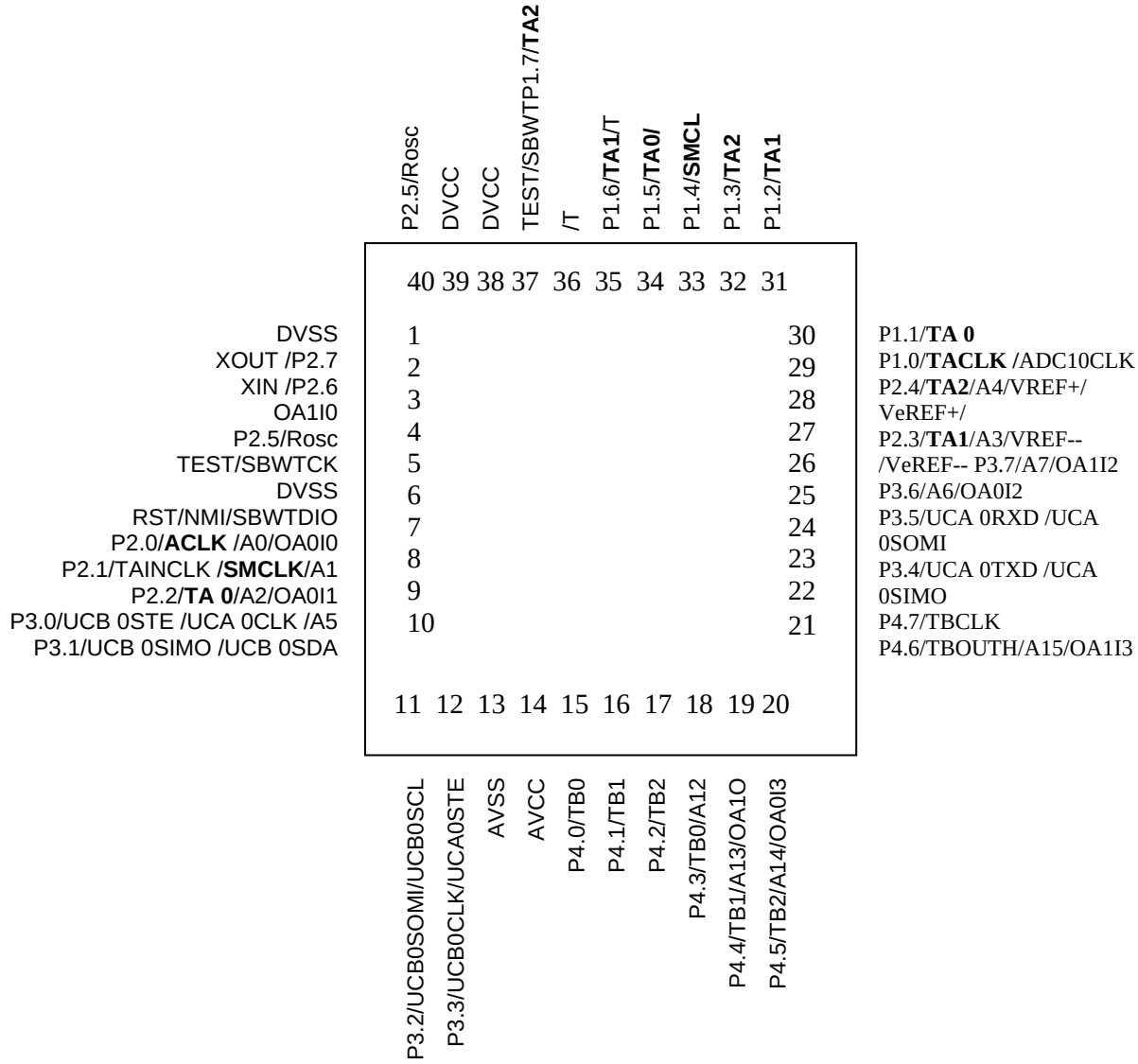


Рисунок 1.2 – Микросхема **MSP430x22x4** с внешними выводами.

Среди внешних выводов любого микроконтроллера имеются выводы для сброса (RST), подключения источника напряжения (VCC, VSS), причем VSS – общая точка. Микроконтроллеры MSP430 содержат аналого-цифровой преобразователь (АЦП, ADC Analog Digital Converter). Выводы AVCC, AVSS подключаются для питания аналоговых цепей, DVCC, DVSS – для цифровых. Функции остальных выводов становятся понятными после изучения архитектуры микроконтроллера.

Микроконтроллеры MSP430 требуют напряжения питания 3,3 В, и потребляют ток 250 мкА/MIPS (270 мкА при 16 МГц). В режимах

ограниченного энергопотребления (сон и глубокий сон) потребляемый ток может быть 1 мкА .

В микроконтроллере применяется *магистральный принцип обмена* информацией между устройствами. Это означает, что все устройства микросхемы соединены с магистралью, через которую и обмениваются информацией по определенным правилам. Магистраль состоит из шины данных DB (Data Bus), шины адреса AB (Address Bus) и шины управления для передачи сигналов управления. Каждая шина предназначена для передачи одного из этих видов информации, представленной в двоичном коде. Каждая шина содержит параллельные проводники по количеству разрядов шины. Шинный формирователь обеспечивает одно из трех состояний каждого проводника шины: 0, 1, и «отключено».

Функциональная схема микроконтроллера MSP430 представлена на рисунке 1.3. Центральное процессорное устройство CPU (Central Processing Unit) выполняет обработку информации, представленной в двоичном коде и передаваемой в процессор по шине данных от внешних устройств и памяти. Таким образом, вначале выполняется *чтение* данных, затем их обработка в CPU и *запись* в память либо во внешние устройства.

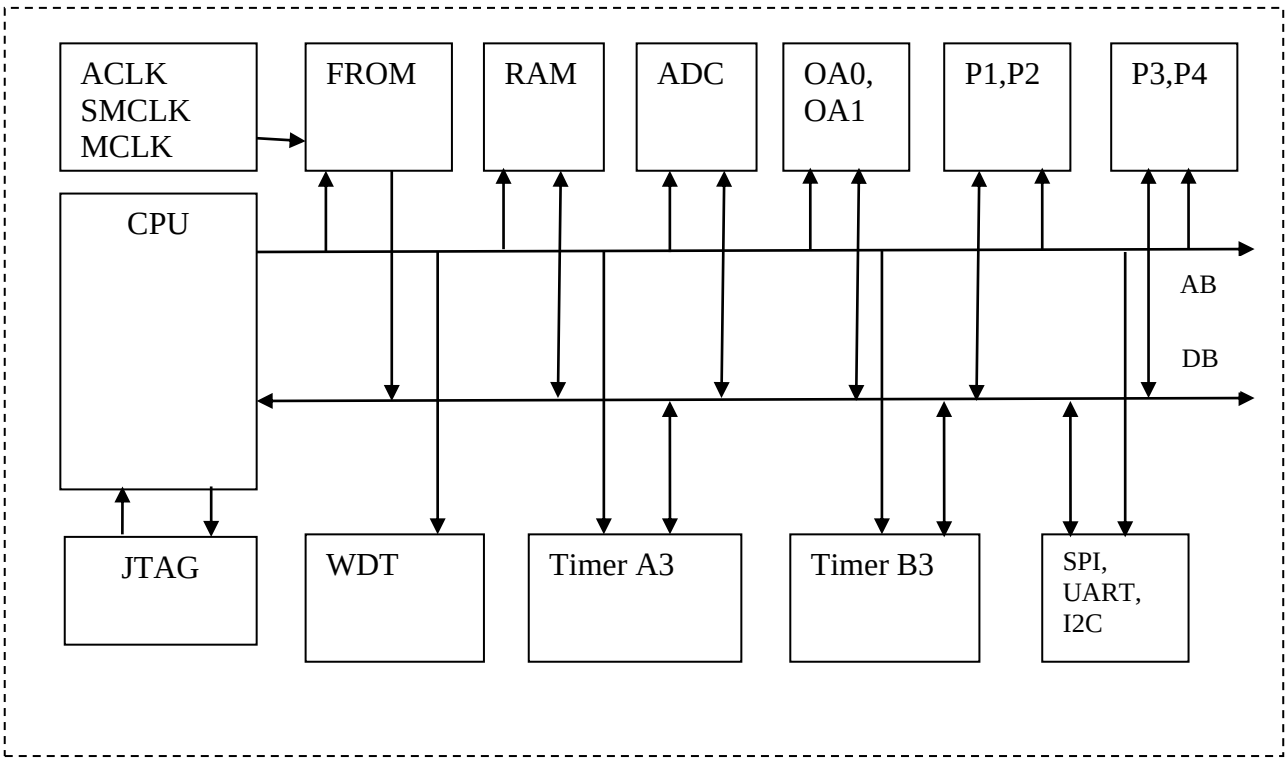


Рисунок 1.3– Функциональная схема **MSP430x22x4**

Адрес формируется в CPU и выводится на шину адреса для чтения либо для записи данных. Тактовый генератор формирует последовательности тактовых импульсов ACLK (32 kHz), SMCLK (Sub-system Master Clock, до 16 MHz), MCLK (Master Clock, 16MHz). Микроконтроллер содержит постоянное запоминающее устройство (ПЗУ, ROM, Read Only Memory) в виде флэш-памяти FROM (Flash ROM) и оперативное запоминающее устройство (ОЗУ, RAM,

Random Access Memory), которые предназначены для хранения программ и данных.

Для ввода аналоговых данных предназначен аналого-цифровой преобразователь ADC(Analog-Digital Converter), а для ввода и вывода цифровых данных – параллельные 8- разрядные порты ввода-вывода P1-P4.

Интерфейс JTAG предназначен для внутрисхемной (без отключения микроконтроллера от внешней схемы) отладки программного обеспечения.

Сторожевой таймер WDT предназначен для вывода микроконтроллера из состояния зависания. Таймеры 16-разрядные Timer A3, Timer B3 предназначены для формирования интервалов времени, времязадающих функций, формирования зависящих от времени сигналов и могут быть использованы в режиме счетчиков. Последовательный интерфейс SPI, UART, I2C предназначен для обмена информацией последовательным двоичным кодом, и дает ряд преимуществ.

Функциональная схема CPU (ЦПУ) представлена на рисунке 1.5. Принцип действия ЦПУ заключается в выполнении *командного цикла*, алгоритм которого реализуется аппаратными или программными средствами в *устройстве управления*. Для этого ЦПУ содержит внутренние регистры специального назначения R0…R3 и регистры R4…R15 общего назначения. Среди регистров специального назначения **R0=PC** (Programmer Counter) - программный счетчик, **R1=SP** (Stack Pointer) - указатель стека, **R2=SR** (Status Register) – регистр состояния.

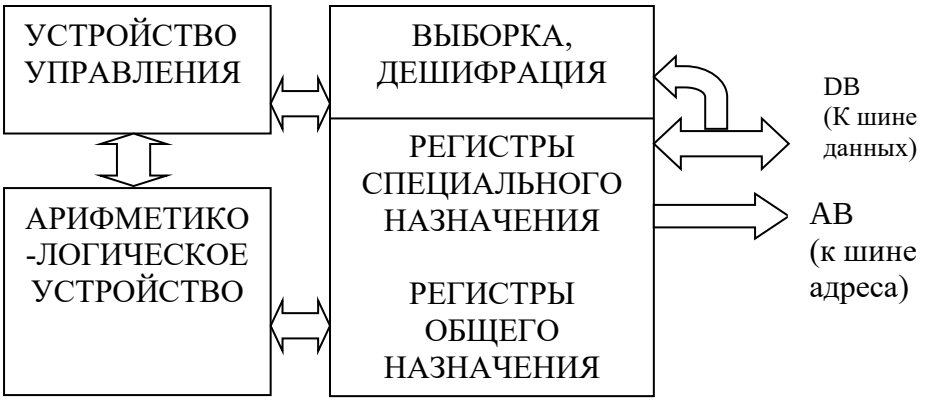


Рисунок 1.4– Центральное процессорное устройство (ЦПУ).

Арифметико-логическое устройство (АЛУ) предназначено для выполнения арифметических и логических операции, и представляет собой комбинационную логическую схему. АЛУ не содержит регистров памяти. Регистры специального назначения необходимы для организации работы микроконтроллера, а регистры общего назначения используются программистами для хранения данных.

Командный цикл выполняется следующим образом. Адрес формируется в ЦПУ *программным счетчиком* **PC.** После включения или сброса микроконтроллера программный счетчик PC указывает начальный адрес **0x8000**. По этому адресу происходит *выборка команды*, то есть команда программы, записанная в память, передается по шине данных в процессор для дешифрации. По результату дешифрации устройство управления организует выполнение команды. После выборки команды программный счетчик **PC**

инкрементируется (увеличивается на 2). Если нет команды останова, процессор переходит к выполнению следующего командного цикла.

1.2. Организация памяти и внутренние регистры процессора

Размер адресного пространства процессора определяется количеством разрядов программного счетчика, равным количеству разрядов шины данных. Так, для 16 разрядного **PC** адресное пространство содержит 2^{16}-1=65 535 адресов, то есть могут быть доступны 64Кб памяти. Однако общий объем адресуемой памяти составляет 128 Кб. Адреса интерфейсных (периферийных) устройств (портов ввода вывода, АЦП, таймеров) включаются в адресное пространство микроконтроллера. Микроконтроллер обладает внутренней памятью. Структура адресного пространства представлена в таблице 1.1.

Внутренние регистры процессора представлены на рисунке 1.5. Всего микроконтроллер имеет 16 регистров по 16 разрядов в каждом, их имена: R0, R1,…,R15. К каждому регистру можно обращаться в программах как по имени (регистровая адресация), так и по адресу (прямая адресация).

Таблица 1.1 Организация памяти

		MSP430F227x
Память *Основная: вектор прерывания* *программ* *Информационная* *Загрузочная*	*объем* *Flash* *Flash* *Flash* *объем* *ROM*	*32KB Flash* *0FFFFh--0FFC0h* *0FFFFh--08000h* *256 Byte* *010FFh--01000h* *1KB* *0FFFh--0C00h*
RAM данных	*Size*	*1KB* *05FFh--0200h*
Периферийные устройства *SFR*	*16-bit* *8-bit* *8-bit*	*01FFh--0100h* *0FFh--010h* *0Fh--00h*

Рисунок 1.5 –Внутренние регистры процессора.

Регистры специального назначения (специальных функций, специальные регистры) - это программный счетчик **PC (R0),** указатель стека **SP (R1),** регистр состояния **SR (R2)** и генератор констант **CG1, CG2 (R2,R3).**

Программный счетчик **PC (R0)** предназначен для генерации адреса команд. После включения питания и после сброса **PC** содержит значение **0x8000** начального адреса области хранения программ. Выполняя командные циклы, **CPU** инкрементирует программный счетчик, значение которого передается на шину адреса для чтения следующей команды.

Указатель стека **SP (R1)** предназначен для организации стековой адресации в заданной области оперативной памяти.

Регистр состояния **SR (R2)** предназначен для хранения флагов **C, Z, N, V** состояния программы после выполнения логической либо арифметической операции. Регистр состояния представлен на рисунке 1.6. Кроме флагов регистр состояния содержит биты управления процессора: общее разрешение прерываний **GIE** (General Interrupt Enable), бит отключения процессора **CPU OFF** и биты управления осциллятором.

15,,,,,,9	8	7	6	5	4	3	2	1	0
-	**V**	SCG1	SCG0	OSC OFF	CPU OFF	GIE	**N**	**Z**	**C**

Рисунок 16– Регистр состояния

Далее расположены регистры общего назначения R4…R15, 8- разрядные периферийные устройства в соответствии со схемой на рисунке 1.5.

1.4. Система команд и способы адресации

Система команд является полностью ортогональной, то есть любую команду можно применять к любым из внутренних регистров и регистров памяти, применяя различные способы адресации для каждого из операндов.

Имеются команды с одним и двумя операндами. Система команд допускает способы адресации, представленные в таблице 2. В таблице приняты обозначения:

- непосредственная адресация; & - прямая адресация; @ - косвенная адресация;

MEM, - адрес памяти; EDE,TONI и т.п. – произвольные имена переменных;

Непосредственный режим адресации может быть применен только для первого операнда.

Таблица 1.2. -Способы адресации

Способ адресации	Синтаксис	Примеры	Комментарии
Регистровый	*MOV Rs,Rd*	*MOV R10,R11*	*R10 ----> R11*
Индексный	*MOV X(Rn),Y(Rm)*	*MOV 2(R5),6R6)*	*M(2+R5)----> M(6+R6)*
Символический (PC relative)	*MOV EDE,TONI*	*MOV EDE,TONI*	*M(EDE) ----> M(TONI)*
Абсолютный	*MOV &MEM,&TCDAT*		*M(MEM)----> M(TCDAT)*
Косвенный	*MOV @Rn,Y(Rm)*	*MOV @R10,Tab(R6)*	*M(R10) ----> M(Tab+R6)*

Составляя программу, следует размещать данные во внутренних регистрах микроконтроллера и памяти с учетом организации адресного пространства, представленной в таблице 1. Система команд представлена в таблице3.

Система команд содержит операции пересылки, арифметические и логические операции, а также команды перехода. Арифметические команды - это суммирование, инкрементирование, вычитание и декрементирование. Умножение выполняется посредством обращения к аппаратному умножителю либо используя про граммы умножения.

При составлении программы можно использовать операции с 16-разрядными словами и байтами. Байтовые команды должны иметь расширение мнемокода***. b***, например, команда пересылки байта из ***R11*** в ***R12*** имеет вид

mov.b R11,R12.

В этом случае команда будет выполнена над младшими байтами регистров. Адрес 16 разрядного слова всегда четный, причем по четному адресу хранится младший байт слова. Данные величиной в байт могут иметь как четный, так и нечетный адрес.

Выполнение команд условных переходов зависит от состояния флагов, хранящихся в регистре состояния SR, а флаги отображают свойства последнего результата работы АЛУ.

1.5. Среда разработки проекта

Запустить программу **CCS.** В результате откроется окно для разработки проекта. Далее можно в верхней строке окна выбрать: **New C/C++project.** Дать имя проекту в открывшемся окне. Выбрать вариант устройства, используемого в проекте MSP430x2xx. Если проект предполагается выполнять на ассемблере, его следует конфигурировать как ассемблерный, иначе - как проект на языке С.

Войти в директорию, выполнить **Project →Add File to Active Project** и выбрать в директории пример файла на ассемблере либо языке С из папки **slac123d** на диске D.

Определить интерфейс связи с устройством. Для этого в файле ***.ccxml** в проекте выбрать вид соединения **TI MSP430 USB1.** Назначить устройство **MSP430F2274.**

Выполнить компиляцию **Project→Build Active Project.** Прочесть сообщения об ошибках и исправить их.

Соединить устройство с системным блоком через **USBx.** Выполнить **Target→Debug Active Project** для записи сформированного машинного кода программ проекта в память устройства.

Вид окна проекта настраивается командой **view.** В открытых окнах можно увидеть содержимое памяти и внутренних регистров процессора, а так же и другую полезную информацию.

1.6. Порядок выполнения работы

1. Прочесть инструкцию.
2. Ознакомиться с устройством на базе микроконтроллера **MSP430.**
3. Включить компьютер и загрузить интегрированную среду разработки **CCS.**
4. Создать проект на языке ассемблера или С.
5. Подключить отладочную плату.
6. Выполнить компиляцию (**Build**) и загрузку проекта в устройство (**Debug**)
7. Настроить окно проекта так, чтобы были видны внутренние регистры микроконтроллера, используемые в программе. (**View→Registers).**
8. Ознакомиться с внутренними регистрами и интерфейсными модулями микроконтроллера до и после выполнения программы.

1.7. Варианты заданий

1. Составить алгоритм и программу формирования интервала времени.
2. Составить алгоритм и программу ввода значений *x, y* через младшую и старшую тетрады порта P2IN и вывода их конъюнкции через P1OUT. Как выполняются логические команды?
3. Для чего нужна косвенная адресация? Составить пример программы с использованием косвенной адресации.
4. Составить пример программы ввода значения *x* и вывода *y = 2x* через параллельный порт, дать полное описание каждой команды.

5. Сформулировать правила написания исходного текста программы на ассемблере.

6. Составить список интерфейсных устройств микроконтроллера **MSP430F2274** с указанием их назначения.

7. Создать проект. Проанализировать директорию проекта и содержащихся в проекте файлов. Дать описание файлов проекта.

1.8. Содержание отчета

1. Цель работы

2.. Указать назначение каждого из специальных регистров и интерфейсных модулей.

4. Выполнить задания из п. 1.7.

1.9. Контрольные вопросы

1.Что относится к ядру и периферии микроконтроллера?

2.Что такое адресное пространство?

3.Каков алгоритм выполнения командного цикла?

4.Как формируется адрес?

5.Что является источником адреса?

6.Записать имена внутренних регистров процессора.

7.Назначение и принцип действия процессора.

8.Каково назначение специальных регистров?

9.Назначение и принцип действия таймера.

10.Назначение и принцип действия параллельных и последовательных портов.

11.Назначение и принцип действия АЦП.

12.Назначение генератора тактовых импульсов, какие тактовые частоты возможны в микроконтроллере?

13.Как ограничивается энергопотребление микроконтроллера?

14.Как увидеть содержимое внутренней памяти микроконтроллера?

15.Как увидеть содержимое портов ввода-вывода?

16Как выполняется сброс процессора, для чего он необходим и что при этом происходит?

17.Как ограничивается энергопотребление микроконтроллера?

Таблица 1.3. Система команд микроконтроллеров MSP430

Мнемокод	**Комментарии**	**Примеры**	**Кол. тактов**
Команды с 2 операндами			
MOV s,d	s→d	MOV R4, R5	**1**
ADD s,d	s+d→d	ADD R7, R6	**1**
ADDC s,d	s+d+c→d		
SUB s,d	d-s→d	SUB R5,&P1OUT	4
SUBC s,d	d-s+c-1→d		
CMP s,d	d-s→установка флагов;	CMP #003h, R2	2
DADD s,d	s+d+c→d		
BIT s,d	**s**∩d→установка флагов;	BIT #001h,&P1OUT	5
BIC s,d	$\overline{\boldsymbol{s}}$∩d→ d		
BIS s,d	$\overline{\mathbf{s}}$Ud→d	BIS R4,0(R12)	4
XOR s,d	**s**$\overline{\forall}$d→d		
AND s,d	**s**$\overline{\cap}$d→d	AND @R4+, R5	2
Команды с 1 операндом			
RRC d	Сдвиг вправо ч. перенос		
RRA d	Сдвиг вправо, мл. бит в перенос		
PUSH s	Запись в стек		
SWPB d	Перестановка байтов		
CALL d	Вызов подпрограммы		
RETI d	Возврат		
SXT d	Заполнение знаком		
Команды перехода			
JZ m	Если 0,(z=1)		
JNZ m	Если не 0,(z=0)		
JC m	Если перенос,(c=1)		
JNC m	Если не перенос,(c=0)		
JN m	Если отрицательно,(N=1)		
JGE m	Если N$\overline{\forall}$V=0		
JL m	Если N$\overline{\forall}$V=1		
JMP m	Без условия		

Флаг V применяется только для знаковых величин.

Расширение .B указывает на операцию с байтами (MOV.B s,d).

Расширение .W указывает на операцию с 16-рядными словами (MOV .W s,d или MOV s,d). Расширение .W можно опустить.

Все команды перехода выполняются за 2 такта.

Все команды с регистровой адресацией выполняются за 1 такт. Другие виды адресации требуют большего числа тактов.

2 ЛАБОРАТОРНАЯ РАБОТА №2. СОЗДАНИЕ ПРОЕКТА АВТОМАТИЗАЦИИ В ИНТЕГРИРОВАННОЙ СРЕДЕ CODE COMPOSER STUDIO™ (CCS) ДЛЯ МИКРОКОНТРОЛЛЕРОВ MSP430

2.1. Цель работы.

Целью работы является выполнение программы в микроконтроллере **MSP430** и анализ результатов выполнения с использованием интегрированной среды **CCS.**

2.2. Устройство eZ430RF2500.

На рисунке 3.1 представлено устройство **eZ430RF2500**, предназначенное для отладки проектов автоматизации на основе микроконтроллера **MSP430**. Устройство **eZ430RF2500** обеспечивает аппаратные и программные возможности беспроводной связи по радиоканалу.

Устройство построено на базе микроконтроллера **MSP430** и микросхемы **CC2500** и позволяет контролировать температуру окружающей среды с помощью терморезистора. Сигнал температуры преобразуется в градусы по Цельсию или по Фаренгейту и может быть передан на расстояние по радиочастотному каналу **RF2500**, формируемому микросхемой **CC2500**. Через интерфейс USB выполняется передача сигнала о температуре на компьютер, а от компьютера на устройство передается энергетическое питание. Если на компьютере инсталлирована программа eZ430RF2500 Sensor Monitor, значение температуры отображается на экране в специальном окне. Устройство eZ430RF2500 позволяет принимать сигналы о температуре от других таких же устройств. Тогда на экране видны значения температуры, передаваемые от всех устройств.

Устройство eZ430RF2500 может быть так же использовано для загрузки и отладки программного обеспечения микроконтроллеров MSP430F2274 с применением среды разработки CCSv4.2 или IAR. Пользователь может выполнять прикладную программу на полной скорости и пошагово.

Интерфейс USB, размещенный на плате устройства, получает сигнал микроконтроллера MSP430 через его интерфейс UART. Имеются внешние выводы, доступные для применений.

Два вывода общего применения подсоединены к зеленому и красному светодиодам для визуализации работы портов. Имеется кнопка для нужд пользователя.

Устройство eZ430RF2500 может быть использовано как автономное, если его соединить с платой батареи с помощью разъема, как показано на рисунке 2.2 и замкнуть пластмассовую перемычку.

2.3. Использование программного обеспечения Code Composer Studio™

Следует создать проект и сделать проект активным: **Set Project as Active.** Для компиляции программ и построения проекта следует выполнить **Project →Build Active Project.** Прочесть список ошибок, если они имеются, и доработать программу, чтобы после повторной компиляции появилось сообщение **0 errors**, означающее отсутствие ошибок.

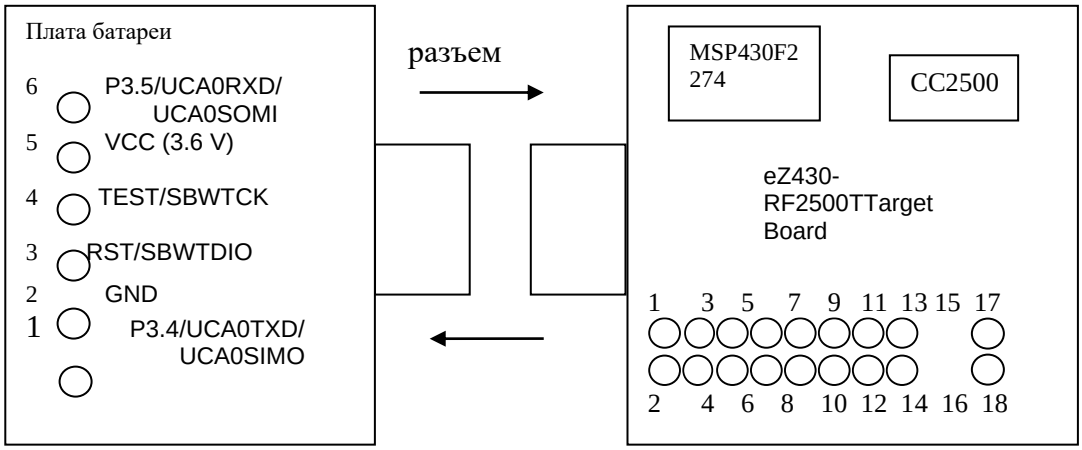


Рисунок 2.2 – Соединение устройства с платой батареи.

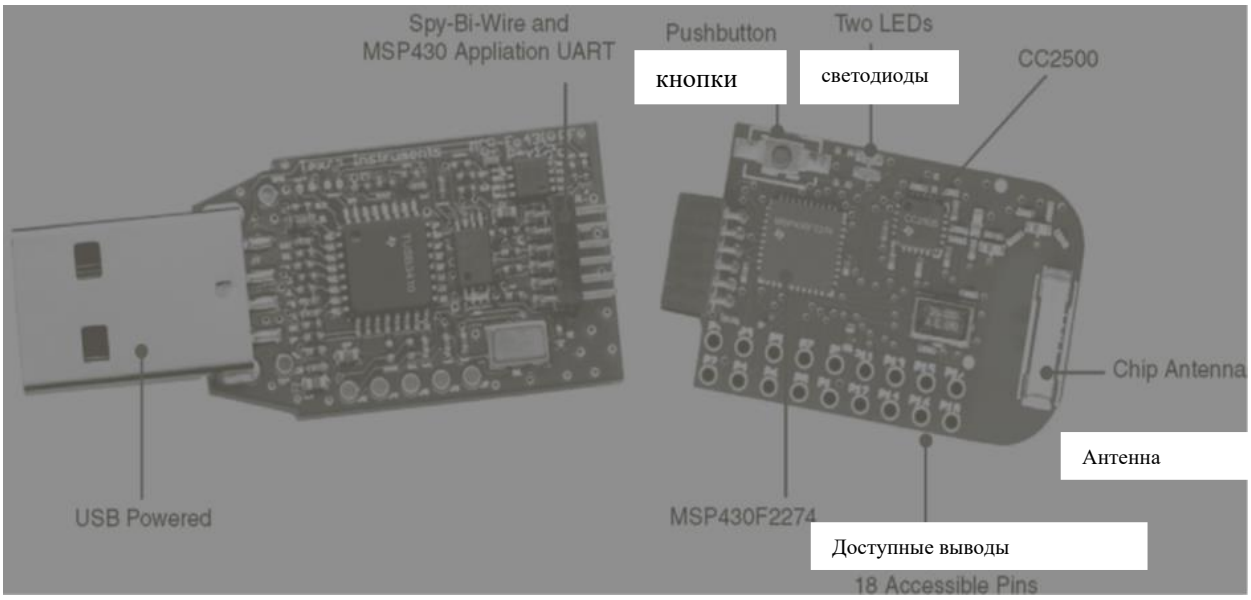


Рисунок 2.3 - Устройство eZ430RF2500.

Для загрузки приложения в устройство (микроконтроллер **MSP430**) и взаимодействия прикладной программы с устройством следует подключить устройство через разъем **USB** и перейти к **Target → Debug Active Project.** По этой команде стирается содержимое памяти целевого устройства, память устройства программируется, устройство перезапускается.

Для конфигурации соединения с устройством в проекте открывают файл ***.ccxml** для выбора интерфейса или для принятия интерфейса по умолчанию.

Чтобы установить связь с устройством служит команда: **Target → Debug.** По этой команде стирается содержимое памяти целевого устройства, в память устройства записывается программное обеспечение активного проекта, устройство перезапускается. Для запуска прикладной программы служит команда: **Target → Run**.

Для остановки режима отладки служит команда: **Target → Terminate All,** после которой связь с устройством прекращается и устройство можно отсоединить.

2.5. Порядок выполнения работы

1. Создать проект и делать проект активным.
2. Откомпилировать проект.
3. Подключить устройство и записать проект в устройство.
4. Выполнить проект в пошаговом режиме, проанализировать результаты выполнения.
5. Внести в проект изменения в соответствии с заданным вариантом (п.2.7).
6. Откомпилировать и выполнить проект, записать содержимое регистров ядра и параллельных портов.

2.6. Содержание отчета

1. Цель работы
2. Состав проекта и назначение его файлов.
3. Алгоритм и программа в соответствии с заданным вариантом с комментариями.
4. Результаты компиляции и выполнения программы с комментариями (содержимое регистров ядра и регистров портов на этапах выполнения программы).

2.7. Варианты задач

1. Составить алгоритм и программу вывода в P1OUT значений 1, 2, 3, … с интервалом времени 0,5 с.
2. Составить алгоритм и программу вывода сигналов на 2 светодиода с интервалом времени 1 с.
3. Составить алгоритм и программу вывода сигналов на 2 светодиода поочередно с интервалом времени 0.3 с.
4. Составить алгоритм и программу вывода сигналов на 2 светодиода в противофазе с интервалом времени 0.2 с.
5. Составить алгоритм и программу вывода сигналов на светодиод с интервалом времени 2 с.
6. Составить алгоритм и программу вывода в параллельный порт заданного значения с заданным интервалом времени.
7. Организовать ввод данных в порт P2 и вывод в порт P1 через 1 с.
8. Составить алгоритм и программу вывода константы через порт P4.
9. Составить алгоритм и программу вывода константы в порт P3.
10. Составить алгоритм и программу вывода константы в порт P2.

2.8. Контрольные вопросы

1. Для чего необходим проект автоматизации?
2. Как создать новый проект?
3. Что содержит проект?
4. Как в проект добавить файл?
5. Как выполнить компиляцию проекта?
6. Как проект записать в память микроконтроллера?
7. Какие настройки проекта можно изменить?

3 ЛАБОРАТОРНАЯ РАБОТА №3. АЛГОРИТМЫ И ПРОГРАММЫ АВТОМАТИЗАЦИИ ДЛЯ МИКРОКОНТРОЛЛЕРОВ

3.1. Цель работы.

Целью работы является разработка алгоритма и программы автоматизации для микроконтроллера

3.2. Разработка алгоритма

Алгоритм для автоматизации выполняется *в реальном времени*. Режим реального времени предполагает, что устройство должно реагировать на внешние события, которые в виде входных сигналов поступают на внешние выводы микроконтроллера. В соответствии со входными сигналами формируется реакция устройства на них в виде выходных сигналов (событий), в нужные моменты времени. Это означает, что алгоритм должен начинаться *инициализационной частью,* в которой определяются режимы работы интерфейсных средств микроконтроллера и формируются начальные условия. Поскольку программа реального времени должна обеспечивать реакцию на входные воздействия в течение всего времени функционирования, алгоритм должен иметь *циклическую часть.*

Циклическая часть должна содержать чтение входных сигналов, расчет и вывод выходных сигналов и, если необходимо, формирование интервалов времени. Пример алгоритма представлен на рисунке 3.1.

3.3. Разработка программы

Пример исходного текста программы на языке С показан далее. Программа предназначена для включения и отключения светодиода с заданным периодом. Программа содержит инициализационную часть, где задается адрес вершины стека, остановлен сторожевой таймер и настроен на вывод один бит порта. В циклической части формируется попеременное включение и отключение светодиода операцией **(^)** (исключающее ИЛИ).

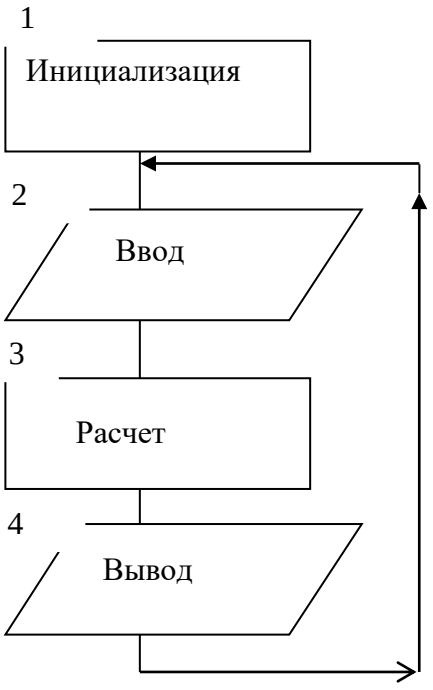


Рисунок 3.1 – Алгоритм управления

```
#include <msp430x22x4.h">

int i;
void main(void)
{
  WDTCTL = WDTPW | WDTHOLD;
   P1DIR |= 0x01;
  for (  ;  ;  )
 {
    P1OUT ^= 0x01;
  i = 50000;
         do (i--);
    while (i != 0);
 }]
```

3.4 Порядок выполнения работы

1. Составить алгоритм и программу в соответствии с заданным вариантом.
2. Открыть проект. В проект добавить файл с текстом программы.
3. Откомпилировать проект **Project → Build Active Project**. При необходимости устранить ошибки.
4. Для взаимодействия прикладной программы с устройством выполнить **Target → Debug Active Project**.
5. Настроить окно проекта так, чтобы были видны внутренние регистры микроконтроллера, используемые в программе. (**View→Debug, View→Registers).**
6. Запустить программу на выполнение **Target →Run.** Убедиться в правильности работы программы. Записать содержимое регистров с промежуточными и конечными результатами, чтобы показать правильность выполнения программы.
7. Остановить выполнение программы**.**
8. Изменить выдержку времени, запустить программу и посмотреть, как это повлияет на работу устройства.

3.5. Содержание отчета

1 Цель работы

2 Схема функциональная, расчет, алгоритм и программа в соответствии с заданным вариантом с комментариями.

3 Результаты компиляции и выполнения программы с комментариями (содержимое регистров ядра и регистров портов на различных этапах выполнения программы).

3.6 Варианты заданий

1. Составить алгоритм и программу последовательного через заданное время включения двух электродвигателей посредством контакторов.
2. Составить алгоритм и программу реверсивного управления электродвигателем.
3. Составить алгоритм и программу включения привода при условии наличия сигналов от двух кнопок ПУСК и отключения от любой из двух кнопок СТОП.
4. Составить алгоритм и программу формирования разгона с постоянным ускорением до заданной скорости по сигналу ПУСК и торможения по сигналу СТОП.
5. Составить алгоритм и программу формирования разгона с постоянным ускорением и затем торможения с постоянным замедлением.
6. Составить алгоритм и программу последовательного включения 2-х приводов через заданное время и их отключения в обратном порядке.
7. Составить алгоритм и программу включения одного из 2-х приводов, через заданное время его отключения и включения второго привода.
8. Составить алгоритм и программу управления возвратно-поступательным движением рабочего органа.

3.7. Контрольные вопросы

1 Как управлять электромагнитными реле и контакторами от микроконтроллера?

2. Как зависят алгоритм и программа от схемы функциональной?

3. Как организовать логическое управление включением и отключением оборудования с использованием прерываний?

4. Как организовать цикличность программным методом?

5. Как организовать цикличность программно-аппаратными методами?

6. Как организовать цикличность от программируемого таймера?

7. Как зависит алгоритм от организации ввода и вывода информации?

4.ЛАБОРАТОРНАЯ РАБОТА №4. РАЗРАБОТКА АЛГОРИТМОВ И ПРОГРАММ С ИСПОЛЬЗОВАНИЕМ ПРОГРАММИРУЕМОГО ТАЙМЕРА

4.1. Цель работы

Целью работы является разработка алгоритмов и программ для микроконтроллера MSP430 с использованием программируемого таймера для задания интервалов времени.

4.2. Разработка алгоритма

Алгоритм должен начинаться *инициализационной частью*, в которой определяются режимы работы интерфейсных средств микроконтроллера и формируются начальные условия

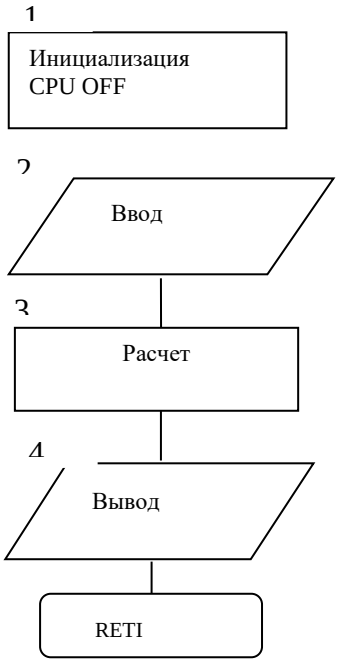


Рисунок 8,1 – Алгоритм с подпрограммой прерывания

Программное обеспечение микроконтроллеров предназначено для режима работы *реального времени*. Режим реального времени предполагает, что алгоритм vдолжен иметь *циклическую часть.*

Формирование интервала времени может выполняться либо программным способом, либо аппаратным, с использованием программируемого таймера. В данной работе изучается *аппаратный* способ формирования интервала времени с использованием программируемого таймера. Поэтому инициализационная часть должна содержать

- инициализацию внешних выводов параллельных портов для ввода и вывода;

- инициализацию таймера для заданного режима работы;

- установку битов регистра состояния, если необходимо управление работой центрального процессорного устройства в режиме прерываний.

В циклической части формируются операции ввода и вывода сигналов, а также необходимые расчеты.

Каждая подпрограмма прерываний формируется на основании алгоритма. Следует учитывать, что подпрограммы прерываний являются независимыми программными единицами, и связаны с главной программой только через данные.

4.3. Таймер TA микроконтроллера MSP430

Программируемый таймер — это устройство, предназначенное для формирования интервалов времени и выполнения функций, связанных с заданием времени. Таймер программируется на выполнение определенных функций путем записи в регистры управления *управляющих слов*. Программируемые таймеры используются как самостоятельные устройства в виде отдельных микросхем, а также входят в состав микроконтроллеров. Обычно в составе микроконтроллера от двух до шести и более программируемых таймеров.

Принцип действия таймера заключается в следующем. Вначале в регистры управления записывают управляющие слова, чтобы задать режим работы таймера. Затем в регистр счета записывают заранее рассчитанное значение, которое при заданной тактовой частоте таймера будет формировать заданный интервал времени. После этого таймер готов к применению. Следовательно, когда появляется сигнал разрешения счета, содержимое регистра счета инкрементируется либо декрементируется при появлении каждого тактового импульса. Счет идет до заданного значения либо до нулевого значения. Обычно в конце счета таймер генерирует запрос на прерывание, что позволяет использовать сформированный интервал времени.

Важнейшими режимами, в которых используются программируемые таймеры, являются режимы ввода и вывода *событий.*

Событием для микроконтроллера может быть:

- появление уровня логической единицы на внешнем выводе;
- появление уровня логического нуля на внешнем выводе;
- появление нарастающего фронта на внешнем выводе;
- появление спадающего фронта на внешнем выводе;
- появление любого фронта на внешнем выводе;

Захватом называют режим *ввода внешнего события* через заданный вход микроконтроллера, сопровождаемый записью в специальный буферный регистр содержимого регистра-счетчика таймера. Таким образом, время появления события фиксируется.

Так, если на вход микроконтроллера поступают импульсы, то режим захвата позволяет измерить их длительность.

Режимом *сравнения* называют режим, при котором организуется появление внешнего события на заданном выходе микроконтроллера в заданный момент времени. Значение регистра-счетчика таймера сравнивается с содержимым регистра модуля захвата-сравнения, и когда их значения совпадают, выводится событие. Режим сравнения является *режимом вывода события.*

В частности, режим сравнения позволяет сформировать импульсы определенной длительности на выходе микроконтроллера.

Таймер TA микроконтроллера MSP430 содержит регистр управления TACTL, 16 разрядный регистр-счетчик TAR и три регистра TACCR0, TACCR1, TACCR2 захвата-сравнения. Для инициализации таймера в регистр управления TACTL таймера и в регистры управления TACCTL0, TACCTL1, TACCTL2 модулей захвата-сравнения следует записать управляющие слова.

Таймер имеет три канала захвата-сравнения, может генерировать широтно-импульсную модуляцию (ШИМ) и формировать интервалы времени.

Таймер имеет 4 режима:

MCx=00 – останов; MCx=10 – непрерывный счет;

MCx=01 – прямой счет; MCx=11 – реверсивный счет.

На рисунке 8.1. показаны диаграммы изменения значения в регистре TAR в различных режимах счета.

Регистр TAR инкрементируется или декрементируется (в зависимости от режима) по нарастающему фронту тактового сигнала. При переполнении регистра TAR может возникать прерывание.

Для тактирования таймера могут использоваться системные тактовые сигналы ACLK (32 кГц) и SMCLK (до 16МГц), а так же внешние сигналы TACLK и INCLK. Выбранный при инициализации тактовый сигнал поступает на таймер через делитель, коэффициент деления которого (1,2,4,8) может быть выбран пользователем при инициализации

Регистры таймера TA, их назначение и адреса представлены в таблице 4.1

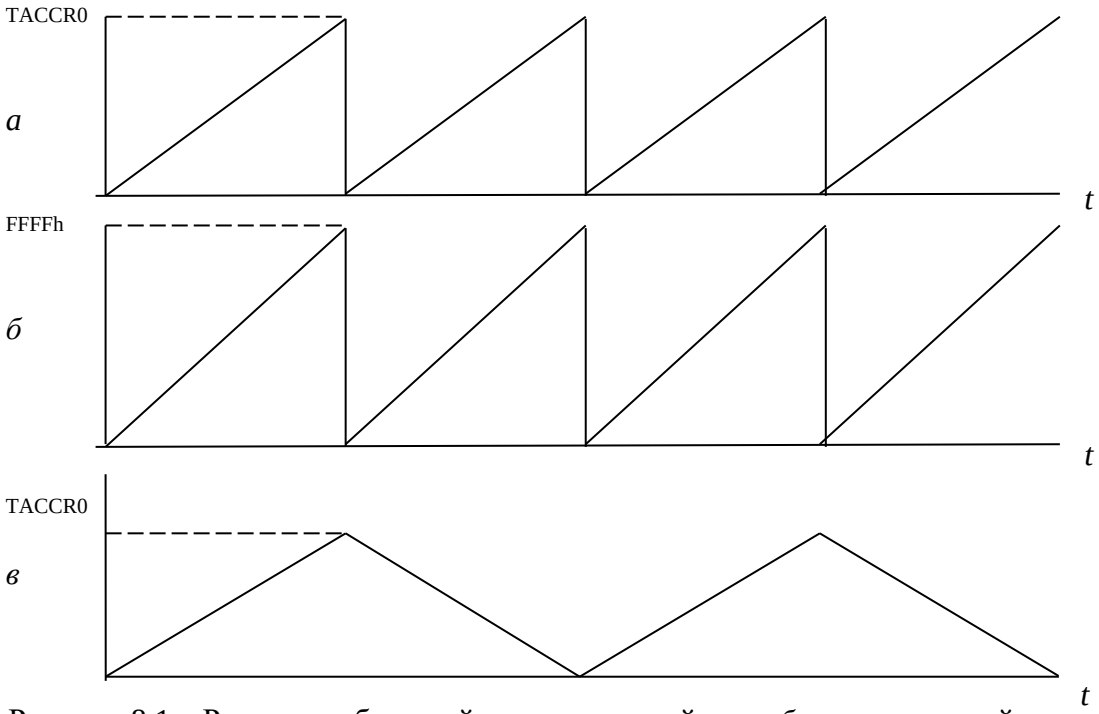


Рисунок 8.1 – Режимы работы таймера, *а* –прямой счет, *б* – непрерывный счет, *в* – реверсивный счет.

Таблица 4.1 - Регистры таймера TA

Регистр	Обозначение	Тип регистра	Адрес
регистр управления	TACTL	Чтение-запись	0160h
регистр-счетчик	TAR	Чтение-запись	0170h
регистры управления захвата-сравнения	TACTLR0, TACTLR1, TACTLR2	Чтение-запись	0162h 0164h 0166h
регистры захвата-сравнения	TACCR0, TACCR1, TACCR2	Чтение-запись	0172h 0174h 0176h
Регистр вектора прерывания	TAIV	Чтение	012Eh

Регистр TACTL управления таймера TA представлен в таблице 4.2.

Таблица 4.2 - Регистр TACTL управления таймера TA.

15-10	9	8	7	6	5	4	3	2	1	0
	TASSELx	IDx	MCx	TMCLR	TAIE	TAIFG				
	выбор источника тактового сигнала 00 TACLK 01 ACLK 10 SMCLK 11 INCLK	коэффициент деления частоты 00 /1 01- /2 10 /4 11 /8	управление режимом таймера 00 –останов 01 –прямой счет 10–непрерывный счет 11 –реверсивный счет	сброс таймера при TMCLR=1,	разрешение прерывания таймера при TAIE=1; 0 – запрет	флаг прерывания таймера.				

Имена и назначение разрядов регистра TACTL:
TASSELx - выбор источника тактового сигнала;
IDx - коэффициент деления частоты;
MCx управление режимом таймера;
TMCLR сброс таймера при TMCLR=1,;
TAIE разрешение прерывания таймера при TAIE=1;
TAIFG флаг прерывания таймера.
Разряды 10-15 и 3 не используются.
Регистр управления захватом-сравнением представлен в таблице 4.3.

Таблица 4.3 - Регистр TACCTLx управления захватом-сравнением таймера TA

15-14	13-12	11	10	8	7 6 5	4	3	2	1	0
CMx	CCIS	SCS	SCCI	CAP	OUTMODx	CCIE	CCI	OUT	COV	CCIFG
режим захвата; 00–нет захвата; 01– захват по нарастающ. фронту; 10- захват по спадающему фронту;	выбор входа захвата-сравн. 00 – CCIxA; 01 – CCIxB; 10 - GND; 11- Vcc;	синхронизация захвата: 0–асинхр. захват; 1 – синхр. захват;	Синхронизирует вход захвата-сравнения. Входной сигнал фиксируется поEQUx и может быть считан;	Режим . захвата-сравнения; 1–захват,	режим работы модуля вывода; 000–сост. OUT; 001–установка; 010–переключение-сброс; 011–установка-сброс; 100–переключение; 101–сброс; 110 – переключение-установка; 111–сброс-установка.	Разреше-ние прерыва-ния захвата-сравн. 0–запрещено, 1 – разрешено;	Вход . захвата-сравнения (значе-ние вх.сигнала);	управление состоянием выхода (0 – низкий уровень, 1 - высокий):	переполнение захвата: 0 – не было переполнения при захвате; 1 - было переполнение. Бит COV должен сбрасываться программно	флаг прерывания захвата сравнения: 0 – не было прерывания, 1 –

Имена и назначение разрядов регистра TACTLRx управления захватом сравнением:

CMx режим захвата;

00 – нет захвата;

01 – захват по нарастающему фронту;

10 - захват по спадающему фронту;

11 - захват по обоим фронтам;

CCISx выбор входа захвата-сравнения;

00 – CCIxA;

01 – CCIxB;

10 - GND;

11 - Vcc;

SCS - синхронизация захвата: 0 – асинхронный захват; 1 - синхронный захват;

SCCI синхронизированный вход захвата-сравнения; входной сигнал фиксируется поEQUx и может быть считан;

CAP режим захвата-сравнения; 1 – захват, 0 – сравнение;

OUTMODx режим работы модуля вывода;

000 – состояние OUT;

001 – установка;

010 – переключение-сброс;

011 – установка-сброс;

100 – переключение;

101 – сброс;
110 – переключение-установка;
111 – сброс-установка.

CCIE разрешение прерывания захвата-сравнения; 0 – запрещено, 1 – разрешено;

CCI вход захвата-сравнения (значение входного сигнала);

OUT управление состоянием выхода (0 – низкий уровень, 1 - высокий):

COV переполнение захвата:0 – не было переполнения при захвате; 1 - было переполнение. Бит COV должен сбрасываться программно.

CCIFG флаг прерывания захвата сравнения: 0 – не было прерывания, 1 – было прерывание.

Внешние выводы, используемые для таймера ТА, выделены на рисунке 4.2.

4.4. Построение программы с использованием программируемого таймера

Программа должна начинаться инициализационной частью, в которой содержится запись управляющих слов в регистры управления интерфейсных устройств.

Для записи управляющих слов можно применять различные способы. Так, в соответствии с данными таблиц 8.2 и 8.3 составляются в двоичном коде слова, определяющие режимы таймера. Эти слова представляются в 16-й системе счисления и записываются в регистры управления с использованием команды MOV. Например, для разрешения прерываний можно использовать управляющее слово 0000 0000 0001 0000_2 = 0010h

```
SetupC0   mov.w   #0010h, &TACCTL0        ; в TACCR0 разрешено прерывание
```

Каждый разряд (бит, группа разрядов, битов) регистра управления имеет свое имя, которому соответствует 16 разрядное двоичное слово, со значениями данном разряде (бите, группе битов). Остальные разряды этого двоичного слова равны нулю, например

#CCIE=0000 0000 0001 0000_2 = 0010h.

Поэтому в команде можно использовать имя бита, например

```
SetupC0 mov.w   #CCIE, &TACCTL0 ; (в TACCR0 разрешено прерывание).
```

Используя имена битов управления, можно формировать управляющее слово с использованием символа суммирования, например, для инициализации таймера применима команда

```
SetupTA mov.w   #TASSEL_2+MC_2,&TACTL ; SMCLK
```

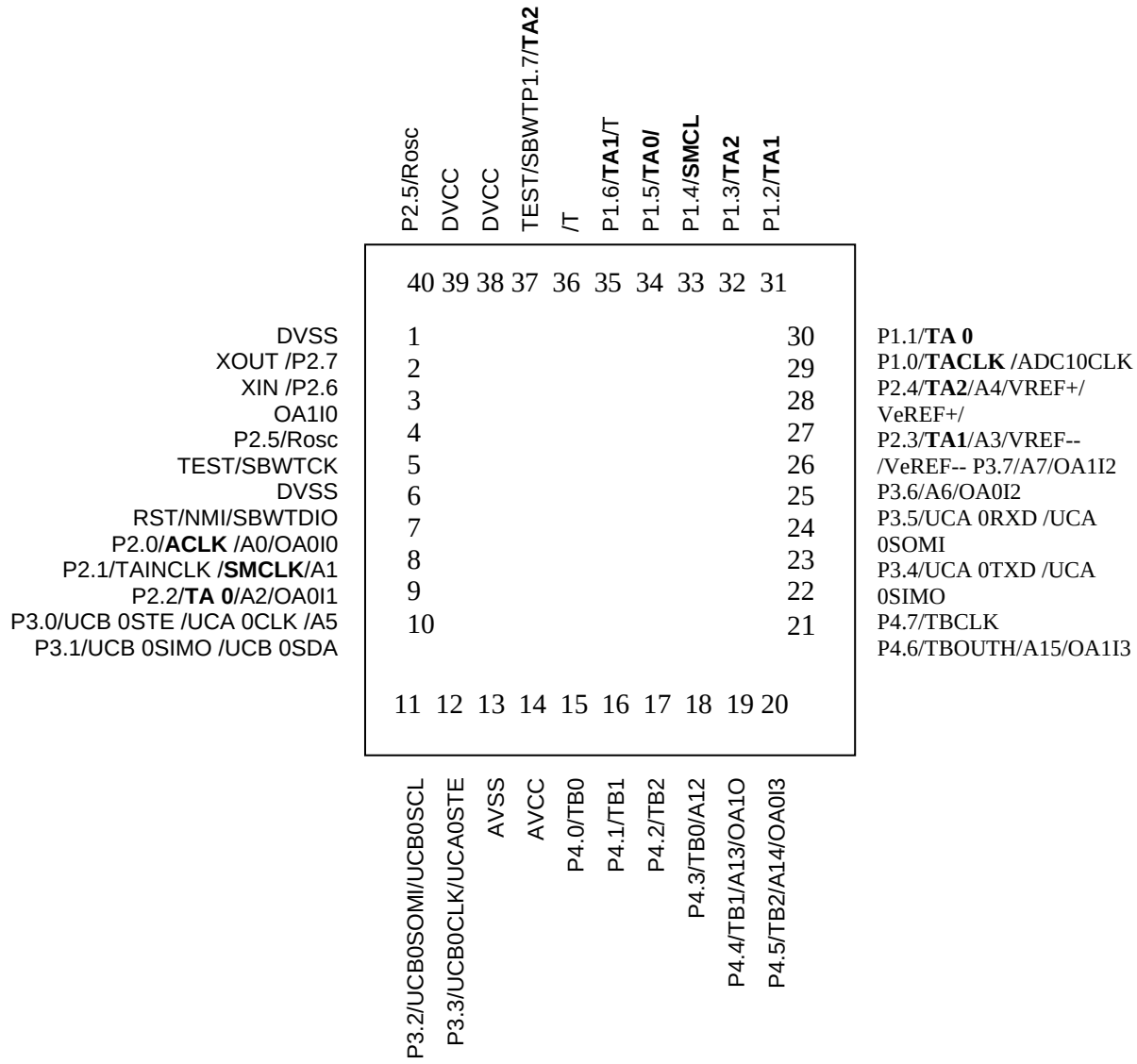


Рисунок 4.2 - Внешние выводы, используемые таймером TA

Здесь #TASSEL_2 означает, что для выбора тактового сигнала использовано значение 10_2 в двоичной системе, равное 2 десятичной системы (основной тактовый сигнал SMCLK), а MC_2 означает, что для выбора режима таймера использовано значение $10_2=2_{10}$, что соответствует режиму непрерывного счета.

Пример исходного текста программы **msp430x22x4_ta_01.asm** на языке ассемблера показан далее. Программа предназначена для включения и отключения светодиода с заданным периодом по сигналу прерывания, который генерирует таймер TA, что указано в комментарии перед текстом программы.

Программа содержит инициализационную часть, где задается адрес вершины стека, остановлен сторожевой таймер и настроен на вывод один бит порта P1. Далее разрешено прерывание модуля захвата-сравнения TACCTL0.

В регистр TACCR0 записывается значение, пропорциональное интервалу времени, который необходимо сформировать.

Выполняется инициализация таймера TA. Выполняется отключение CPU и разрешение прерывания. В результате происходит прерывание при каждом переполнении таймера. Поскольку прерывание выводит процессор из состояния останова, периодически выполняется подпрограмма прерывания. В подпрограмме прерывания организовано переключение младшего бита порта P1. Этим формируется попеременное включение и отключение светодиода операцией **(^)** (исключающее ИЛИ).

Прямой счет (*MC_1)* использован в примере программы **msp430x22x4_ ta_02,** которая обеспечивает переключение бита *P1*.0 параллельного порта *P*1 при выполнении подпрограммы прерывания по вектору *TIMERA0_VECTOR* нулевого канала захвата-сравнения таймера *TA* с частотой, обратно пропорциональной значению 20000, записываемому в регистр сравнения *TACCR0.* Далее представлен текст программы.

```
#include <msp430x22x4.h>
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;              // Stop WDT
  P1DIR |= 0x01;                       // P1.0 output
  TACCTL0 = CCIE;                       // TACCR0 interrupt enabled
  TACCR0 = 20000;
  TACTL = TASSEL_2 + MC_1;              // SMCLK, up mode
  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0 w/ interrupt
}

// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  P1OUT ^= 0x01;                       // Toggle P1.0
}
```

Непрерывный счет (*MC_2)* использован в примере программы **msp430x22x4_ ta_,** которая обеспечивает переключение бита *P1*.0 параллельного порта *P*1 при выполнении подпрограммы прерывания по вектору *TIMERA0_VECTOR* нулевого канала захвата-сравнения таймера *TA* с частотой, обратно пропорциональной значению 50000, записываемому в регистр сравнения *TACCR0.* Далее представлен текст программы.

```
#include <msp430x22x4.h>
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;              // Stop WDT
  P1DIR = 0x01;                       // P1.0 output
  TACCTL0 = CCIE;                       // TACCR0 interrupt enabled
  TACCR0 = 50000;
  TACTL = TASSEL_2 + MC_2;              // SMCLK, contmode

  __bis_SR_register(LPM0_bits + GIE);       // Enter LPM0 w/ interrupt
}
// Timer A0 interrupt service routine
#pragma vector=TIMERA0_VECTOR
__interrupt void Timer_A (void)
{
  P1OUT ^= 0x01;                       // Toggle P1.0
  TACCR0 += 50000;}                  // Add Offset to TACCR0
```

Реверсивный счет (*MC_3*) использован в примере программы **msp430x22x4_ ta_13,** которая обеспечивает переключение бита *P1*.1 параллельного порта *P*1 в режиме OUTMOD_4 вывода событий (переключение) нулевого канала *TA*0 захвата-сравнения таймера *TA* с частотой, обратно пропорциональной значению 2 * 250, если 250 записано в регистр сравнения *TACCR0.* Бит *P1*.1 параллельного порта *P*1 переключается по прерываниям аппаратным способом, так, что подпрограмма прерывания не требуется. Далее представлен текст программы.

```
//*************************************************************
#include <msp430x22x4.h>
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;              // Stop WDT
  P1DIR |= 0x02;                          // P1.1 output
  P1SEL |= 0x02;                          // P1.1 option select
  TACCTL0 = OUTMOD_4;                    // TACCR0 toggle mode
  TACCR0 = 250;                           //
  TACTL = TASSEL_2 + MC_3;               // SMCLK, up-downmode
  __bis_SR_register(CPUOFF);              // CPU off
}
// Toggle P1.1;}
```

4.5. Порядок выполнения работы

1. Составить алгоритм и программу в соответствии с заданным вариантом.
2. Открыть проект. В проект добавить файл исходного текста программы.
3. Откомпилировать проект **Project → Build Active Project**. При необходимости устранить ошибки. Для взаимодействия прикладной программы с устройством выполнить **Target → Debug Active Project**.
4. Настроить окно проекта. (**View→Debug** , **View→Registers).**
5. Запустить программу на выполнение в пошаговом режиме**.**
6. Внести изменения в исходный текст**,** чтобы программа соответствовала заданному варианту.
7. Выполнить п.3-6 для измененной программы и записать содержимое регистров.
8. Остановить выполнение программы и записать содержимое регистров**.**
9. Изменить выдержку времени, запустить программу и посмотреть, как это повлияет на работу устройства.

4.6. Варианты заданий

1. Составить алгоритм и программу для переключений вывода P1.0 порта P1 с частотой 16 Гц в режиме непрерывного, прямого и реверсивного счета.
2. Составить алгоритм и программу для переключений 2-х выводов порта P1 с частотой 32 Гц в режиме прямого счета, используя настройки OUTMODE.

3. Составить алгоритм и программу формирования последовательности импульсов частотой 200 Гц в режиме реверсивного счета, используя настройку OUTMODE.

4. Составить алгоритм и программу формирования 2-х последовательностей импульсов частотой 32 КГц в режиме реверсивного счета, используя настройки OUTMODE.

5. Составить алгоритм и программу формирования последовательности импульсов частотой 160 Гц в режиме непрерывного счета.

6. Составить алгоритм и программу формирования последовательности импульсов с заданным периодом и со скважностью 0,5 в режиме прямого счета, используя OUTMODE.

7. Составить алгоритм и программу последовательного включения таймером 2х силовых ключей и их отключения в обратной последовательности в режиме реверсивного счета.

8. Составить алгоритм и программу вывода двух сигналов заданной частоты через P1.0 и P1.1 в противофазе в режиме реверсивного счета.

9. Составить алгоритм и программу вывода двух сигналов заданной частоты через P1.0 и P1.1 со сдвигом на четверть периода.

10. Составить алгоритм и программу формирования последовательности импульсов частотой 40 Гц.

11. Сформировать режим для реверсивного счета и и захвата событий.

12. Составить алгоритм и программу формирования импульсов на выходе P1.0 в режиме непрерывного счета.

4.7. Содержание отчета

1. Цель работы.
2. Условие задачи.
3. Алгоритм и программа с комментариями.
4. Результат выполнения программы в виде таблицы со значениями, записанными в регистрах.

4.8. Контрольные вопросы

1. Как инициализировать таймер?
2. Как использовать режим захвата?
3. Как использовать режим сравнения?
4. Как изменить режим работы таймера?
5. Как формируют интервал времени?
6. Каково назначение подпрограммы прерывания?
7. Когда появляется запрос прерывания от таймера?

5.ЛАБОРАТОРНАЯ РАБОТА №5. РАЗРАБОТКА АЛГОРИТМОВ И ПРОГРАММ МИКРОКОНТРОЛЛЕРА MSP430 С ИСПОЛЬЗОВАНИЕМ АНАЛОГО-ЦИФРОВОГО ПРЕОБРАЗОВАТЕЛЯ ADC10.

5.1. Цель работы

Целью работы является разработка алгоритмов и программ для микроконтроллера MSP430 с использованием для ввода сигналов аналого-цифрового преобразователя **ADC10,** а также анализ режимов функционирования **ADC10.**

5.2. Аналого-цифровой преобразователь ADC10 микроконтроллера MSP430

Аналого-цифровой преобразователь **ADC10** предназначен для преобразования изменяющейся во времени аналоговой величины на входе АЦП в двоичный код. В результате преобразования **ADC10** формирует 10 – разрядные двоичные значения, пропорциональные входной величине, которые сохраняются в специальном регистре **ADC10MEM** для дальнейшего использования. Запуск аналого-цифрового преобразования может быть программным способом, либо от таймера TA.

Аналого-цифровой преобразователь **ADC10** микроконтроллера **MSP430** имеет внешние выводы для ввода сигналов, которые показаны на рисунке 5.1 и выделены.

Функциональная схема **ADC10** представлена на рисунке 5.2. Модуль **ADC10** содержит 10- разрядное ядро с регистром последовательного приближения, блок управления выборкой (БУВ), источник опорного напряжения (ИОН) и контроллер пересылки данных DTC, который позволяет сохранять результаты преобразования в пределах адресного пространства, минуя центральное процессорное устройство (CPU).

Модуль **ADC10** допускает скорость преобразования $2*10^5$ выборок в секунду. Время выборки можно задать программно. Запуск преобразования выполняется программно или от таймера. Внутренний генератор опорного напряжения может быть настроен на 1.5 *V* или 2.5 *V*. Следовательно, входной аналоговый сигнал не должен по модулю превосходить соответственно 1.5 *V* или 2.5 *V*. Возможен внешний источник опорного сигнала. На модуль **ADC10** могут быть поданы до 12 входных аналоговых сигналов. Тем не менее, в каждый момент времени идет преобразование лишь одного входного сигнала. Если необходимо выполнить преобразование нескольких величин, поступающих по различным каналам ввода, **ADC10** выполняет это последовательно, подключаясь поочередно к различным входам. Модуль допускает 4 режима преобразования, и каждому режиму соответствует двоичный код:

Однократный одноканальный – 00;

Однократный последовательный – 01;

Циклический одноканальный – 10;

Циклический последовательный - 11.

Для настройки модуля **ADC10** применяются регистры управления **ADC10CTL0 ADC10CTL1,** структура которых представлена в таблицах 5.1, 5.2. Для инициализации аналого-цифрового преобразователя **ADC10** необходимо в

управляющие регистры записать информацию о выбираемых режимах работы. Если выборка и преобразование запускаются от таймера, необходимо так же инициализировать таймер, записывая в его управляющие регистры информацию о режиме работы.

Таблица 5.1. регистр управления ADC10CTL0

15 14 13	12 11	10	9	8
SREFx	ADC10SHTx	ADC10SR	REFOUT	REFBURST
Выбор источника опорного напряжения 000 – V_{R+}=Vcc; V_R-=Vss 001 - V_{R+}=V_{REF+}; V_R-=Vss 010 V_{R+}=Ve_{REF+}; V_R-=Vss 011 V_{R+}=буферизованное Ve_{REF+}; V_R-=Vss 100 V_{R+}=Vcc; V_R-=V_{REF-}|Ve_{REF} 101 V_{R+}= V_{REF+}; V_R-=V_{REF-}|Ve_{REF} 110 V_{R+}=Ve_{REF+}; V_R-=V_{REF-}|Ve_{REF} 111 V_{R+}=буферизованное Ve_{REF+}; V_R-=V_{REF-}|Ve_{REF}	Время выборки, тактов 00 4 01 8 10 16 11 64	Скорость преобразования, c^{-1} 0 до $200*10^3$ 1 до $50*10^3$	Выход опорного напряжения 1- вкл 0 откл	Управление буфером встроенного источника опорного напряжения: 0 вкл. 1 вкл. на время преобр.

7	6	5	4	3	2	1	0
MSC	REF2_5v	REFON	ADC10ON	ADC10IE	ADC10IFG	ENC	ADC10SC
Многокр. преобр., 0 запускается по нарастающ. фронту SHI; 1 таймер выборки запускается по первому нарастающее. фронту SHI, остальные выборки – по окончании предыдущего преобр.;	0 1,5V 1 2,5 V	Генератор опорного напряжения 0 выкл. 1 включен	Вкл. модуля ADC10 0 выкл. 1 включен	Разрешение прерывания модуля ADC10 0 запрещено 1 разрешено.	Флаг прерывания. Устанавливается при записи результата преобразования в ADC10MEM. 0 не было запроса прерывания 1 есть запрос.	Разрешение преобразования 0 запрещено;1 разрешено	Запуск выборки преобр. 0 не начинать 1 начать.

Таблица 5.2. регистр управления ADC10CTL1

15 14 13 12	11 10	9	8	7 6 5	4 3	2 1	0
INCHx	SHSx	ADC10DF	ISSN	ADC10DIVx	ADC10SSELx	CONSEQx	ADC10BUSY
Выбор входного канала 0000 A0 0001 A1 … 0111 A7 1000 Ve_{REF+} 1001V_{REF-}/ Ve_{REF-} 1010 датчик температуры 1011 AVcc	Источник сигнала запуска 00 ADC10SC 01 выв.1 ТА 10 выв.2 ТА	Формат результата 0 двоичный 1-дополнит. код	Инвертирование сигнала запуска 0 не инвертировать 1 инвертировать	Коэффициент деления тактового сигнала 000 1 001 2 010 3 011 4 … 111 8	Выбор источника тактового сигнала 00 ADC10SC 01 ACLK 10 MCLK 11 SMCLK	Выбор режима преобразования 00 однокр. одноканальн.01 однокр. последоват. 10 циклич. одноканальн. 11 циклич. последоват.	занятость 0 нет активности 1 занят

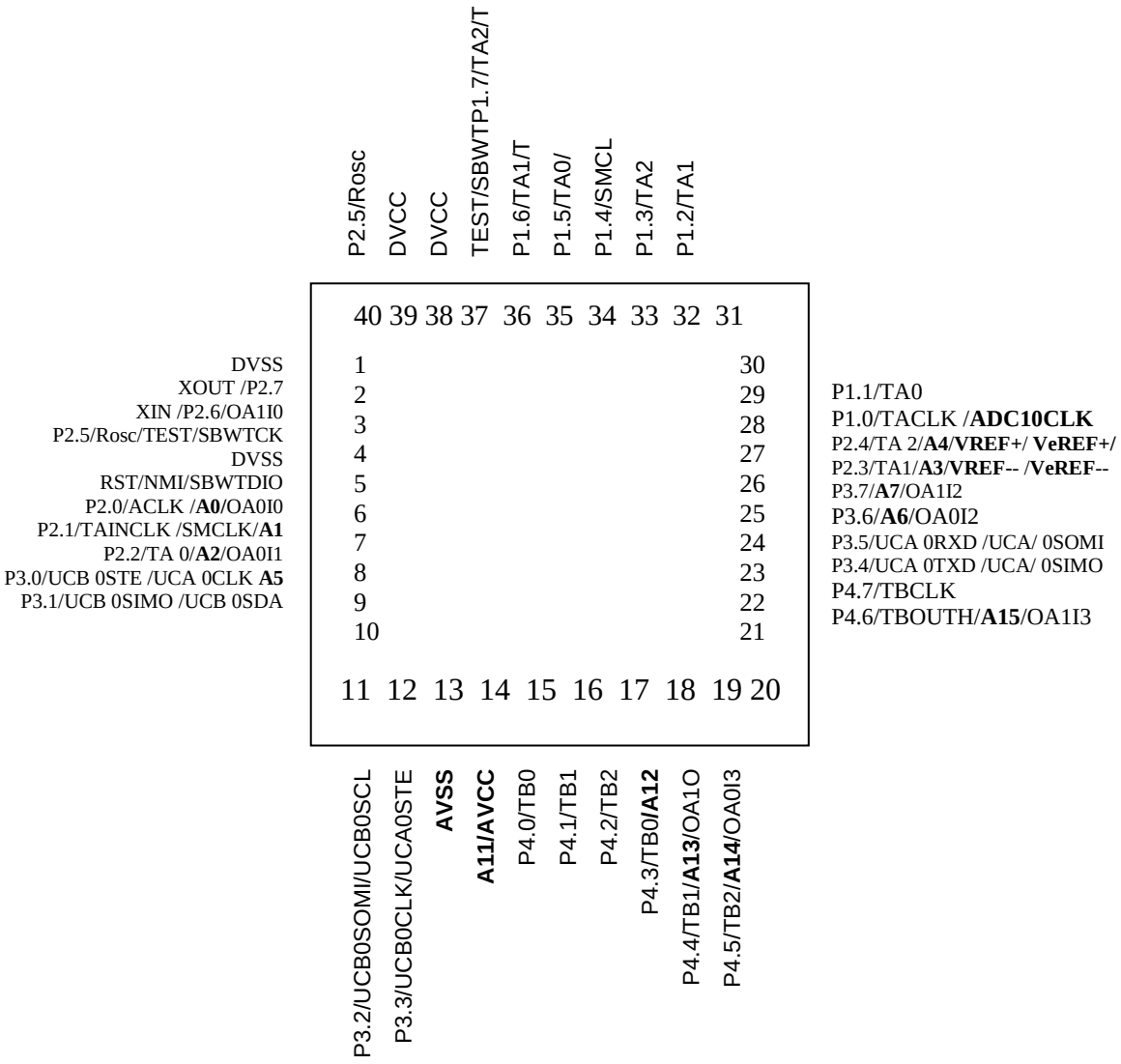


Рисунок 5.1 – Внешние выводы, используемые **ADC10.**

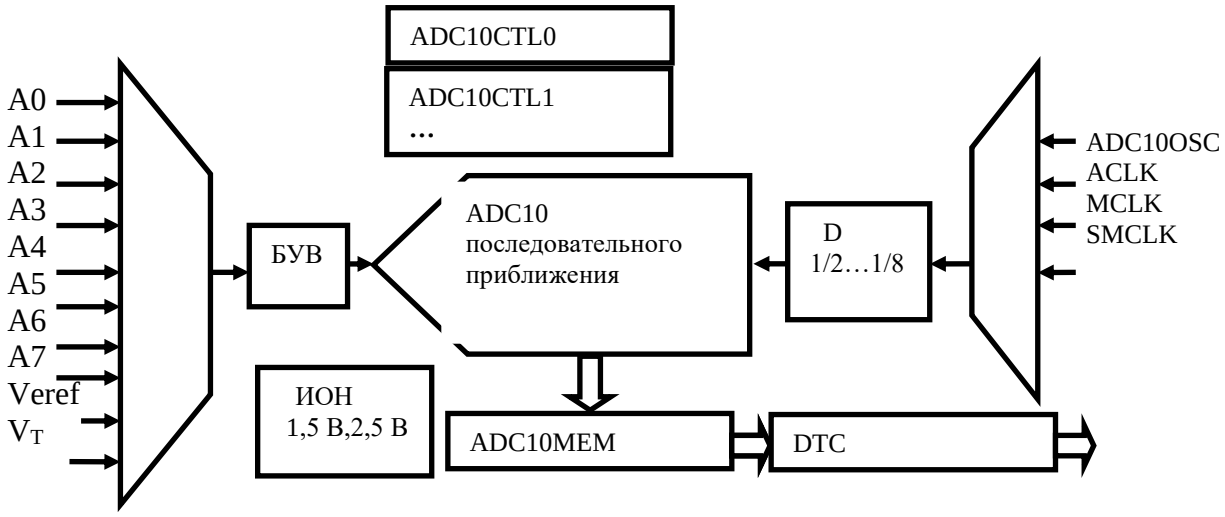


Рисунок 5.2 – Функциональная схема **ADC10.**

Аналого-цифровое преобразование складывается из двух этапов: выборка и собственно преобразование. На этапе выборки аналоговый сигнал сравнивается с опорным напряжением от ИОН (рисунок 5.2). Переходный процесс, продолжается в течение времени

$$t_s = T\ ln(2^{m+1}), \qquad (5.1)$$

где $T = (Rs + R_1)\ C_1$ - постоянная времени входной RC - цепи **ADC10,** m =10 – количество разрядов преобразователя, Rs – сопротивление внешнего источника, R_1, C_1 – сопротивление и емкость входной цепи. За это время формируется установившееся значение с точностью, соответствующей m =10 двоичным разрядам. Двоичное значение получается на основании выражения

$$x_u = (2^m - 1)\ (u - U_{REF-})/(U_{REF+} - U_{REF-}). \qquad (5.2)$$

Здесь x_u - двоичное значение входного аналогового сигнала, u – входной аналоговый сигнал, U_{REF+} = **V_{ref}** - значение опорного сигнала, $U_{REF+} - U_{REF-}$ - разность потенциалов на выходе источника опорного сигнала. Учитывая, что значение **V_{ref}** может быть при инициализации (Таблица 5.1. Регистр управления ADC10CTL0) настроено на 1,5V либо на 2,5 V, входной сигнал ограничен одним из этих значений. Из (5.2) видно, что наибольшее значение x_u = 1023_{10} = 0x03FF достигается для входного сигнала, равного опорному.

5.3. Программа с использованием аналого-цифрового преобразователя ADC10

Рассматривается пример программы **msp430x22x4_adc10_01.c,** исходный текст которой написан на языке С. Программа предназначена для выполнения аналого-цифрового преобразования сигнала **AVcc/2** от источника напряжения для аналоговых цепей на входе A11 ADC10 (рисунок 5.1). На вывод P1.0 выводится 1, если достигается значение A0 > 0.5*AVcc.

```
#include <msp430x22x4.h>
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ON, interrupt enabled
  ADC10AE0 |= 0x01;                         // P2.0 ADC option select, разрешение входа A0
  P1DIR |= 0x01;                            // Set P1.0 to output direction

  for (;;)
  {
    ADC10CTL0 |= ENC + ADC10SC;             // Sampling and conversion start
    __bis_SR_register (CPUOFF + GIE);       // LPM0, ADC10_ISR will force exit
    if (ADC10MEM < 0x1FF)
      P1OUT &= ~0x01;                       // Clear P1.0 LED off
    else
      P1OUT |= 0x01;                        // Set P1.0 LED on
  }
}
// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
  __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

Директива **#include <msp430x22x4.h>** препроцессора подсоединяет к программе головные файлы, где содержатся адреса периферийных устройств микроконтроллера указанного типа. Программа содержит инициализационную часть, где задается адрес вершины стека, остановлен сторожевой таймер, выполнена инициализация **ADC10,** настроен на вывод бит порта P1.0. Чтобы **ADC10** работал в однократном одноканальном режиме, принимается **CONSEQx** = 00 по умолчанию.

В программе предусматривается запись в регистр управления **ADC10CTL1** значения **ADC10SHT_2** что, в соответствии с таблицей 5.2, означает время выборки 16 тактов.

Циклическая часть представлена циклом for (; ;). Это цикл, повторяющийся бесконечное число раз. Повторяющиеся операторы заключены в фигурные скобки. Вначале оператором ADC10CTL0 |= ENC + ADC10SC; разрешается и запускается преобразование. Далее следует операция

___bis_SR_register (CPUOFF + GIE);

отключения CPU и общее разрешение прерываний. Процессор пребывает в режиме LPM0 пониженного энергопотребления, а периферийные устройства продолжают функционировать. По окончании очередного преобразования ADC10 запрашивает прерывание. По запросу прерывания CPU выходит из режима LPM0 и выполняет подпрограмму прерывания, в которой выполняется команда **__bic_SR_register_on_exit (CPUOFF**); ассемблера, по которой процессор включается. Этим создается возможность выполнения команд основного цикла, следующих после **__bis_SR_register** (CPUOFF + GIE). Проверяется условие *A*0 > 0.5*AVcc, и в случае его выполнения в порт выводится 1. Директива **#pragma vector=ADC10_VECTOR** препроцессора описывает прерывание от **ADC10.**

5.4. Порядок выполнения работы

1. Составить алгоритм и программу в соответствии с заданным вариантом. Использовать приеденный пример программы, как прототип.
2. Открыть проект. В проект добавить файл **msp430x22x4_adc10_01.c** текста программы, скопировав его из папки **slac123d.**
3. Откомпилировать проект **Project → Build Active Project**. При необходимости устранить ошибки. Для взаимодействия прикладной программы с устройством выполнить **Target → Debug Active Project**.
4. Настроить окно проекта так, чтобы были видны внутренние регистры микроконтроллера, используемые в программе. (**View → Debug**, **View→Registers).**
5. Запустить программу на выполнение в пошаговом режиме, затем в непрерывном режиме. Записать содержимое регистра SR и регистров управления **ADC10** до и после инициализации.
6. Убедиться, что при изменении опорного сигнала содержимое **ADC10MEM** изменяется.
7. Записать содержимое регистров **ADC10** до и после возникновения прерывания.

8. Внести изменения в исходный текст **msp430x22x4_adc10_01.c,** чтобы программа соответствовала заданному варианту.

9. Выполнить п.3-7 для измененной программы.

10. Изменить время выборки, и посмотреть, как это повлияет на результат.

5.5. Варианты заданий

1. Составить алгоритм и программу переключений выводов P1.0, P1.1 порта P1 при достижении заданного значения сигнала на входе.
2. Составить алгоритм и программу для вывода результата преобразования в P1.
3. Составить алгоритм и программу для вывода преобразованного сигнала в дополнительном коде в два параллельные порта.
4. Составить алгоритм и программу для ввода сигнала в АЦП и вывода в порт P1 округленного до 1 байта значения в двоичном коде.
5. Составить алгоритм и программу для ввода сигнала в АЦП и вывода в порт P1 младшего байта з в двоичном коде.
6. Организовать циклическое одноканальное аналого-цифровое преобразование с различными длительностями выборки.
7. Составить алгоритм и программу формирования последовательного циклического преобразования опорного сигнала 1,5 *V*.
8. Составить алгоритм и программу последовательного однократного преобразования
9. Составить алгоритм и программу вывода сигнала температуры через P1.
10. Сформировать управляющее слово для циклического последовательного режима преобразования опорного сигнала 2,5 *V* и использовать в программе.

5.6 Содержание отчета

1. Цель работы.
2. Условие варианта задания.
3. Алгоритм для заданного варианта и программа с комментариями.
4. Содержимое регистров микроконтроллера до, во время и после выполнения программы, до и после прерывания с пояснениями.

5.7. Контрольные вопросы

1. Какие возможны режимы преобразования?
2. Как инициализировать АЦП для каждого из режимов преобразования?
3. Как взаимодействуют АЦП и центральный процессор?
4. Как сохраняется результат преобразования?
5. Как изменить режим работы АЦП?
6. Как формируют интервал времени преобразования?
7. Каково назначение подпрограмм прерывания?

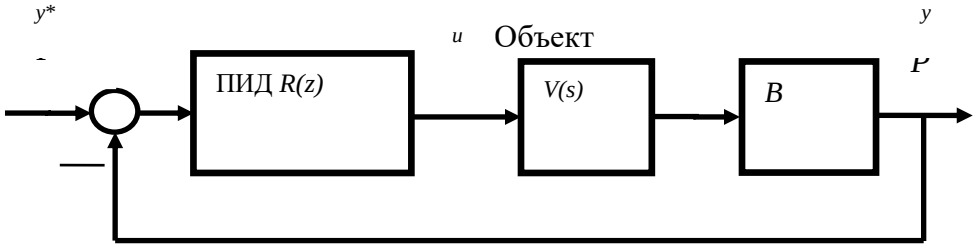
6. ЛАБОРАТОРНАЯ РАБОТА №6. АЛГОРИТМЫ И ПРОГРАММЫ РАСЧЕТА ВЫХОДНЫХ ВЕЛИЧИН РЕГУЛЯТОРОВ, ИНТЕГРИРУЮЩИХ И ДИФФЕРЕНЦИРУЮЩИХ ЗВЕНЬЕВ

6.1. Цель работы

Целью работы является анализ работы микроконтроллера MSP430 при выполнении программы расчета выходных величин регуляторов, когда входные сигналы поступают на аналоговые входы.

6.2 Расчётные выражения для сигналов на выходе динамических звеньев

Обычно ПИД регуляторы применяются в одноконтурных системах с управлением по выходу, то есть с обратной связью по выходной регулируемой величине (рисунок 6.1). Примером являются электроприводы турбомеханизмов с обратной связью по напору.



6.1 – Структура управления с ПИД регулятором

Структура на рисунке 6. 1 содержит ПИД регулятор с ПФ $K(z)$, объект управления с ПФ $V(s)$ и датчик температуры B безынерционный, причем $K(z)$ имеет вид

$$R(z) = k_I T_C z^{-1}/(1 - z^{-1}) + k_P + k_D(1 - z^{-1})/T_C. \qquad (6.1)$$

Здесь k_I, k_P, и k_D являются коэффициентами усиления интеграла ошибки, ошибки и ее производной соответственно. В то же время объект описывается ПФ $V(s) = M_P(s)/N_P(s)$ как непрерывное звено.

Частными случаями ПИД- регулятора являются регуляторы пропорционально- интегрирующий (ПИ), пропорционально- интегрирующий (ПИ), пропорционально- дифференцирующий (ПД) и П-регулятор.

На вход ПИ-регулятора поступает сигнал ошибки $e_k = (y^* - y_k)$. Для выходной величины ПИ- регулятора справедливо выражение

$$u_k = u_{k-1} + b_1 e_k + (T_C b_0 - b_1) e_{k-1}. \qquad (6.2)$$

В простейшем случае объект управления имеет передаточную функцию интегрирующего звена, и поэтому описывается разностным уравнением

$$y_{k+1} = y_k + T_C u_k.$$

Далее приводится пример программы с использованием аналого-цифрового преобразования для ввода сигнала задания y^* = ADC10MEM. Сигнал управления рассчитывается по выражению (6.2)

```
#include "msp430x22x4.h"
  unsigned int i,k;
    int y[150], ee, e1 = 0;                    // ee = y* – y;  y* =ADC10MEM
    int u = 0, um = 0x01FF;
    int bb1 = 20, b1 = 1, R = 20;     // REG,   b0/b1 = R; (R*TS-1)b1 = bb1
void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop WDT
  ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, interrupt enabled
  ADC10AE0 |= 0x01;           // P2.0 ADC option select
  P1DIR |= 0x01;                  // Set P1.0 to output direction

  for (;;)
  {y[0] = 0x64;
    ADC10CTL0 |= ENC + ADC10SC;         // Sampling and conversion start
    __bis_SR_register(CPUOFF + GIE);        // LPM0, ADC10_ISR will force exit
    for (k = 0;k < 200;k++)            //    infinite cycle
      {        P1OUT ^= 0x03;
        ee = ADC10MEM  - y[k];
     u += b1 * ee + bb1 * e1 ;
        if(u < -um) u = -um;
        if(u > um) u = um;
        y[k+1] = y[k] + u/100;    //
       e1 = ee;
   for (i = 1250;i > 0 ;i--);      //     delay value                 // Clear P1.0 LED off
    }}                    // Set P1.0 LED on
  }
// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{   __bic_SR_register_on_exit(CPUOFF);        // Clear CPUOFF bit from 0(SR)
}
```

Для понимания настроек **ADC10** следует пользоваться справочными таблицами 5.1 и 5.2 лабораторной работы №5 для регистров управления ADC10CTL0 и ADC10CTL1. Для управления выборкой используется таймер TA в режиме прямого счета с тактовой частотой SM CLK 1 MHz.

Программное обеспечение состоит из двух программных единиц: главной программы с инициализационной и циклической частями и подпрограммы прерываний от **ADC10.** Оператор __bis_SR_register (CPUOFF + GIE); обеспечивает выполнение команды ассемблера bis установки битов CPUOFF, GIE в регистре SR.

6.3. Порядок выполнения работы

2 Вывести расчетные выражения для расчета выходной величины регулятора в соответствии с заданным вариантом.

3 Составить алгоритм и программу в соответствии с заданным вариантом.

4 Проанализировать работу представленного примера программы с использованием аналого-цифрового преобразователя и ПИ- регулятора.

5 Создать проект, добавить файл текста **msp430x22x4_adc10_01.c** программы и доработать его в соответствии с заданным вариантом.

6 Откомпилировать проект **Project → Build Active Project**. При необходимости устранить ошибки. Для выполнения прикладной программы выполнить **Target → Debug Active Project**.

7 Настроить окно проекта так, чтобы были видны внутренние регистры микроконтроллера, используемые в программе. (**View → Debug**, **View→Registers).**

8 Запустить программу на выполнение в пошаговом режиме, затем в непрерывном режиме. Записать содержимое регистров **ADC10**, а также значение рассчитанного сигнала на выходе регулятора и на выходе объекта.

9 Вывести график выходной величины. Tool→Graf→Signl Time. В открывшемся окне задать размер массива (buffer size, display data size), его начальный адрес и количество бит элемента массива.

10 Записать содержимое регистров **ADC10** до и после возникновения прерывания.

11 Изменить время выборки, и посмотреть, как это повлияет на результат.

6.4. Варианты заданий

1. Дана программа расчета выходной величины контура регулирования с ПИ- регулятором. Дать подробное описание функционирования микроконтроллера при выполнении программы.
2. Внести изменения в данную программу, чтобы получить ПИД-регулятор. Подробно описать функционирование программы.
3. Для данной программы с ПИ- регулятором описать функционирование микроконтроллера при выполнении программы. Организовать вывод графика сигнала управления.
4. Составить подпрограмму прерывания для вывода **ADC10MEM** в параллельный порт.
5. Составить полное описание режима работы **ADC10** и указать значения всех битов регистров управления.
6. Составить алгоритм программы **и у**величить период преобразований в два раза.

7 Составить алгоритм программы и **у**меньшить период преобразований в два раза.

6.5 Содержание отчета

1. Цель работы.
2. Выражения для расчета выходной величины регулятора.
3. Алгоритм и программа с комментариями.
4. Результат выполнения программы в виде графика выходной величины с пояснениями.

6.6. Контрольные вопросы

1. Как выполнить запуск преобразования от ТА?
2. Как выполняются выборка и преобразование?
3. Когда АЦП запрашивает прерывание?
4. Как происходит возврат из прерывания?
5. Как АЦП взаимодействует с памятью данных?

6. Как формируют интервал времени выборки?
7. Как формируется интервал времени преобразования?
8. Как организовать вывод графика?

7. ЛАБОРАТОРНАЯ РАБОТА №7. ИСПОЛЬЗОВАНИЕ ПОСЛЕДОВАТЕЛЬНОГО ИНТЕРФЕЙСА ДЛЯ ПРИЕМА И ПЕРЕДАЧИ ДАННЫХ

7.1. Цель работы

Целью работы является разработка программы на языке C/C++ для микроконтроллера **MSP430F2274,** используя для обмена данными с внешними устройствами последовательный интерфейс**.**

7.2. Последовательный интерфейс микроконтроллера MSP430F2274

Модуль универсального последовательного интерфейса **USCI** (Universal Serial Communicative Interface, универсальный последовательный коммуникационный интерфейс) позволяет организовать обмен информацией с внешними устройствами по интерфейсу **UART** (Universal Asynchronous Receiver Translator, универсальный асинхронный приемопередатчик). Возможен синхронный обмен по интерфейсам **SPI** (Serial Programmable Interface, последовательный программируемый интерфейс) и **I2C** (Interface 2-Conductors', двухпроводной интерфейс). Функциональная схема модуля последовательного интерфейса показана на рисунке 7.1.

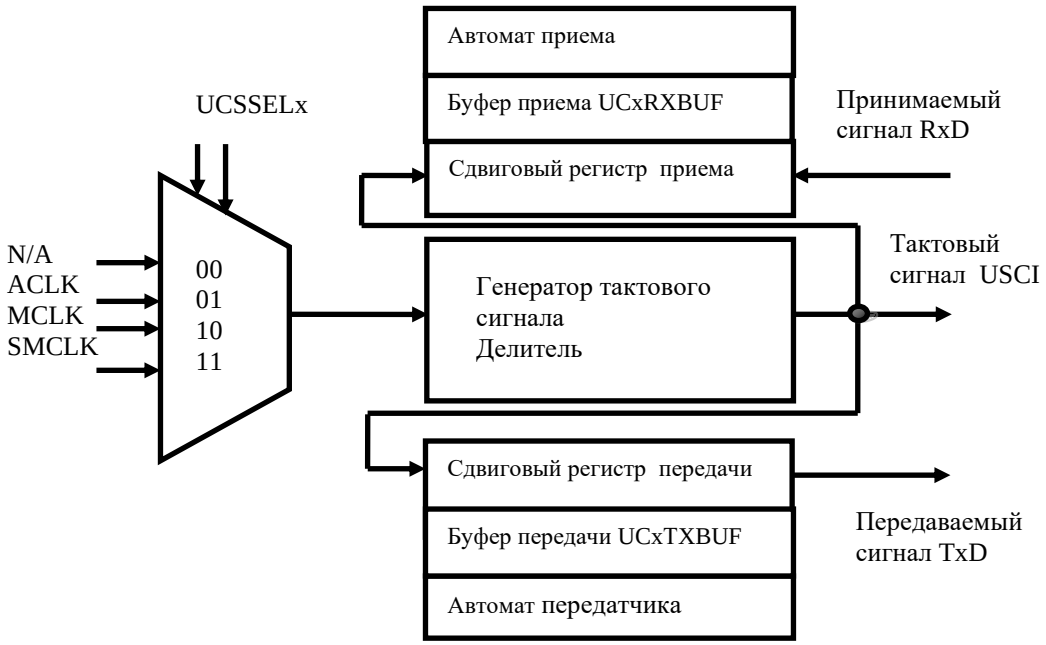


Рисунок 7.1 – Функциональная схема **USCI.**

Основным устройством модуля последовательного интерфейса является сдвиговый регистр, который формирует последовательность 7 или 8 передаваемых битов в режиме передачи информации.

В режиме приема информации сдвиг выполняется таким образом, что последовательность принимаемых битов записывается в сдвиговый регистр для дальнейшей записи в память и использования. В составе модуля имеются функциональные узлы для организации трехпроводного интерфейса **SPI** и двухпроводного интерфейса **I2C,** а так же для использования прерываний.

В режиме **SPI** используются регистры управления и состояния модуля **USCI_A0**, приведенные в таблице 7.1.

Таблица 7.1. Регистры управления и состояния модуля USCI_A0

Регистры	Назначение	Тип	Адрес
UCA0CTL0 **UCA0CTL1**	Регистры управления	Чтение, запись	060h 061h
UCA0BR0 **UCA0BR1**	Регистры управления скоростью обмена (содержат 16 битовый коэффициент деления)	Чтение, запись	062h 063h
UCA0MCTL	Регистр управления модуляцией	Чтение, запись	064h
UCA0STAT	Регистр состояния	Чтение, запись	065h
UCA0RXBUF **UCA0TXBUF**	Регистр буфера приема передачи	Чтение Чтение, запись	066 067h

В таблице 7.1. показано, что все регистры доступны для чтения и записи, кроме буфера приема-передачи данных, доступного только для чтения.

Таблица 7.2 Регистры управления модуля USCI_A0

UCA0CTL0

7	6	5	4	3	2 1	0
UCCKPH	**UCCKPL**	**UCMSB**	**UC7BIT**	**UCMST**	**UCMODEx**	**UCSYNC=1**
Фаза тактового сигнала 0 – данные выставляются по 1-му фронту, считываются – по 2-му; **1** - – данные считываются по 1-му фронту, выставляются – по 2-му;	Полярность тактового сигнала. 0 – низкий уровень неактивного состояния. 1 – высокий уровень неактивного состояния	Порядок передачи битов 0 – младший бит первый; 1- - Старший бит первый.	Размер данных 0 – 8 бит 1 – 7 бит	0 – ведомый 1 - ведущий	Режим 00 – 3-х проводной 01 – 4-х проводной 10 - -//- 11- **I2C**	

Таблица 7.3 Регистры управления модуля USCI_A0

UCA0CTL1

7 6	5	4 3 2 1	0
UCCSELx	**USCI_A0**	**Unused**	**UCSWRST**
Выбор источника тактового стигнала в режиме ведущего 00 ------ 01 – ACLK 10 - SMCLK 11 - SMCLK	0 – не активен 1 - активен	Не используется	1 - Программный сброс модуля 0 – рабочее состояние

В синхронном режиме для связи микроконтроллеров с внешними устройствами используются выводы (выделены на рисунке 7.2).:

UCxSIMO - ведомый - ввод, ведущий - вывод; (P3.4/UCA0TXD/UCA0SIMO)

UCxSOMI - ведомый - вывод, ведущий - ввод;(P3.5/UCA0RXD/UCA0SOMI)

UCxCLK - тактовый сигнал ведущего; (P3.0/UCB0STE/UCA0CLK/A5)

UCxSTE - разрешение передачи данных (в 4 проводном режиме).

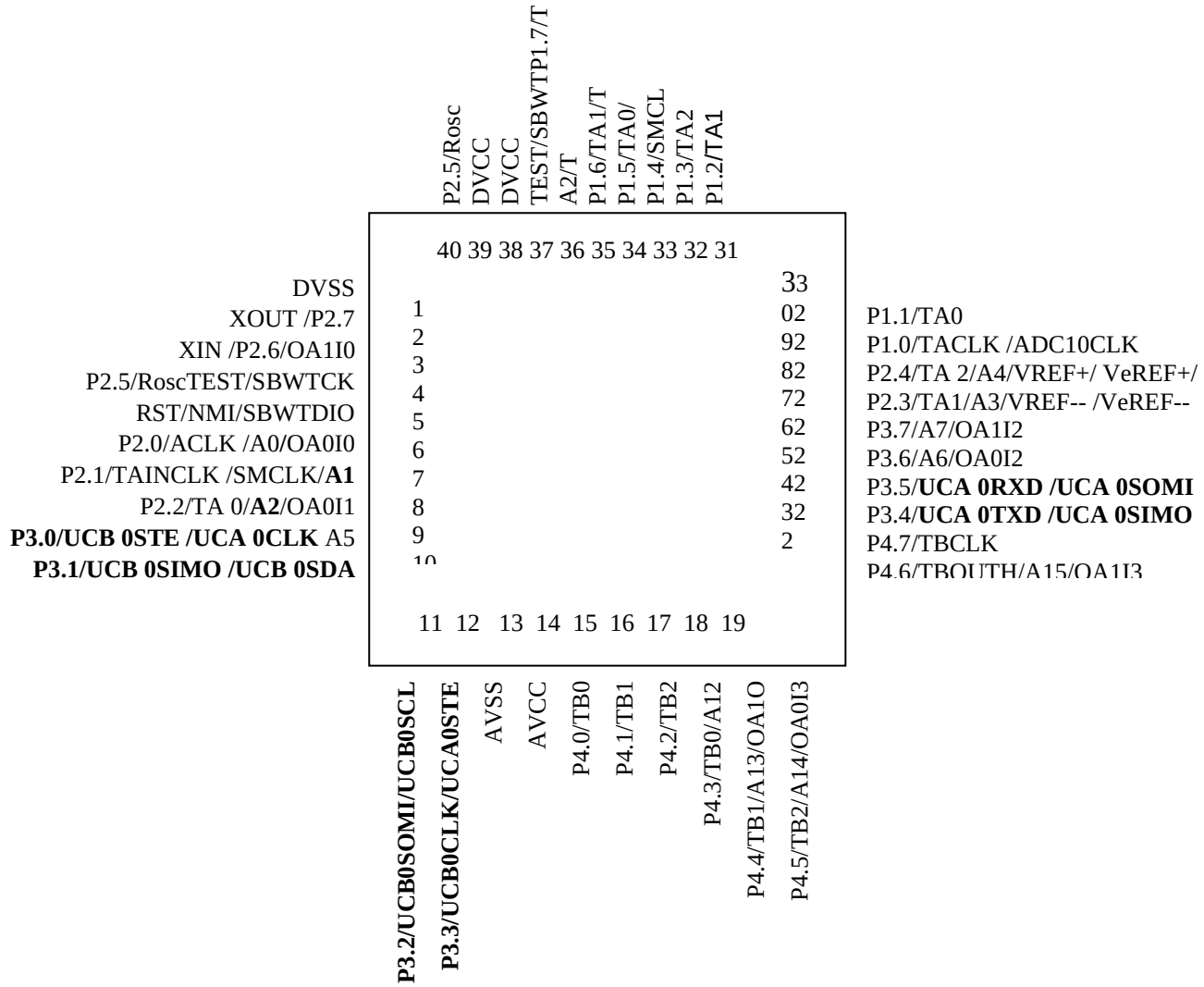


Рисунок 7.2 – Внешние выводы, используемые интерфейсом SPI

Режим **SPI** включается, если установить бит UCSYNC, а разновидность **SPI** (3-х или 4-х проводной) определяется битом UCMODEx.

7.3. Пример программы

Текст программы должен содержать инициализационную часть, где отключается сторожевой таймер, биты параллельного порта настраиваются на передачу данных от **USCI,** инициализируется последовательный интерфейс для работы в режиме **SPI,** задается начальное значение данным. Далее представлена программа, предназначенная для передачи данных по последовательному интерфейсу **SPI**.

```
#include "msp430x22x4.h"
unsigned char Data;
volatile unsigned int i;

void main(void)
{
  WDTCTL = WDTPW + WDTHOLD;                 // Stop watchdog timer
  P3SEL |= 0x11;                          // P3.0,4 USCI_A0 option select
  UCA0CTL0 |= UCCKPH + UCMSB + UCMST + UCSYNC;  // 3-pin, 8-bit SPI master
  UCA0CTL1 |= UCSSEL_2;                     // SMCLK
  UCA0BR0 |= 0x02;
  UCA0BR1 = 0;
  UCA0MCTL = 0;
  UCA0CTL1 &= ~UCSWRST;                     // **Initialize USCI state machine**
  Data = 0x0FF;                           // Load initial data

  while(1)
  {
    Data++;                               // Increment Data value
    while (!(IFG2 & UCA0TXIFG));            // USCI_A0 TX buffer ready?
    UCA0TXBUF = Data;                       // Byte to SPI TXBUF
    for(i = 0xFFFF; i > 0; i--);            // Delay
  }
}
```

Текст программы начинается директивой препроцессора. Далее объявлены переменные

unsigned char Data; // байтовая величина без знака

volatile unsigned int i;// целая величина временного хранения

В отдельной строке пишется заглавие программы.

void main(**void**)

После заглавия следуют операторы инициализационной части. Далее организован повторяющийся цикл преобразования данных для передачи по последовательному интерфейсу с выдержкой времени. В каждом цикле проверяется флаг UCA0TXIFG готовности данных в буфере передачи.

7.4. Порядок выполнения работы

1. Прочесть инструкцию.
2. Составить проект языке C/C++ с использованием рассмотренной в п.3 программы.
3. Откомпилировать проект и записать в устройство.
4. Запустить программу на выполнение в непрерывном и пошаговом режимах.
5. Проследить по шагам за работой устройства последовательного интерфейса. Записать содержимое регистров **USCI** на каждом шаге.
6. Внести в программу изменения в соответствии с заданным вариантом.
7. Откомпилировать программу, записать в устройство и выполнить.
8. Записать результат выполнения программы в таблицу 12.

Таблица 12.2

Номер шага	Значение переменной	Значение в буфере передачи

7.5 .Варианты заданий

Составить алгоритм и программу языке C/C++ для передачи по последовательному интерфейсу значений *y* для значений x,.изменяющихся в интервале от 0000h до 0050h.

7.6 Содержание отчета

1. Цель работы.
2. Условие задания в соответствии с вариантом.
3. Алгоритм программы с комментариями.
4. Исходный текст программы с комментариями.
5. Результаты выполнения программы в виде таблицы 12.2 для ряда значений *y(x).*

7.7. Контрольные вопросы

1. Как функционирует последовательный интерфейс?
2. Какие режимы последовательного интерфейса имеет микроконтроллер?
3. Как организовать обмен данными двух устройств через SPI?
4. Какие режимы следует задать для инициализации последовательного интерфейса?
5. Как организовать вывод и ввод данных через последовательный интерфейс программным способом?
6. Как организовать вывод и ввод данных через последовательный интерфейс программно-аппаратным способом?
7. Пояснить работу программы.

8.ЛАБОРАТОРНАЯ РАБОТА №8. ИЗУЧЕНИЕ АРХИТЕКТУРЫ МИКРОКОНТРОЛЛЕРА TMS320F28335 ДЛЯ ЧАСТОТНОГО УПРАВЛЕНИЯ ЭЛЕКТРОДВИГАТЕЛЯМИ

8.1. Цель работы

Микроконтроллер **TMS320F28335** предназначен для цифрового управления электроприводами посредством полупроводниковых преобразователей электрической энергии, для управления в энергоустановках, в транспорте, для промышленных и медицинских применений.

Архитектура микроконтроллера **TMS320F28335** представлена его функциональной схемой и системой команд. Цель работы заключается в изучении архитектуры микроконтроллера, что необходимо пользователю при создании программного обеспечения и разработке схем принципиальных электрических управления электроприводами.

8.2. Функциональная схема микроконтроллера

Функциональная схема микроконтроллера представлена на рисунке 13.1. Микроконтроллер **TMS320F28335** имеет ядро **C2000** с 32 разрядным словом данных и является первым из промышленных микроконтроллеров, содержащим сопроцессор **FPU** для вычислений с плавающей точкой. Тактовая частота микроконтроллера - до 150 MHz и производительность до 300 MFLOPS. Микроконтроллер совместим по выводам со всеми микросхемами серии F283xx. Регистры ядра представлены в таблице 13.1.

Микросхема содержит 512 Кб флэш-памяти и оперативную память. Оба вида памяти имеют защищенные области, выделенные на схеме и имеющие ключи защиты. Для управления доступом служит модуль защиты программ (Code Security Module)

Быстродействующий 12-битный аналого-цифровой преобразователь ADC12 позволяет вводить аналоговые сигналы от датчиков обратных связей по 16 входным каналам. Для формирования и вывода слова состояния силовых ключей в зависимости от сигнала управления в микроконтроллере имеется широтно-импульсный модулятор (ШИМ, PWM, Pulse Width Modulator).

Для ввода двух последовательностей импульсов от энкодера имеются устройство eCAP захвата с шестью входами и устройство eQEP квадратурного счета импульсов с двумя входами. Поскольку в режиме захвата фиксируется интервал времени между импульсами, есть возможность определить значение частоты импульсов энкодера, что позволяет организовать обратные связи по скорости. Микросхема содержит 2–х проводной последовательный интерфейс I2C, а также интерфейсы SPI, SCI. Устройства FIFO (First Input First Output) с 16 уровнями позволяют в 16 буферах временно сохранять передаваемые через интерфейс данные. Микросхема имеет стандартный 2–х проводной последовательный интерфейс CAN для связи с внешними устройствами.

Общее количество внешних выводов – 176, внешних выводов общего применения 88 (GPIO, General Propose Input-Output), описание представлено в таблице 13.1.

Микропроцессоры 28335 имеют следующие периферийные устройства.

- **ePWM:** (enhanced Pulse Width Modulator, улучшенный широтно-импульсный модулятор, ШИМ): Модуль ePWM генерирует последовательность импульсов постоянной заданной частоты с длительностью, пропорциональной сигналу управления на входе ePWM; имеет выводы HRPWM для высокоскоростного вывода сигналов. Регистры ePWM имеют прямой доступ к памяти через шину DMA.

- **eCAP:** (enhanced capture peripheral улучшенное устройство захвата событий) использует 32-бит регистра таймера и регистры для четырех программируемых в режиме захвата входов событий. Это устройство может быть так же сконфигурировано для генерации ШИМ сигнала.

- eQEP: (enhanced QEP) – устройство квадратурного счета, использует 32-бит счетчика положения в режиме низких скоростей, устройство захвата и измерение высокой скорости, используя 32-битный таймер.

Это периферийное устройство имеет сторожевой таймер для выявления зависаний и логику выявления ошибок входа . logic to identify simultaneous edge transition in QEP signals.

- ADC: (Analog to Digital Converter, аналого-цифровой преобразователь) является 12-битным, 16-канальным. Он содержит два устройства для одновременного захвата и квантования. Регистры ADC поддерживаются прямым доступом к памяти (DMA).

Устройство имеет следующие виды последовательных коммуникационных интерфейсов.

- **eCAN**: Улучшенная версия двухпроводного интерфейса CAN (Controller Area Network, контроллер распределенных сетей). Интерфейс разработан в 1982 году фирмой Robert Bosch GmbH для автомобильной промышленности. Интерфейс предназначен для связи устройств на расстоянии не более 40 м со скоростью передачи до 1 Мбит/с. Сеть длиной 1500 м позволяет получить скорость передачи 10 Кбит/с. Имеет 32 буфера приема сообщений (mailboxes), запись времени сообщения, и совместима с CAN 2.0B.

- McBSP: (Multichannel Buffered Serial Port, многоканальный буферируемый последовательный порт) связан с линией E1/T1, допускает применение для стерео и аудио устройств. Регистры приема и передачи поддерживаются DMA, что значительно сокращает ресурсы времени для обслуживания этого интерфейса. Каждый модуль McBSP может быть сконфигурирован при необходимости как SPI.

- SPI: (Serial Programmable Interface, последовательный программируемый интерфейс) является высокоскоростным синхронным последовательным портом ввода и вывода, допускающим программируемую длину передаваемого слова до 16 бит. Порт SPI предназначен и применяется для связи между цифровыми сигнальными процессорами и внешними устройствами. Содержит внешнее устройство ввода вывода или расширение периферии, сдвиговые регистры, драйверы дисплеев, и АЦП. Связь нескольких устройств производится в режиме ведущий- ведомый (master/slave). Устройство SPI содержит 16 уровневый буфер приема передачи FIFO (First Input First Output, первым введен, первым выведен) чтобы сократить обслуживание прерываний.

- SCI: (Serial Communications Interface, последовательный коммуникационный интерфейс) является двухпроводным асинхронным последовательным портом, известным как UART (Universal Asynchronous Receiver Translator, универсальный асинхронный приемопередатчик). Устройство SCI содержит 16 уровневый буфер приема передачи FIFO чтобы сократить обслуживание прерываний.

- I2C: (Inter-Integrated Circuit, объединяющая цепь, предложена в Philips Semiconductors, версия 2.1). Модуль обеспечивает последовательный интерфейс между вычислительными и другими устройствами, связанными двухпроводной шиной. Внешние устройства могут передавать и принимать до 8 бит данных. Устройство SCI содержит 16 уровневый буфер приема передачи FIFO обслуживания прерываний.

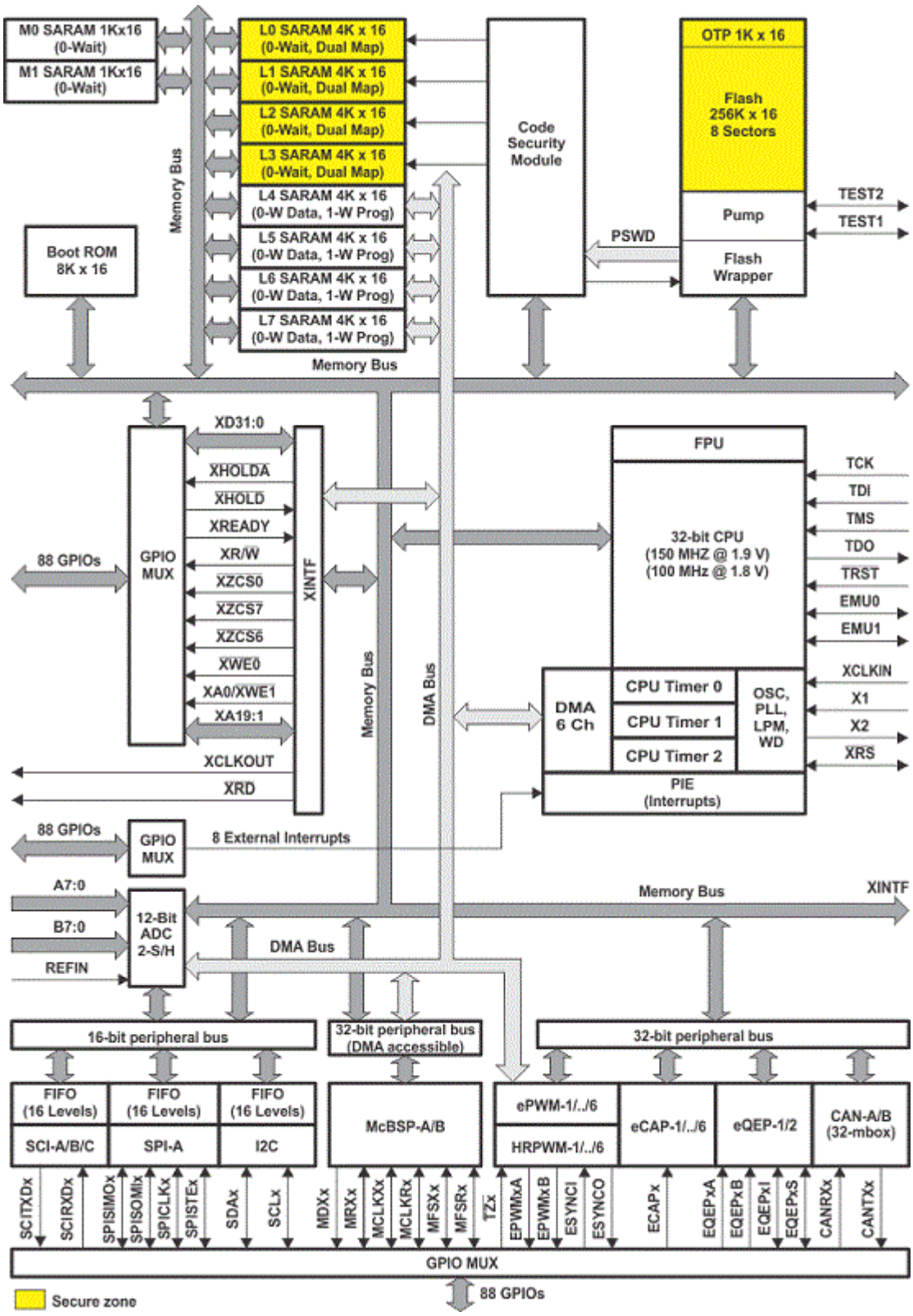


Рисунок 8.1 – Функциональная схема микроконтроллера

8.3. Порядок выполнения работы

1. Прочесть инструкцию и изучить архитектуру микроконтроллера **TMS320F28335.**
2. Включить компьютер и загрузить интегрированную среду разработки **CCSv4.**
3. Открыть проект, имеющийся в директории на языке С.
4. Откомпилировать проект **HVACI_Sensorless_F2833x** и выполнить **Debug.**
5. Настроить окно проекта так, чтобы были видны внутренние регистры микроконтроллера. (**View→Registers).**
6. Ознакомиться с регистрами **CPU** микроконтроллера.
7. Запустить проект на выполнение. Выполнится программа **HVACI_Sensorless_DevInt_ F2803x.c** инициализации периферийных устройств.
8. Записать изменившееся содержимое регистров **CPU** микроконтроллера и регистров периферийных устройств.
9. Выполнить задание в соответствии с заданным вариантом.

8.4. Варианты заданий

1. Описать структуру микроконтроллера и назначение входящих в него устройств.
2. Описать назначение и принцип действия средств последовательного интерфейса.
3. Как используются входы и выходы GPIO общего применения? Привести пример.
4. Как организовать ввод сигнала энкодера?
5. Как организовать ввод аналоговых сигналов?
6. Как увидеть содержимое области данных микроконтроллера, где находятся регистры периферийных устройств.
7. Какие регистры содержит последовательный порт I2C и где они расположены?
8. Как увидеть содержимое области данных микроконтроллера?
9. Какие регистры содержит последовательный порт SCI и где они расположены в памяти?
10. Для чего нужна косвенная адресация? Привести примеры использования косвенной адресации.
11. Для программы, приведенной в проекте, дать описание директив препроцессора, которые используются для настройки периферийных модулей.
12. Дать описание возможных режимов последовательного интерфейса.
13. Составить описание периферийных устройств микроконтроллера с указанием адресов их регистров.
14. Как выполняется настройка системы PIE прерываний и каковы источники прерываний?
15. Как ограничивается энергопотребление микроконтроллера?

8.5. Содержание отчета

1. Цель работы, вариант задания.
2. Описать структуру микроконтроллера.
3. Описать внутренние регистры процессора и назначение специальных регистров.
4. Структура проекта.
5. Выполнить задание в соответствии с вариантом.

8.6 Контрольные вопросы

1. Что относится к ядру и периферии микроконтроллера?
2. Как в микроконтроллере выполнены требования к интерфейсу, необходимые для управления электроприводами?
3. Что такое адресное пространство?
4. Каково назначение **CPU** и **FPU**?
5. Что является источником адреса?
6. Для чего нужна шина **DMA**?
7. Каково назначение специальных регистров?
8. Назначение и таймера в системе управления электроприводами.
9. Как увидеть содержимое внутренней памяти программ и данных микроконтроллера?
10. Как увидеть содержимое регистров интерфейсных устройств?

Таблица 13.1 Регистры ядра C2000 их адреса

Регистры CPU	комментарии
ACC = 0xFFFF0000 AH = 0xffff AL = 0x0	аккумулятор
P = 0xFFFFFFFF PH = 0xffff PL = 0xffff	Регистр результата
XT = 0x00000000 TH = 0x0 TL = 0x0	Регистр умножения
XAR0 = 0x00000000 AR0H = 0x0 AR0 = 0x0 XAR1 = 0x0000FFFF AR1H=0x0 AR1 = 0xFFFF XAR2 = 0x00000000AR2H = 0x0 AR2 = 0x0 XAR3 = 0x00000000AR3H = 0x0 AR3 = 0x0 XAR4 = 0x00000000AR4H = 0x0 AR4 = 0x0 XAR5 = 0x00000000AR5H = 0x0 AR5 = 0x0 XAR6 = 0x00000000AR6H = 0x0 AR6 = 0x0 XAR7 = 0x000002AA AR7H= 0x0 AR7=0x2AA	Регистры общего назначения
PC = 0x008000	Программный счетчик
IC = 0x008000	
RPC = 0x00932A	Программный счетчик возврата
ST0 = 0x0098 OVC = 0x0 PM = 0x1 V = 0x0 N = 0x0 Z = 0x1 C = 0x1 TC = 0x0 OVM = 0x0 SXM = 0x0	Регистр состояния 0
ST1 = 0xCA0B ARP = 0x6 XF = 0x0 M0M1MAP = 0x1 CNF = 0x0 OBJMODE = 0x1 AMODE = 0x0 IDLESTAT = 0x0 EALLOW = 0x0 LOOP = 0x0 SPA = 0x0 VMAP = 0x1 PAGE0 = 0x0 DBGM = 0x1 INTM = 0x1	Регистр состояния 1
DP = 0x0000 SP = 0x0484 IER = 0x0000 IFR = 0x0000 DBGIER	Указатель на данные Указатель стека Разрешение прерываний Флаги прерываний Разрешение прерываний в режиме Debug

Регистры FPU

R0H	Регистр с плавающей точкой 0 (Floating point register 0)
R1H	Регистр с плавающей точкой 1 (Floating point register 1)
R2H	Регистр с плавающей точкой 2 (Floating point register 2)
R3H	Регистр с плавающей точкой 3 (Floating point register 3)
R4H	Регистр с плавающей точкой 4 (Floating point register 4)
R5H	Регистр с плавающей точкой 5 (Floating point register 5)
R6H	Регистр с плавающей точкой 6 (Floating point register 6)
R7H	Регистр с плавающей точкой 7 (Floating point register 7)
STF	Регистр состояния **FPU** (Floating point status register)

Адреса периферийных устройств

Устройство содержит четыре области (кадра) для регистров периферии. Адресное пространство распределено следующим образом:
Кадр 0: устройства, соединенные напрямую с шиной памяти CPU. (Таблица 13-8).
Кадр 1 Периферийные устройства 32-битной периферийной шины (Таблица 13-9).
Кадр 2: Периферийные устройства 16-битной периферийной шины(Таблица 13-10).
Кадр 3: Периферийные устройства, соединенные с периферийной шиной 32-бит DMA-(прямого доступа к памяти, (Таблица 13-11)).

Регистры кадра 0

Device Emulation Registers x00 0880 – 0x00 09FF, 38, EALLOW protected
FLASH Registers(3) 0x00 0A80 – 0x00 0ADF 96, EALLOW protected
Code Security Module Registers 0x00 0AE0 – 0x00 0AEF,16 EALLOW protected
ADC registers (dual-mapped) 0x00 0B00 – 0x00 0B0F 16, Not EALLOW protected
0 wait (DMA), 1 wait (CPU), read only
XINTF Registers 0x00 0B20 – 0x00 0B3F 32 EALLOW protected
CPU-Timer 0, CPU-Timer 1,
CPU-Timer 2 Registers 0x00 0C00 – 0x00 0C3F 64 Not EALLOW protected
PIE Registers 0x00 0CE0 – 0x00 0CFF, 32 Not EALLOW protected
PIE Vector Table 0x00 0D00 – 0x00 0DFF, 256 EALLOW protected
DMA Registers 0x00 1000 – 0x00 11FF, 512 EALLOW protected

Регистры Кадра 0 имеют доступ к словам 16-бит и 32-бита. Если регистр защищен (EALLOW), запись невозможна при выполнении EALLOW. Команда EDIS запрещает запись to из регистра с запорченным содержимым. Регистры флэш-памяти защищены модулем защиты (Code Security Module,CSM).

Регистры кадра 1

eCAN-A Registers 0x00 6000 – 0x00 61FF 512
eCAN-B Registers 0x00 6200 – 0x00 63FF 512

ePWM1 + HRPWM1 registers	0x00 6800 – 0x00 683F	64
ePWM2 + HRPWM2 registers	0x00 6840 – 0x00 687F	64
ePWM3 + HRPWM3 registers	0x00 6880 – 0x00 68BF	64
ePWM4 + HRPWM4 registers	0x00 68C0 – 0x00 68FF	64
ePWM5 + HRPWM5 registers	0x00 6900 – 0x00 693F	64
ePWM6 + HRPWM6 registers	0x00 6940 – 0x00 697F	64
eCAP1 registers	0x00 6A00 – 0x00 6A1F	32
eCAP2 registers	0x00 6A20 – 0x00 6A3F	32
eCAP3 registers	0x00 6A40 – 0x00 6A5F	32
eCAP4 registers	0x00 6A60 – 0x00 6A7F	32
eCAP5 registers	0x00 6A80 – 0x00 6A9F	32
eCAP6 registers	0x00 6AA0 – 0x00 6ABF	32
eQEP1 registers	0x00 6B00 – 0x00 6B3F	64
eQEP2 registers	0x00 6B40 – 0x00 6B7F	64
GPIO registers	0x00 6F80 – 0x00 6FFF	128

Регистры кадра 2

System Control Registers	0x00 7010 – 0x00 702F	32
SPI-A Registers	0x00 7040 – 0x00 704F	16
SCI-A Registers	0x00 7050 – 0x00 705F	16
External Interrupt Registers	0x00 7070 – 0x00 707F	16
ADC Registers	0x00 7100 – 0x00 711F	32
SCI-B Registers	0x00 7750 – 0x00 775F	16
SCI-C Registers	0x00 7770 – 0x00 777F	16
I2C-A Registers	0x00 7900 – 0x00 793F	64

Регистры кадра 3

McBSP-A Registers (DMA)	0x5000 – 0x503F	64
McBSP-B Registers (DMA)	0x5040 – 0x507F	64
ePWM1 + HRPWM1 (DMA	0x5800 – 0x583F	64
ePWM2 + HRPWM2 (DMA)	0x5840 – 0x587F	64
ePWM3 + HRPWM3 (DMA)	0x5880 – 0x58BF	64
ePWM4 + HRPWM4 (DMA)	0x58C0 – 0x58FF	64
ePWM5 + HRPWM5 (DMA)	0x5900 – 0x593F	64
ePWM6 + HRPWM6 (DMA)	0x5940 – 0x597F	64

Модули ePWM, HRPWM могут быть отнесены к кадру 3 для доступа к DMA. Для этого бит 0 (MAPEPWM) в регистре MAPCNF (адрес 0x702E) должен быть установлен 1. Этот регистр защищен EALLOW. Если бит равен 0, то модули ePWM, HRPWM отнесены к кадру 1.

Встроенные функции

Ассемблер поддерживает многие встраиваемые функции.
Далее x, y и z имеют тип float, n - int. Функции $cvi, $int and $sgn возвращают целую величину, остальные – с плавающей точкой (float). Угол в тригонометрических функциях – в радианах.

$acos(*x*) Returns cos-1(*x*) in range [0, π], *x* [-1, 1]
$asin(*x*) Returns sin-1(*x*) in range [-π/2, π/2], *x* [-1, 1]
$atan(*x*) Returns tan-1(*x*) in range [-π/2, π/2]
$cos(*x*) Returns the cosine of *x*
$cosh(*x*) Returns the hyperbolic cosine of *x*
$fabs(*x*) Returns the absolute value |*x*|
$floor(*x*) Returns the largest integer not greater than *x*, as a float
$fmod(*x*, *y*) Returns the floating-point remainder of *x*/*y*, with the same sign as *x*
$int(*x*) Returns 1 if *x* has an integer value; else returns 0. Returns an integer.
$ldexp(*x*, *n*) Multiplies *x* by an integer power of 2. That is, *x* □□2n
$log(*x*) Returns the natural logarithm ln(*x*), where *x*>0
$max(*x*, *y*, .*z*) Returns the greatest value from the argument list
$min(*x*, *y*, .*z*) Returns the smallest value from the argument list
$pow(*x*, *y*) Returns *xy*

2.5. Задания для практических занятий

1 Разработать алгоритм программного обеспечения реального времени в режиме программного ввода-вывода.

2 Разработать алгоритм для программного обеспечения реального времени в режиме программно-аппаратного (по прерываниям) ввода-вывода от параллельных портов.

3 Разработать алгоритм и программу формирования сигнала задания для разгона электродвигателя с постоянным ускорением.

4 Разработать алгоритм и программу формирования сигнала задания для разгона и торможения электродвигателя.

5 Разработать алгоритм и программу формирования широтно-импульсной модуляции от таймера.

6 Разработать алгоритм и программу формирования таймером двух последовательностей импульсов с заданным сдвигом по фазе.

7 Разработать алгоритм и программу формирования широтно-импульсной модуляции от таймера с шириной импульсов, пропорциональной входному сигналу.

8 Разработать алгоритм и программу логического управления механизмами подачи и главного движения металлорежущего станка от микроконтроллера в зависимости от состояния кнопок пульта управления.

9 Разработать алгоритм и программу ввода аналоговых сигналов задания и обратной связи по давлению в микроконтроллер и расчета сигнала управления на выходе ПИД- регулятора в системе управления давлением.

10 Разработать алгоритм и программу расчета сигнала задания скорости, и передачу сигнала задания по прерываниям через последовательный интерфейс.

11 Разработать алгоритм и программу ввода аналоговых сигналов от датчиков в микроконтроллер и вывода их цифровых значений через асинхронный интерфейс *UART*.

12 Разработать алгоритм и программу ввода аналоговых сигналов задания и обратной связи по давлению в микроконтроллер и расчета сигнала управления на выходе ПИД- регулятора в системе управления давлением

3 РАЗДЕЛ КОНТРОЛЯ ЗНАНИЙ

Текущая аттестация выполняется по результатам практических занятий и подготовки к выполнению каждой лабораторной работы. Для текущей проверки знаний применяются контрольные вопросы в конце каждой из лабораторных работ.

Итоговая аттестация (экзамен) выполняется по ответам на вопросы экзаменационного билета с применением контрольных вопросов и задач.

Контрольные вопросы и задачи к экзамену

1. Структура и принцип действия микроконтроллераа. Выполнение процессором командного цикла. Система команд.
2. Как формируется адрес команды в фазе выборки?
3. Какова структура команды и какие есть способы адресации?
4. Какая информация хранится в специальных регистрах?
5. Организация подпрограмм. Стековая адресация и стековая области памяти.
6. Как организовать подпрограмму и обращение к ней?
7. Как используется стековая адресация в организации подпрограмм?
8. Что происходит в программном счетчике *PC* при обращении к подпрограмме и возврате?
9. Принципы обмена информацией. Синхронный и асинхронный обмен.
10. Программная, аппаратно-программная и аппаратная организация обмена информацией.
11. Система прерываний микроконтроллера.
12. Режим работы последовательного интерфейса SPI.
13. Режим работы последовательного интерфейса I2C.
14. Старт-стопный ввод-вывод данных через UART.
15. Прерывания от последовательного интерфейса и их использование в автоматизации.
16. Принцип действия программируемого таймера.
17. Ввод и вывод событий микроконтроллером.
18. Работа таймера в режиме захвата при вводе сигнала энкодера.
19. Работа таймера в режиме сравнения при формировании широтно-импульсной модуляции (ШИМ).
20. Прерывания от таймера.
21. Ввод и вывод данных с применением прерываний.
22. Организация прямого доступа к памяти.
23. Цифро-аналоговое преобразование (ЦАП). Принципы построения ЦАП.
24. Аналого-цифровое преобразование (АЦП). Принципы построения АЦП.
25. Когда АЦП запрашивает прерывание?
26. Микроконтроллер как средство сбора информации от датчиков технологических величин.
27. Микроконтроллер как динамическое звено в системе управления.
28. Алгоритмы и программы расчета выходных сигналов регуляторов.
29. Способы повышения производительности микропроцессоров. Конвейерное выполнение команд. Кэш-память, ее назначение и принцип действия.

30. Процессоры с сокращенным набором команд (RISC) и с полным набором команд (CISC), их преимущества и недостатки.
31. Как организовать функцию в языке *C*?
32. Как использовать указатель для доступа к элементам массива?
33. Как задать начальные значения элементам массива?
34. Чем отличается структура данных от массива?
35. В чем отличие структуры данных и объединения?
36. Как получить доступ к полям структуры для записи и чтения?
37. Как и для чего применяются директивы препроцессора?
38. В каком месте программы располагаются объявления и описания переменных?
39. Какова структура программы на языке С?
40. Построить пример применения условного оператора.
41. Построить пример применения оператора цикла.
42. Построить пример применения оператора переключения.
43. Каковы области применения параллельного и последовательного интерфейсов?
44. Каковы области применения различных видов последовательного интерфейса?
45. Чем отличается дискретный регулятор от непрерывного?
46. Записать дискретные передаточные функции интегрирующего, дифференцирующего и пропорционального звеньев.
47. Записать дискретные передаточные функции П, ПИ и ПИД- регуляторов.
48. Как выполнить расчет параметров регуляторов?
49. Как сформировать алгоритм преобразования сигнала регулятором?
50. Каков принцип действия контроллера нечеткой логики?
51. Преимущества и недостатки контроллера нечеткой логики.
52. Где целесообразно применять контроллеры нечеткой логики?
53. Каков принцип действия искусственного нейрона?
54. Как соединяются нейроны в сеть?
55. Где возможно применение ИНС в автоматике?
56. Как организовать ввод на вход микроконтроллера сигналов от реле уровня, давления, температуры?
57. Параллельная обработка информации как способ повышения производительности микроконтроллеров.
58. Микроконтроллер как средство реализации интеллектуального управления.

Задачи

1. Составить алгоритм формирования интервала времени.
2. Составить алгоритм реверсивного управления АД.
3. Составить алгоритм ПИД- регулятора.
4. Составить алгоритм численного интегрирования.
5. Составить алгоритм численного дифференцирования.
6. Составить алгоритм регулятора с заданной передаточной функцией.
7. Соединить два микроконтроллера последовательным интерфейсом на параллельную работу.

4 ВСПОМОГАТЕЛЬНЫЙ РАЗДЕЛ

МИКРОКОНТРОЛЛЕРЫ В АВТОМАТИЗАЦИИ

Учебная программа учреждения высшего образования по учебной дисциплине для специальности

7-06-0713-04 «Автоматизация»

Минск 2024 г.

Учебная программа составлена на основе образовательного стандарта ОСВО 7-06 0713-04 2023 и учебного плана специальности 7-06-0713-04 «Автоматизация» (рег № II с ФИТР 45 д-1/уч.) утв. 18.04.2023 г.

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

Учебная программа по учебной дисциплине «Микроконтроллеры в автоматизации» разработана для специальности 7-06-0713-04 «Автоматизация».

Целью изучения учебной дисциплины является овладение знаниями и умениями для использования микроконтроллеров в автоматизации промышленных установок.

Основными задачами учебной дисциплины являются:

- изучение принципа действия микроконтроллеров: системы команд микроконтроллера, принципов и методов составления алгоритмов и программ реального времени для использования в системах автоматики;

- изучение интегрированной среды разработки микроконтроллеров и ее использование для программирования микроконтроллеров.

Знания по микроконтроллерам используются в проектировании систем автоматизации, при анализе и модификации систем автоматизации производства.

Учебная дисциплина базируется на знаниях, полученных при изучении дисциплин: «Высшая математика», «Физика», «Информатика», «Теоретические основы электротехники» и «Теория автоматического управления». Знания и умения, полученные при изучении дисциплины «Микроконтроллеры в автоматизации», применимы при подготовке магистерской диссертации и необходимы при проектировании систем автоматизации, при анализе и модификации систем автоматизации производства.

В результате изучения учебной дисциплины магистрант должен:

знать:

– принцип действия микроконтроллеров, запоминающих устройств, аппаратных средств интерфейса, универсальные пакеты программ;

- принципы и методы составления алгоритмов и программ реального времени для использования в автоматизации;

-язык программирования С и его применение для программирования микроконтроллеров;

уметь:

- разрабатывать алгоритмы и программное обеспечение реального времени для автоматизации;

-использовать интегрированную среду разработки для программирования микроконтроллера;

-выполнять разработку схем принципиальных электрических систем управления с микроконтроллерами;

иметь навык:

-использования современных программных средств микроконтроллеров с универсальными пакетами компьютерной поддержки инженерного анализа и расчетов.

Освоение данной учебной дисциплины обеспечивает формирование следующих компетенций:

СК-7 Выполнять технико-экономическое обоснование проектных решений.

Согласно учебным планам на изучение учебной дисциплины отведено:

- для очной формы образования всего 136 ч., из них аудиторных - 50 часов;
- для заочной формы образования всего 136 ч., из них аудиторных - 12 часов.

Распределение аудиторных часов по курсам, семестрам и видам занятий приведено ниже.

Таблица 1. Очная форма образования

Курс	Семестр	Лекции, ч.	Лабораторные занятия, ч.	Практические занятия, ч.	Форма промежуточной аттестации
1	2	18	16	16	экзамен

Таблица 2. Заочная форма образования

Курс	Семестр	Лекции, ч.	Лабораторные занятия, ч.	Практические занятия, ч.	Форма промежуточной аттестации
2	3	4	4	4	экзамен

СОДЕРЖАНИЕ учебного материала

Раздел I. Введение.

Тема 1.1. Основные понятия и определения. Принципы построения и функционирования микроконтроллеров.

Краткие исторические сведения. Принципы построения и функционирования микроконтроллеров. Ядро и периферия микроконтроллера. Алгоритм работы процессора при выполнении командного цикла

Архитектура микроконтроллеров: функциональная схема и система команд. Магистральный принцип организации обмена информацией в микроконтроллерах.

Назначение, области применения и технико-экономическая эффективность микроконтроллеров в автоматизации.

Классификация вычислительных систем с параллельной обработкой информации, параллелизм на уровне команд и на уровне алгоритмов. Архитектура микропроцессоров с сокращенным набором команд (RISC) и с полным набором команд (CISC). Конвейерное выполнение команд. Кэш-память, ее назначение и принцип действия.

Режимы энергопотребления микроконтроллеров.

Раздел II. Система команд и программирование микроконтроллеров.

Тема 2.1. Система команд и алгоритмы реального времени. Составление алгоритмов и программ.

Система команд микроконтроллера. Алгоритмы и программы автоматизации как программы реального времени. Инициализационная и циклическая части алгоритма и программы.

Интегрированная среда разработки проектов автоматизации для микроконтроллеров.

Применение языка высокого уровня С для программирования микроконтроллера. Директивы препроцессора, ключевые слова, комментарии языка С. Выражения и операторы языка С.

Структура программы на языке С. Простейшая программа для микроконтроллера на С.

Этапы компиляции программы на С.

Алгоритмы и программы для микроконтроллеров. Разработка алгоритмов инициализационной и циклической частей программ реального времени. Организация ветвлений и циклов в алгоритмах и программах.

Использование стековой области памяти при организации подпрограмм.

Примеры алгоритмов и программ логического управления оборудованием.

Примеры алгоритмов и программ управления в системах с обратными связями и регуляторами.

Раздел III. Организация обмена информацией (интерфейс).

Тема 3.1. Принципы и организация обмена информацией в микроконтроллерах. Параллельный и последовательный интерфейс

Программная, аппаратно-программная и аппаратная организация обмена. Режимы прерываний и прямого доступа к памяти.

Организация ввода-вывода информации программным способом через параллельный и последовательный интерфейс в микроконтроллере.

Организация ввода-вывода информации с использованием прерываний. Параллельная работа микроконтроллеров. Последовательные интерфейсы UART, I2C и SPI.

Тема 3.2. Программируемый таймер и его применение. Режимы захвата и сравнения в микроконтроллерах. Примеры их применения

Назначение и использование программируемого таймера. Структура, принцип действия и режимы работы таймера. Прерывания от таймера и их применение.

Виды событий микроконтроллера. Режимы захвата и сравнения как ввод и вывод событий микроконтроллером с применением программируемых таймеров. Организация прерываний в режимах захвата и сравнения. Примеры применений режимов захвата и сравнения.

Тема 3.3. Аналого- цифровое и цифро-аналоговое преобразования.

Принцип действия цифро-аналоговых преобразователей (ЦАП). Аналого-цифровые преобразователи (АЦП), их назначение, принципы действия, функциональные схемы и применение в автоматизации.

Раздел IV Применение микроконтроллеров в системах автоматизации.

Тема 4.1. Цифровые фильтры, их передаточные функции, алгоритмы и программы расчета выходной величины.

Определение требуемого быстродействия микроконтроллера. Выбор числа разрядов слова данных по требуемой точности системы управления.

Алгоритм и программа расчета выходной величины цифрового фильтра. Цифровое дифференцирование и интегрирование. Методы расчета параметров цифровых П, ПИ, и ПИД- регуляторов. Алгоритмы и программы расчета выходных сигналов цифровых П, ПИ, и ПИД регуляторов.

Тема 4.2. Применение последовательного интерфейса в системах автоматизации.

Структура и принцип действия последовательного интерфейса. Преимущества, недостатки и области применения различных видов последовательного интерфейса. Синхронный и асинхронный интерфейс. Прерывания от устройств последовательного интерфейса. Примеры применения последовательного интерфейса.

Тема 4.3. Применение микроконтроллеров для частотного управления электроприводами.

Требования к специализированным микроконтроллерам для управления преобразователями электрической энергии и частотноуправляемыми электроприводами переменного тока. Требования к программному обеспечению.

Структура микроконтроллера для управления преобразователями электрической энергии и электроприводами, особенности периферийных устройств.

Структура программного обеспечения для управления электроприводами

Тема 4.4. Алгоритмы интеллектуального, энергосберегающего и ресурсосберегающего управления оборудованием.

Методы искусственного интеллекта в автоматизации. Микроконтроллер как средство интеллектуального управления. Применение микроконтроллера для рационализации энергопотребления и ресурсосбережения.

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА УЧЕБНОЙ ДИСЦИПЛИНЫ
очная форма обучения

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов			Количество часов СР	Форма контроля знаний
		Лекции	Практические занятия	Лабораторные занятия		
1	2	3	4	6	8	9
	2 семестр					
1	Введение					
1.1	Основные понятия и определения. Принципы построения и функционирования микроконтроллеров.	2				
	Практическое занятие №1. Алгоритмы и программы автоматизации для микроконтроллера		2			Отчет
	Лабораторная работа №1. Архитектура 16-разрядного микроконтроллера			2		Отчет
2	Система команд и программирование микроконтроллеров					
2.1	Система команд и алгоритмы реального времени. Составление алгоритмов и программ.	2				
	Практическое занятие №2. Создание проектов автоматизации и программ для микроконтроллера		2			Отчет
	Лабораторная работа №2. Создание проекта автоматизации в интегрированной среде Code Composer Studio™ (CCS) для микроконтроллеров MSP430			2		Отчет
3	Организация обмена информацией (интерфейс).					
3.1.	Принципы и организация обмена информацией в микроконтроллерах. Параллельный и	2				

	последовательный интерфейс.					
	Практическое занятие №3. Составление алгоритмов и программ логического управления оборудованием.		2			Отчет
	Лабораторная работа №3. Алгоритмы и программы автоматизации для микроконтроллеров			2		Отчет
3.2	Программируемый таймер и его применение. Режимы захвата и сравнения в микроконтроллерах. Примеры их применения.	2				
	Практическое занятие №4. Разработка алгоритмов и программ с использованием порограммируемого таймера.		2		5	Отчет
	Лабораторная работа №4 Разработка алгоритмов и программ с использованием порограммируемого таймера			2		Отчет
3.3	Аналого- цифровое и цифро-аналоговое преобразования	2				
	Практическое занятие №5. Использование аналого-цифрового преобразователя в автоматике.		2			Отчет
	Лабораторная работа №5. Разработка алгоритмов и программ микроконтроллера MSP430 с использованием аналого-цифрового преобразователя ADC10.			2		Отчет
4	Применение микроконтроллеров в системах автоматизации.					
4.1	Цифровые фильтры, их передаточные функции, алгоритмы и программы расчета выходной величины.	2				
	Практическое занятие №6. Алгоритмы и программы расчета выходной величины регуляторов.		2			Отчет
	Лабораторная работа №6. Алгоритмы и программы расчета выходной величины регуляторов, интегрирующих и дифференцирующих звеньев.			2		Отчет
4.2	Применение последовательного интерфейса в системах автоматизации	2				
	Практическое занятие №7. Применение последовательного интерфейса для приема и передачи данных в системах автоматизации.		2			Отчет
	Лабораторная работа №7. Применение последовательного интерфейса для приема и передачи данных			2		Отчет
4.3	Применение микроконтроллеров для частотного управления электроприводами.	2				
	Практическое занятие №8. Изучение архитектуры микроконтроллера TMS320F28335 для частотного управления электроприводами.		2			Отчет

	Лабораторная работа №8. Изучение архитектуры микроконтроллера TMS320F28335 для частотного управления электроприводами.			2		Отчет
4.4	Алгоритмы интеллектуального, энергосберегающего и ресурсосберегающего управления оборудованием.	2				
	Итого за семестр	18	16	16	86	экзамен
	Всего аудиторных часов	50				

УЧЕБНО-МЕТОДИЧЕСКАЯ КАРТА УЧЕБНОЙ ДИСЦИПЛИНЫ

заочная форма обучения [1]

Номер раздела, темы	Название раздела, темы	Количество аудиторных часов			Количество часов СР	Форма контроля знаний
		Лекции	Практические занятия	Лабораторные занятия		
	2 семестр					
1	Введение					
1.	Основные понятия и определения. Принципы построения и функционирования микроконтроллеров	2				конспект
	Практическое занятие №1. Алгоритмы и программы автоматизации для микроконтроллера		2			Отчет
	Лабораторная работа №1. Архитектура 16-разрядного микроконтроллера			2		Отчет
2	Система команд и программирование микроконтроллеров					
2.1	Система команд и алгоритмы реального времени. Составление алгоритмов и программ. Принципы и организация обмена информацией в микроконтроллерах. Параллельный и последовательный интерфейс.	2		2		
	Практическое занятие №2. Составление алгоритмов и программ логического управления оборудованием.		2			отчет
	Лабораторная работа №3. Алгоритмы и программы автоматизации для микроконтроллеров			2		отчет
	Итого за семестр	4	4	4	124	
	Всего аудиторных часов	12				экзамен

[1] Темы учебного материала, не указанные в Учебно-методической карте, отводятся на самостоятельное изучение студентом.

Информационно-методическая часть

Средства диагностики результатов учебной деятельности

Оценка уровня знаний магистранта на экзамене производится по десятибалльной шкале в соответствии с критериями, утвержденными Министерством образования Республики Беларусь.

Для оценки достижений магистранта рекомендуется использовать следующий диагностический инструментарий:

– устный и письменный опрос во время лабораторных занятий;

– выполнение заданий по отдельным темам;

– защита выполненных на лабораторных занятиях индивидуальных заданий;

– собеседование при проведении индивидуальных и групповых консультаций.

Методические рекомендации по организации и выполнению самостоятельной работы студентов

При изучении дисциплины рекомендуется использовать следующие формы самостоятельной работы:

– решение задач на составление алгоритмов и программ автоматики;

– составление алгоритмов и программ управления оборудованием;

– разработка схем принципиальных соединений микроконтроллера в системе автоматизации;

– проработка тем (вопросов), вынесенных на самостоятельное изучение, используя рекомендуемую литературу.

–

СПИСОК ЛИТЕРАТУРЫ

Основная литература

1. Опейко О. Ф. Микропроцессорные средства в автоматизированном электроприводе / О. Ф. Опейко, Ю.Н. Петренко // Учебное пособие / Амалфея - Минск 2008. - 340 с.
2. Опейко О. Ф. Лабораторные работы по курсу «Микропроцессорные средства в автоматизированном электроприводе» Учебно-методическое пособие для студентов специальности 1 53.01.05 «Автоматизированные электроприводы». Учебное электронное издание. / О. Ф. Опейко - Минск, БНТУ 2016.

Дополнительная литература

3. Гируцкий И. И. Микропроцессорная техника систем автоматизации учебно-методическое пособие / И. И. Гируцкий, А. Г. Сеньков. – Минск : БГАТУ, 2022. – 224 с.
3. Семейство микроконтроллеров MSP430x2xx. Архитектура, программирование, разработка приложений / пер.с англ. Евстифеева А. В. – М.: Додэка-XXI, 2010. – 544 с.: ил. – (Серия «Мировая электроника»).

4. www.ti.com/msp430 (дата доступа 13.01.2022)
5. eZ430-RF2500 Development Tool User's Guide/Literature Number: SLAU227E September 2007. www.ti.com/msp430 (дата доступа 13.01.2022)
6. MSP430 Assembly Language Tools User's Guide (SLAU131) www.ti.com/msp430 (дата доступа 13.01.2022).
7. MSP430 Optimizing C/C++ Compiler User's Guide (SLAU132) www.ti.com/msp430 (дата доступа 13.01.2022)
8. Язык С++. Учебное пособие / И.Ф. Астахова, С. В. Власов, В. В. Фертиков, А.В. Ларин. – Мн.: Новое знание, 2003. – 203 с.
9. Страуструп Б. Язык программирования С. М.; СПб.: БИНОМ – Невский диалект, 1999.
10. Шупляк В.И. С++. Практический курс: учеб. Пособие / В.И. Шупляк. – Минск: Новое знание, 2008. – 576 с.
11. Керниган Б., Д. Ритчи Язык программирования С.,1978
12. Kernighan Brian W., D. M. Ritcie The C programming language (ANSI C) Prentice-Hall Software series Second edition1988.
13. Draft ANSI C Standard (ANSI X3J11/88-090) (May 13, 1988), Third Public Review.
14. "ANSI Standards Action Vol. 36, #48" (PDF). American National Standards Institute. 2005-12-02. Archived from the original (PDF) on 2016-03-04. Retrieved 2009-08-06.
15. www.arm.com (дата доступа 13.01.2022)
16. TMS320F28335, F28334, F28332, F28235, F28234, F28232 Digital Signal Controllers / Texas Instruments. Literature Number: SPRS439 June 2007–Revised August 2012. 188 P.
17. www.ti.com (дата доступа 13.01.2022).
18. www.STMicroelectronics.com (дата доступа 13.01.2022)
19. Анхимюк, В.Л. Теория автоматического управления / В.Л. Анхимюк, О.Ф. Опейко, Н.Н. Михеев Мн.: Дизайн ПРО, 2002. – 343 с.
20. Куо Б. Теория и проектирование цифровых систем управления: Пер. с англ.– М.: Машиностроение, 1986. – 448 с.
21. Jury, E.I. Inners and Stability of Dynamic Systems. // E.I. Jury, / A Willey-Interscience Publications, John Willey & Sons. New York-London-Sydney-Toronto, 1974.
22. Опейко О. Ф. Синтез регулятора тока системы векторного управления асинхронным электродвигателем // Вісник КДУ імені Михайла Остроградського.Випуск 1/2014(84) - С 9-14.
23. Опейко, О. Ф. Робастный синтез дискретных ПИД регуляторов для объектов с интервальными параметрами / Мехатроника, автоматизация, управление, том 19, № 6, 2018. – с. 374--379.
24. Опейко, О. Ф. Синтез управления в двухконтурной дискретной системе = Control synthesis for two loops discret system / О. Ф. Опейко // Системный анализ и прикладная информатика. - 2018. – №1. - С. 22-26.
25. Головко В.А., Краснопрошин В. В. Нейросетевые технологии обработки данных : учеб. пособие / Минск : БГУ, 2017. – 263 с.
26. www.mathWork.com ((дата доступа 13.01.2022).)
27. https://en.www.wikiwersity.org>wiki>Fuzzy_Logic (дата доступа 22.03.2023)
28. https://en.www.wikiwersity.org>wiki>Genetic_algorithm (дата доступа 22.03.2023).