МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

Белорусский национальный технический университет

Лабораторные работы по курсу «Микропроцессорные средства в автоматизированном электроприводе»

Учебно-методическое пособие для студентов специальности 53 01 05 «Автоматизированные электроприводы»

Электронный учебный материал

Минск ◊ БНТУ ◊ 2016

УДК-51.001.57:621(075.8)

Автор:

О. Ф. Опейко

Рецензент: А. А. Несенчук

Учебно-методическое пособие по тринадцати лабораторным работам позволяет изучить 16-разрядный микроконтроллер MSP430 и интегрированную среду разработки для создания проектов автоматизации на основе микроконтроллера. Учебно-методическое пособие может быть использовано для выполнения лабораторных работ и самостоятельной работы студентов как дневного, так и заочного отделений высших учебных заведений.

Белорусский национальный технический университет
пр-т Независимости, 65, г. Минск, Республика Беларусь
Тел.(017) 232-77-52 факс (017) 232-91-37

Регистрационный № БНТУ/ФИТР46-4.2016

Содержание

1. Лабораторная работа №1. Архитектура 16 разрядного микроконтроллера MSP430

1.1. Цель работы

Целью работы является изучение архитектуры микроконтроллера MSP430. 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 друними микроконтроллерами. Это ультра низкое 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.

1.2. Функциональная схема и принцип действия микроконтроллера MSP430

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. Подлежащая обработке информация вначале должна быть введена в устройство, преобразуясь в двоичный код. Следующим этапом будет обработка информации, например вычисления, анализ. Вводимая информация, результаты обработки информации, в том числе и промежуточные данные могут быть сохранены в запоминающем устройстве. Обработанная информация выводится с помощью устройства вывода (рисунок 1.1).

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).

Архитектурой вычислительного устройства называется его внутренняя структура (функциональная схема) в совокупности с системой команд. Изучение архитектуры позволяет оценить возможность использования устройства для определенных применений и способы применения, в частности написания программ.

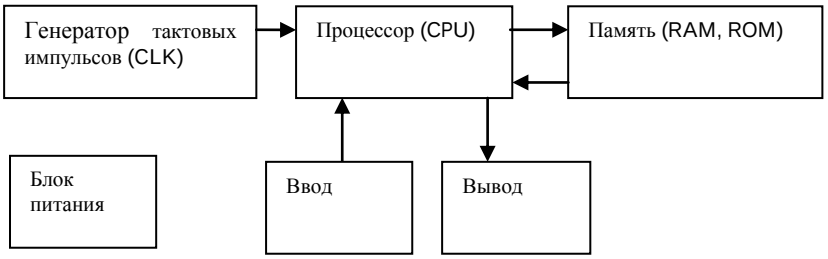


Рисунок 1.1 – Вычислительное устройство.

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 показаны на рисунках 1.2, 1.3 для двух исполнений корпуса. 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.

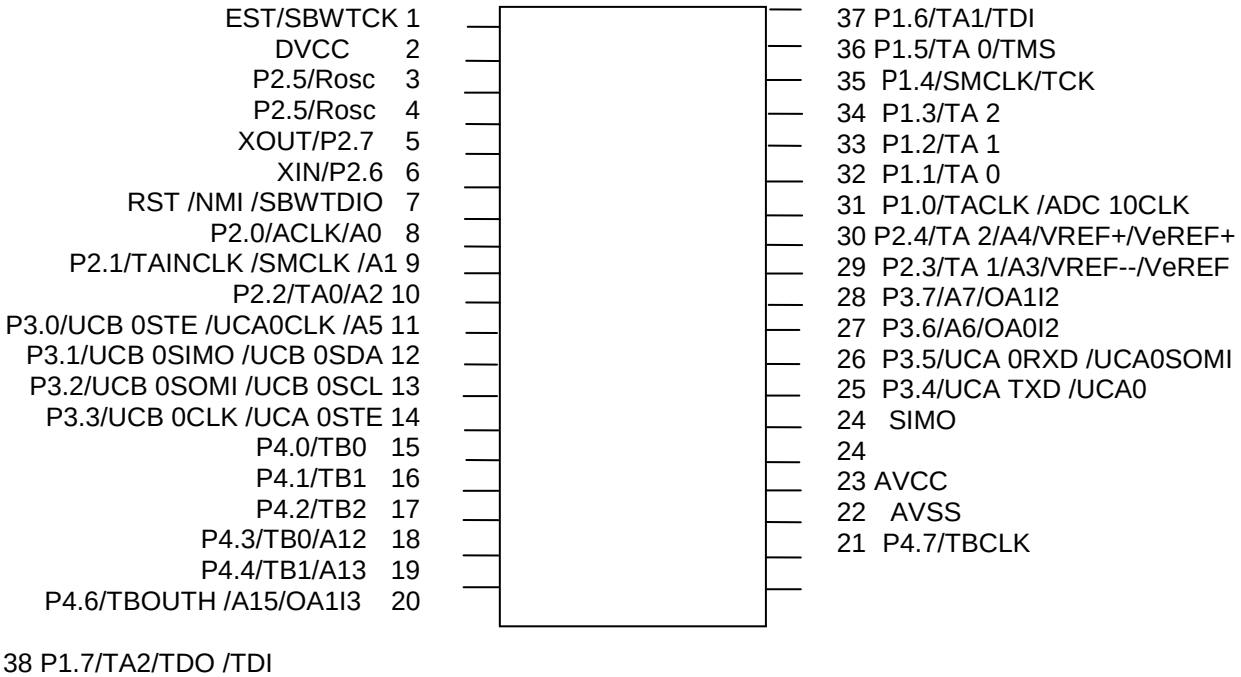


Рисунок 1.2 – Микросхема MSP430x22x4 с 38 выводами.

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 В, и потребляют ток 250

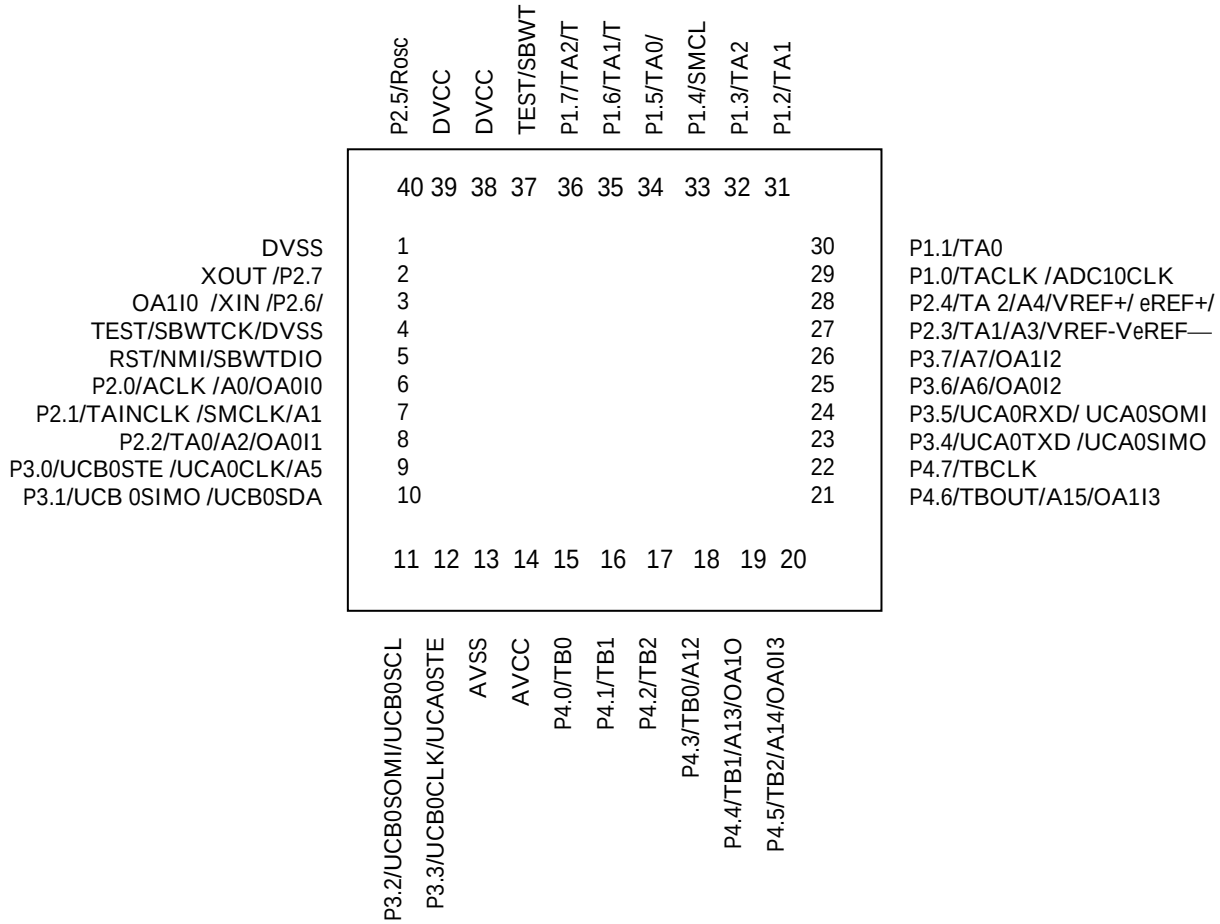


Рисунок 1.3 – Микросхема MSP430x22x4 с 40 выводным корпусом

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 1.4. 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.

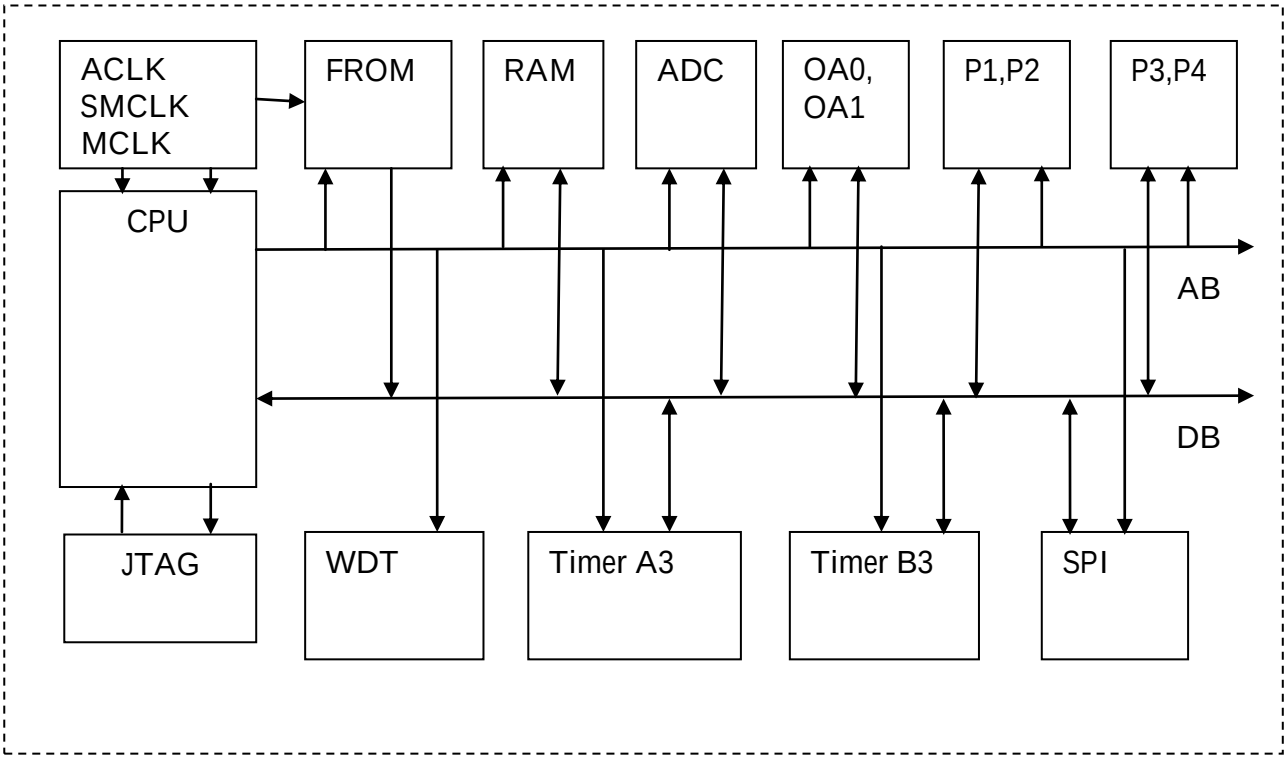


Рисунок 1.4– Функциональная схема MSP430x22x4

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 (Serial Programmable Interface) предназначен для обмена информацией последовательным двоичным кодом, что 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.

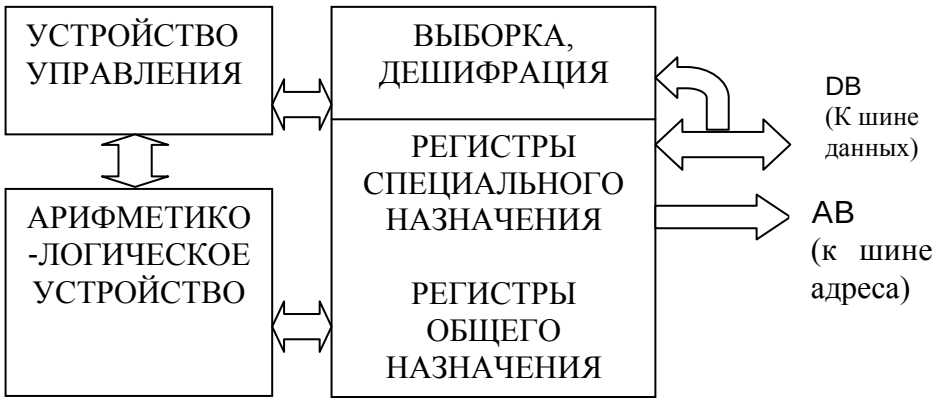


Рисунок 1.5– Центральное процессорное устройство (ЦПУ).

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.

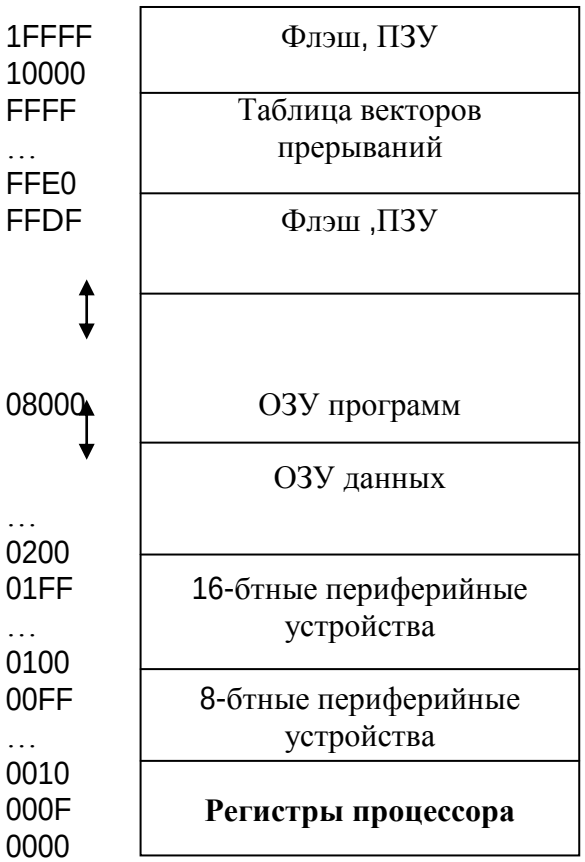
1.2. Организация памяти и внутренние регистры процессора

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.

Организация памяти **Таблица 1**.1

		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	*16-bit* *8-bit* *8-bit SFR*	*01FFh--0100h* *0FFh--010h* *0Fh--00h*

На рисунке 1.6.показан примерный вид распределения адресного пространства между устройствами в различных микроконтроллерах MSP430.



На рисунке 1.6 стрелками показаны границы областей, которые могут отличаться у различных микросхем семейства MSP430. Область адресов 0000-000F отводится внутренним регистрам процессора, специальным и общего назначения.

Внутренние регистры процессора представлены на рисунке 1.7. 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).

По младшим адресам, начиная от 0000, расположены 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).

Рисунок 1.6 –Карта памяти.

Рисунок 1.7 –Внутренние регистры процессора.

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 1.8. Кроме флагов регистр состояния содержит биты управления процессора: общее разрешение прерываний GIE (General Interrupt Enable), бит отключения процессора CPU OFF и биты управления осциллятором.

Далее расположены регистры общего назначения R4…R15, 8- разрядные периферийные устройства и так далее в соответствии со схемой на рисунке 1.7.

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

Рисунок 1.8– Регистр состояния

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),6(R6)*	*M(2+R5)----> M(6+R6)*
Символический (относительно PC)	*MOV EDE,TONI*	*MOV EDE,TONI*	*M(EDE) ----> M(TONI)*
Абсолютный	*MOV &MEM,&TCDAT*	*MOV &0200 h , &0202h*	*M(0200)----> M(0202)*
Косвенный	*MOV @Rn,Y(Rm)*	*MOV @R10,0(R6)*	*M(R10) ----> M(0+R6)*
Косвенный автоинкрементный	*MOV @Rn+,Rm*	*MOV @R10+,R11*	*M(R10) ----> R11* *R10 + 2----> R10*
Непосредственный	*MOV #X,R11*	*MOV #45R11*	*#45 ----> R11)*

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 так же команды перехода. Арифметические команды это суммирование, инкркментирование, вычитание и декрементирование. Умножение выполняется посредством обращения к аппаратному умножителю.

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 действовать одним из двух способов.

1. В верхней строке окна выбрать: Project →**Import Existing CCS /CCE Eclipse** Project. В появившемся окне нажать Browse с целью назначить директорию: D/MSP430/ccsv4/msp430examples/msp430x2xxASM Example. Нажать **ОК**. В левой части окна проекта появится название директории активного проекта.
2. В верхней строке окна выбрать: New C/C++project. Дать имя проекту в открывшемся окне.

Дважды нажать next для открытия страницы настроек проекта (CCS Settings page). Выбрать вариант устройства, используемого в проекте MSP430x2xx. Если проект предполагается выполнять на ассемблере, его следует конфигурировать как ассемблерный. Для этого следует выбрать Configure as an assembly only project.

Войдя в директорию, открыть файл конфигурации msp430F2101.ccxml. Выполеить Project →Add File to Active Project и выбрать в директории пример файла на ассемблере.

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 .При правильной настройке среды разработки и окон 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 основе имеющегося в директории examples готового проекта на языке ассемблера иди готового файла программы.
5. Настроить окно проекта так, чтобы были видны внутренние регистры микроконтроллера, используемые в программе. (View→Registers).
6. Ознакомиться с внутренними регистрами и интерфейсными модулями микроконтроллера.

1.7. Варианты заданий

1. Составить алгоритм командного цикла и указать устройства, задействованные на каждом шаге командного цикла.
1. Записать имена и назначение внутренних регистров процессора.
2. Назначение и принцип действия WDT.
3. Назначение и принцип действия таймера.
4. Назначение и принцип действия параллельных и последовательных портов.
5. Назначение и принцип действия АЦП.
6. Назначение генератора тактовых импульсов и его использование в микроконтроллере.

7. Как увидеть содержимое внутренней памяти микроконтроллера?
8. Как увидеть содержимое внутренних регистров процессора микроконтроллера?
9. Как настроить порт на ввод и на вывод данных?
10. Сколькими регистрами снабжен параллельный порт, каково назначение этих регистров?
11. Как выполняются арифметические команды?
12. Охарактеризовать способы адресации. Привести примеры.
13. Как выполняются логические команды?
14. Для чего нужна косвенная адресация? Привести примеры использования косвенной адресации.
15. Для программы, приведенной в примере проекта, дать полное описание каждой команды.
16. Сформулировать правила написания исходного текста программы на ассемблере.
17. Составить список интерфейсных устройств микроконтроллера MSP430F2274 с указанием их назначения.
18. Как выполняется сброс процессора, для чего он необходим и что при этом происходит?
19. Как ограничивается энергопотребление микроконтроллера?

1.8. **Содержание отчета**

1. Цель работы
2. Функциональная схема микроконтроллера MSP430F2274.
3. Структура адресного пространства и внутренние регистры процессора. Указать назначение каждого из специальных регистров и интерфейсных модулей.
4. Выполнить задания из п.7.

1.9.**Контрольные вопросы**

1.Что относится к ядру и периферии микроконтроллера?
2.Что такое адресное пространство?
3.Каков алгоритм выполнения командного цикла?
3.Как формируется адрес?
4.Что является источником адреса?
5.Записать имена и адреса внутренних регистров процессора.
6.Назначение и принцип действия процессора.
7.Каково назначение специальных регистров?
8.Назначение и принцип действия таймера.
9.Назначение и принцип действия параллельных и последовательных портов.
10.Назначение и принцип действия АЦП.
11.Назначение генератора тактовых импульсов, какие тактовые частоты возможны в микроконтроллере?
12.Как увидеть содержимое внутренней памяти микроконтроллера?
13.Как увидеть содержимое портов ввода-вывода?

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	$\bar{s}$∩d→ d		
BIS s,d	sUd→d	BIS R4,0(R12)	4
XOR s,d	s∀d→d		
AND s,d	**s**∩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∀V=0		
JL m	Если N∀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. Интегрированная среда Code Composer Studio™ v4.2 (CCS) для микроконтроллеров MSP430 и создание проекта автоматизации

2.1. Цель работы.

Целью работы является изучение интегрированной среды разработки (ИСР, английское название Integrated Development Environment, IDE) и создание нового проекта. Интегрированная среда разработки служит для создания систем автоматики на базе микроконтроллеров. Программные и аппаратные средства IDE обычно создаются производителем микроконтроллеров для ограниченного ряда микроконтроллеров и помогают пользователю в их применении.

2.2. Интегрированная среда разработки программ для микроконтроллеров

Интегрированная среда разработки систем автоматики содержит программное обеспечение и аппаратные средства. Программное обеспечение ИСР предназначено для создания *проекта автоматизации* на базе микроконтроллера, используя персональный компьютер. Проект автоматизации обычно содержит один или несколько программных модулей пользователя и библиотечные программы среды разработки. Каждая из создаваемых программ может быть написана на языке высокого уровня либо в мнемокоде ассемблера для указанного типа микроконтроллера. Организация пользовательского программного обеспечения в виде проекта автоматизации облегчает пользование библиотеками программных средств. Библиотеки программных средств могут содержать, в частности, программы-драйверы аппаратных средств интерфейса.

Первым шагом разработки является создание нового проекта либо загрузка в рабочую область готового проекта с целью его использования, и, возможно, модификации. Далее проект должен пройти компиляцию, в результате которой из исходных текстов программ, написанных на языке программирования, формируется программа в машинном коде. В процессе компиляции генерируются сообщения об ошибках в проекте. После исправления всех ошибок и повторной компиляции проект готов к загрузке в память микроконтроллера, что предполагает использование аппаратных средств ИСР.

Производители некоторых микроконтроллеров обеспечивают их ИСР, которая позволяет выполнить *эмуляцию* (симуляцию) микроконтроллера, то есть воспроизвести программными средствами работу микроконтроллера с возможностью анализа содержимого памяти и внутренних регистров в процессе выполнения программы. Эмуляция означает имитационное моделирование работы микроконтроллера в программной среде персонального компьютера. Используя эмуляцию, разработчик имеет возможность разработать программное обеспечение и выполнить его отладку на персональном компьютере. Следующим этапом является запись программы, представленной в двоичном коде, в память микроконтроллера, что предполагает использование аппаратных средств ИСР.

Аппаратные средства ИСР обычно представлены демонстрационной или отладочной платой, содержащей разъем для установки микроконтроллера и стандартные средства интерфейса для подключения к персональному компьютеру.

Пакет программ CCSv.4.2 (Code Composer Studio™ v4.2)предназначен для разработки программного обеспечения систем автоматики на основе микросхем фирмы Texas Instruments. В данной лабораторной работе рассматривается применение CCS для микроконтроллеров MSP430.

2.3. **Инсталляция программного обеспечения** Code Composer Studio™ v**4.2 для микроконтроллеров** MSP430

Для установки программного обеспечения необходимы системные ресурсы персонального компьютера, представленные в таблице 1.1.

В начале загружается с диска все программное обеспечение и обязательно все архивные файлы. После этого, чтобы инсталлировать с DVD Code Composer Studio™ v4., следует выполнить CCS_4.2.x.x.x.exe. Необходимо следовать инструкции, показанной на экране.

Драйверы аппаратных средств эмуляторов USB JTAG для серии MSP-FET430UIF и eZ430 устанавливаются автоматически при инсталляции CCS. Драйверы параллельных портов не устанавливаются автоматически, но могут быть выбраны вручную при инсталляции.

Таблица 2.1. **Системные ресурсы**

	Рекомендуемые системные ресурсы	Минимальные системные ресурсы
Процессор	Dual Core 1.5 GHz	1.5 GHz
RAM	2 GB	1 GB
Свободное место на диске	2 GB	300 MB (зависит от объема задачи и выбирается при инсталляции)
Операционная система	Microsoft® Windows® XP with SP2 (32/64 bit) or Microsoft® Windows® XP with SP2 (32/64 bit) or	Windows Vista® with SP1 (32/64 bit) or Windows Vista® (32 bit) or Windows 7® (32/64 bit) Windows 7® (32/64 bit)

Драйверы и компоненты ИСР для параллельных портов MSP-FET430PIF выбираются вручную. Архив CCS_x_x_x.zip должен быть для этого полностью извлечен в персональный компьютер.

2.4. **Использование программного обеспечения Code Composer Studio™ v4.2 для создания проектов**.

Программное обеспечение CCS v4.2 включает файлы на языках C и ASM, а так же целые сконфигурированные проекты.

Далее описывается последовательность создания и загрузки проекта в устройство, его выполнения.

1.Запуск Code Composer Studio выполняется в следующем порядке:

Start → All Programs → Texas Instruments → Code Composer Studiov4.2.4 → Code Composer Studio v4.

Появляется окно Select a workspace, где следует нажать Browse, чтобы выбрать директорию на диске D . Откроется окно, необходимое для создания нового проекта.

2. Для создания нового проекта следует выбрать File → New → CCS Project.

3. В появившемся окне набрать имя проекта (только латинские буквы и цифры) и нажать next.

4. Установить тип проекта для MSP430.

5. Дважды нажать next для открытия страницы настроек проекта (CCS Settings page). Выбрать вариант устройства MSP430F2274, используемого в проекте.

6. Добавить пример программы (the flashing LED code example) в проект, выбирая Project → Add Files to Active Project. Пример программы расположен в директории **…** \ccs4\msp430\examples. Чтобы найти пример программы, пригодный для выбранного типа устройства, используйте таблицу 2.2.

Таблица 2.2. **Примеры программ**

Устройства MSP430	Примеры программ
MSP430x1xx	<...>\msp430x1xx\C-Source\msp430x1xx.c
MSP430x2xx	<...>\msp430x2xx\C-Source\msp430x2xx.c
MSP430x4xx	<...>\msp430x4xx\C-Source\msp430x4xx.c
MSP430x5xx	<...>\msp430x5xx\C-Source\msp430x5xx.c
MSP430L092	<...>\msp430x5xx\C-Source\msp430l092.c

7. Если используется устройство с USB Flash, как, например, MSP-FET430UIF или eZ430, то они конфигурируются по умолчанию. Интерфейс для разработки может быть изменен с помощью файлов конфигурации *.ccxml в проекте. Меню содержит подключение TI MSP430 USB1.

Следует выбрать "save file" в ниспадающем меню или нажать **“save”** чтобы запомнить изменение конфигурации.

8. Для компиляции программ следует выполнить Project → Build Active Project. Последуют сообщения об ошибках. Необходимо исправить ошибки и повторить компиляцию.

9. Устройство должно быть подключено к разъему USB системного блока компьютера. Для загрузки приложения в устройство следует выполнить Target → Debug Active Project. По этой команде стирается содержимое памяти целевого устройства, память устройства программируется, устройство перезапускается.

10.Для запуска прикладной программы служит команда: Target → Run (F8) или нажать кнопку Play на панели инструментов.

Если указанных средств запуска нет в окне проекта или они не активны, следует изменить вид окна View→ Debug.

Если указанных средств запуска нет в окне проекта или они не активны, это может являться признаком отсутствия связи с устройством. Если между CCS и устройством нет связи, следует обратиться к документу SLAU278 на сайте www.ti.com.

Создание проекта закончено, если команды Target → Run активны и могут быть использованы.

Для запуска приложения следует выполнить: → Run (F8) или нажать кнопку Play на панели инструментов. Если между CCS и устройством нет связи, следует обратиться к FAQ Debugging #1.

Использовать пример готового проекта можно, выбирая Project → Import Existing CCS/CCE Eclipce Project. Пример проекта расположен в директории <D/MSP/**…** \ccs4\msp430\examples \example projects.

Пример исходного текста программы на ассемблере имеет вид

```
;-------------------------------------------------------------------------
        .cdecls C,LIST,"msp430x21x1.h"  ; головной файл (адреса       периф.устройств микросхемы

        .text                                                   ;    начало программы
;-------------------------------------------------------------------------
RESET           mov.w   #300h,SP                        ;    инициализация стека
StopWDT         mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; остановка таймера WDT
SetupP1         bis.b   #001h,&P1DIR                        ; разряд P1.0 – на вывод;
```

```
Mp          xor.b   #001h,&P1OUT                    ; переключение P1.0
Wait        mov.w   #0500,R15                       ; запаздывание R1
L1          dec.w   R15                             ; декремент (R15 - 1)
            jnz     L1                              ; интервал истек?
            jmp     Mp                              ; повторение цикла
                        ;
;-----------------------------------------------------------------------
;       Interrupt Vectors
;-----------------------------------------------------------------------------------------------
        .sect   ".reset"                            ; MSP430 RESET Vector
        .short  RESET           ;
        .end
```

Последние версии документов для MSP430™, необходимые при разработке приложений, доступны на сайте (www.ti.com/msp430). Полное описание устройств и разработки программного обеспечения CCS на ассемблере или языке C, можно прочесть в документах [3-5].

В проект можно добавить новый файл File → New → Source File. Следует ввести имя файла с расширением .c, если файл на языке C, или .asm., если файл на ассемблере. Если же нужно использовать имеющийся файл, следует выбрать Project → Add Files to Active Project и найти в директории нужный файл и добавить его в папку проекта.

Для упрощения преобразования кода следует использовать файлы с расширением .h, в которых для каждого устройства определены имена регистров и битов. Для включения файлов .h, соответствующих целевому устройству, добавляют строку #include <msp430xyyy.h> для файлов на языке C и .cdecls C,LIST,"msp430xyyy" для ассемблерных программ, где xyyy указывает на номер партии MSP430.

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

Для остановки режима отладки служит команда: Target → Terminate All (красная кнопка), после которой устройство можно отсоединить.

Перед выходом из интегрированной среды CCS рекомендуется закрыть все окна. Команда File → Exit позволяет выйти из CCS.

2.5. Порядок выполнения работы

1. Войти в интегрированную среду разработки проектов и ознакомиться с возможностями интегрированной среды.
2. Создать проект. Изучить структуру проекта и входящие в него файлы.
3. Открыть каждый файл из директории проекта и ознакомиться с его содержанием.
4. Записать назначение файлов и охарактеризовать содержимое.
5. Открыть файл исходного текста программы на языке ассемблера.
6. Создать новую программу на языке ассемблера в соответствии с заданным вариантом.
7. Откомпилировать программу, записать диагностические сообщения об ошибках.

2.6. Содержание отчета

1. Цель работы.
2. Директория проекта и назначение каждого файла в нем.
2. Алгоритм и текст программы в соответствии с заданным вариантом на ассемблере с комментариями.
3. Диагностические сообщения об ошибках и их исправление.

2.7. Варианты заданий

1. Составить алгоритм и программу формирования интервала времени 0 .5 с.
2. Составить алгоритм и программу формирования интервала времени 1 с.
3. Составить алгоритм и программу формирования интервала времени 0.3 с.
4. Составить алгоритм и программу формирования интервала времени 0.2 с.
5. Составить алгоритм и программу формирования интервала времени 2 с..
6. Составить алгоритм и программу вывода в параллельный порт значения из R4.
7. Организовать ввод данных в порт P2 и вывод через порт P1 введенных данных.
8. Составить алгоритм и программу записи константы в R8 и вывода ее через P1.
9. Составить алгоритм и программу записи константы в R7, используя непосредственную адресацию, и вывод ее через порт P1.
10. Составить алгоритм и программу записи константы в параллельный порт, используя прямую адресацию.
11. Составить алгоритм и программу записи константы в параллельный порт, используя косвенную адресацию.
12. Составить алгоритм и программу записи в регистры R4 и R5 двух констант и вывода их суммы *y* через P1.
13. Составить алгоритм и программу записи в P1 членов геометрической прогрессии 1,2,4, …с интервалом 1 с.
14. Составить алгоритм и программу ввода *x* через P2 и вывода суммы *x+R4* через P1.
15. Составить алгоритм и программу ввода *x* через P2 и вывода суммы 4**x+R4* через P1.
16. Составить алгоритм и программу ввода *x* через P2 и вывода 2**x-R5* через P1.
17. Составить алгоритм и программу вывода разности 2**R7-R5* через P1.
18. Составить алгоритм и программу ввода *x* через P1 и вывода суммы *x+R14* через P2.
19. Составить алгоритм и программу ввода *x* через P2 и вывода суммы 2**x+R5* через P1.
20. Составить алгоритм и программу ввода *x* через P2 и вывода 2**x-R15* через P1.
21. Составить алгоритм и программу вывода разности 3**R7-R5* через P1.
22. Составить алгоритм и программу вывода разности *R7-2*R5* через P1.

2.8. **Контрольные вопросы**

1. Какова последовательность создания проекта?
2. Что означает компиляция проекта?
3. Пояснить алгоритм и программу.
4. Из каких папок и файлов состоит проект?
5. Как интегрированная среда позволяет управлять выполнением программы в режиме отладки?
6. Какие способы адресации использованы в программе, и как выполняются соответствующие команды?
7. Какова последовательность создания проекта и что при этом происходит в интегрированной среде?
8. Как увидеть содержимое внутренних регистров процессора микроконтроллера?

3Лабораторная работа №3. Выполнение программы в микроконтроллере MSP430

3.1. Цель работы.

Целью работы является выполнение программы в микроконтроллере MSP430 и анализ результатов с использованием интегрированной среды CCS. Результаты выполнения программы отображаются в регистрах ядра и интерфейсных устройств, которые доступны в интегрированной среде.

3.2. 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**.

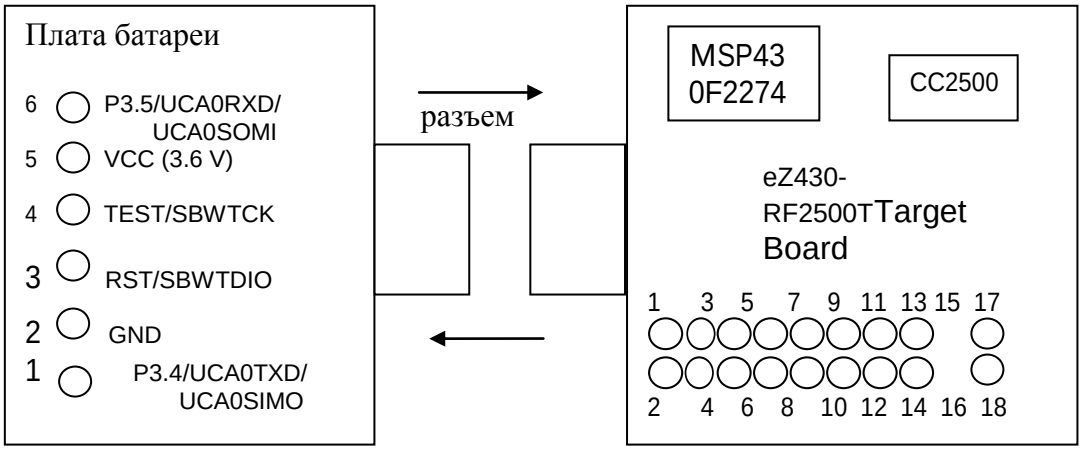


Рисунок 3.1 – Соединение устройства с платой батареи.

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 (их 21), 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 3.1 и замкнуть пластмассовую перемычку.

3.3. **Использование программного обеспечения** Code Composer Studio™ v4.2

Сделать проект, созданный в лабораторной работе 2, активным: выделить его в директории левого окна проектов и в ниспадающем списке выбрать 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.

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 запуска прикладной программы служит команда: Target → Run (F8) или нажать кнопку Play на панели инструментов.

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. Команда File → Exit позволяет выйти из CCS. Перед выходом из CCS закройте все окна.

3.4.**Настройка среды разработки проекта**

Как правило, проект может быть компилирован и передан в устройство с использованием настройки по умолчанию. Изменение настройки необходимо для наиболее рационального использования всех возможностей интегрированной среды разработки.

Настройка активного проекта доступна через Project → Properties. Рекомендуется и применяется следующая настройка. Целевое устройство специфицируют, выбирая Project → Properties → CCS Build Settings → Device Variant. Автоматически выбираются соответствующие командный файл компоновки и библиотека для поддержки выполнения программ (Linker Command File and Runtime Support Library).

Разработка приложений MSP430 требует отключения сторожевого таймера, который по умолчанию включен и сбрасывает устройство. Для этого используют WDTCL = WDTPW + WDTHOLD. Эту операцию лучше расположить перед инициализацией, что выполняется перед главной процедурой.

Если сторожевой таймер не отключен, он формирует сброс и теряется возможность записи программы в память устройства. Далее показан пример отключения сторожевого таймера на С.

```
{
/* Вставьте инициализацию*/
WDTCTL = WDTPW + WDTHOLD; // Stop Watchdog timer
/*===================================*/
/* Choose if segment initialization */
/* should be done or not. */
```

```
/* Return: 0 to omit initialization */
/* 1 to run initialization */
/*================================*/
return (1);   }
```

На языке ассемблера данная операция имеет вид

```
StopWDT    mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; Stop WDT
```

Для управления видом окна проекта в верхней части окна имеется средство настройки view, которое позволяет выводить в окно проекта необходимую информацию.

3.5. **Порядок выполнения работы**

1. Сделать проект, созданный в лабораторной работе 2, 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.3.7).
6. Откомпилировать и выполнить измененный проект, записать содержимое регистров ядра и параллельных портов.

3.6. **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 различных этапах выполнения программы).

3.7. **Варианты задач**

1. Составить алгоритм и программу вывода в параллельный порт 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 1 светодиод с интервалом времени 0.2 с.
5. Составить алгоритм и программу вывода сигналов на 1 светодиод с интервалом времени 2 с.
6. Составить алгоритм и программу вывода в параллельный порт заданного в R4 значения интервалом времени 0 .5 с..
7. Организовать ввод информации в порт P2 и вывод через порт P1 введенных данных с интервалом 1 с.
8. Составить алгоритм и программу записи константы в R8 и вывода ее через параллельный порт P1 с интервалом 2 с.
9. Составить алгоритм и программу записи константы в R7, используя непосредственную адресацию, и вывод ее через порт P1 с интервалом 1 с.
10. Составить алгоритм и программу записи константы в параллельный порт, используя прямую адресацию, с интервалом 0.5 с.

11. Составить алгоритм и программу записи константы в параллельный порт, используя косвенную адресацию с интервалом 0.5 с..
12. Составить алгоритм и программу записи в регистры R4 и R5 двух констант и вывода их суммы *y* через P1 с интервалом 3 с.
13. Составить алгоритм и программу записи в P1 членов геометрической прогрессии 2,4,8, …с интервалом 2 с.
14. Составить алгоритм и программу ввода *x* через P2 и вывода суммы *x+R4* через P1 с интервалом 1 с.
15. Составить алгоритм и программу ввода *x* через P2 и вывода суммы 4**x+R4* через P1 с интервалом 1 с.
16. Составить алгоритм и программу ввода *x* через P2 и вывода разности 2**x-R5* через P1 с интервалом 1 с.
17. Составить алгоритм и программу вывода разности 2**R7-R5* через P1 с интервалом 1 с.
18. Составить алгоритм и программу ввода *x* через P1 и вывода суммы *x+R14* через P2 с интервалом 1 с.
19. Составить алгоритм и программу ввода *x* через P2 и вывода суммы 2**x+R5* через P1 с интервалом 1 с.
20. Составить алгоритм и программу ввода *x* через P2 и вывода разности 2**x-R15* через P1 с интервалом 1 с.

3.8. **Контрольные вопросы**

1. Как открыть новый проект?
2. Из чего складывается проект?
3. Как в проект добавить файл?
4. Как использовать готовый проект?
5. Какие настройки проекта можно изменить?

4. **Лабораторная работа №4**. **Разработка алгоритмов и программ на ассемблере с использованием подпрограмм**

4.1. **Цель работы**

Целью работы является разработка алгоритмов и программ на ассемблере с использованием подпрограмм, настройка проекта и отладка программного обеспечения проекта на ассемблере с использованием устройства eZ430RF2500.

4.2. **Организация подпрограмм**

Подпрограммой называется программный модуль, исходный текст которого записывается отдельно от основной (главной) программы, а загрузочный модуль, образуемый в результате компиляции подпрограммы, хранится в отдельной области памяти. К подпрограмме можно обращаться из других программных модулей, в том числе и из подпрограмм. Обычно не допускается обращение из подпрограммы к этой же подпрограмме.

Подпрограммы позволяют структурировать программное обеспечение. Обычно в подпрограмме выполняется операция или законченная функция. Например, интервал времени можно организовать с помощью подпрограммы. Особенно эффективно использование подпрограмм для часто повторяющихся операций.

Для организации подпрограмм используется *стековая адресация*, когда указателем адреса служит указатель стека SP. В оперативной памяти данных отводится специальная область для стековой адресации. В микроконтроллере MSP430F2274 эта область начинается адресом 0x0300h. Для записи в стек и извлечения из стека используются команды

PUSH s SP-2 → SP, s → @SP, MOV @SP+, d.

Здесь s – регистр-источник слова данных, d – регистр-приемник слова данных. Источником и приемником слова данных может быть любой из регистров ядра R0…R15

В исходном тексте программы на ассемблере можно записывать команду извлечения из стека POP d , которая эмулируется командой пересылки MOV @SP+, d.

При стековой адресации последнее слово данных, записанное в стек, остается единственным доступным для чтения из стека, так как именно его адрес хранится в SP. После считывания этого слова данных, то есть извлечения из стека в регистр микроконтроллера, становится доступным для считывания записанное перед ним слово данных, и так далее. Таким образом, запись данных в стек может выполняться в произвольном порядке, однако извлечение из стека должно выполняться в порядке, обратном записи в стек.

Текст каждой подпрограммы должен начинаться меткой, которая символизирует адрес начала подпрограммы. Далее могут идти команды записи в стек содержимого регистров ядра и затем команды операций подпрограммы. В конце подпрограммы должны быть команды извлечения из стека в порядке, обратном записи в стек.

Текст каждой подпрограммы должен заканчиваться командой возврата из подпрограммы RET.

Для обращения к подпрограмме служит команда CALL d вызова подпрограммы, в которой могут быть использованы различные виды адресации, например

CALL #wait ; вызов с использованием метки wait (адреса начала подпрограммы)

CALL R5 ; вызов подпрограммы по адресу, записанному в R5 (косвенная адресация)

CALL @R5 ; косвенная адресация по косвенному содержимому (в R5 должен быть записан адрес памяти, где хранится адрес начала подпрограммы)

По команде CALL содержимое программного счетчика PC записывается в стековую область памяти. В программный счетчик записывается адрес начала подпрограммы. Происходит переход к выполнению подпрограммы. В конце подпрограммы выполняется команда возврата RETURN. По команде RETURN из стека извлекается старое содержимое программного счетчика. Поэтому происходит возврат к адресу команды, следующей за командой CALL вызова данной подпрограммы.

Пример текста программы с обращением к подпрограмме и с текстом подпрограммы представлен далее.

```
        .cdecls C,LIST,"msp430x21x1.h"                    ; адреса периферийных устройств
        .text                    ; исходный текст
;-------------------------------------------------------------------------
RESET           mov.w   #300h,SP                          ; инициализация стека
StopWDT         mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; остановка таймера WDT
ssP1            mov.b    #001h,&P1DIR                    ; инициализация разряда P1.0 – на вывод;
M               xor.b    #001h,&P1OUT                    ; переключение P1.0
                call     #W                              ; интервал времени
                jmp    M                                 ; повторение цикла
                ;
W                                       ; подпрограмма формирования интервала времени
        mov.w   #05000,R15              ; интервал пропорционален 5000
L1      dec.w   R15                     ; R15 - 1
        jnz     L1                      ; если интервал не истек, L1
        ret                             ; возврат
```

```
;       Interrupt Vectors   векторы прерываний
;-----------------------------------------------------------------------
        .sect   ".reset"                ; MSP430 RESET Vector
        .short  RESET                   ;
        .end
```

Программа предназначена для переключения светодиода, подключенного к выводу P1.0 порта P1, с заданным в регистре R15 интервалом времени с помощью команды *xor.b* логической операции «исключающее или». Для формирования интервала времени используется подпрограмма, в которой организована запись в R15 значения, пропорционального интервалу времени, и декрементирование регистра R15.

4.3. **Порядок выполнения работы**

1. Открыть проект, созданный в лабораторной работе 3 в соответствии с заданным вариантом. В директории проекта содержится файл с исходным текстом программы на языке ассемблера.
2. В текст программы внести изменения, а именно, для формирования интервала времени использовать подпрограмму.
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→Registers).
6. Запустить программу на выполнение в пошаговом режиме и записать содержимое специальных регистров и регистров, используемых в программе, на каждом шаге. Обратить внимание на изменения в указателе стека.
7. Запустить программу на выполнение Target →Run. Записать содержимое регистров до и после выполнения программы.
8. Остановить выполнение программы Target → Terminate All.
9. Изменить выдержку времени, запустить программу и посмотреть, как это повлияет на работу устройства.

4.4. **Содержание отчета**

1. Цель работы
2. Алгоритм главной программы и подпрограммы.
3. Исходный текст программ на языке ассемблера с комментариями.
4. Содержимое регистров микроконтроллера до выполнения подпрограммы, во время выполнения подпрограммы и после выполнения подпрограммы.
5. Выполнение задания в соответствии с вариантом.

4.5. **Варианты заданий.**

1. Описать содержимое внутренней памяти микроконтроллера.
2. Привести примеры специальных регистров и их назначение.
3. Как рассчитать значение, записываемое в регистр для формирования интервала времени ?
4. Как можно выполнить программу в пошаговом режиме?
5. Как использовать точки останова?
6. Дать описание откомпилированного проекта и указать назначение его частей.

7. Дать описание интерфейсных устройств микроконтроллера.
8. Записать значения регистра R0 в процессе выполнения программы и дать объяснение.
9. Записать значения регистра R2 в процессе выполнения программы и дать объяснение.
10. Записать значения регистров параллельных портов в процессе выполнения программы.
11. Как изменить конфигурацию проекта?
12. Сформулировать правила синтаксиса для исходного текста на ассемблере.
13. Сформулировать правила использования прямой и косвенной адресации.
14. Сформулировать формирование циклических вычислений в программе.
15. Как программно сформировать интервал времени?
16. Дать описание логических команд программы.
17. Дать описание арифметических команд программы.
18. Дать описание команд условных переходов в программе.
19. Дать описание команд циклического сдвига.
20. Как увидеть содержимое памяти программ и данных микроконтроллера?

4.6. **Контрольные вопросы**

1. Как используется стековая адресация?
2. Как обратиться к подпрограмме?
3. Что происходит в программном счетчике при обращении к подпрограмме и при возврате из подпрограммы?
4. Как выполняется команда CALL #mm?
5. Как выполняется команда RETURN?
6. Пояснить алгоритм и программу.
7. Пояснить смысл каждой команды.

5.**Лабораторная работа №5**. **Разработка алгоритмов и программ на ассемблере для микроконтроллера** MSP430

5.1.**Цель работы.**

Целью работы является разработка алгоритмов и программ на ассемблере для микроконтроллера MSP430 с использованием интегрированной среды CCSv4.2. Отладка программного обеспечения проекта может быть выполнена с использованием устройства eZ430RF2500.

5.2. **Разработка алгоритма**

Алгоритм должен начинаться *инициализационной частью,* в которой определяются режимы работы интерфейсных средств микроконтроллера и формируются начальные условия. Программное обеспечение микроконтроллеров предназначено для режима работы *реального 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. 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 5.1.

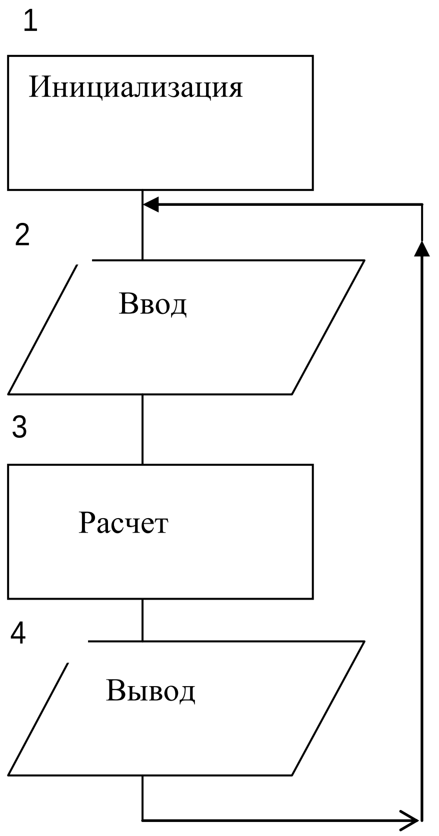


Рисунок 5.1. – Пример алгоритма

5.3. **Разработка программы**

Программа составляется на основании алгоритма и таблицы команд (см. Приложение) на языке ассемблера. Система команд является полностью ортогональной, то есть любую команду можно применять к любым из внутренних регистров, при различных способах адресации. Система команд допускает способы адресации, представленные в таблице 1.2. В таблице приняты обозначения:

Непосредственный режим адресации может быть применен только для первого операнда.

Составляя программу, следует размещать данные во внутренних регистрах микроконтроллера и памяти с учетом организации адресного пространства, представленной в таблице 1.1.

При составлении программы можно использовать операции с 16-разрядными словами и байтами. Байтовые команды должны иметь расширение мнемокода *.b*, например, *mov.b R11,R12.* Тогда операция выполняется над младшими байтами операндов.

Пример исходного текста программы на языке ассемблера показан далее. Программа предназначена для включения и отключения светодиодов 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 светодиодов операцией ***xor*** (исключающее ИЛИ).

```
;***************************************************************************
          .cdecls C,LIST,"msp430F2274.h"  ; головной файл устройства
;-,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,,
          .text                ;исходный текст
;............................................................................................
RESET     mov.w  #300h,SP                      ; инициализация стека
StopWDT   mov.w  #WDTPW+WDTHOLD,&WDTCTL        ; остановка таймера WDT
          bis.b  #003h,&                       ; разряды P1.0, P1.1– на вывод;
M         xor.b  #002h,&P1OUT                  ; переключение   P1.1
          mov.w  #0FFFh,R15                    ; запаздывание R15
L1        dec.w  R15                           ; декремент R15
          jnz    L1                            ; интервал истек?
          xor.b  #003h, &P1OUT                 ; переключение   P1.0, P1.1
          mov.w  #0FFFh,R15                    ; запаздывание R15
L2        dec.w  R15                           ; декремент R15
          jnz    L2
```

```
                jmp     M                       ; повторение цикла
;-------------------------------------------------------------------- ------
;       Interrupt Vectors
;-------------------------------------------------------------------- -------
        .sect   ".reset"                        ; MSP430 RESET Vector
        .short  RESET           ;
        .end
```

В системе команд отсутствует команда умножения, однако для перемножения двух операндов их достаточно поместить в регистры аппаратного умножителя, если таковой имеется в используемом микроконтроллере. После этого в регистрах *RESL0, RESH1* результата можно прочесть 32 разрядный результат. Младшее слово результата хранится в *RESL0,* старшее – в *RESH1*. Первый сомножитель может быть размещен в одном из 4-х регистров, которые определяют вид операции умножения.

MPY - умножение без знака,

MPYS - умножение со знаком,

MAC- умножение с накоплением без знака,

MACS - умножение с накоплением со знаком.

Второй сомножитель следует поместить в регистр *OP2* второго операнда. Примеры записи сомножителей для разных видов умножения следуют далее.

```
MOV #01234h,&MPY  ;     первый операнд    MOV.B #012h,&0130h ;
MOV #05678h,&OP2 ;      второй операнд    MOV.B #034h,&0138h ;

MOV #01234h,&MPYS ;                       MOV.B #012h,&0132h ;
MOV #05678h,&OP2 ;                        MOV.B #034h,&0138h ;

MOV #01234h,&MAC ;                        MOV.B #012h,&0134h ;
MOV #05678h,&OP2 ;                        MOV.B #034h,&0138h ;

MOV #01234h,&MACS ;                       MOV.B #012h,&0136h ; первый операнд MOV.B
MOV #05678h,&OP2 ;                        #034h,R5 ; временное хранение 2го операнда
                                          MOV R5,&OP2 ; второй операнд
```

Пример умножения имеет вид.

```
MOV #RESLO, R5 ;            адрес RESLO в R5 для косвенной адресации
MOV &OPER1, &MPY ;              первый операнд
    MOV &OPER2, &OP2   ;            2-й операнд
    NOP ;                           один цикл
    MOV @R5+, &xxx ;                запись RESLO в регистр xxx
    MOV @R5, &xxx ;                 запись RESHI
```

Здесь *xxx* может быть внутренним регистром микропроцессора или оперативной памяти, регистром порта или другого интерфейсного устройства. Пустая команда *NOP* предназначена для ожидания готовности результата умножения в течение одного цикла.

Если микроконтроллер не содержит аппаратного умножителя, умножение целых чисел a и b выполняется программно как многократное суммирование одного из сомножителей столько раз, каков другой сомножитель

$$c = a * b = \sum_{1}^{a} b = \sum_{1}^{b} a = a + a + \cdots + a.$$

Деление целых чисел c и a выполняется программно как многократное вычитание делителя из делимого.

$$b = \frac{c}{a} = (c - a) - a \cdots - a.$$

На каждом шаге вычитания анализируется знак результата. Вычитание повторяется, пока этот результат положителен; при получении отрицательного результата процесс прекращается. Подсчитывается количество вычитаний, которое и равно результату целочисленного деления.

5.4.Порядок выполнения работы

1. Составить алгоритм и программу в соответствии с заданным вариантом.
2. Открыть проект. В директории проекта содержится файл на языке ассемблера. В него записать составленный текст 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.

5.5. Варианты заданий

1. Составить алгоритм и программу последовательного через заданное время включения трех электродвигателей посредством контакторов.
2. Составить алгоритм и программу плавного пуска за заданное время двух электродвигателей: после разгона первого двигателя должен начаться разгон второго.
3. Составить алгоритм и программу реверсивного управления электродвигателем.
4. Составить алгоритм и программу включения привода при условии наличия сигналов от двух кнопок ПУСК и отключения от любой из трех кнопок СТОП.
5. Составить алгоритм и программу формирования разгона с постоянным ускорением до заданной скорости по сигналу ПУСК и торможения свободным выбегом по сигналу СТОП.
6. Составить алгоритм и программу формирования разгона с постоянным ускорением и затем торможения с постоянным замедлением.
7. Составить алгоритм и программу формирования последовательности импульсов с заданным периодом и с заданной скважностью.
8. Составить алгоритм и программу последовательного включения 3-х приводов через заданное время и их отключения в обратном порядке.

9. Составить алгоритм и программу включения одного из 2-х приводов, через заданное время его отключения и включения второго привода.

10. Составить алгоритм и программу вывода через параллельный порт последовательности значений: R12*2, R12*3, R12*4, R12*5,... через произвольный интервал времени.

11. Составить алгоритм и программу формирования последовательности импульсов с периодом 2 с.

12. Составить алгоритм и программу формирования последовательности прямоугольных импульсов линейно возрастающей амплитуды.

13. Составить алгоритм и программу формирования разгона с постоянным ускорением по сигналу пуска и торможения свободным выбегом по сигналу остановки.

14. Составить алгоритм и программу формирования разгона с постоянным ускорением, торможения и реверса.

15. Составить алгоритм и программу формирования импульса заданной в регистре R12 длительности и паузы заданной в регистре R11 длительности.

16. Составить алгоритм и программу расчета произведения содержимого регистров R11 и R13 и вывода через параллельный порт.

17. Составить алгоритм и программу управления возвратно-поступательным движением рабочего органа.

18. Составить алгоритм и программу управления пуском и остановкой рабочего органа.

19. Составить алгоритм и программу расчета суммы содержимого регистров R11, R12 и R13 и вывода суммы через параллельный порт.

20. Составить алгоритм и программу расчета разности содержимого регистров R11, R12 и вывода через параллельный порт.

5.6. **Содержание отчета**

1. Цель работы.
2. Условие задачи.
3. Расчетные выражения. Функциональная схема управления (для управления электроприводами)
4. Алгоритм и программа в виде таблицы с комментариями.
5. Содержимое регистров микроконтроллера во время и после выполнения программы.
6. Результат выполнения программы с комментариями.

5.7.**Контрольные вопросы**

1. Что записывают в инициализационной части программы?
2. Как использовать регистровую и косвенную адресацию?
3. Как выполняются команды над байтами и словами?
4. Как выполнить умножение?
5. Как формируют интервал времени?
6. Как увидеть содержимое внутренней памяти микроконтроллера?
7. Как увидеть содержимое регистров ядра микроконтроллера?

6. Лабораторная работа №6. Разработка алгоритмов и программ формирования функциональных зависимостей

6.1. Цель работы

Целью работы является разработка алгоритмов и программ формирования функциональных зависимостей для микроконтроллера MSP430 с использованием арифметических и логических операций. Функциональные зависимости используются в системах управления электроприводами для формирования сигналов управления (в нелинейных регуляторах, в частности, при необходимости ограничения тока и скорости электропривода допустимыми значениями), для координатных преобразований в системах векторного управления и в других случаях.

Отладка программного обеспечения проекта может быть выполнена с использованием устройства eZ430RF2500.

6.2. Формирование функциональных зависимостей

В системах управления электроприводами формирование функциональных зависимостей применяется в устройствах управления для формирования сигналов управления. Так, нелинейные функциональные зависимости с насыщением применяются для ограничения сигналов управления по модулю. Нелинейность с зоной нечувствительности применяется в обратных связях с отсечкой. Для координатных преобразований векторного управления необходимы тригонометрические функции (синус и косинус), которые так же являются нелинейными зависимостями. Наибольшее значение имеют функциональные преобразования $y = f(x)$, с одной независимой переменной $x(t)$, как показано на рисунке 6.1.

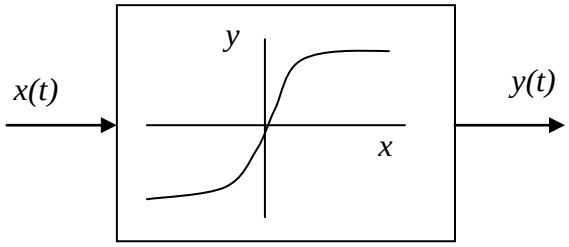


Рисунок 6.1 - Функциональное преобразование

Входной сигнал *x(t)* преобразования может вводится через устройства ввода или формироваться внутри микроконтроллера в процессе расчетов. Для входного сигнала, представленного в двоичном коде, известны возможные пределы его изменения

$$x \in (x_0, x_n).$$

Выходной сигнал *y(t)* преобразования так же представлен в двоичном коде и изменяется во времени, поскольку зависит от переменного входного сигнала. Произвольная функциональная зависимость $y = f(x)$, $x \in (x_0, x_1)$ может быть вычислена в режиме реального времени известными методами.

Шестнадцатиразрядный микроконтроллер допускает только целочисленные вычисления. Для организации вычислений с плавающей точкой необходимо применение специального программного обеспечения, что значительно увеличивает время выполнения программы и требует дополнительного объема оперативной памяти. Чтобы с достаточной точностью представить функциональную зависимость в 16-разрядном устройстве необходимо масштабирование. От выбранного масштаба зависит относительная погрешность при

целочисленных вычислениях. Например, функция $y = sinx$ при целочисленном представлении y и x обратится в нуль везде за исключением точек $x = (2k+1)\,\pi/2$, $(k = 0,1,2,3\cdots)$, в которых функция принимает значения ± 1. Целесообразно функцию представить в масштабе таким образом, чтобы относительная погрешность δ ее целочисленного представления не превышала допустимой величины

$$y = Y_m sin(ax)$$

Здесь Y_m, a масштабные коэффициенты. Например, если количество разрядов $m = 8$, $Y_m = 100,\ a = (\pi/2)/100 = 0{,}0157$, то расчетное выражение принимает вид

$$y = 100sin(0.0157x) \cong 100sin(x/60)$$

Здесь аргумент и функция могут принимать значения в пределах одного байта ($-128 \le x \le 127,\ -100 \le y \le 100$).

Кусочно-линейная аппроксимация предполагает приближенную замену исходной нелинейной зависимости кусочно-линейной зависимостью, как показано на рисунке 6.2. Для каждого линейного участка при $x \in (x_i, x_{i+1})$ будет справедливо расчетное выражение

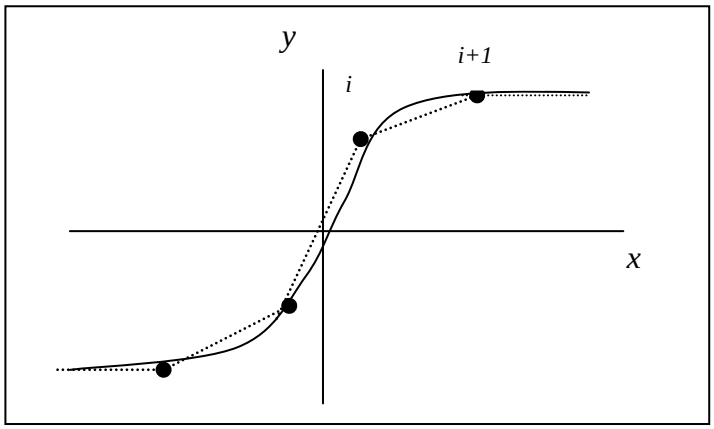


Рисунок 6.2 – Кусочно-линейная аппроксимация

$$y = y_i + k_i(x - x_i),$$

где коэффициент наклона линейного участка $k_i = (y_{i+1} - y_i)/(x_{i+1} - x_i)$, $(i = 0,1,\dots,n)$.

Разложение в ряд Тейлора позволяет получить приближенное выражение для расчета функции $y = f(x\ )$ с любой требуемой точностью, зависящей от количества учитываемых членов ряда Тейлора

$$y = f(x_0) + f^{(1)}_{x=x_0}(x - x_0) + \frac{1}{2!}f^{(2)}_{x=x_0}(x - x_0)^2 + \frac{1}{3!}f^{(3)}_{x=x_0}(x - x_0)^3 + \cdots + \frac{1}{k!}f^{(k)}_{x=x_0}(x - x_0)^k + \cdots$$

Здесь *($^{(k)}$)* означает производную по *x* порядка *k*. Производная определяется в заданной начальной точке x_0, и является известной числовой величиной.

Зная аналитическое выражение функции $y = f(x\ )$, благодаря ряду Тейлора можно рассчитать значения $y = f(x\ )$,, применяя операции умножения, деления и суммирования, что является возможным для микроконтроллера.

Для организации операции деления можно применить табличное задание гиперболической зависимости в соответствии с выражением $h = x_{max}/z$, где z - заданный целочисленный аргумент, h - обратная ему величина, $x_{max} = 2^{m-1} - 1$ масштабный коэффициент, m - количество разрядов слова данных.

Результаты разложения некоторых функций в ряд Тейлора представлены в таблице 6.1.

Разложение функций в ряд Тейлора

Таблица 6.1.

$y = Y_m sin(ax)$	$y = Y_m\left(ax - \frac{1}{3!}(ax)^3 + \frac{1}{5!}(ax)^5 - \frac{1}{7!}(ax)^7 + \cdots\right)$
$y = Y_m cos(ax)$	$y = Y_m\left(1 - \frac{1}{2!}(ax)^2 + \frac{1}{4!}(ax)^4 - \frac{1}{6!}(ax)^6 + \cdots\right)$
$y = Y_m tg(ax)$	$y = Y_m\left(ax + \frac{1}{3}(ax)^3 + \frac{2}{15}(ax)^5 + \frac{17}{315}(ax)^7 + \cdots\right)$
$y = Y_m ctg(ax)$	$y = Y_m\left(\frac{1}{ax} - \left(\frac{1}{3}(ax) \quad + \frac{1}{45}(ax)^3 - \frac{2}{945}(ax)^5 + \cdots\right)\right)$
$y = Y_m ln(1 \pm ax)$	$y = Y_m\left(\pm ax - \frac{1}{2}(ax)^2 \pm \frac{1}{3}(ax)^3 - \frac{1}{4}(ax)^4 + \cdots\right)$
$y = Y_m ln\left(\frac{1+ax}{1-ax}\right)$	$y = 2Y_m\left(ax + \frac{1}{3}(ax)^3 + \frac{1}{5}(ax)^5 + \frac{1}{7}(ax)^7 + \cdots\right)$
$y = Y_m ln(ax)$	$y = 2Y_m\left(\frac{1-ax}{1+ax} + \frac{1}{3}\left(\frac{1-ax}{1+ax}\right)^2 + \frac{1}{5}\left(\frac{1-ax}{1+ax}\right)^5 + \cdots\right), \quad x > 0$
$y = \frac{1}{1-x}$	$y = 1 + x + x^2 + x^3 + \cdots$
$y = \frac{1}{1+x}$	$y = 1 - x + x^2 - x^3 + \cdots$
$y = Y_m exp(ax)$	$y = Y_m\left(1 + ax + \frac{1}{2!}(ax)^2 + \frac{1}{3!}(ax)^3 + \frac{1}{4!}(ax)^4 + \cdots\right)$
$y = Y_m a^x$	$y = \left(1 + xlna + \frac{1}{2!}(xlna)^2 + \frac{1}{3!}(xlna)^3 + \frac{1}{4!}(xlna)^4 + \cdots\right)$

Табличное задание функции путем записи ее в ПЗУ или ОЗУ. Значения функции на заданном интервале изменения независимой переменной могут быть вычислены и записаны в память в виде массива, что позволит ускорить выполнение программы. Преимуществом данного способа является его универсальность, а недостатком – ограничения по точности представления функции, а так же по объему требуемой памяти.

При табличном задании аргумент применяется в качестве относительного адреса, а соответствующее значение функции записывается по этому адресу. Для получения необходимой точности аргумент и функцию представляют в масштабе: $x = m_x X \quad y = m_y Y$. Здесь $x, y -$ значения входной и выходной величины в двоичном коде, $X, Y -$ значения входной и выходной величины в физических единицах измерения, m_x, m_y - масштабные коэффициенты. Результат расчета выходной величины по входной величине записывают в таблицу 6.2.

Таблица 6.2.

Вход			Выход		
X	*x*		Y	*y*	
	десятичное	*h*		*десятичное*	*h*
0.0 … 1.0	0. … 127	0x00 … 0x7F	0.0 … 5.0	0.0 … 255	0x00 … 0xFF

Шестнадцатеричные значения входа и выхода, обозначенные *h*, используются для формирования адреса и данных массива значений *y* в памяти.

Составляя программу, следует размещать данные во внутренних регистрах микроконтроллера и памяти с учетом организации адресного пространства, представленной в таблице 1.1.

6.3. Загрузка проекта в устройство и запуск проекта на выполнение

Создание нового проекта можно выполнить, выбирая File → New → CCS Project, ввести имя проекта и нажать next, установить тип проекта для MSP430.

Следует дважды нажать next для открытия страницы настроек проекта (CCS Settings page). Выбрать вариант устройства, используемого в проекте MSP430x2xx. Если проект выполнен на ассемблере, его следует конфигурировать как ассемблерный. Для этого следует выбрать Configure as an assembly only project.

В созданный проект следует добавить файл исходного текста программы на ассемблере, который расположен в директории <D/MSP/**....**

Следует выполнить Project → Build Active Project.

Для взаимодействия прикладной программы с устройством выполняют Target → Debug Active Project. По этой команде стирается содержимое памяти целевого устройства, память устройства программируется, то есть в память записывается активный проект, устройство перезапускается.

Для запуска приложения следует выполнить: → Run (F8) или нажать кнопку Play на панели инструментов. Если между CCS и устройством нет связи, следует обратиться к документу SLAU278 на сайте www.ti.com.

Для запуска прикладной программы служит команда: Target → Run или кнопка ⮚

Для остановки прикладной программы служит кнопка □□. Команда окончания отладки: Target → II.

Команда File → Exit позволяет выйти из CCS.

6.4. Порядок выполнения работы

1. Составить алгоритм и программу в соответствии с заданным вариантом.
2. Открыть проект. В директории проекта содержится файл на языке ассемблера. В него записать составленный текст 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. Записать содержимое регистров, где формируются *x* и *y*.
7. По результатам расчета построить график *y(x)*.

6.5. Варианты заданий

Составить алгоритм и программу для функционального преобразования с использованием арифметических операций и табличного задания функций.

1. $y = x^3$;
2. $y = x^3 + ax^2 + 1$;
3. $y = x^3 + 1$;

4. $y = |x^3 + ax^2|$;
5. $y = \left(\left(x - sign(x)\right)^2 - 1\right) sign(x)$;
6. $y = 2\left(\left(x - sign(x)\right)^2 - 1\right) sign(x)$;
7. $y = x^2 sign(x)$;

Составить алгоритм и программу для функционального преобразования с использованием кусочно-линейного представления функций.

8. $y = max\left(x - 10, min(0, x + 10)\right)$;
9. $y = max\left(x - 1, min(0, x + 1)\right)$;
10. $y = min\left(10, max\left(-10, max\left(x - 1, min(0, x + 1)\right)\right)\right)$;
11. $y = x \quad sign(x)$;
12. $y = \left(\left(x + sign(x)\right)^2 - 1\right) sign(x)$;
13. $y = min\left(10, max(-10, x)\right)$;
14. $y = min\left(10, max\left(-10, (x - 1)\right)\right)$;
15. $y = tan\, x$;
16. $y = \tan^{-1} x$;
17. $y = \mathrm{Y}\sin ax$;
18. $y = a\sqrt{x}$;
19. $y = a\sqrt{1 + x^2} + 1$;
20. $y = a\left(1 - exp(-bx \quad)\right)$;

6.6. Содержание отчета

1. Цель работы
2. Условие задачи.
3. Расчетные выражения.
4. График функциональной зависимости (примерный вид).
5. Алгоритм и программа в виде таблицы с комментариями.
6. Результат выполнения программы в виде таблицы значений аргумента и функции в регистрах, либо в виде графика.

6.7. Контрольные вопросы

1. Как выполнить деление целых чисел?
2. Как использовать регистровую и косвенную адресацию?
3. Как выполнить умножение?
4. Как определить наибольшее из двух чисел?
5. Как формируют функциональные зависимости?
6. Как сохранить массив во внутренней памяти микроконтроллера?
7. Как организовать возведение переменной в степень?
8. Как увидеть содержимое внутренней памяти микроконтроллера?

7. Лабораторная работа №7. Запись в память функциональных зависимостей и вывод графиков

7.1. Цель работы.

Целью работы является использование программ формирования функциональных зависимостей для записи функциональных зависимостей в память микроконтроллера MSP430 и вывода графиков.

7.2. Адресное пространство и область хранения данных пользователя

Размер адресного пространства процессора определяется количеством разрядов программного счетчика, равным количеству разрядов шины адреса. Так, для 16 разрядного PC адресное пространство содержит 2^{16}-1=65 535 адресов, то есть могут быть доступны 64Кб памяти. Однако общий объем адресного пространства составляет 128 Кб, потому что все адреса четные, и адресуется всегда младший байт. Адреса интерфейсных (периферийных) устройств (портов ввода вывода, АЦП, таймеров) включаются в адресное пространство микроконтроллера. Структура адресного пространства представлена в таблице 1.1.

Следует обратить внимание на область *RAM 05FFh--0200h.* Именно в этой области могут сохраняться данные пользователя, то есть исходные данные, промежуточные результаты расчета и конечные результаты.

Для записи данных в память удобно использовать косвенную адресацию. В мнемокоде ассемблера косвенная адресация регистра-источника и регистра-приемника отображается разными способами, что отражено в таблице 7.2.

Если массив данных сохранен в памяти, то возможно его представление в виде графика.

Область адресов 0000-000F отводится внутренним регистрам процессора, специальным и общего назначения. По адресам 0000-0003 расположены 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).

7.3. Табличное задание функции путем записи ее в ПЗУ или ОЗУ

Значения функции на заданном интервале изменения независимой переменной могут быть вычислены и записаны в память в виде массива, что позволит ускорить выполнение программы. Преимуществом данного способа является его универсальность, а недостатком – ограничения по точности представления функции, а так же по объему требуемой памяти.

При табличном задании аргумент применяется в качестве относительного адреса, а соответствующее значение функции записывается по этому адресу. Для получения необходимой точности аргумент и функцию представляют в масштабе: $x = m_x X \quad y = m_y Y$. Здесь $x, y -$ значения входной и выходной величины в двоичном коде, $X, Y -$ значения входной и выходной величины в физических единицах измерения, m_x, m_y - масштабные коэффициенты. Результат расчета выходной величины по входной величине записывают в таблицу 7.1.

Таблица 7.1.

аргумент			функция		
X	*x*		Y	*y*	
	десятичное	*h*		*десятичное*	*h*
0.0 … 1.0	0. … 127	0x00 … 0x7F	0.0 … 5.0	0.0 … 255	0x00 … 0xFF

Шестнадцатеричные значения, обозначенные *h*, используются для формирования адреса и данных массива значений *y* в памяти.

Для косвенной адресации операнда-приемника следует использовать регистры R13 – R15.

Составляя программу, следует размещать данные во внутренних регистрах микроконтроллера и памяти с учетом организации адресного пространства, представленной в таблице 1.1.

В системе команд отсутствует команда умножения, однако для перемножения двух операндов их достаточно поместить в регистры аппаратного умножителя. После этого в регистрах *RESL0, RESH1* результата можно прочесть 32 разрядный результат. Младшее слово результата хранится в *RESL0,* старшее – в *RESH1*. Первый сомножитель может быть размещен в одном из 4-х регистров, которые определяют вид операции умножения.

MPY - умножение без знака, *MPYS* - умножение со знаком, *MAC*- умножение с накоплением без знака, *MACS* - умножение с накоплением со знаком. Второй сомножитель следует поместить в регистр *OP2* второго операнда. Умножение двух целых чисел может быть выполнено как многократное суммирование..

Деление целых чисел может быть заменено циклическим вычитанием. Используя умножение и деление, гладкую функцию всегда можно вычислить, например, представляя ее рядом Тейлора.

7.4.**Вывод графика**

График функции можно получить, если записать значения функции в оперативную память, начиная с адреса 0200h. После этого можно использовать программное средство Graph интегрированной среды, задавая начальный адрес массива, длину слова данных, размер массива (количество точек графика).

Для этого после выполнения расчета по программе, которая предусматривает запись массива чисел в память, следует выполнить: Tool***s*** → **Graph** → **Single** Time, нажать левую кнопку мыши на слове **Single** Time. В результате откроется окно задания параметров графика.

В окне задания параметров графика многие значения принимаются по умолчанию, размер буфера (Acquisition Buffer Size) следует задать 20.

Для параметра Dsp Data Type следует задать 16 bit для 16 разрядных данных, 8 bit для восьмиразрядных данных.

Начальный адрес Start Address задается 0x0200 и более.

Размер массива выводимых данных Display Data Size указывается в соответствии с размером массива, формируемым программой.

Свойства графика можно сохранить, нажав кнопку Export. Необходимо, чтобы имя файла имело расширение .graphProp. Доступ к сохраненным свойствам выполняется через кнопку Import.

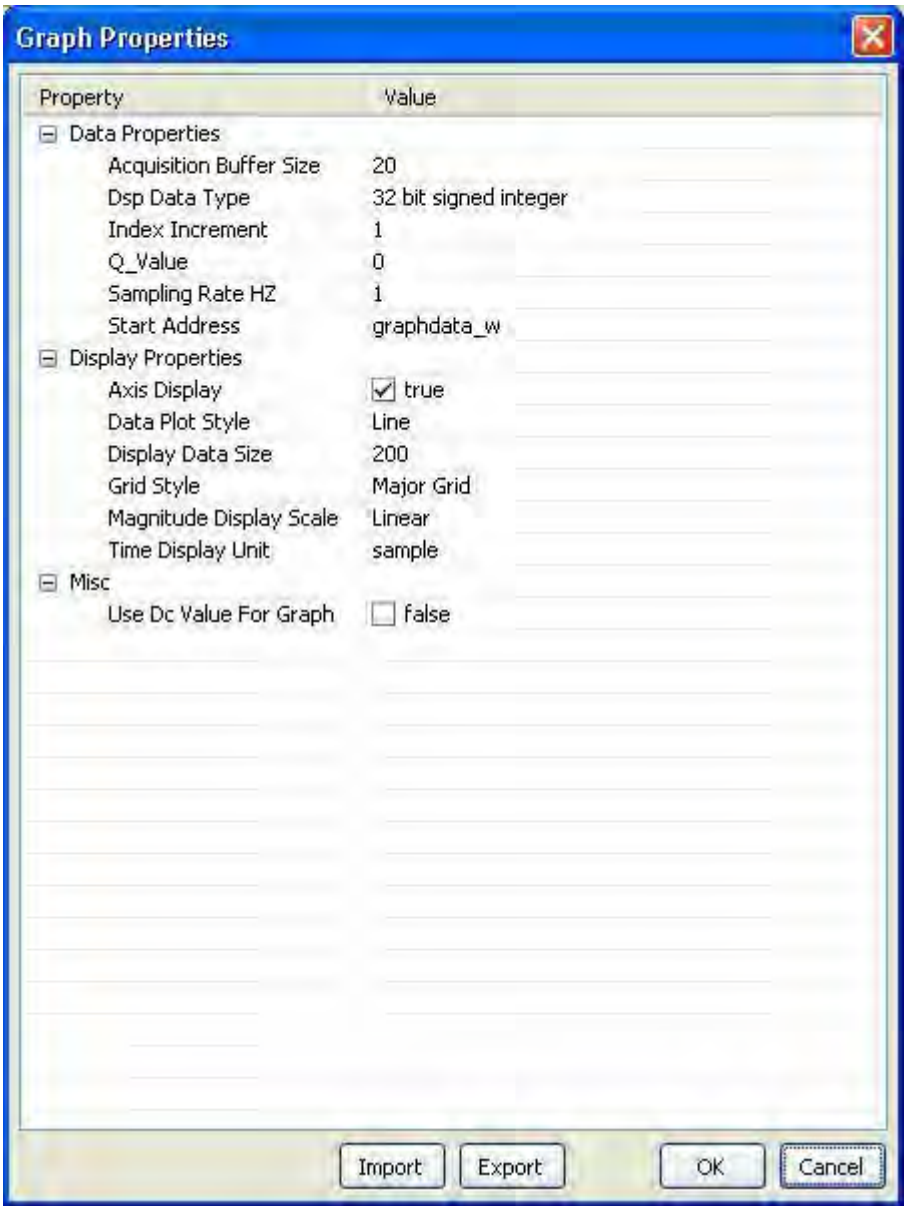


Рисунок 7.2 – Окно задания параметров графика

Следует установить параметры:

- Размер буфера данных Acquisition Buffer Siz;
- Тип данных Dsp Data Type (8 бит, 16 бит, 32 бит) должен соответствовать типу, рассчитываемому программой;
- Начальный адрес Start Address (значение адреса или имя массива)
- Размер выводимых данных Display Data size.

После нажатия OK открывается окно графика, в котором при правильной настройке свойств графика можно увидеть график функции.

7.5. Порядок выполнения работы

1. В соответствии с заданным в лабораторной работе 6 вариантом составить алгоритм и программу, записать в микроконтроллер.
2. Запустить программу на выполнение (Target →Run).

3. Записать результаты выполнения программы. Настроить окно проекта так, чтобы были видны внутренние регистры микроконтроллера, используемые в программе, (View→Debug, View→Registers) а так же область памяти, где программа сохраняет результаты расчета (View→Memory).

4. Убедиться, что результаты расчета сохранены в памяти.

5. Вывести график рассчитанной зависимости и представить график в отчете.

7.6.**Варианты заданий**

Составить алгоритм и программу для функционального преобразования с использованием арифметических операций и табличного задания функций.

1. $y = x^3$;
2. $y = x^3 + ax^2 + 1$;
3. $y = x^3 + 1$;
4. $y = |x^3 + ax^2|$;
5. $y = \left(\left(x - sign(x)\right)^2 - 1\right) sign(x)$;
6. $y = 2\left(\left(x - sign(x)\right)^2 - 1\right) sign(x)$;
7. $y = x^2 sign(x)$;

Составить алгоритм и программу для функционального преобразования с использованием кусочно-линейного представления функций.

8. $y = max\left(x - 10, min(0, x + 10)\right)$;
9. $y = max\left(x - 1, min(0, x + 1)\right)$;
10. $y = min\Big(10, max\Big(-10, max\big(x - 1, min(0, x + 1)\big)\Big)\Big)$;
11. $y = x \quad sign(x)$;
12. $y = \left(\left(x + sign(x)\right)^2 - 1\right) sign(x)$;
13. $y = min\left(10, max(-10, x)\right)$;
14. $y = min\left(10, max\left(-10, (x - 1)\right)\right)$;
15. $y = tan\, x$;
16. $y = \tan^{-1} x$;
17. $y = \mathrm{Ysin}\, ax$;
18. $y = a\sqrt{x}$;
19. $y = a\sqrt{1 + x^2} + 1$;
20. $y = a\left(1 - exp(-bx \quad)\right)$;

7.7.**Содержание отчета**

1. Цель работы
2. Условие задачи. Расчетные выражения.
3. Алгоритм и программа.
4. Содержимое области памяти, где записан массив данных.
5. Параметры настройки для вывода графика.
6. График функциональной зависимости.

7.8. **Контрольные вопросы**

1. Как записать данные в память?
2. Как использовать регистровую и косвенную адресацию?
3. Как выполнить умножение и деление?
4. Как представить функцию массивом данных?
5. Как записать функцию в память?
6. Как сохранить массив во внутренней памяти микроконтроллера?
7. Как содержимое внутренней памяти микроконтроллера представит в виде графика?
8. Как управлять видом графического окна?

7.9.**Порядок выполнения работы**

1. В соответствии с заданным в лабораторной работе 6 вариантом составить алгоритм и программу, записать в микроконтроллер.
2. Запустить программу на выполнение (Target →Run).
3. Записать результаты выполнения программы. Настроить окно проекта так, чтобы были видны внутренние регистры микроконтроллера, используемые в программе, (View→Debug, View→Registers) а так же область памяти, где программа сохраняет результаты расчета (View→Memory).
4. Убедиться, что результаты расчета сохранены в памяти.
5. Вывести график рассчитанной зависимости и представить график в отчете.

7.10. **Варианты заданий**

Составить алгоритм и программу для функционального преобразования с использованием арифметических операций и табличного задания функций.

1. $y = x^3;$
2. $y = x^3 + ax^2 + 1;$
3. $y = x^3 + 1;$
4. $y = |x^3 + ax^2|;$
5. $y = \left(\left(x - sign(x)\right)^2 - 1\right) sign(x);$
6. $y = 2\left(\left(x - sign(x)\right)^2 - 1\right) sign(x);$
7. $y = x^2 sign(x);$

Составить алгоритм и программу для функционального преобразования с использованием кусочно-линейного представления функций.

8. $y = max\left(x - 10, min(0, x + 10)\right);$
9. $y = max\left(x - 1, min(0, x + 1)\right);$
10. $y = min\left(10, max\left(-10, max\left(x - 1, min(0, x + 1)\right)\right)\right);$
11. $y = x \quad sign(x);$
12. $y = \left(\left(x + sign(x)\right)^2 - 1\right) sign(x);$
13. $y = min\left(10, max(-10, x)\right) \;;$
14. $y = min\left(10, max\left(-10, (x - 1)\right)\right) \;;$
15. $y = tan\, x;$
16. $y = \tan^{-1} x;$
17. $y = \mathrm{Y}\sin ax;$
18. $y = a\sqrt{x};$
19. $y = a\sqrt{1 + x^2} + 1;$
20. $y = a\left(1 - exp\left(-bx \quad \right)\right);$

7.11. **Содержание отчета**

1. Цель работы
2. Условие задачи. Расчетные выражения.
3. Алгоритм и программа.
4. Содержимое области памяти, где записан массив данных.
5. Параметры настройки для вывода графика.
6. График функциональной зависимости.

7.12.**Контрольные вопросы**

1. Как записать данные в память?
2. Как использовать регистровую и косвенную адресацию?
3. Как выполнить умножение и деление?
4. Как представить функцию массивом данных?

5. Как записать функцию в память?
6. Как сохранить массив во внутренней памяти микроконтроллера?
7. Как содержимое внутренней памяти микроконтроллера представит в виде графика?
8. Как управлять видом графического окна?

8.**Лабораторная работа №8**. **Формирование интервалов времени с использованием программируемого таймера**

8.1. **Цель работы**

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

8.2.**Разработка алгоритма**

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

Программное обеспечение микроконтроллеров предназначено для режима работы *реального 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 (событиями на входе микроконтроллера) формируется реакция устройства на них в виде выходных сигналов (событий), в нужные моменты времени. Для этого либо алгоритм должен иметь циклическую часть, либо можно использовать прерывания через равные интервалы времени от таймера.

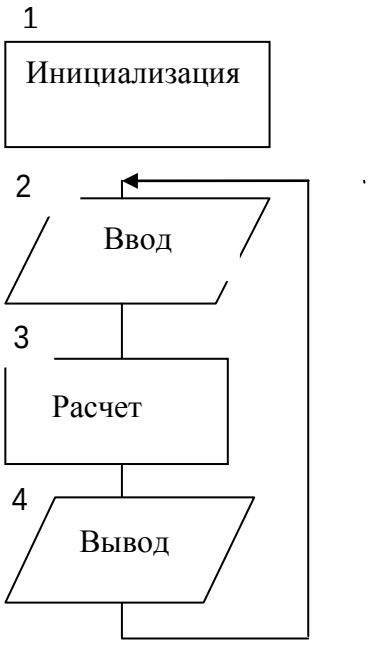


Рисунок 8,1 – Алгоритмы главной программы (1) и подпрограммы прерываний (2,3,4).

Формирование интервала времени может выполняться аппаратным способом, с использованием программируемого таймера. В этом случае инициализационная часть должна содержать

- инициализацию внешних выводов параллельных портов для ввода и вывода событий;
- инициализацию таймера для заданного режима работы;
- инициализацию стека, если необходимы подпрограммы;
- установку битов регистра состояния, если необходимо управление работой центрального процессорного устройства в режиме прерываний.

В циклической части формируются операции ввода и вывода сигналов, а так же 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.

8.3.**Таймер ТА 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 так же входят в состав микроконтроллеров. Обычно в составе микроконтроллера может быть 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 ТА 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

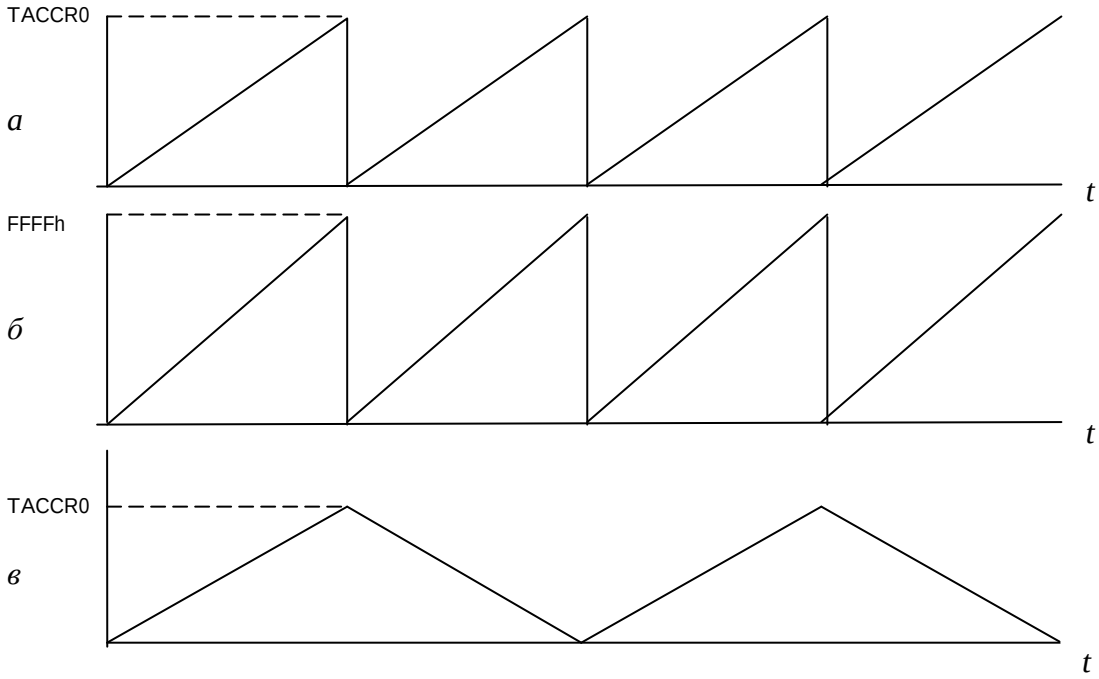


Рисунок 8.1 – Режимы работы таймера, *а* –прямой счет, *б* – непрерывный счет, *в* – реверсивный счет.

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 8.1. Регистр TACTL управления таймера TA представлен в таблице 8.2.

Таблица 8.1 - **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

Таблица 8.2 - **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 TACTLRx управления захватом-сравнением представлен в таблице 8.3. Его 9 разряд не используется.

Таблица 8.3 - **Регистр** TACCTL x **управления захватом-сравнением таймера TA**

15-14	13-12	11	10	9	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; 11 - захват по обоим фронтам;	выбор входа захвата-сравн. 00 – CCIxA; 01 – CCIxB; 10 - GND; 11- Vcc;	синхронизация захвата: 0–асинхр. захват; 1-синхр. захват;	Синхр. вход захвата-сравнения. Входной сигнал фиксируется поEQUx и может быть считан;		Режим . захвата-сравнения; 1–захват; 0–сравнение;	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 – было прерывание.

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

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.
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 TA, представлены на рисунке 8.2 и выделены.

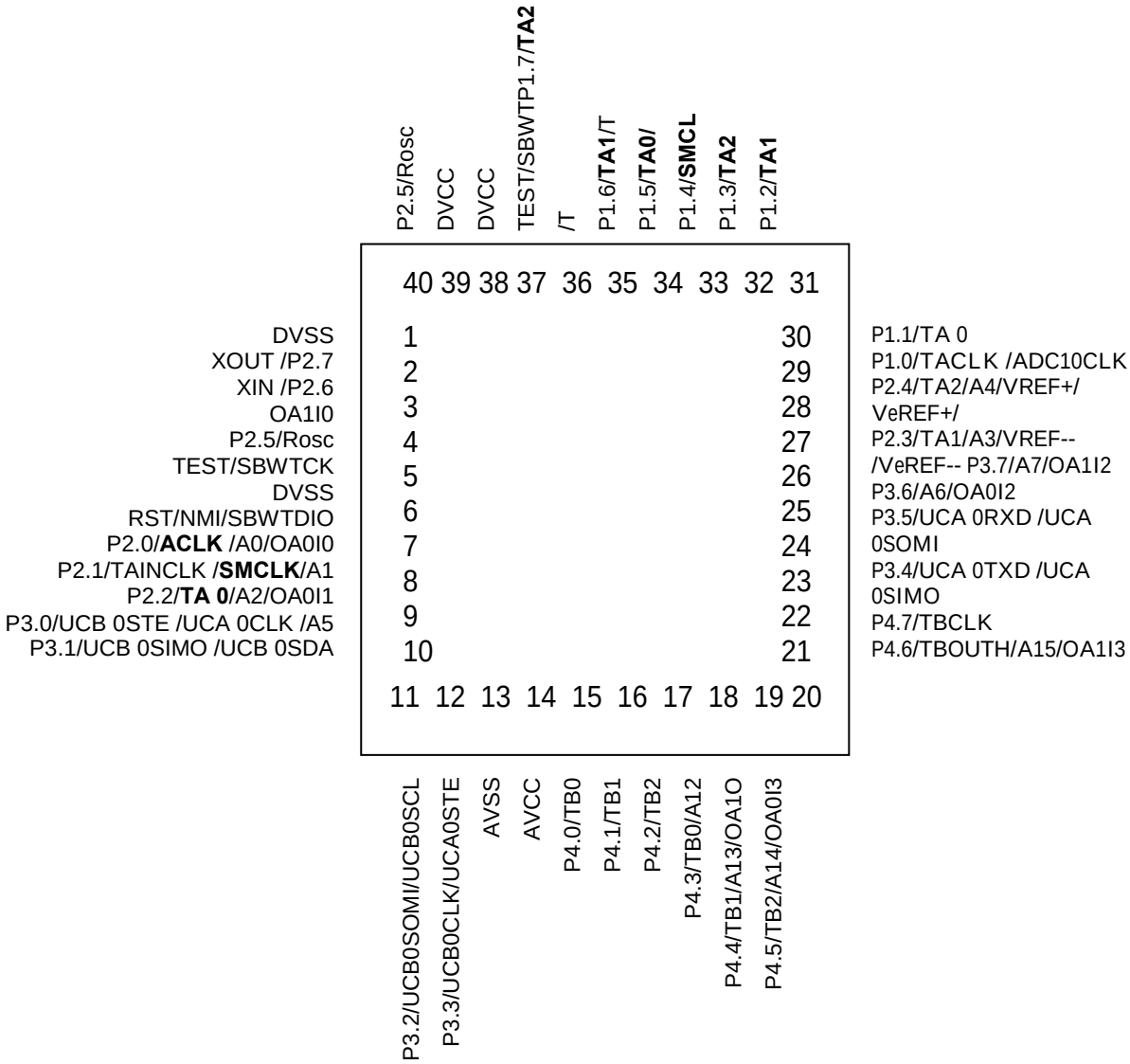


Рисунок 8.2 - Внешние выводы, используемые таймером TA

8.4.**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, contmode
```

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 ***xor*** (исключающее ИЛИ).

```
;*******************************************************************
 .cdecls C,LIST,  "msp430x22x4.h"

;-------------------------------------------------------------------------------------------------------
        .text                                ;
;-------------------------------------------------------------------------------------------------------
RESET           mov.w   #300h,SP                    ; инициализация стека
StopWDT         mov.w   #WDTPW+WDTHOLD,&WDTCTL  ; остановка  WDT
                bis.b   #001h,&P1DIR               ; инициализация бита P1.0 на вывод
                mov.w   #CCIE,&TACCTL0         ; TACCTL0, разрешение прерываний
                mov.w   #50000,&TACCR0         ; интервал времени
                mov.w   #TASSEL_2+MC_2,&TACTL   ; сигнал SMCLK –иниц. таймера,
M               bis.w   #CPUOFF+GIE,SR          ; CPU откл.,  прерывания разрешены
                nop                                 ;
;-------------------------------------------------------------------------------------------------------
TA0_ISR;    переключение  P1.0
;-------------------------------------------------------------------------
                xor.b   #001h,&P1OUT            ; переключение  P1.0
                add.w   #50000,&TACCR0         ; прибавить к  TACCR0
                reti                                ;  возврат
                        ;
;-------------------------------------------------------------------------
;       Векторы прерывания
;-------------------------------------------------------------------------
        .sect   ".reset"            ; MSP430 RESET Vector
        .short  RESET                   ;
        .sect   ".int09"            ; Timer_A0 Vector
        .short  TA0_ISR             ;
              .end
```

8.5.**Порядок выполнения работы**

1. Составить алгоритм и программу в соответствии с заданным вариантом.
2. Открыть проект. В проект добавить файл msp430x22x4_ta_01.asm исходного текста программы на языке ассемблера.
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 в пошаговом режиме. Записать содержимое регистра SR и регистров управления таймером TACTL, TACCLx до и после инициализации.
7. . Записать содержимое регистров таймера до и после возникновения прерывания.
8. Внести изменения в исходный текст msp430x22x4_ta_01.asm, чтобы программа соответствовала заданному варианту.
9. Выполнить п.3-7 для измененной программы.
10. Остановить выполнение программы.

11. Изменить выдержку времени, запустить программу и посмотреть, как это повлияет на работу устройства.

8.6.**Варианты заданий**

1. Составить алгоритм программы msp430x22x4_ta_01.asm. Рассчитать период и частоту переключений. Увеличить период переключений в два раза.
2. Составить полное описание режима работы таймера и указать значения всех битов регистров управления TACTL, TACCLx. Составить программы для организации работы таймера в каждом из трех режимов.
3. Составить алгоритм и программу для переключений вывода P1.1 порта P1.
4. Составить алгоритм программы msp430x22x4_ta_01.asm. Уменьшить период переключений в два раза.
5. Составить алгоритм и программу для переключений выводов P1.0, P1.1 порта P1 с заданной частотой.
6. Составить алгоритм и программу для переключений вывода P1.0 порта P1 с частотой 16 Гц.
7. Составить алгоритм и программу для переключений выводов P1.0, P1.1 порта P1 с частотой 32 Гц.
8. Составить алгоритм и программу формирования последовательности импульсов частотой 200 Гц.
9. Составить алгоритм и программу формирования последовательности импульсов частотой 160 Гц.
10. Составить алгоритм и программу формирования последовательности импульсов с заданным периодом и со скважностью 0,5.
11. Составить алгоритм и программу последовательного включения таймером 2х силовых ключей и их отключения в обратной последовательности.
12. Составить алгоритм и программу вывода двух сигналов заданной частоты через P1.0 и P1.1 в противофазе.
13. Составить алгоритм и программу вывода двух сигналов заданной частоты через P1.0 и P1.1 со сдвигом на четверть периода.
14. Составить алгоритм и программу формирования последовательности импульсов частотой 40 Гц на выходе P2.5.
15. Составить алгоритм и программу формирования последовательности прямоугольных импульсов на выходе P2.1.
16. Сформировать управляющее слово для реверсивного счета и применить его в программе.
17. Составить алгоритм и программу формирования импульсов на выходе P1.0 в режиме непрерывного счета.
18. Составить алгоритм и программу формирования импульсов на выходе P1.0 в режиме прямого счета.
19. Составить алгоритм и программу формирования на P1.0 сигнала частоты 40 Гц, а на P1.1 – 80 Гц.
20. Составить алгоритм и программу формирования сигнала частотой 8 КГц.

8.7. **Содержание отчета**

1. Цель работы.
2. Условие задачи.
3. Алгоритм и программа в виде таблицы с комментариями.
4. Содержимое регистров микроконтроллера до, во время и после выполнения программы, до и после прерывания.

5. Результат выполнения программы в виде таблицы со значениями, записанными в регистрах.

8.8.**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?

9.**Лабораторная работа №9. Ввод сигналов с использованием аналого-цифрового преобразователя (АЦП) и программируемого таймера TA**

9.1. **Цель работы**

Аналого-цифровой преобразователь ADC10 предназначен для преобразования изменяющейся во времени аналоговой величины на входе АЦП в двоичный код. Преобразователь ADC10 в результате преобразования формирует 10 –разрядные двоичные значения, пропорциональные входной величине. Для запуска аналого-цифрового преобразования может быть использован таймер **TA**.

Целью работы является разработка алгоритмов и программ на ассемблере для микроконтроллера MSP430 с использованием для ввода сигналов АЦП ADC10 и таймера TA.

9.2.**Таймер TA микроконтроллера** MSP430

Таймер TA микроконтроллера MSP430 при аналого-цифровом преобразовании может быть использован как источник сигнала запуска преобразования. Аналого-цифровое преобразование занимает определенное время. Таймер TA может быть настроен на формирование периодических сигналов запуска преобразования аналоговой величины в двоичный код. Период сигналов запуска должен быть не менее времени аналого-цифрового преобразования.

Таймер TA микроконтроллера MSP430 содержит регистр управления TACTL, 16 разрядный регистр-счетчик TAR и три регистра, TACCR0 захвата-сравнения. Для инициализации таймера в регистр управления TACTL таймера и в регистр управления TACCTL0 модуля захвата-сравнения следует записать управляющие слова для формирования интервала времени, необходимого для аналого-цифрового преобразования.

Для того, чтобы с помощью таймера генерировать периодическую последовательность сигналов, можно использовать режим *сравнения*, то есть вывода событий.

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. В регистр TACCR0 захвата-сравнения следует записать значение, пропорциональное требуемому времени периода. Коэффициент пропорциональности зависит от частоты тактового сигнала, поступающего на таймер.

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 делитель таймера, коэффициент деления которого (1,2,4,8) определяется при инициализации таймера.

Регистры таймера ТА, их назначение и адреса, необходимые для инициализации таймера, представлены в таблицах 8.1, 8.2, 8.3.

9.3. **Аналого-цифровой преобразователь** ADC10 **микроконтроллера** MSP430

Модуль ADC10, функциональная схема которого представлена на рисунке 9.1 содержит 10 - разрядное ядро с регистром последовательного приближения, блок управления выборкой, генератор опорного напряжения 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 ИОН может быть настроен на 1.5 В или 2.5 В. Следовательно, входной аналоговый сигнал не должен по модулю превосходить соответственно 1.5 В или 2.5 В. Возможен внешний источник опорного сигнала. На модуль ADC10 могут быть поданы до 8 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, управляемые блоком БУВ управления выборкой.

Однократный одноканальный режим– однократное преобразование одного входного сигнала;

Однократный последовательный– режим однократное последовательное преобразование каждого из входных сигналов (по каждому каналу);

Циклический одноканальный режим – циклически повторяется с заданной частотой преобразование по одному каналу (одного входа);

Циклический последовательный режим - циклически повторяется с заданной частотой преобразование по всем каналам.

Для настройки модуля ADC10 применяются регистры управления ADC10CTL0, ADC10CTL1, структура которых представлена в таблицах 9.1, 9.2. Внешние выводы ADC10 на рисунке 9.2 выделены. Это входы A**0…**A7 аналоговых сигналов, общая точка аналоговых сигналов AVSS, подключение источника питания аналоговых сигналов AVCC, внешний тактовый сигнал ADC10CLK для преобразователя ADC10.

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 В, а при Ref2_5V=0 внутреннее опорное напряжение равно 1,5 В. Это напряжение может быть использовано внутри микросхемы или выведено через V_{REF+} при REFOUT=1. На схеме выделены выходы VREF+ и VeREF+ опорных сигналов Уровни V_+ и V_- могут задаваться внешними источниками, подключаемыми к **A3** и **A4** соответственно (рисунок 9.1).

Отключение буфера вывода порта исключает появление паразитного тока и, таким образом, уменьшает энергопотребление микросхемы. Биты ADC10AEx позволяют выполнить отключение буфера вывода.

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.

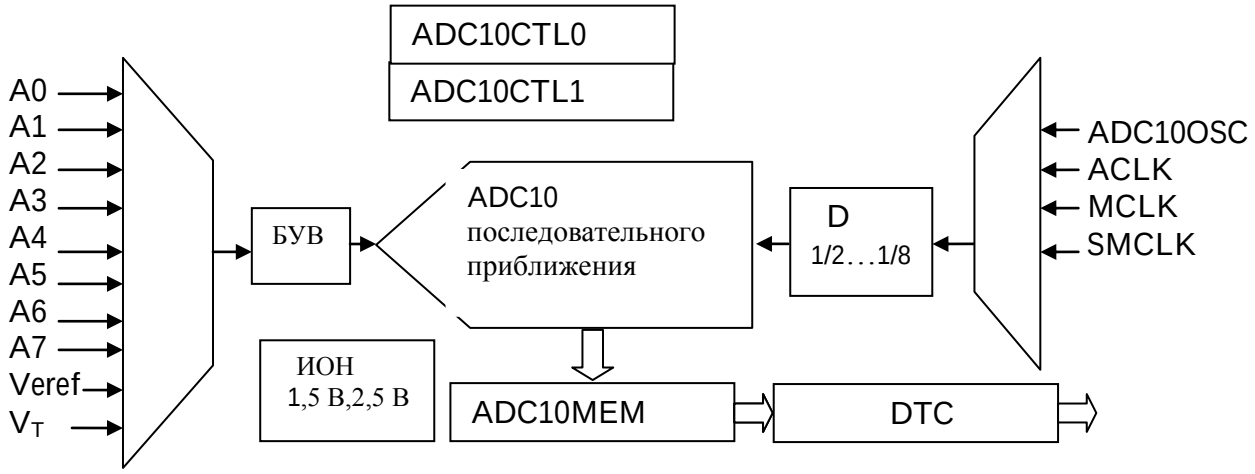


Рисунок 9.1 – Функциональная схема ADC10.

9.4. **Построение программы с использованием аналого-цифрового преобразователя** ADC10

Пример исходного текста программы msp430x22x4_adc10_03.asm на языке ассемблера показан далее. Программа предназначена для выполнения аналого-цифрового преобразования сигнала датчика температуры, встроенного в микроконтроллер и вычисления градиента температуры. Датчик температуры имеет на выходе сигнал напряжения, пропорциональный температуре, и опрашивается каждые 120 мс. Разность температур сравнивается с заданной фиксированной величиной. Если разность температур больше 2^oC, то на вывод P1.0 выводится сигнал, равный 1.

Таблица 9.1. **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 00 4 такта 01 8 тактов 10 16 тактов 11 64 такта	Скорость преобразов. 0 до $2*10^5$ с 1 до $50*10^3$ с	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

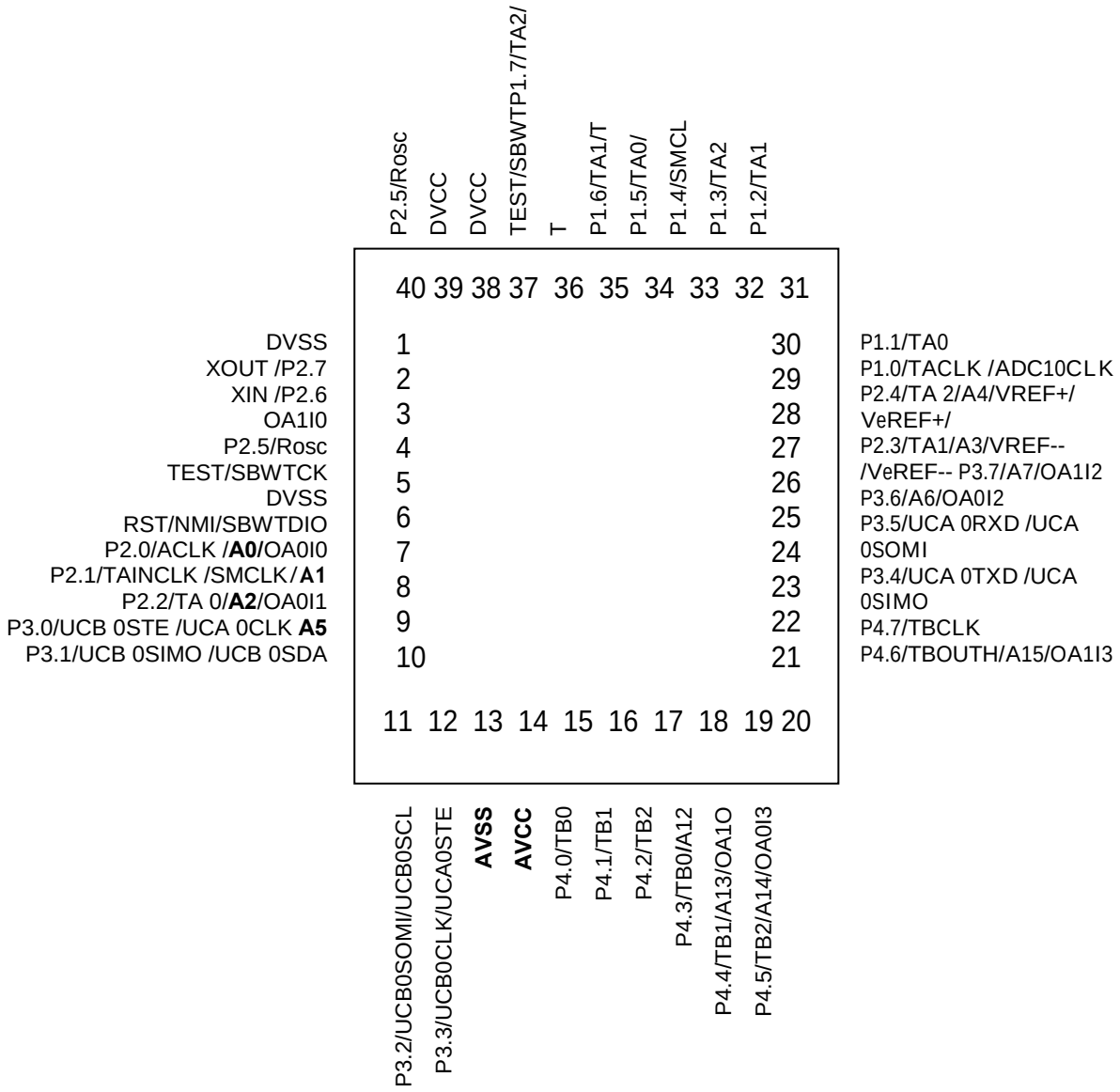


Рисунок 9.2 – Внешние выводы, используемые ADC10.

Таблица 9.2. **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+} 1001 V_{REF-}/ Ve_{REF-} 1010 датчик температуры	Источник сигнала запуска 00 ADC10SC 01 выв.1 TA 10 выв.2 TA	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

ADC10 работает в циклическом одноканальном режиме. Источником сигнала запуска преобразования является таймер **TA**.

Перед началом исходного текста имеются две директивы ассемблера

```
.cdecls C,LIST,  "msp430x22x4.h"

DeltaOn .set  3
```

Первая директива указывает на тип микроконтроллера, что определяет адреса периферийных устройств для компиляции исходного текста программы. Вторая директива ассемблера задает переменной с именем DeltaOn значение, равное 3.

Программа содержит инициализационную часть, где задается адрес вершины стека, остановлен сторожевой таймер, выполнена инициализация таймера **TA** и ADC10, настроен на вывод один бит порта **P1**. В регистр TACCR**0** записывается значение, пропорциональное интервалу времени, который необходимо сформировать.

Циклическая часть программы начинается с метки ADC_Wait. Если первое преобразование произошло, его результат записывается в R4. Циклическая часть оканчивается переходом в режим низкого энергопотребления и разрешением прерываний. После этого микроконтроллер должен работать в режимах прерываний от таймера и АЦП. Подпрограммы прерываний следуют после текста основной программы.

```
;-------------------------------------------------------------------------------
 .cdecls C,LIST,  "msp430x22x4.h"                ; головной файл (адреса периф. устройств)

DeltaOn .set  3                          ;  задание значения 3 переменной  DeltaOn
;-------------------------------------------------------------------------------
        .text                                    ; Сброс
;-------------------------------------------------------------------------------
```

```
RESET    mov.w  #300h,SP                          ; инициализация SP
         mov.w  #WDTPW+WDTHOLD,&WDTCTL            ; стоп WDT
                                                  ; иницивлизация ADC10
         mov.w  #ADC10DIV_3+INCH_10+SHS_1+CONSEQ_2,&ADC10CTL1
         mov.w  #SREF_1+ADC10SHT_3+REF2_5V+ADC10IE+REFON+ADC10ON,&ADC10CTL0
         mov.w  #30,&TACCR0                       ; Delay to allow Ref to settle
         bis.w  #CCIE,&TACCTL0                    ; режим сравнения, разрешение прерывания
         mov.w  #TACLR+MC_1+TASSEL_2,&TACTL       ; инициализация TA, SMCLK
         bis.w  #LPM0+GIE,SR                      ; режим LPM0 малого потребления
                                                  ; энергии ,разрешение прерывания TA
         bic.w  #CCIE,&TACCTL0                    ; запрещение прерывания TA
         bis.w  #ENC,&ADC10CTL0       ;
         mov.w  #OUTMOD_4,&TACCTL1                ; перекл.  по сигналу EQU1 (TAR = 0)
         mov.w  #TASSEL_2+MC_2,&TACTL             ; SMCLK, режим счета
W        bit.w  #ADC10IFG,&ADC10CTL0              ; первое преобразование?
         jnc    W                                 ; ожидание
         mov.w  &ADC10MEM,R4                      ; чтение 1-го значения ADC
         add.w  #DeltaOn, R4          ;
         clr.b  &P1OUT                            ; сброс P1OUT
         bis.b  #01h,&P1DIR                       ; настройка P1.0 на вывод
                        ;
M        bis.w  #LPM0+GIE,SR                      ; вход в LPM0, разрешение прерываний
         nop                                      ; только для debug
;-------------------------------------------------------------------------------------------------------------------
TA0_ISR;   ISR for TACCR0
;-------------------------------------------------------------------------------------------------------------------
         clr.w  &TACTL              ; сброс регистра управления TA
         bic.w  #LPM0,0(SP)          ; выход из режима LPM0 по возврату
         reti                       ;
;-------------------------------------------------------------------------------------------------------------------
ADC10_ISR;
;-------------------------------------------------------------------------------------------------------------------
         cmp.w  R4,&ADC10MEM          ; ADC10MEM = A10 > R4
         jlo    ADC_ISR_1             ; Again
         bis.b  #01h,&P1OUT          ; P1.0 = 1
         reti                       ;
ADC_ISR_1  bic.b  #01h,&P1OUT          ; P1.0 = 0
       reti                         ;
;-------------------------------------------------------------------------------------------------------------------
;        Interrupt Vectors
;-------------------------------------------------------------------------------------------------------------------
       .sect   ".reset"             ; MSP430 RESET Vector
       .short  RESET                ;
       .sect   ".int05"             ; ADC10 Vector
       .short  ADC10_ISR            ;
       .sect   ".int09"             ; Timer_A0 Vector
       .short  TA0_ISR              ;
       .end
```

9.5. **Порядок выполнения работы**

1. Составить алгоритм и программу в соответствии с заданным вариантом.
2. Открыть проект. В проект добавить файл исходного текста программы msp430x22x4_adc10_03.asm на языке ассемблера.
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 в пошаговом режиме, затем в непрерывном режиме. Записать содержимое регистра SR и регистров управления ADC10 и таймером TACTL, TACCLx до и после инициализации.
7. Убедиться, что при изменении температуры содержимое ADC10MEM изменяется.
8. Записать содержимое регистров ADC10 до и после возникновения прерывания.
9. Внести изменения в исходный текст msp430x22x4_adc10_03.asm, чтобы программа соответствовала заданному варианту.
10. Выполнить п.3-7 для измененной программы.
11. Остановить выполнение программы.
12. Изменить выдержку времени, запустить программу и посмотреть, как это повлияет на работу устройства.

9.6. **Варианты заданий**

1. Составить алгоритм и программу переключений вывода P1.0 порта P1 при достижении заданной температуры.
2. Составить алгоритм и программу для включения вентилятора при достижении заданной температуры.
3. Составить алгоритм и программу для вывода значений температуры в двоичном коде через 240 мс.
4. Дать подробное описание функционирования программы.
5. Составить алгоритм программы msp430x22x4_adc10_03.asm. Увеличить интервал температуры в два раза.
6. Составить полное описание режима работы таймера и ADC10 и указать значения всех битов регистров управления этих устройств.
7. Составить алгоритм программы msp430x22x4_adc10_03.asm. Увеличить период преобразований в два раза.
8. Составить алгоритм и программу для ввода сигнала от АЦП и вывода в порт P1 значения температуры в двоичном коде.
9. Дать подробное описание функционирования подпрограмм прерывания.
10. Организовать однократное аналого-цифровое преобразование.
11. Составить алгоритм и программу формирования последовательного преобразования.
12. Составить алгоритм и программу последовательного циклического преобразования
13. Записать содержимое регистров таймера и АЦП в пошаговом режиме в процессе выборки и преобразования.
14. Составить алгоритм и программу вывода сигнала опорного напряжения через P1.

15. Составить алгоритм и программу вывода сигнала опорного напряжения через P1, и сигнала температуры через P2.

16. Составить алгоритм и программу формирования 1 на выходе P1.1 при достижении заданной температуры.

17. Сформировать управляющее слово для циклического последовательного режима преобразования.

18. Составить управляющее слово для однократного последовательного режима преобразования.

19. Составить управляющее слово для циклического одноканального режима преобразования.

20. Составить подпрограмму прерывания для вывода ADC10MEM в параллельный порт

9.7. **Содержание отчета**

1. Цель работы.
2. Результаты выполнения исходной программы.
3. Условие варианта задания.
4. Алгоритм и программа заданного варианта в виде таблицы с комментариями.
5. Содержимое регистров микроконтроллера во время и после выполнения программы, до и после прерывания.
6. Результат выполнения программы.

9.8. **Контрольные вопросы**

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

10.Лабораторная работа №10. Разработка программ для микроконтроллера на языке C/C++

10.1.Цель работы

Язык C/C++ находит широкое применение в современном программировании. На языке C/C++ написано около семидесяти процентов программного обеспечения во всем мире.

Язык C, используемый для программирования микроконтроллеров и для системного программирования, усовершенствован Бьярном Страуструпом, создавшим объектно-ориентированный язык C++. Язык C++ полностью унаследовал и расширил возможности языка C.

Целью работы является изучение синтаксиса языка C/C++ и правил описания переменных, составление программы на языке C/C++ для микроконтроллеров.

10.2.Компиляция программы. Директивы препроцессора

Компиляция программы на языке C/C++ начинается с работы препроцессора. Препроцессор это служебная программа, которая выполняет предварительную обработку исходного текста программы пользователя. При этом выполняется включение в программу указанных в директивах файлов, создание макроопределений, условная компиляция, разрешение специальных директив. 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 программистом в исходный текст программы в любом месте и действует до конца программы или до отменяющей директивы. Директивы препроцессора показаны в таблице 10.1

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 условной компиляции позволяют выполнить компиляцию только при соблюдении определенного в директиве условия.

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

Таблица 10.1

Директива	Действие директивы	Пример
#include “ имя файла “	Включение файла-заголовка в исходный файл	#include “ msp430x21x1.h“
#define “имя макроса“ ‘фрагмент кода‘ #undef “имя макроса“	Создание нового макроопределения Отмена макроопределения	#define “Z“ ‘sqrt(R*R + X*X)‘ #undef “Z“
#if **константное выражение** #ifdef идентификатор #ifndef идентификатор #else #endif	Условная компиляция	#if **константное выражение** #ifdef идентификатор #ifndef идентификатор #else #endif
#line **целая константа** “ имя файла “ #error **сообщение об ошибке”**	Вставить из указанного файла строку с номером, заданным целой константой сообщение об ошибке	#line 3 “AZ.c “ #error zero_ div
#asm …. #endasm	Далее - команды ассемблера Конец фрагмента на ассемблере	#asm RRC R4 RRC R4 ADD.W #04, R4 #endasm
#pragma	Управление специфическими возможностями компилятора	#pragma CODE_SECTION (**Motor_Control, “secureRamFuncs”**); /*Размещение программы Motor_Control в защищенной области оперативной памяти*/

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

10.3.**Структура программы на C/C++**

Текст программы начинается директивами препроцессора. Далее в отдельной строке пишется заглавие программы. После заглавия следуют операторы программной единицы, обрамленные фигурными скобками. Перед первым выполняемым оператором в каждой программной единице (программе, подпрограмме) должны быть описания всех применяемых 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 /* */. Главная программа может иметь вид

```
#include “ msp430x21x1.h“ // включение заголовочного файла с адресами периферийных устройств
void main (void)          /*главная программа обозначается ключевым словом main, void означает отсутствие передаваемых данных*/
{
int  i, j = 4, k = 2;     //целые переменные, 16 разрядов, со знаком
char a,b;                 // целые переменные, 8 разрядов, со знаком
```

```
float32   X = 0;          // переменная  X с плавающей точкой, 32 разряда
float32   Y[4]            //массив из 4 элементов с плавающей точкой, 32 разряда
a = 4;              //далее выполняются вычисления, определяются элементы массива Y и переменная  X
b = k * a;
for  (i = 0; i < 4; i++)   // цикл повторится 4 раза
  {                        // начало цикла
          Y[i] = b + i;
           X = X + Y[i];     // сумма элементов массива
  }                        // конец цикла
}
```

Переменные бывают *глобальные и локальные.* Локальные переменные действуют только внутри подпрограммы, 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.

Произвольное выражение превращается в оператор, если его закончить точкой с запятой.

Арифметические операции описывают следующие арифметические действия:

- замена знака, вычитание;

++ инкремент	* умножение
- - декремент;	/ деление
+ сложение	% остаток от деления

Операции сравнения выполняют проверку на равенство, неравенство и результат сравнения, имея тип int, принимает значения 1 (истинно) или 0 (ложно). 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 операции служат для выполнения следующих действий:

! - инверсия

&& - коньюнкция

\\ - дизьюнкция

Результат принимает значения 1 (истинно) либо 0 (ложно).

Битовые операции выполняются над одноименными битами 8 и 16 разрядных слов.

~ - инверсия (один аргумент)	*^ - исключающее или*
& - коньюнкция	
\ - дизьюнкция	

10.4.**Порядок выполнения работы**

1. Прочесть инструкцию.
2. Составить алгоритм и программу на языке С/С++ по заданному варианту. Можно использовать пример готовой программы на языке С.
3. Откомпилировать программу и записать в устройство исполняемый файл.
4. Изучить образовавшийся при этом ассемблерный файл.
5. Исходный текст на языке С, текст на ассемблере и комментарии записать в таблицу 10.2.
6. Запустить программу на выполнение в непрерывном и пошаговом режимах.
7. Записать результаты выполнения программы в виде таблицы значений (Таблица 10.3).

Таблица 10.2

Исходный текст на языке С	текст на ассемблере	Комментарии

Таблица 10.3

Имена регистров	Значения в регистрах		

10.5.**Варианты заданий**

Составить алгоритм и программу языке С/С++ для расчета заданного выражения для ряда значений x и вывода результата в параллельный порт.

1. $y = 2ax^2$.
2. $y = 2 + ax^3$.
3. $y = 2 * sqrt(ax)$.
4. $y = 2 * sqrt(a^2 + x^2)$.
5. $y = Ym * sin(ax)$.
6. $y = Ym * ln(ax)$.
7. $y = 1 + x + ax^3$.
8. $y = 2(1 + ax^2)$.
9. $y = 3(1 + x - ax^2)$.
10. $y = 4|1 + x - x^2|$.
11. $y = 3 * sqrt(a^3 + x^3)$.
12. $y = 4 * \left|(^{a\,3} + x^3)\right|$.
13. $y = 5 * (a^2 + x^2)$.
14. $y = 7 * |a^2 - x^2|$.
15. $y = 8 * (1 + x^2)$.
16. $y = 3(1 + 2x - 3x^2)$.
17. $y = 2(1 + x - 3x^3)$.
18. $y = 9(1 - 3x^3)$.
19. $y = 11(|1 + 2x^3|)$.
20. $y = 13(1 + x^3)$.

10.6.Содержание отчета

1. Цель работы.
2. Условие задания в соответствии с вариантом
3. Алгоритм и исходный текст программы, текст программы на ассемблере, созданный компилятором с комментариями в виде таблицы 10.2.
4. Результаты выполнения программы в виде таблицы для ряда значений *y(x)* и графика функции.

10.7.Контрольные вопросы

1. Как и для чего применяются директивы препроцессора?
2. В каком месте программы располагаются объявления и описания переменных?
3. Как описать переменные различных типов?
4. Каковы ограничения на назначение имен переменных?
5. Какие части должна содержать программа на языке C/C++ ?

11.Лабораторная работа №11. Разработка алгоритмов и программ на языке C для микроконтроллера MSP430 с использованием аналого-цифрового преобразователя ADC10.

11.1. Цель работы

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 для дальнейшего использования. Для запуска аналого-цифрового преобразования может быть применен программный способ либо таймер TA.

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

11.2. Аналого-цифровой преобразователь ADC10 микроконтроллера MSP430

Функциональная схема ADC10 представлена на рисунке 9.1. Аналого-цифровой преобразователь ADC10 микроконтроллера MSP430 имеет внешние выводы для ввода сигналов, которые показаны на рисунке 9.2 и выделены.

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 В или 2.5 В. Следовательно, входной аналоговый сигнал не должен по модулю превосходить соответственно 1.5 В или 2.5 В. Возможен внешний источник опорного сигнала. На модуль ADC10 могут быть поданы до 8 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 11.1, 11.2. 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 11.1. **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 02 4 такта 03 8 тактов 12 16 тактов 13 64 такта	Скорость преобр. 0 до $20*10^4$ 1 до $5*10^4$ в секунду	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 1,5 В 1 2,5 В	Генератор опорного напряжения 0 выкл. 1 вкл.	Вкл. модуля ADC10 0 выкл. 1 вкл.	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 11.2. **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+} 1001 V_{REF-}/ Ve_{REF-} 1010 датчик температуры	Источник сигнала запуска 00 ADC10SC 01 выв.1 TA 10 выв.2 TA	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 не активен 1 занят

11.3. **Программы с использованием аналого-цифрового преобразователя** ADC10

Рассматривается пример программы msp430x22x4_adc10_01.c, исходный текст которой написан на языке C. Программа предназначена для выполнения аналого-цифрового преобразования сигнала AVcc от источника напряжения для аналоговых цепей на входе микроконтроллера. На вывод P1.0 выводится 1, если достигается значение A0 > 0.5*AVcc.

Текст программы начинается директивой **`#include "msp430x22x4.h"`** препроцессора, которая подсоединяет к программе головной файл, где содержатся адреса периферийных устройств микроконтроллера указанного типа.

```
//*******************************************************************************************
#include "msp430x22x4.h"  ;              // головной файл

void main(void)
{
 WDTCTL = WDTPW + WDTHOLD;              // стоп WDT
 ADC10CTL0 = ADC10SHT_2 + ADC10ON + ADC10IE; // ADC10ON, разрешение прерываний
 ADC10AE0 |= 0x01;                      // выбор P2.0 как внешнего вывода ADC
 P1DIR |= 0x01;                         // настройка на вывод   P1.0

 for ( ; ; )
 {
  ADC10CTL0 |= ENC + ADC10SC;           // запуск преобразования
  __bis_SR_register ( CPUOFF + GIE );   // режим малого потребления LPM0, выход по прерыванию ADC10_ISR
  if  ( ADC10MEM  <  0x1FF )
    P1OUT  &=  ~0x01;                   // сброс P1.0, LED отключен
  else
    P1OUT  |=  0x01;                    //  установка  P1.0,   LED включен
 }
}
// ADC10 подпрограмма прерывания
#pragma vector   =   ADC10_VECTOR
__interrupt void ADC10_ISR ( void )
{
 __bic_SR_register_on_exit ( CPUOFF );      // сброс CPUOFF   из  0(SR)
}
```

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 9.4 означает время выборки 16 тактов. Источником сигнала запуска преобразования является ADC10SC, для этого в регистр ADC10CTL0 записываются биты ENC и ADC10SC.

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.

Затем проверяется условие A0 > 0.5*AVcc, и в случае его выполнения в порт выводится 1.

После циклической части директива препроцессора описывает прерывание от ADC10, а в регистре состояния освобождается бит CPUOFF отключения CPU.

Следующий пример исходного текста на языке C программы msp430x22x4_adc10_02.c представлен далее. Данная программа предназначена для выполнения аналого-цифрового преобразования сигнала 1.5V Vref.

```
//*******************************************************************************
#include "msp430x22x4.h"                        // головной файл
      void main ( void )                       // начало главной программы
      {
 WDTCTL = WDTPW + WDTHOLD;                     // стоп WDT
 ADC10CTL0 = SREF_1 + ADC10SHT_2 + REFON + ADC10ON + ADC10IE;
 TACCR0 = 30;                                  // запаздывание, чтобы установился сигнал Ref
 TACCTL0 |= CCIE;                              // прерывание в режиме сравнения
 TACTL = TASSEL_2 + MC_1;                      // TACLK = SMCLK, тактовый сигнал.
 __bis_SR_register (CPUOFF + GIE);             // LPM0, выход по TA0_ISR
 TACCTL0 &= ~CCIE;                             // запрет прерывания от таймера
 ADC10AE0 |= 0x01;                             // выбран вывод P2.0 для  ADC10AE0
 P1DIR |= 0x01;                                // настройка P1.0 для вывода

 for (;;)
 {
  ADC10CTL0 |= ENC + ADC10SC;                  //   запуск преобразования
  __bis_SR_register(CPUOFF + GIE);             // режим LPM0, выход по ADC10_ISR
  if (ADC10MEM < 0x88)                         // ADC10MEM = A0 > 0.2V?
    P1OUT &= ~0x01;                            // сброс P1.0
  else
    P1OUT |= 0x01;                             // установка P1.0
 }
}

// ADC10 interrupt service routine
#pragma vector=ADC10_VECTOR
__interrupt void ADC10_ISR(void)
{
 __bic_SR_register_on_exit(CPUOFF);        // сброс CPUOFF от  0(SR)
```

```
}

#pragma vector=TIMERA0_VECTOR
__interrupt void TA0_ISR(void)
{
 TACTL = 0;
 LPM0_EXIT;                    // выход из режима LPM0 по возврату
}
```

11.4. **Порядок выполнения работы**

1. Составить алгоритм и программу в соответствии с заданным вариантом.
2. Открыть проект. В проект добавить файл текста программы на языке С.
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 в пошаговом режиме, затем в непрерывном режиме. Записать содержимое регистра SR и регистров управления ADC10 и таймером TACTL, TACCLx до и после инициализации.
7. Убедиться, что при изменении входного сигнала содержимое ADC10MEM изменяется.
8. Записать содержимое регистров ADC10 до и после возникновения прерывания.
9. Внести изменения в исходный текст **msp430x22x4_adc10_01.c,** чтобы программа соответствовала заданному варианту.
10. Выполнить п.3-7 для измененной программы.
11. Остановить выполнение программы.
12. Изменить выдержку времени, запустить программу и посмотреть, как это повлияет на работу устройства.

11.5. **Варианты заданий**

1. Составить алгоритм и программу переключений вывода P1.1 порта P1 при достижении заданного значения сигнала на входе.
2. Составить алгоритм и программу для включения двигателя при достижении заданной входной величины.
3. Составить алгоритм и программу для вывода преобразованного сигнала в двоичном коде через 250 мс в параллельный порт.
4. Составить алгоритм и программу для ввода сигнала от АЦП и вывода в порт P1 значения в двоичном коде.
5. Дать подробное описание функционирования подпрограмм прерывания.
6. Организовать однократное двухканальное аналого-цифровое преобразование.
7. Составить алгоритм и программу формирования последовательного многократного преобразования.
8. Составить алгоритм и программу последовательного однократного преобразования
9. Составить алгоритм программы msp430x22x4_adc10_01.c. Уменьшить интервал в два раза.

10. Записать содержимое регистров АЦП в пошаговом режиме в процессе выборки и преобразования.

11. Составить алгоритм и программу вывода сигнала опорного напряжения через P1.

12. Составить алгоритм и программу вывода сигнала напряжения через P2, температуры через P1.

13. Сформировать управляющее слово для циклического последовательного режима преобразования и использовать в программе.

14. Составить управляющее слово для однократного последовательного режима преобразования и применить в программе.

15. Дать подробное описание функционирования микроконтроллера при выполнении программы msp430x22x4_adc10_01.c..

16. Составить управляющее слово для циклического одноканального режима преобразования и применить в программе.

17. Составить подпрограмму прерывания для вывода ADC10MEM в параллельный порт

18. Составить полное описание режима работы ADC10 и указать значения всех битов регистров управления.

19. Составить алгоритм программы msp430x22x4_adc10_02.c. Увеличить период преобразований в два раза.

20. Дать подробное описание функционирования микроконтроллера при выполнении программы msp430x22x4_adc10_02.c..

11.6 **Содержание отчета**

3. Цель работы.
4. Результаты выполнения исходной программы.
5. Условие варианта задания.
6. Алгоритм для заданного варианта и программа в виде таблицы с комментариями.
7. Содержимое регистров микроконтроллера до, во время и после выполнения программы, до и после прерывания.
8. Результат выполнения программы с пояснениями.

11.7. **Контрольные вопросы**

6. Какие возможны режимы преобразования?
7. Как инициализировать АЦП для каждого из режимов преобразования?
8. Как взаимодействуют АЦП и центральный процессор ?
9. Как сохраняется результат преобразования?
10. Как изменить режим работы АЦП?
11. Как формируют интервал времени преобразования?
12. Каково назначение подпрограмм прерывания?

12.**Лабораторная работа №12**. **Использование последовательного интерфейса для приема и передачи данных**

12.1.**Цель работы**

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

12.2.**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 12.1.

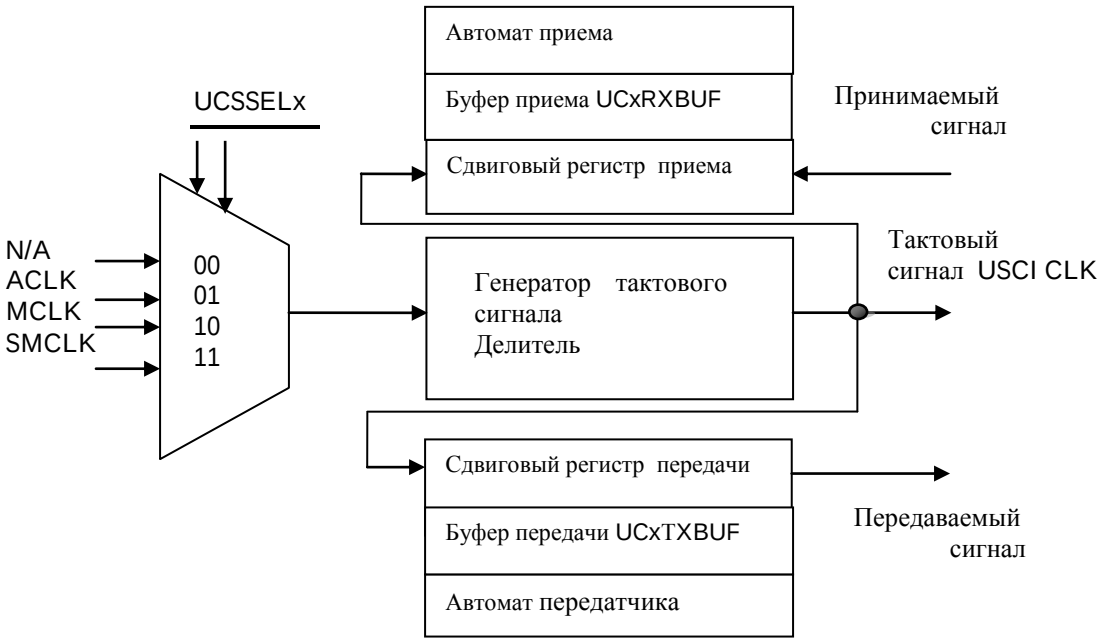


Рисунок 12.1 – Функциональная схема USCI.

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 12.1.

Таблица 12.1.Регистры управления и состояния модуля USCI_**A0**

Регистры	Назначение	Тип	Адрес
UC**A0**CTL0 UC**A0**CTL1	Регистры управления	Чтение, запись	060h 061h
UC**A0**BR0 UC**A0**BR1	Регистры управления скоростью обмена (содержат 16 битовый коэффициент деления)	Чтение, запись	062h 063h
UC**A0**MCTL	Регистр управления модуляцией	Чтение, запись	064h
UC**A0**STAT	Регистр состояния	Чтение, запись	065h
UC**A0**RXBUF UC**A0**TXBUF	Регистр буфера приема передачи	Чтение Чтение, запись	066 067h

В таблице 12.1. показано, что все регистры доступны для чтения и записи, кроме буфера приема-передачи данных, доступного только для чтения.

Регистры управления модуля USCI_**A0** имеют следующий вид.

UC**A0**CTL0

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	

UC**A0**CTL1

7 6	5 4 3 2 1	0
UCCSELx	-------	UCSWRST
Выбор источника тактового стигнала в режиме ведущего 00 ------ 01 – ACLK 10 - SMCLK 11 - SMCLK		1 - Программный сброс модуля 0 – рабочее состояние

В синхронном режиме для связи микроконтроллеров с внешними устройствами используются выводы:

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, когда на одной шине несколько ведущих устройств).

Режим SPI включается, если установить бит UCSYNC , а разновидность SPI (3-х или 4-х проводной) определяется битом UCMODEx.

Внешние выводы устройства последовательного интерфейса выделены на рисунке 12.2.

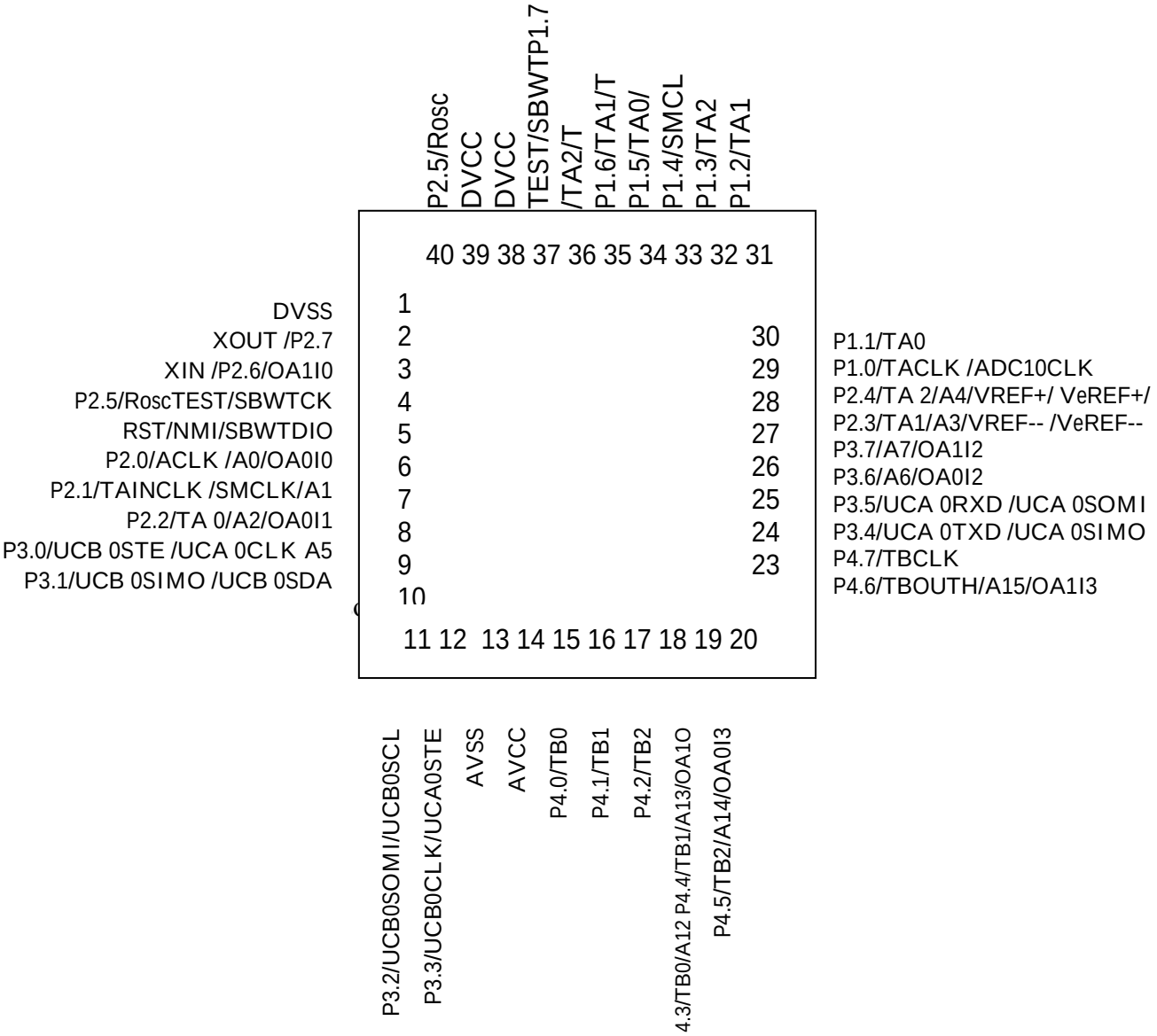


Рисунок 12.2 – Внешние выводы, используемые интерфейсом SPI

12.3. **Пример программы на C/C++**

Текст программы содержит 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"          // головной файл
unsigned char Data;                 // описание переменных
volatile unsigned int i;            //описание переменных

void main(void)
{
 WDTCTL = WDTPW + WDTHOLD;              // стоп WDT
 P3SEL |= 0x11;                    // P3.0,4 USCI_A0 выбор вывода
 UCA0CTL0 |= UCCKPH + UCMSB + UCMST + UCSYNC;  // 3-вывода, 8-бит, SPI ведущий
 UCA0CTL1 |= UCSSEL_2;               //источник тактового сигнала SMCLK
 UCA0BR0 |= 0x02;
 UCA0BR1 = 0;
 UCA0MCTL = 0;
 UCA0CTL1 &= ~UCSWRST;                 // **инициализация USCI **
 Data = 0x0FF;                     // загрузка начальных данных

 while(1)                          //  начало цикла
 {
  Data++;                                 // инкрементирование
  while (!(IFG2 & UCA0TXIFG));          // USCI_A0 TX готов ли буфер?
  UCA0TXBUF = Data;                       // запись байта в SPI TXBUF
  for(i = 0xFFFF; i > 0; i--);            // интервал времени
 }                                 //  конец  цикла
}
```

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.

12.4.**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

12.5.**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.

1. $y = 2ax^2$.
2. $y = 2 + ax^2$.
3. $y = 2 * x)$.
4. $y = 2 * (a + x)$.
5. $y = Ym * x$.
6. $y = Ym * ln(ax)$.
7. $y = 1 + x + a$
8. $y = 2(1 + ax^3)$.
9. $y = 3(x - a)$.
10. $y = 4|1 + x|$.
11. $y = 3 * (a + x)$.
12. $y = 4 * |(a + x)|$.
13. $y = 5 * (a + x)$.
14. $y = 7 * |a - x|$.
15. $y = 8 * (1 + x)$.
16. $y = 3(1 + 2x - 3x^2)$.
17. $y = 2(1 + x - 3x^2)$.
18. $y = 9(1 - 3x^2)$.
19. $y = 11(|1 + 2x^2|)$.
20. $y = 13(1 + x^2)$.

12.6.**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)*.

12.7.**Контрольные вопросы**

1. Как функционирует последовательный интерфейс?
2. Как организовать вывод и ввод данных через последовательный интерфейс?
3. Как организовать обмен данными двух устройств через SPI?
4. Какие режимы следует задать для инициализации последовательного интерфейса?
5. Пояснить работу программы.

13.**Лабораторная работа №13**. **Изучение архитектуры микроконтроллера** TMS320F28335

13.1. **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.

13.2.**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 так же 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 13.1 – Функциональная схема микроконтроллера

13.3.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.

13.4.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. Записать содержимое регистров ADC12 микроконтроллера и пояснить смысл.
7. Записать содержимое регистров PWM микроконтроллера и указать режим, соответствующий этой настройке.
8. Описать настройку длительности импульса PWM микроконтроллера и задание частоты импульсов.
9. Как увидеть содержимое области данных микроконтроллера, где находятся регистры периферийных устройств.
10. Какие регистры содержит последовательный порт I2C и где они расположены?
11. Как увидеть содержимое области данных микроконтроллера?
12. Какие регистры содержит последовательный порт SCI и где они расположены в памяти?
13. Охарактеризовать способы адресации. Привести примеры.
14. Как выполняются логические и арифметические команды?
15. Для чего нужна косвенная адресация? Привести примеры использования косвенной адресации.
16. Для программы, приведенной в проекте, дать описание директив препроцессора, которые используются для настройки периферийных модулей.
17. Дать описание возможных режимов последовательного интерфейса.
18. Составить описание периферийных устройств микроконтроллера с указанием адресов их регистров.
19. Как выполняется настройка системы PIE прерываний и каковы источники прерываний?
20. Как ограничивается энергопотребление микроконтроллера?

13.5. **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.

13.6.**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. Назначение и принцип действия процессора.
7. Каково назначение специальных регистров?
8. Назначение и принцип действия таймера.
9. Назначение и принцип действия параллельных и последовательных портов.
10. Назначение и принцип действия АЦП.
11. Как увидеть содержимое внутренней памяти программ и данных микроконтроллера?
12. Как увидеть содержимое регистров интерфейсных устройств?

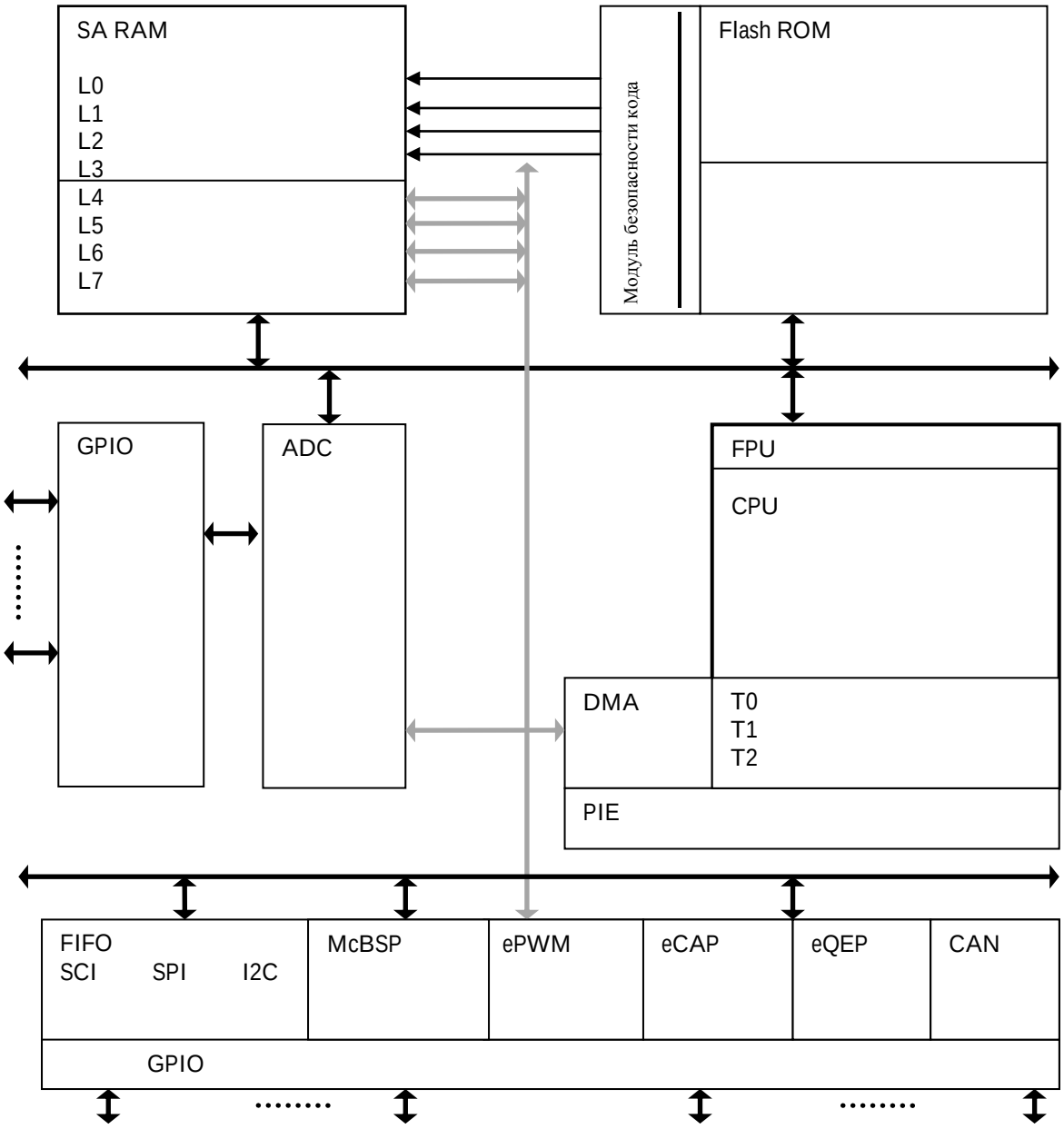


Рисунок 13.1- Функциональная схема TMS320F28335

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 = 0x00000000 AR2H = 0x0 AR2 = 0x0 XAR3 = 0x00000000 AR3H = 0x0 AR3 = 0x0 XAR4 = 0x00000000 AR4H = 0x0 AR4 = 0x0 XAR5 = 0x00000000 AR5H = 0x0 AR5 = 0x0 XAR6 = 0x00000000 AR6H = 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

Register	Description
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)).

Таблица 13-8. Регистры кадра 0

ИМЯ	АДРЕС	(x16)	ТИП ДОСТУПА
Device Emulation Registers	0x00 0880 – 0x00 09FF	384	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) 0 wait (DMA), 1 wait (CPU), read only	0x00 0B00 – 0x00 0B0F	16	Not EALLOW protected
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).

Таблица 13-9. Регистры кадра 1

ИМЯ	АДРЕС	(x16)
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

Таблица 13-10. Регистры кадра 2

ИМЯ	**АДРЕС**	(x16)
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

Таблица 13-11. Регистры кадра 3

ИМЯ	**АДРЕС**	(x16)
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.

13-**2. Арифметические и логические операторы)**

1 +	Унарный плюс	-	Унарный минус
~	1s complement	!	Логическое нет
2 *	Умножение	/	Деление
%	Модуль	3 +	Суммирование
-	Вычитание	4 <<	Сдвиг влево
>>	Сдвиг вправо	5 <	Меньше
<=	Не больше	>	Больше
>=	Не меньше		

Ассемблер

Операторы =[=] равно; != Не равно; & Побитное AND; ^ Побитное OR (XOR); | Побитное OR;

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.

Встроенные функции

Функция	**Описание**
$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]
$atan2(*x*, *y*)	Returns tan-1(*y*/*x*) in range [-π, π]

$ceil(*x*)	Returns the smallest integer not less than *x*, as a float
$cos(*x*)	Returns the cosine of *x*
$cosh(*x*)	Returns the hyperbolic cosine of *x*
$cvf(*n*)	Converts an integer to a float
$cvi(*x*)	Converts a float to an integer. Returns an integer.
$exp(*x*)	Returns the exponential function e*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
$log10(*x*)	Returns the base-10 logarithm log10(*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*
$round(*x*)	Returns *x* rounded to the nearest integer
$sgn(*x*)	Returns the sign of *x*. Returns 1 if *x* is positive, 0 if *x* is zero, and -1 if *x* is negative. Returns an integer.
$sin(*x*)	Returns the sine of *x*
$sinh(*x*)	Returns the hyperbolic sine of *x*
$sqrt(*x*)	Returns the square root of x, x□0
$tan(*x*)	Returns the tangent of *x*
$tanh(*x*)	Returns the hyperbolic tangent of *x*
$trunc(*x*)	Returns *x* truncated toward 0

Декодирование с оптимизацией

Команда...	**кодируется**...
MOV AX, *#8Bit*	MOVB AX, *#8Bit*
ADD AX, *#8BitSigned*	ADDB AX, *#8BitSigned*
CMP AX, *#8Bit*	CMPB AX, *#8Bit*
ADD ACC, *#8Bit*	ADDB ACC, *#8Bit*
SUB ACC, *#8Bit*	SUBB ACC, *#8Bit*
AND AX, *#8BitMask*	ANDB AX, *#8BitMask*
OR AX, *#8BitMask*	ORB AX, *#8BitMask*
XOR AX, *#8BitMask*	XORB AX, *#8BitMask*
MOVH *loc*, ACC << 0	MOV *loc*, AH
MOV *loc*, ACC << 0	MOV *loc*, AL
MOVL XAR*n*, *#8Bit*	MOVB XAR*n*, *#8Bit*
MOV *SP++, ACC.	PUSH ACC
MOV P, #0	MPY P, T, #0
SUB *loc*, *#16BitSigned*	ADD *loc*, *#-16BitSigned*
ADDB SP, *#-7Bit*	SUBB SP, *#7Bit*
ADDB *aux*, *#-7Bit*	SUBB *aux*, *#7Bit*
SUBB AX, *#8BitSigned*	ADDB AX, *#-8BitSigned*
PUSH IER	MOV *SP++, IER
POP IER	MOV IER, *--SP
PUSH ACC	MOV *SP++, ACC
POP ACC	MOV ACC, *--SP
PUSH XAR*n*	MOV *SP++, XAR*n*
POP XAR*n*	MOV XAR*n*, *--SP
PUSH *#16Bit*	MOV *SP++, *#16Bit*
MPY ACC, T, *#8Bit*	MPYB ACC, T, *#8Bit*
MOVW AX, *#8Bit*	MOV AX, *#8Bit*
ADDW AX, *#8Bit*	ADD AX, *#8Bit*
CMPW AX, *#8Bit*	CMP AX, *#8Bit*
ADDW ACC, *#8Bit*	ADD ACC, *#8Bit*
SUBW ACC, *#8Bit*	SUB ACC, *#8Bit*
JMP *8BitOffset*, *cond*	B *8BitOffset*, *cond*

Список использованных и рекомендуемых источников

1. Code Composer Studio™ v4.2 User's Guide for MSP430™ User's Guide / Literature Number: SLAU157S May 2005–Revised August 2011Copyright © 2005–2011, Texas Instr. Incorp. 2011. - 52 p.
2. Опейко О.Ф. , Петренко Ю.Н. Микропроцессорные средства в автоматизированном электроприводе. Минск, Амалфея, -2008.
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. Code Composer Studio™ v4.2 User's Guide for MSP430™ User's Guide / Literature Number: SLAU157S May 2005–Revised August 2011Copyright © 2005–2011, Texas Instruments Incorporated 2011. - 52 p.
5. www.ti.com/msp430
6. www.ti.com/support.
7. MSP430 Assembly Language Tools User's Guide (SLAU131)
8. MSP430 Optimizing C/C++ Compiler User's Guide (SLAU132).
9. Wiki page (FAQ)
10. E2E Community support forums for the MSP430 and Code Composer Studio v4.2.MSP430.
11. eZ430-RF2500 Development Tool User's Guide/Literature Number: SLAU227E September 2007.
12. www.ti.com/tms320f2833x
13. SPRS439—TMS320F28335, F28334, F28332, F28235, F28234, F28232 Digital Signal Controllers
14. SPRU513—TMS320C28x Assembly Language Tools v5.1.0 User's Guide (Current) SPRU513C—TMS320C28x Assembly Language Tools User's Guide (October 2007)
15. SPRU514—TMS320C28x Optimizing C/C++ Compiler Users Guide (Current) SPRU514C—TMS320C28x Optimizing C/C++ Compiler v5.0.0 (September 2007)
16. SPRUEO2—TMS320C28x Floating Point Unit and Instruction Set Reference Guide
17. SPRU430—TMS320C28x DSP CPU and Instruction Set Reference Guide
18. SPRU608—TMS320C28x Instruction Set Simulator Technical Overview
19. SPRU566—TMS320x28xx, 28xxx DSP Peripheral Reference Guide
20. SPRAA85—Programming TMS320x28xx and 28xxx Peripherals in C/C++
21. SPRAAM0—Getting Started With TMS320C28x Digital Signal Controllers
22. SPRAAN9—C28x FPU Primer
23. SPRAAP6—An Overview of Designing Analog Interface With TM320F28xx/28xxx DSCs
24. SPRB176—TMS320C2000 Digital Signal Controllers (Product Brochure)
25. Getting Started with C2000 Controllers (Internet)
26. Supercharged 32-Bit Controllers—C280xx Digital Signal Controllers (Internet)
27. SPRB167—Motor Control Overview Presentation
28. SPRB166—TMS320C2000 Platform DMC Overview.
29. SPRT461 (*)—TMS320C2000 Experimenter's Kit Overview
30. Язык С++. Учебное пособие / И.Ф. Астахова, С. В. Власов, В.В. Фертиков, А.В. Ларин. – Мн.: Новое знание, 2003. – 203 с.
31. Страуструп Б. Язык программирования М.; СПб.: БИНОМ – Невский диалект, 1999.
32. Шупляк В.И. С++. Практический курс : учеб. Пособие / В.И. Шупляк. – Минск : Новое знание, 2008. – 576 с.