Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Уфимский государственный авиационный технический университет»

Кафедра электромеханики

ПРОГРАММИРОВАНИЕ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

Лабораторный практикум по дисциплине «Микропроцессорные устройства в электроэнергетических объектах»

Уфа 2021

Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное
образовательное учреждение высшего образования
«Уфимский государственный авиационный технический университет»

Кафедра электромеханики

ПРОГРАММИРОВАНИЕ МИКРОПРОЦЕССОРНЫХ УСТРОЙСТВ

Лабораторный практикум
по дисциплине
«Микропроцессорные устройства
в электроэнергетических объектах»

Учебное электронное издание локального доступа

Уфа 2021

Авторы-составители: И. И. Ямалов, Д. Р. Фаррахов, Р. Р. Фаизова

Программирование микропроцессорных устройств : лабораторный практикум по дисциплине «Микропроцессорные устройства в электроэнергетических объектах» : [Электронный ресурс] / Уфимск. гос. авиац. техн. ун-т ; [авт.-сост. : И. И. Ямалов, Д. Р. Фаррахов, Р. Р. Фаизова]. – Уфа : УГАТУ, 2021. – URL: https://www.ugatu.su/media/uploads/MainSite/Ob%20universitete/Izdateli/El_izd/2021-102.pdf

Цель лабораторного практикума заключается в усвоении навыков обращения с числами в двоичном представлении для закрепления понимания работы микроконтроллера на аппаратном уровне, изучении основных характеристик и архитектуры микроконтроллера MSP430, изучении интерфейса отладочного модуля MSP430 LaunchPad, изучении основных характеристик таймера, изучении основных приемов программирования микроконтроллера MSP430, в том числе с использованием прерываний и АЦП.

Предназначен для студентов, обучающихся по направлению подготовки магистров 13.04.02 Электроэнергетика и электротехника.

Рецензент канд. техн. наук, доцент А. Р. Валеев

При подготовке электронного издания использовались следующие программные средства:

- Adobe Acrobat – текстовый редактор;
- Microsoft Word – текстовый редактор.

Авторы-составители: *Ямалов Ильнар Илдарович,*
Фаррахов Данис Рамилевич,
Фаизова Радмила Ризовна

Компьютерная верстка: *А. А. Шарипова*
Программирование и компьютерный дизайн: *М. В. Южакова*

Подписано к использованию: 07.07.2021
Объем: 2.17 Мб.

ФГБОУ ВО «Уфимский государственный авиационный технический университет»
450008, Уфа, ул. К. Маркса, 12.
Тел.: +7-908-35-05-007
e-mail: rik@ugatu.su

ВВЕДЕНИЕ

Тематика лабораторного практикума охватывает основную часть дисциплины «Микропроцессорные устройства в электроэнергетических объектах», в частности ознакомление с основными принципами и методами работы с программным продуктом, интерфейсами Code Composer Studio. В лабораторном практикуме используются современные технологии проектирования и средства автоматизации для разработки электроэнергетических и электротехнических систем. В теоретической части приведены принципы построения и функционирования микроЭВМ, структурная организация микроконтроллеров; определение методам адресации и системам команд, основам проектирования программного обеспечения встроенных систем; так же содержатся примеры интерфейсов ввода-вывода данных.

Лабораторный практикум включает анализ методов адресации и систем команд; содержит выводы по работе в среде инструментальных средств разработки программного обеспечения встроенных систем; формируются технические задания и сопоставляются алгоритмы работы микропроцессорной системы.

Отчет по лабораторной работе должен быть выполнен с соблюдением ГОСТ, иметь титульный лист. В верхней части титульного листа указывается название университета и кафедры. Затем название и номер лабораторной работы, номер группы и фамилия выполнившего ее студента.

В оформлении каждой лабораторной работы необходимо соблюдать следующие требования:

- сформулировать цель работы;
- построить алгоритм процесса;
- привести код программы согласно заданию (листинги);
- сделать выводы по лабораторной работе;
- ответить на контрольные вопросы.

ЛАБОРАТОРНАЯ РАБОТА № 1

Изучение микроконтроллера MSP430.
Порты ввода-вывода

Цель работы

Изучение портов ввода-вывода микроконтроллера MSP430 (рис. 1).

Краткие теоретические сведения

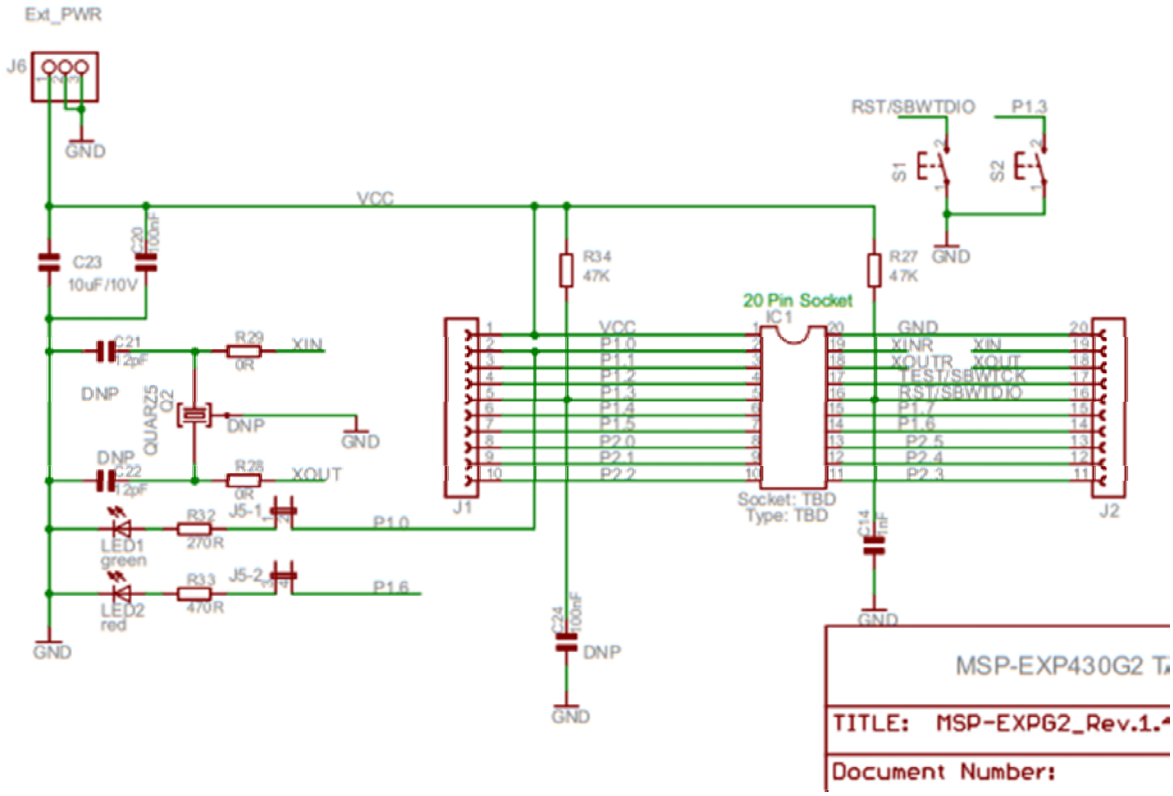

Рис. 1. Порты ввода/вывода

Микроконтроллеры семейства MSP430 могут иметь до восьми портов ввода/вывода **P1…P8**. Все порты содержат **по восемь выводов** (линий). Каждый из выводов порта индивидуально конфигурируется в качестве входа или выхода. Запись и чтение линий ввода/вывода также может осуществляться в индивидуальном порядке.

Порты **P1** и **P2** поддерживают внешние прерывания. Для каждого из выводов портов P1 и P2 можно индивидуально разрешить прерывание и сконфигурировать его так, чтобы оно генерировалось по нарастающему или спадающему фронту входного сигнала. Все

линии ввода/вывода порта **P1** назначены одному вектору прерываний, а все линии порта **P2** – другому вектору.

Цифровые порты ввода/вывода обладают следующими возможностями:

независимые индивидуально программируемые линии ввода/вывода;

любые комбинации входов или выходов; индивидуально конфигурируемые прерывания от выводов портов P1 и P2;

раздельные регистры данных для входов и выходов; индивидуально конфигурируемые внутренние подтягивающие резисторы.

Конфигурирование цифровых портов ввода/вывода осуществляется пользовательской программой. Настройка функционирования цифровых портов осуществляется с помощью нескольких специализированных регистров.

Регистр данных входа PxIN.

Каждый бит регистра **PxIN** отражает уровень входного сигнала на соответствующем выводе порта, если этот вывод используется в качестве цифрового входа/выхода:

Бит = 0: Входной сигнал имеет НИЗКИЙ уровень. **Бит = 1**: Входной сигнал имеет ВЫСОКИЙ уровень.

Регистр данных выхода PxOUT.

Значение каждого бита регистра PxOUT определяет состояние соответствующего вывода порта, если этот вывод сконфигурирован как цифровой выход, и внутренний подтягивающий резистор не используется.

Бит = 0: Выходной сигнал имеет НИЗКИЙ уровень.

Бит = 1: Выходной сигнал имеет ВЫСОКИЙ уровень.

Если используется **внутренний подтягивающий резистор**, то значение бита регистра **PxOUT** определяет тип «подтяжки» на соответствующем выводе порта:

Бит = 0: Вывод подтягивается к общему проводу.

Бит = 1: Вывод подтягивается к питанию.

Регистр направления PxDIR.

Значение каждого бита регистра **PxDIR** определяет направление передачи данных соответствующего вывода порта, независимо от выбранной для этого вывода функции. Если вывод используется каким-либо периферийным модулем, то бит регистра **PxDIR**должен быть установлен в соответствии с требованиями данного модуля.

Бит = 0: Вывод порта переключается на вход.

Бит = 1: Вывод порта переключается на выход.

Регистр включения подтягивающих резисторов PxREN.

Каждый бит регистра **PxREN** подключает или отключает внутренний подтягивающий резистор соответствующего вывода порта. Тип «подтяжки» определяется соответствующим битом регистра **PxOUT**.

Бит = 0: Подтягивающий резистор отключен.

Бит = 1: Подтягивающий резистор подключен.

Неиспользуемые выводы микроконтроллера необходимо сконфигурировать **как выходы портов** ввода/вывода и оставить неподключенными, чтобы избежать появления «плавающих» входов и снизить ток потребления устройства. Значение бита **PxOUT**для такого вывода может быть любым, поскольку вывод никуда не подключен. В качестве альтернативы, чтобы избежать появления «плавающего» входа, можно к неиспользуемому выводу подключить внутренний подтягивающий резистор, установив соответствующий бит регистра **PxREN**.

Описание лабораторной установки

Лабораторная установка представляет собой плату отладочного модуля MSP430 LaunchPad (MSP-EXP430G2). Плата LaunchPad подключается к порту USB компьютера с помощью прилагаемого кабеля. В состав лабораторной установки входит также пьезокерамический излучатель, предназначенный для получения звуковых эффектов.

Внешний вид платы LaunchPad приведен на рис. 2.

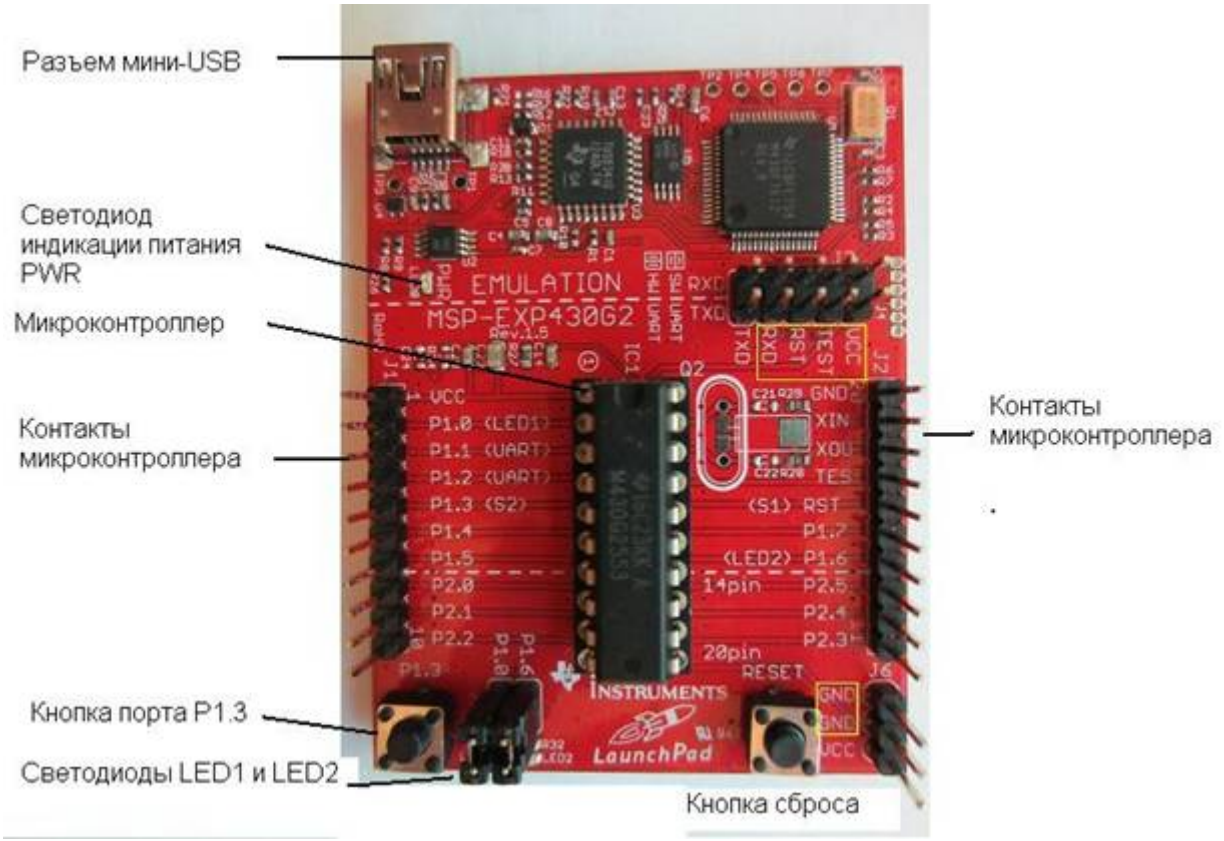

Рис. 2. Внешний вид платы LaunchPad

Порядок выполнения работы

Шаг 1. Откройте интегрированную среду разработки Code Composer Studio. Для операционной системы Windows7/10: *Пуск→Все программы→Texas Instruments→Code Composer Studio 7.x.x→Code Composer Studio 7.x.x*. После запуска будет показано окно выбора пути рабочего окружения (директории, в которой будут храниться все проекты с программным кодом). Оставьте поле «Workspace» без изменений, подтвердите выбор кнопкой «ОК», откроется главное окно системы (рис. 3). Среда Code Composer Studio построена на базе системы разработки Eclipse, поэтому на устаревших компьютерах запуск может занять некоторое время.

Шаг 2. Подключите отладочную плату к USB порту компьютера, должен загореться зеленый индикаторный светодиод с подписью «PWR».

Шаг 3. Создайте новый проект, выбрав следующие пункты главного меню: *File→New→CCS Project*, откроется окно настройки параметров проекта (рис. 4). В левом выпадающем меню «Target» выберите пункт «MSP430GXXX», а в правом – «MSP430G2553».

В поле «Project name» введите Название вашей группы и порядковый номер, например «e109m1», остальные поля оставьте без изменений. После этого подтвердите свои действия кнопкой «Finish». В левой части главного окна, в списке «Project Explorer» появиться новый проект.

Имя текущего рабочего проекта в системе Code Composer Studio выделяется полужирным шрифтом.

Шаг 4. После успешного создания нового проекта, в основном поле редактирования исходного кода автоматически создастся главный файл проекта. Традиционно для программного обеспечения, которое пишется на языке программирования С, этот файл носит название «main.c». Рассмотрим более подробно содержимое этого файла (листинг 1).

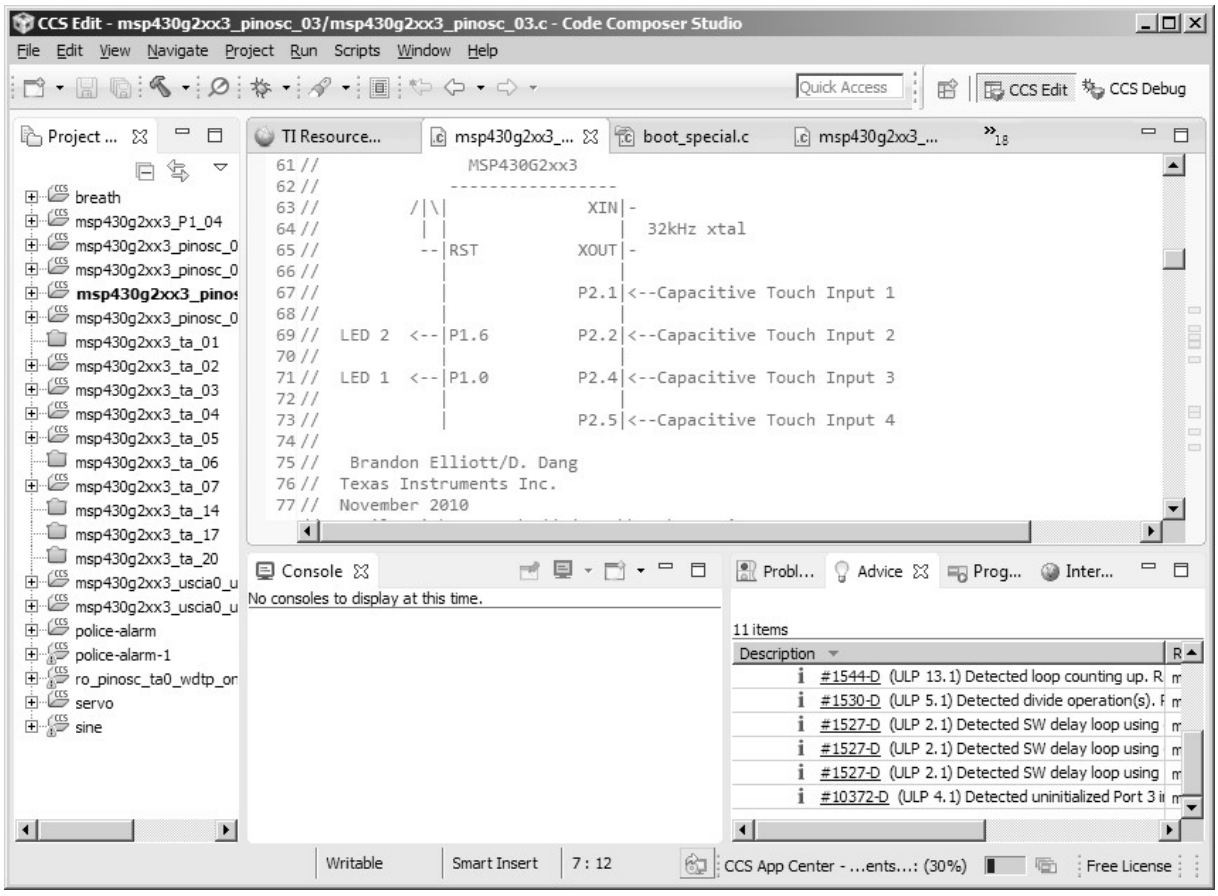

Рис. 3. Главное окно среды Code Composer Studio

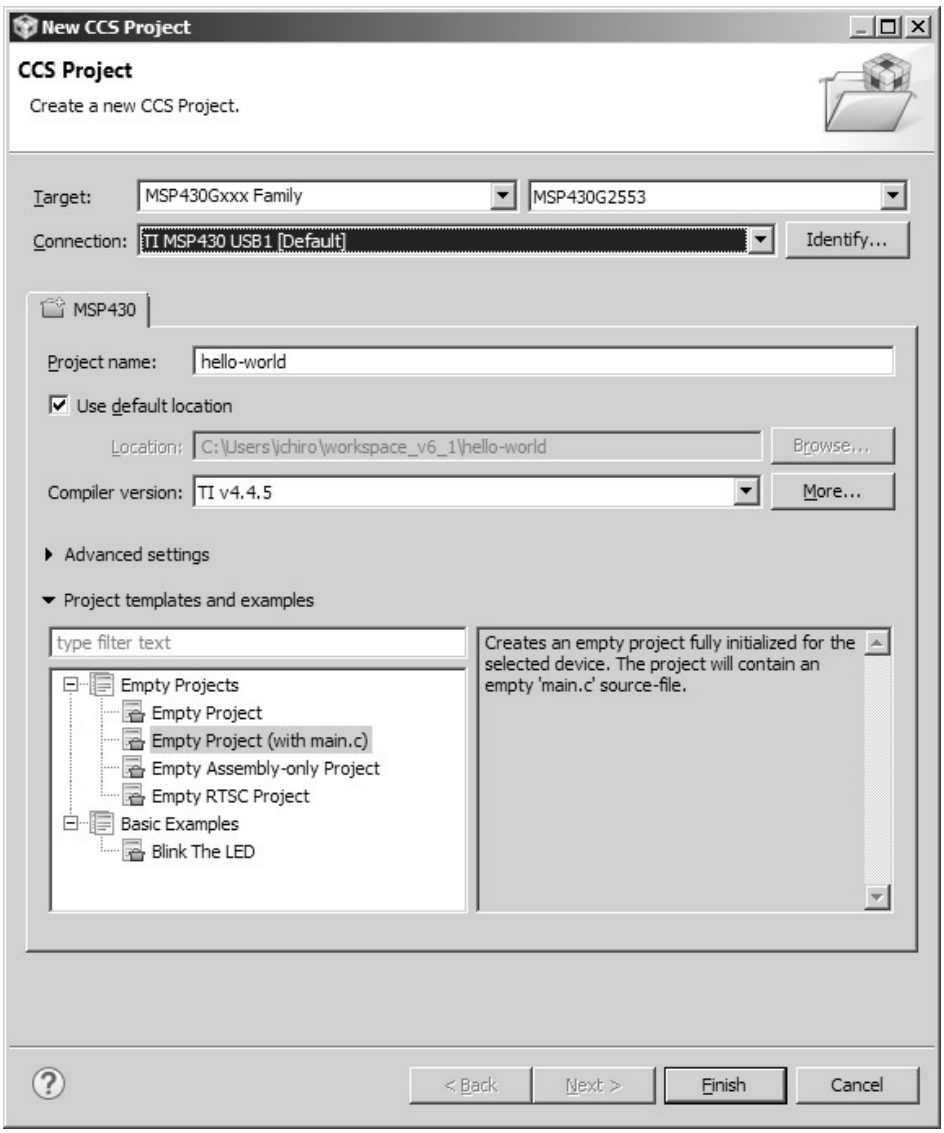

Рис. 4. Окно настрой параметров проекта

Листинг 1

```
1 #include <msp430.h>
2
3       int main(void)
4       {
5       WDTCTL = (WDTPW | WDTHOLD);
6       }
```

Директива **#include <msp430.h>** подключает заголовочный файл с описанием периферийных устройств конкретного чипа семейства MSP430, выбранного на этапе конфигурирования проекта. В нашем случае будут автоматически использоваться настройки для

MSP430G2553. В строке 3 определяется основная функция программы, задающая точку входа в программу. Она всегда должна называться «**main**».

В отличие от десктопных программ, программы для микроконтроллеров работают на «голом железе». Поэтому неважно значение какого типа будет возвращать эта функция и какие параметры будет принимать на вход. Все это имеет смысл только, если программа взаимодействует с операционной системой. Тем не менее, для удобства отладки рекомендуется в сигнатуре функции указывать целый тип возвращаемого значения (**int**). Директиву **return** можно опускать, предупреждений в логе компилятора при этом не возникает.

В строке 5 осуществляет настройка периферийного регистра **WDTCTL** [7, с. 347], отвечающего за работу сторожевого таймера. Сторожевой таймер (англ. *Watchdog timer*) – это аппаратно реализованная схема контроля над зависанием системы. Представляет собой таймер, который периодически сбрасывается контролируемой системой. Если сброса не произошло в течение некоторого интервала времени, происходит принудительная перезагрузка системы.

Битовая маска **(WDTPW | WDTHOLD)**, где **WDTPW** [7, с. 347] – восьмибитный пароль таймера (**0x5A00**), а **WDTHOLD** [7, с. 347] – флаг, останавливающий таймер. Отключение сторожевого таймера обязательно для любой программы, за исключением случаев, когда этот таймер используются не по прямому назначению. Значения, записанные в регистр **WDTCTL** по умолчанию таковы, что микроконтроллер будет перезагружен раньше, чем начнется выполнение основной программы.

Шаг 5. Запустите программу, нажав клавишу F11 (кнопка ). Система Code Composer Studio автоматически скомпилирует вашу программу, соберет ее и передаст на исполнение отладчику, который запишет ее в память микроконтроллера и остановит поток выполнения на первой строке.

При этом изменится вид главного окна. Если говорить в терминах системы Eclipse, будет изменена перспектива. Для Code Composer Studio определено всего две перспективы: перспектива кода (обозначена «CCS Edit») и перспектива отладки («CCS Debug»). Перспектива отладки включает в себя окно исходного кода, окно

с информацией о регистрах, выражениях и текущих переменных (рис. 5), окно процедур отладки (рис. 6), окно дисассемблера (рис. 7), окно терминала и вывода консоли (рис. 8).

Например, в окне регистров вы можете увидеть значения, которые были записаны в регистр **WDTCTL**. Сейчас там нули, но, если вы нажмете клавишу F5 (перейдете на один шаг вперед по коду), значение флага **WDTHOLD** будет изменено на единицу. Изменившиеся биты регистра и весь регистр при этом будут окрашены в желтый цвет (рис. 9).

Шаг 6. Переведите программу в режим постоянного выполнения, нажав клавишу F8 (кнопка ). Теперь программа работает в бесконечном цикле, но ничего не делает. Измените код программы в соответствии с листингом 2.

В строке 05 мы настраиваем направление работы нулевого вывода порта один. Как видно из кода, за это отвечает восьмибитный регистр **P1DIR** [7, с. 329]. Побитовая операция |= устанавливает в единицу те биты регистра, для которых в маске (переменная справа) соответствующие позиции также равны единице. Так, конструкция (1<<0) имеет значение **0x0001,** (1<<2) имеет значение **0x0100** (это значение задано в заголовочном файле **msp430.h**), поэтому получаем:

В строке 06 задается основной бесконечный цикл программы.

В строке 08 мы по аналогии со строкой 5 меняем нулевой бит регистра **P1OUT** [7, с. 329], отвечающего за запись значений в порт один. Тем самым даем команду включить красный светодиод.

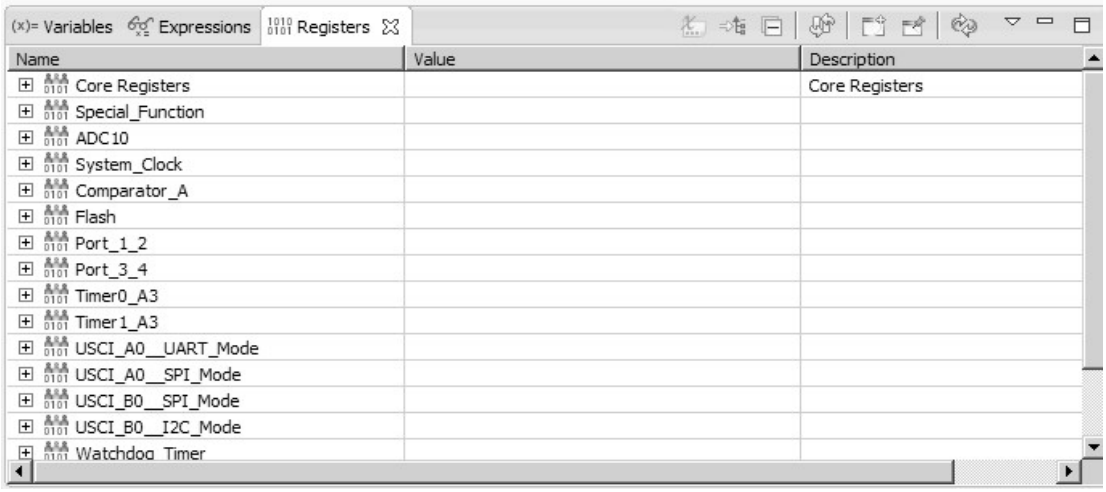

Рис. 5. Окно информации о регистрах, переменных и выражениях

Рис. 6. Окно процедур отладки

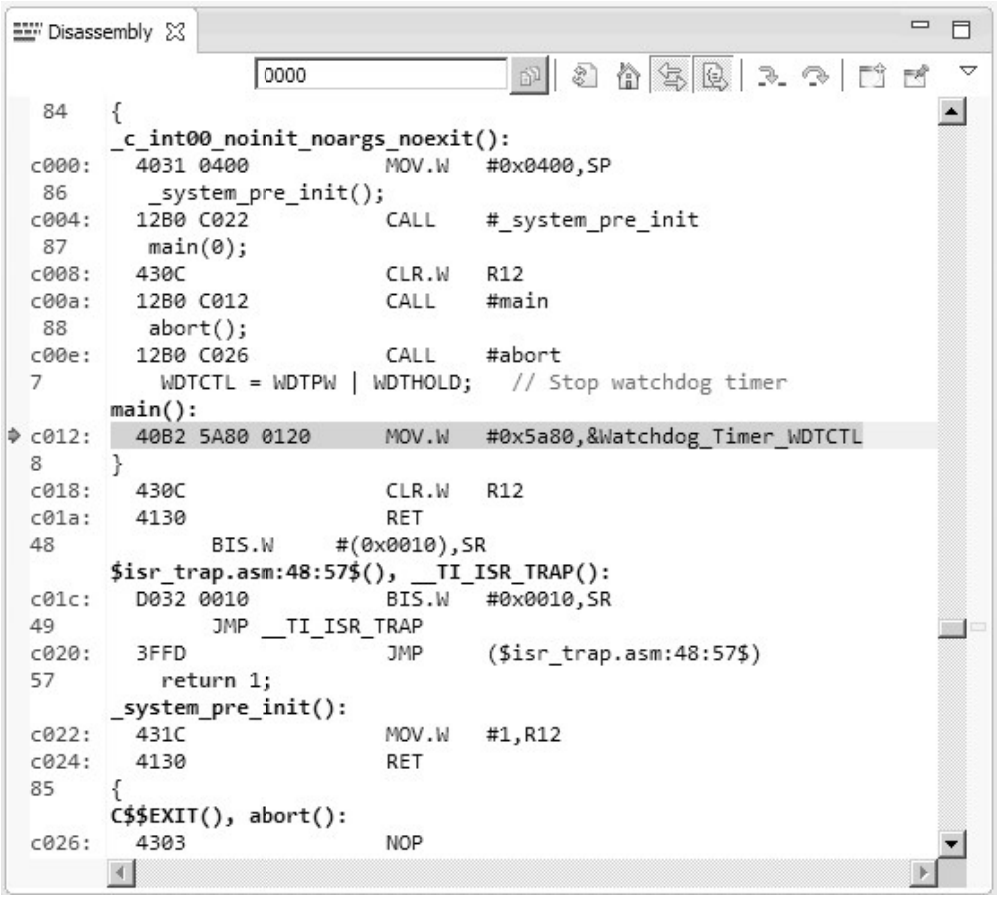

Рис. 7. Окно дисассемблера

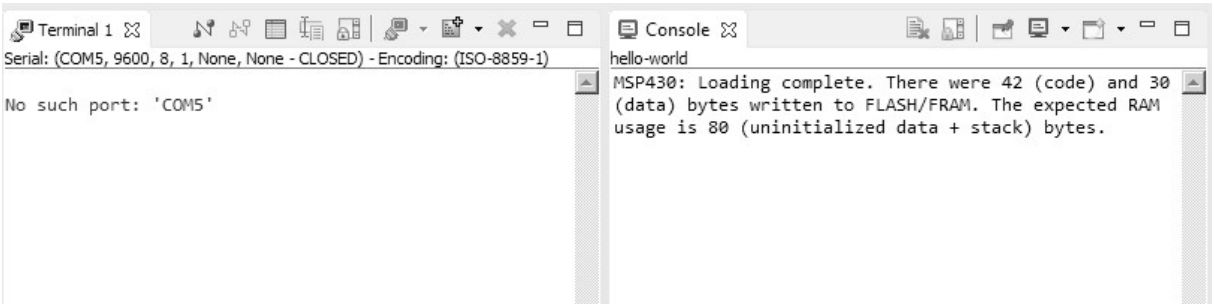

Рис. 8. Окно терминала и вывода консоли

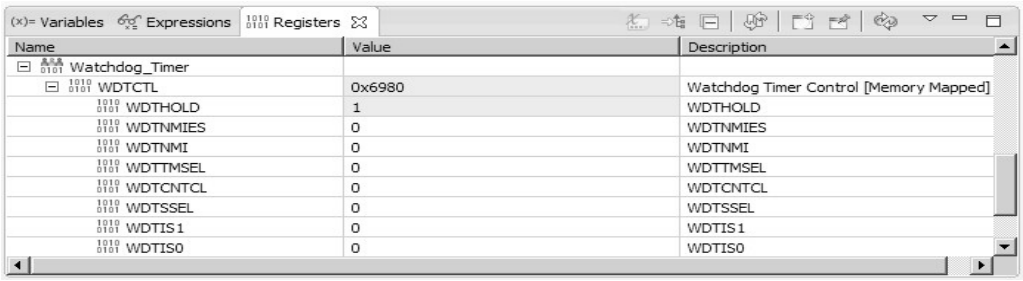

Рис. 9. Содержимое регистра **WDTCTL** после изменения

Листинг 2

```
01   #include <msp430g2211.h>
02   void main(void)
03   {
04           WDTCTL = WDTPW | WDTHOLD;
05           P1DIR |= (1<<0);
06           while(1)
07          {
08           P1OUT |= (1<<0);
09           }
10   }
```

Задание: *изменить листинг таким образом, чтобы загорелся зеленый светодиод (адрес красного и зеленого светодиода см. на рис. 10). Изменить листинг таким образом, чтобы загорелись оба светодиода.*

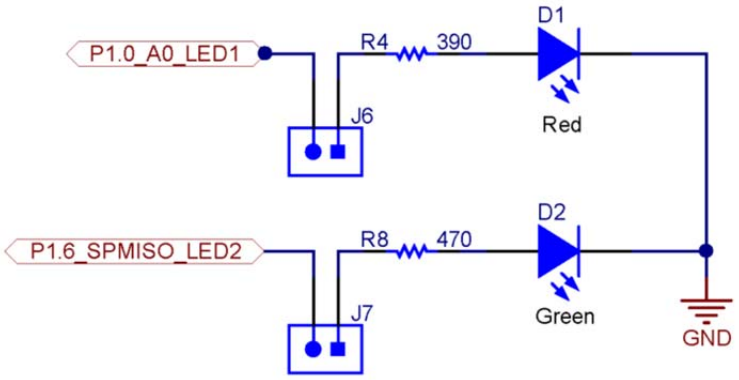

Рис. 10. Подключение светодиодов к МП

Шаг 7. Перезапустите программу (F11→Yes→F8). Если все сделано правильно, красный светодиод должен загореться.

Шаг 8. Измените код проекта в соответствии с листингом 3.

Листинг 3

```
01  #include <msp430.h>
02  int main(void)
03  {
04    WDTCTL = (WDTPW | WDTHOLD);
05    int i = 0;
06    P1DIR |= (1<<0);
07     while(1)
08    {
09
10                    for (i=0; i < 10000; i++)
11                       {
12                       P1OUT |= (1<<0); // включить св.
13                       }
14                    for (i=0; i < 10000; i++)
15                             {
16                       P1OUT &=~ (1<<0); //выключить св.
17                             }
18
19
20     }
21  }
```

Задание: *изменить листинг таким образом, чтобы увеличить или уменьшить частоту моргания светодиода.*

Следующим шагом мы попытаемся поработать с выводом порта, получающим сигнал на вход. В качестве источника сигнала будет использоваться тактовая кнопка, подключенная к выводу P1.3 отладочной платы.

Листинг 4

```
01  #include <msp430.h>
02  void main(void) {
03     WDTCTL = WDTPW | WDTHOLD;
04     P1OUT = 0;
05     P1DIR |= (1<<0);
06     P1REN |= (1<<3); //разрешаем подтяжку
07     P1OUT |= (1<<3); //подтяжка вывода P1.3 вверх
08     while(1) {
09        if((P1IN & (1<<3)) == (1<<3))  // ждем нажатия кнопки
10        {
11           P1OUT &=~(1<<0);
12        }
13        else
14        {
15           P1OUT |= (1<<0);
16        }
17
18     }
19  }
```

Задание: *Получите индивидуальное задание у преподавателя.*

Контрольные вопросы

1. Назовите основные типы переменных, их размерности.
2. Приведите таблицы истинности побитовых операторов И, ИЛИ, НЕ.
3. Перечислите регистры данных, их применение.
4. Какое число запишется в регистр P1DIR после выполнения операции: P1DIR &=~(1<<4) ?
5. Поясните структуру оператора if/else.
6. Поясните структуру цикла for.

ЛАБОРАТОРНАЯ РАБОТА № 2

Прерывания и таймеры

Цель работы

Изучение основных характеристик таймера. Изучение основных приемов программирования микроконтроллера MSP430 с использованием прерываний.

Краткие теоретические сведения

Прерывания представляют собой сигналы о наступлении какого-либо события, которые периферийные устройства передают центральному процессору. При поступлении такого сигнала центральный процессор останавливает свою работу, сохраняя текущее состояние в оперативной памяти (как правило, сохраняется содержимое регистров), после чего управление передается процедуре обработчика прерывания. После того, как процедура выполнена, процессор восстанавливает свое состояние из памяти и продолжает работу.

Микроконтроллеры, в отличие от настольных систем, зачастую работают в автономном режиме, питаясь не от сети, что приводит к необходимости экономии электроэнергии.

Типовой сценарий работы микроконтроллера следующий. В основном цикле работы микроконтроллер ничего не делает, находясь в одном из режимов пониженного энергопотребления. Для микроконтроллеров семейства MSP430 существует пять таких режима, в самом экономном из которых микроконтроллер потребляет не более 100 нА энергии. Срабатывание одного из прерываний от периферийных устройств выводит его из состояния «сна» для выполнения кода обработчика, после чего он вновь переходит в режим пониженного энергопотребления.

Второй важной особенностью прерываний является возможность организации аппаратной многопоточности. Именно этот режим будет рассмотрен в данной лабораторной работе. Мы объединим код обоих примеров из предыдущей работы таким образом, что обработка нажатия кнопки будет в одном прерывании, а переключение состояния светодиода в другом. При этом частота переключения будет задаваться не с помощью задержки в основном цикле, а посредством специального периферийного устройства, именуемого таймером.

Порядок выполнения работы:

Шаг 1. Откройте проект из предыдущей работы и измените код в соответствии с листингом 5.

Листинг 5

```
01 #include <msp430.h>
02
03   int main(void)
04   {
05   WDTCTL = (WDTPW | WDTHOLD);
06   P1DIR |= (1<<0);
07   TACCTL0 |= CCIE;
08   TACCR0 = 62500 - 1;
09   TACTL = (TASSEL_2 | MC_1 | ID_3);
10
11   __bis_SR_register(LPM0_bits | GIE);
12   while (1);
13   }
14   #pragma vector=TIMER0_A0_VECTOR
15   __interrupt void isr_timer_a(void)
16   {
17   P1OUT ^= (1<<0);
18   }
```

Задание: *изменить листинг таким образом, чтобы частота мигания светодиода стала 2 Гц, 12 Гц.*

В строке 7 настраивается блок **TACCTL0** [7, с.372], отвечающий за первый регистр захвата/сравнения таймера А (всего таких блоков три). Флаг **TACCTL0** разрешает прерывания от этого блока. Таймер – это просто счетный механизм, привязанный к импульсам тактового сигнала от системы синхронизации. Таймер А является шестнадцатибитным таймером. Это значит, что он считает от **0** до двоичного значения **0b1111111111111111**, или шестнадцатеричного **0xFFFF**, или десятичного **65535**. В строке 8 заполняется регистр **TACCR0** [7, с. 371]. В режиме сравнения в него записывается значение, по достижении которого таймер должен

подать сигнал центральному процессору, то есть **TACCR0** используется для указания верхней границы счета.

В строке 9 записывается значение основного регистра управления таймером A **TACTL** [7, с. 370]. Значение **TASSEL_2** [7, с. 370] задает тактовый генератор. Для микроконтроллеров архитектуры MSP430 существует три вида тактовых генераторов:

1) ***MCLK*** [7, с. 24] – *Master Clock*, основной тактовый сигнал, он управляет вычислительным ядром и очередью команд программы.

2) ***SMCLK*** [7, с. 24] – *Sub-Main Clock*, дополнительный тактовый сигнал, используется для периферии. Его частота может совпадать с MCLK, а может отличаться, зависит от приложения.

3) ***ACLK*** [7, с. 24] – *Auxilliary Clock*, вспомогательный тактовый сигнал, обычно он генерируется извне микроконтроллера, и используется для периферии.

В данном случае тактирование будет происходить от **SMCLK**. Параметр **MC_1** задает режим прямого счета от нуля до значения, записанного в регистре **TACCR0** с мгновенным возвратом в ноль (рис. 11). Также возможно считать в непрерывном режиме, то есть от нуля до **65535** (рис. 12), а также в режиме реверсивного счета, то есть от нуля до значения регистра **TACCR0** и опять до нуля (рис. 13). Работа таймера начинается сразу после включения.

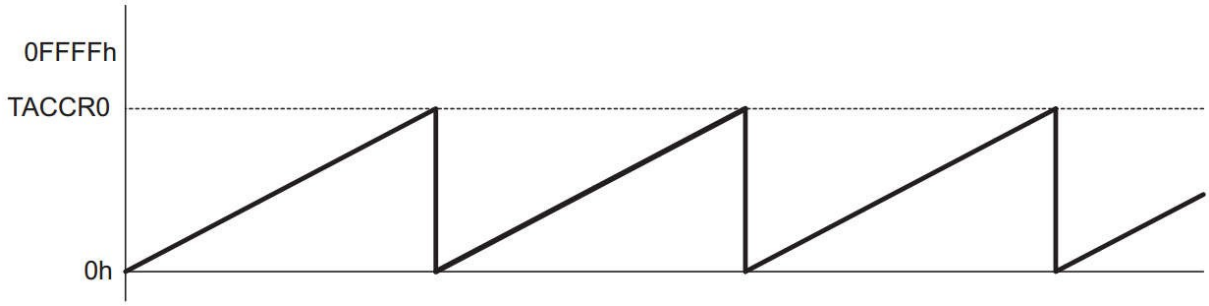

Рис. 11. Режим прямого счета таймера

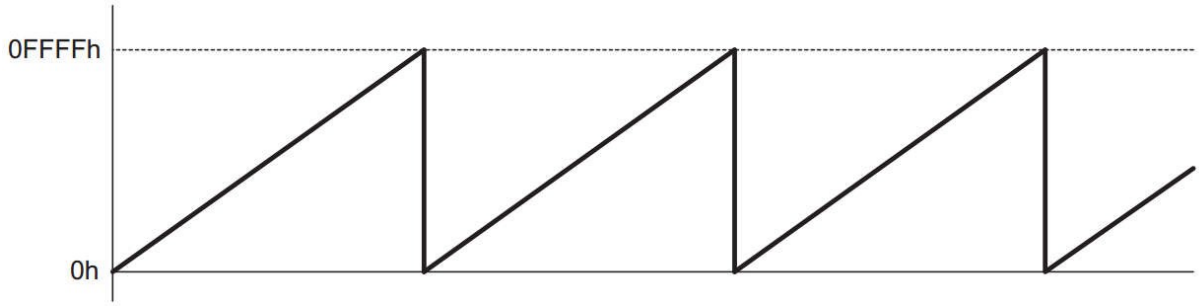

Рис. 12. Режим непрерывного счета таймера

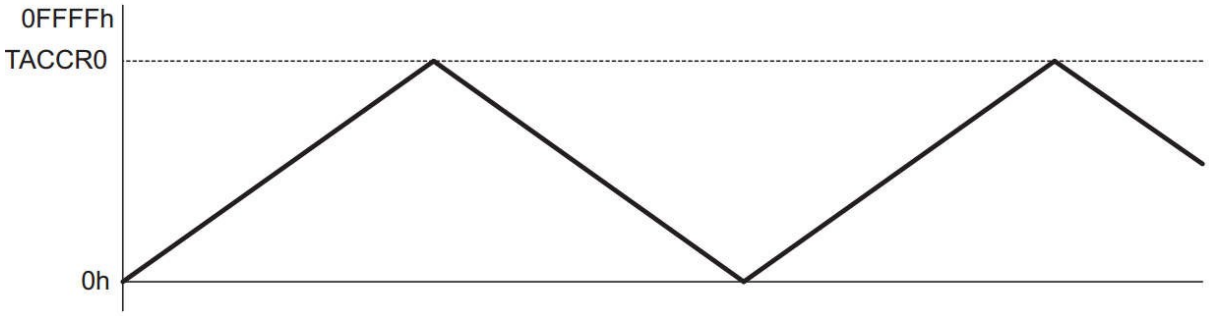

Рис. 13. Режим реверсивного счета таймера

Параметр **ID_3** задает делитель тактового сигнала, в данном случае частота тактового сигнала **SMCLK**, калиброванного на частоту примерно 1 МГц (встроенные источник тактового сигнала с цифровым управлением является не слишком точным по сравнению с внешними резонаторами) будет поделена на 2^3, то есть составит ~125 КГц.

В строке 11 вызывается функция **__bis_SR_register**, задающая значения статусного регистра SR. Первым в нее передается флаг **LPM0_bits**, задающий нулевой уровень пониженного энергопотребления, при котором отключается центральный процессор и основной тактовый генератор **MCLK**, а **SMCLK** и **ACLK** остаются активными. Существуют еще четыре режима пониженного энергопотребления [7, с. 39]:

1) ***LPM1*** – Центральный процессор и тактовый сигнал MCLK выключены. Генератор постоянного тока выключен, если встроенные источник тактового сигнала с цифровым управлением не используется для формирования сигнала **SMCLK**, сигнал **ACLK** активен.

2) ***LPM2*** – Центральный процессор, тактовые генераторы MCLK, SMCLK, а также источник тактового сигнала с цифровым управлением выключены. **ACLK** и генератор постоянного тока активны.

3) ***LPM3*** – Центральный процессор, тактовые генераторы MCLK, SMCLK, а также источник тактового сигнала с цифровым управлением выключены. Генератор постоянного тока выключен, ACLK активен.

4) ***LPM4*** – Центральный процессор и все тактовые сигналы выключены, микроконтроллер находится в состоянии поддержания оперативной памяти.

Флаг **GIE** является флагом общего разрешения маскированных прерываний, то есть прерываний, которые можно запрещать установкой соответствующих битов в регистре управления.

В строках 14 и 15 определяется процедура обработчика прерываний. **TIMER0_A0_VECTOR** – адрес вектора прерывания. Имя обработчика выбирается произвольно, но хорошим тоном считается включать в это имя префикс ISR (сокр. от англ. *Interrupt Service Routine* – процедура обработки прерывания). В теле обработчика прерывания уже традиционно изменяется состояние светодиода.

Шаг 2. Запустите программу на исполнение, красный светодиод должен начать мигать с частотой, равной:

$$\frac{SMCLK\ /\ ID\_3}{2 \cdot 62500} = \frac{1000000\,\text{МГц} / 8}{125000} \cong 1\,\text{Гц}.$$

Шаг 3. Прерывания могут быть вызваны любым периферийным устройством, являющимся частью рассматриваемого микроконтроллера. Измените код программы в соответствии с листингом 6, чтобы реализовать обработчик прерывания от порта ввода вывода общего назначения.

Строки 1–8 не требуют пояснений, в строке 9 устанавливается флаг, разрешающий прерывания от вывода 3 порта 1, то есть для тактовой кнопки. В строке 10 устанавливается флаг задающий фронт импульса, в данном случае прерывание будет происходить при переходе из высокого логического состояния в низкое, то есть по заднему фронту сигнала.

В строке 11 очищается флаг прерывания, если этого не сделать, обработчик сработает сразу. В строке 12 устанавливается флаг общего разрешения прерываний, а центральный процессор переходит в режим пониженного энергопотребления. В строках 14 и 15 задается определение обработчика прерывания, а в строках 16–19 тело обработчика. Обратите внимание на необходимость каждый раз очищать флаг прерывания по выходу из обработчика, если этого не делать, микроконтроллер зациклится в этом обработчике.

Листинг 6

```
01 #include <msp430.h>
02
03    int main(void)
04    {
05    WDTCTL = (WDTPW | WDTHOLD);
06    P1DIR = (1<<0);
07    P1OUT = (1<<3);
08    P1REN |= (1<<3);
09    P1IE |= (1<<3);
10    P1IES |= (1<<3);
11    P1IFG &=~(1<<3);
12    __bis_SR_register(LPM4_bits | GIE);
13    }
14    #pragma vector=PORT1_VECTOR
15    __interrupt void isr_port_1(void)
16    {
17    P1OUT ^= (1<<0);
18    P1IFG &=~(1<<3);
19    }
```

Задание: *изменить листинг таким образом, чтобы загорались оба светодиода.*

изменить листинг таким образом, чтобы загорелись оба светодиода попеременно.

В листинге 7 ознакомьтесь с работой цикла switch.

Листинг 7

```
01  #include <msp430.h>
02  int i = 0;
03  int main(void)
04  {
05  WDTCTL = (WDTPW | WDTHOLD);
06  P1DIR = (1<<0)|(1<<6);
07  TACCTL0 |= CCIE;
08  TACCR0 = 62500 - 1;
09  TACTL = (TASSEL_2 | MC_1 | ID_3);
10
11  __bis_SR_register(LPM0_bits | GIE);
12  while (1);
13  }
14
15  #pragma vector=TIMER0_A0_VECTOR
16  __interrupt void isr_timer_a(void)
17  {
18      i++;
19      switch (i)
20      {
21          case 1:
22          P1OUT |= (1<<6);
23          P1OUT &=~ (1<<0);
24          break;
25          case 5:
26          P1OUT |= (1<<0);
27          P1OUT &=~ (1<<6);
28          i =0;
29          break;
30      }
31  }
```

Задание: *поясните, почему тот или иной светодиод в Листинге 6 горит дольше другого.*

Напишите программу, которая реализует логику работы трехцветного светофора (красный 4 с→ красный с желтым 1 с → зеленый 3 с → зеленый мигающий 3 раза по 1 с → желтый 1 с → красный 3 с).

При нажатии на тактовую кнопку светофор должен переходить в ночной режим (желтый мигающий), при повторном нажатии – возвращаться в обычный режим.

Для выполнения работы соберите следующую электрическую схему (рис. 14).

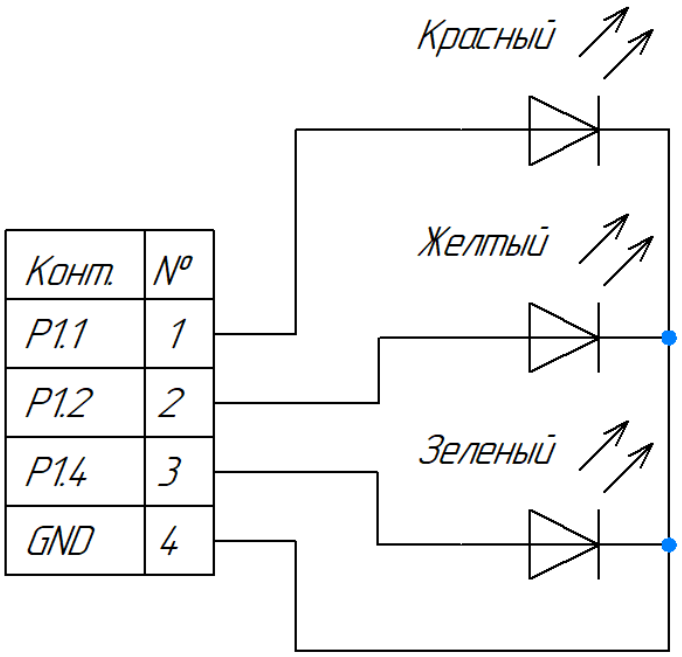

Рис. 14. Схема подключения светодиодов

Программный код должен быть написан с применением прерываний, рекомендуется построить его по принципу конечного автомата с применением оператора **switch**.

Контрольные вопросы

1. Перечислите режимы счета таймера.
2. Назовите основные типы прерываний. Какие строки листинга являются обработчиком прерывания, какие отвечают за инициализацию?
3. Приведите таблицу истинности побитового оператора исключающее ИЛИ.
4. Поясните структуру цикла switch.

ЛАБОРАТОРНАЯ РАБОТА № 3

Широтно-импульсная модуляция

Цель работы

Изучение основных характеристик широтно-импульсной модуляции.

Краткие теоретические сведения

Широтно-импульсная модуляция (ШИМ) представляет собой метод изменения мощности, подводимой к нагрузке, посредством изменения скважности импульсов при неизменной частоте. В наиболее общем смысле, ШИМ – это процесс быстрого переключения состояния вывода порта вводавывода, при котором усредненное напряжение на выводе пропорционально отношению времени, когда на вывод подается высокий логический уровень, к времени, когда подается низкий (рис. 15).

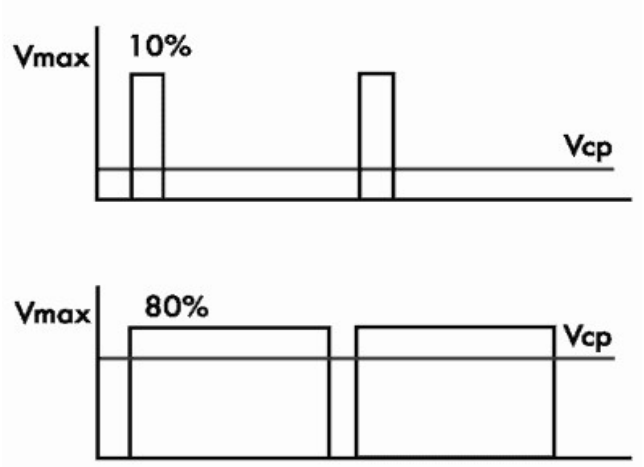

Рис. 15. Широтно-импульсная модуляция

Например, при частоте модуляции 50 Гц (период 20 мс) и скважности 10 %, после фильтрации ($V_{ср}$) на выводе микроконтроллера мы получим разность потенциалов равную

3,6 В × 0,1 = 36 мВ, а при скважности 80 % – 2,88 В.

Широтно-импульсная модуляция находит самое широкое применение при проектировании встроенных систем, что обуславливается простой реализации «в железе», а также возможностью гибкого управления вторичными источниками питания с высоким КПД. Также ШИМ используется в задачах управления, например, недорогие сервоприводы и контроллеры бесколлекторных двигателей принимают на вход ШИМ сигнал,

скважность которого определяет угол поворота сервопривода или скорость вращения мотора соответственно.

В данной лабораторной работе будут рассмотрены различные режимы работы ШИМ в микроконтроллерах архитектуры MSP430.

Порядок выполнения работы:

Шаг 1. Модифицируйте код программы в соответствии с листингом 8.

Листинг 8

```
#include <msp430.h>

    int main(void)
    {
    WDTCTL = (WDTPW | WDTHOLD);
    P1DIR |= BIT0;
    TA0CCTL0 = CCIE;
    TA0CCTL1 = CCIE;
    TA0CCR0 = 10000 - 1;
    TA0CCR1 = 100;
    TA0CTL = (TASSEL_2 | MC_1);
    __bis_SR_register(LPM0_bits | GIE);
    }
    #pragma vector=TIMER0_A0_VECTOR
    __interrupt void isr_timer_a0(void)
    {
    P1OUT |= BIT0;
    }
    #pragma vector=TIMER0_A1_VECTOR
    __interrupt void isr_timer_a1(void)
    {
        switch (TA0IV) {
        case 2:
        P1OUT &= ~BIT0;
        break;
        case 4: break;
        case 10: break;
        }
        }
```

Задание: *изменить листинг таким образом, чтобы увеличить или уменьшить частоту моргания светодиода.*

Изменить листинг таким образом, чтобы увеличить или уменьшить скважность моргания светодиода.

В листинге представлена реализация так называемой программной широтно-импульсной модуляции (англ. *Software PWM*), в данном случае используются только стандартные обработчики прерываний от таймера.

В строках 7 и 8 разрешаются прерывания от первого и второго регистров захвата сравнения. Всего таймер А имеет три регистра захвата сравнения, причем первый (**`TA0CCR0`**) задает период в режимах прямого и обратного счета, а два остальных могут быть использованы для формирования более одного события на период. Отдельно обрабатывается событие переполнения счетчика таймера в режиме непрерывного счета, о чем будет сказано ниже.

В строке 9 задается период ШИМ, в строке 10 – скважность. Для данного случая скважность равна 10 %. Вы можете менять значение регистра **`TA0CCR1`** по своему усмотрению, единственное ограничение – оно не должно превышать период. В 11 строке конфигурируется режим работы таймера А: тактирование от **`SMCLK`** и режим прямого счета. В строке 12 поднимается флаг общего разрешения прерываний, а центральный процессор переводится в первый режим пониженного энергопотребления.

Далее по коду идут два обработчика прерываний от таймера А. Здесь необходимо пояснить, как организована обработка прерываний для вспомогательных регистров захвата сравнения. Первый обработчик соответствует регистру периода **`TA0CCR0`**, а второй – **`TA0CCR1`**, **`TA0CCR2`** и флага переполнения. Все они объединены в один вектор **`TA0IV`**.

Обработчик первого прерывания аналогичен подобному из предыдущей работы. В нем мы просто устанавливаем высокий уровень сигнала на выводе, соединенном со светодиодом, то есть инициируем начало импульса. Второй обработчик содержит код проверки вектора **`TA0IV`**, реализованный с помощью оператора switch. Если значение вектора равно **`2`** (двоичное представление **`0b0000000000000010`**), значит счетчик достиг значения, записанного в регистре **`TA0CCR1`**. Возникло событие,

по наступлению которого мы можем выполнить какой-либо произвольный код. В данном случае мы переводим вывод на низкий логический уровень, завершая импульс ШИМ. После чего принудительно выходим из обработчика (оператор **`break`**).

Аналогичным образом мы можем обработать события, возникающие при достижении счетчиком значения регистра **`TA0CCR2`** (**`case 4`**, строка 26) и по переполнению счетчика (**`case 10`**, строка 27). Может показаться, что строки 26 и 27 не нужны, так как мы не обрабатываем соответствующие случаи, тем не менее, хорошим тоном программирования микроконтроллеров являет создание таких пустых обработчиков. Это может помочь при отладке сложных программ.

В литературе также рекомендуется добавлять в эти обработчики пустые бесконечные циклы, а также создавать обработчики с такими циклами для всех реализованных в микроконтроллере векторов прерываний. Это обуславливается тем, что в случае аппаратного сбоя (сработало неинициализированное прерывание, причина неизвестна), центральный процессор перейдет в обработчик и «зависнет» там в бесконечном цикле, что будет легко отследить в отладчике.

Данный метод часто используется в программировании и называется «максимизацией ошибки». Программа не должна «замалчивать» свои сбои, любой сбой должен приводить к полному краху программы. Так его гораздо легче будет воспроизвести, а значит проще и быстрее исправить. Общее правило здесь следующее: лучше пусть программа не работает вообще, чем работает иногда правильно, иногда неправильно и сложно определить из-за чего это происходит.

Шаг 2. Модифицируйте код программы в соответствии с листингом 9.

Задание: *изменить листинг таким образом, чтобы увеличить или уменьшить частоту моргания светодиода.*

Изменить листинг таким образом, чтобы увеличить или уменьшить скважность моргания светодиода.

Листинг 9

```
01 #include <msp430.h>
02
03  int main(void)
04  {
05  WDTCTL = (WDTPW | WDTHOLD);
06  P1DIR |= BIT2;
07  P1SEL |= BIT2;
08  CCR0 = 512 - 1;
09  CCTL1 = OUTMOD_7;
10  CCR1 = 384;
11  TACTL = (TASSEL_2 | MC_1);
12
13  __bis_SR_register(CPUOFF);
14  }
```

В листинге рассматривается пример аппаратной реализации широтно-импульсной модуляции (англ. *Hardware PWM*). В данном случае за формирование импульсов отвечает блок таймера А, работающий автономно и асинхронно по отношению к центральному процессору. Вывод сигнала осуществляется за счет использования альтернативной функции порта ввода вывода общего назначения.

В строке 7 задается альтернативная функция порта P1.2. В соответствии с листом данных на процессор MSP430G2553 для чипа в корпусе DIP20 этот порт связан с Timer0_A Out1 [9, с. 6]. В строке 8 в регистр **CCR0** записывается значение периода. В строке 9 настраиваются параметры второго регистра захвата сравнения. Всего существует 8 режимов ШИМ:

1) ***OUTMOD_0*** – вывод.
2) ***OUTMOD_1*** – установка.
3) ***OUTMOD_2*** – переключение/сброс.
4) ***OUTMOD_3*** – установка/сброс.
5) ***OUTMOD_4*** – переключение.
6) ***OUTMOD_5*** – сброс.
7) ***OUTMOD_6*** – переключение/установка.
8) ***OUTMOD_7*** – сброс/установка.

Графическое представление всех рассмотренных видов ШИМ сигнала для случая, когда таймер работает в режиме прямого счета дано на рис. 16.

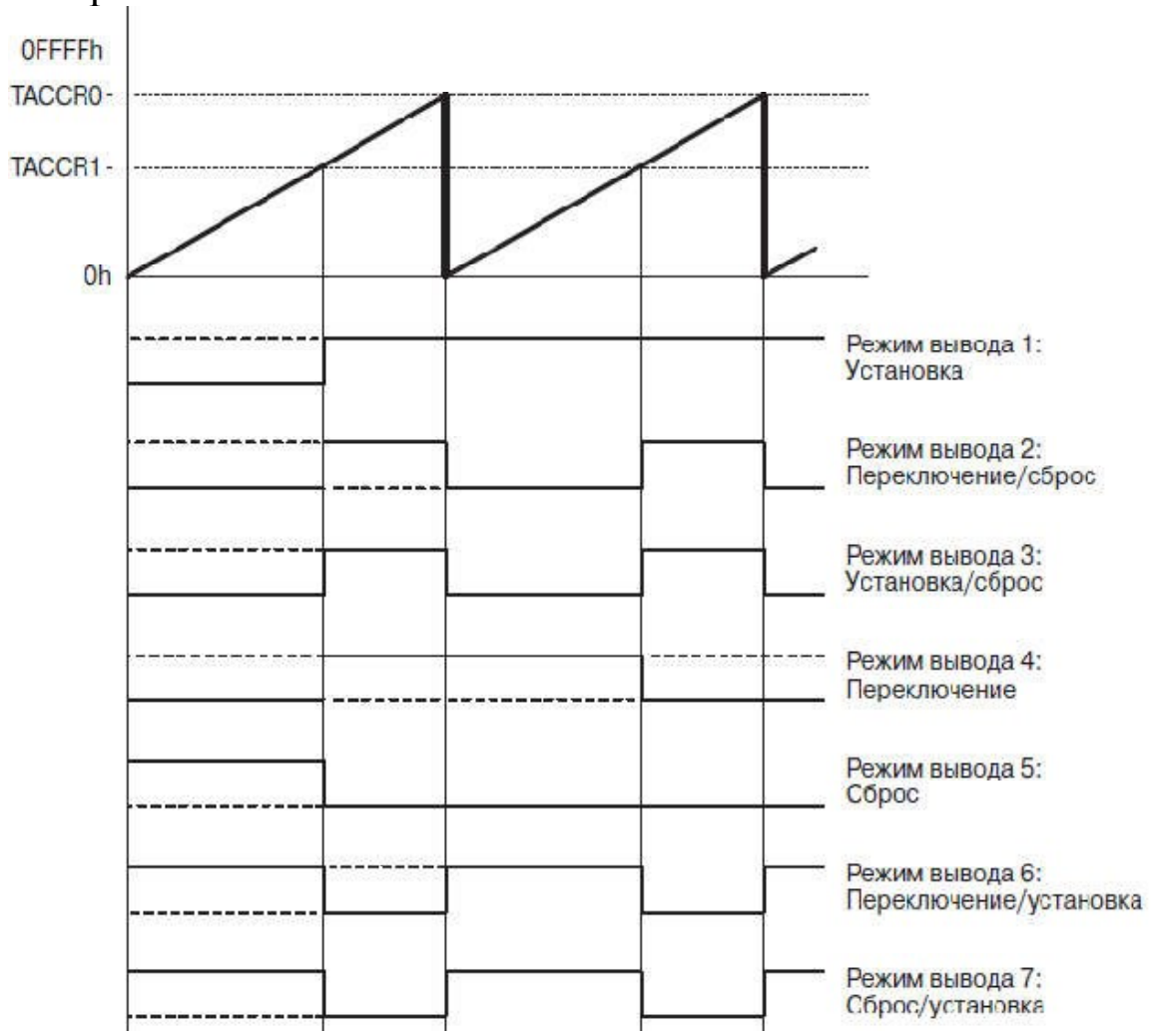

Рис. 16. Режимы формирования ШИМ

В строке 10 задается значение регистра **CCR1**, определяющего скважность сигнала, в данном примере скважность равна 75 %. Увидеть данное значение можно, подключив к выводу P1.2 индикаторный светодиод по схеме из предыдущей работы, либо воспользовавшись измерительными приборами: в простейшем случае достаточно вольтметра, при желании увидеть форму и длительность сигнала рекомендуется использовать логический анализатор или осциллограф (последний предпочтительнее).

В строке 11 задается тактирование от генератора **SMCLK** и режим прямого счета. В строке 12 центральный процессор отключается, то есть переходит в первый режим пониженного энергопотребления. **CPUOFF** является синонимом **LPM0_bits**. Прерывания выключены.

Задание: *обеспечить движение гусеничной платформы в различных направлениях, по заданию преподавателя. Основой для программы может являться листинг, реализующий логику работы трехцветного светофора из Л.р. № 3.*

Внимание: *для того чтобы сберечь двигатели гусеничной платформы, после каждого движения делайте ОСТАНОВКИ (движение вперед – остановка – поворот – остановка – движение вперед)*

1. Для управления гусеничной платформой используется драйвер L298N. Необходимы следующие выводы: INPUT 1, INPUT 2, INPUT 3, INPUT 4, ENABLE A, ENABLE B (рис. 17).
2. Правая и левая гусеницы управляются отдельно. *in 1, in 2, enA* выводы для правой. *in 3, in 4, enB* для левой.
3. Для движения гусеницы вперед элементы F должны выдать на выходе «1». Для движения гусеницы назад элементы R должны выдать на выходе «1».
4. Составьте таблицы управления каждой гусеницы назад и вперед (1 и 0).

	Rforward	Rreverse	Rstop
in1			
in2			
enA			

	Lforward	Lreverse	Lstop
in3			
in4			
enB			

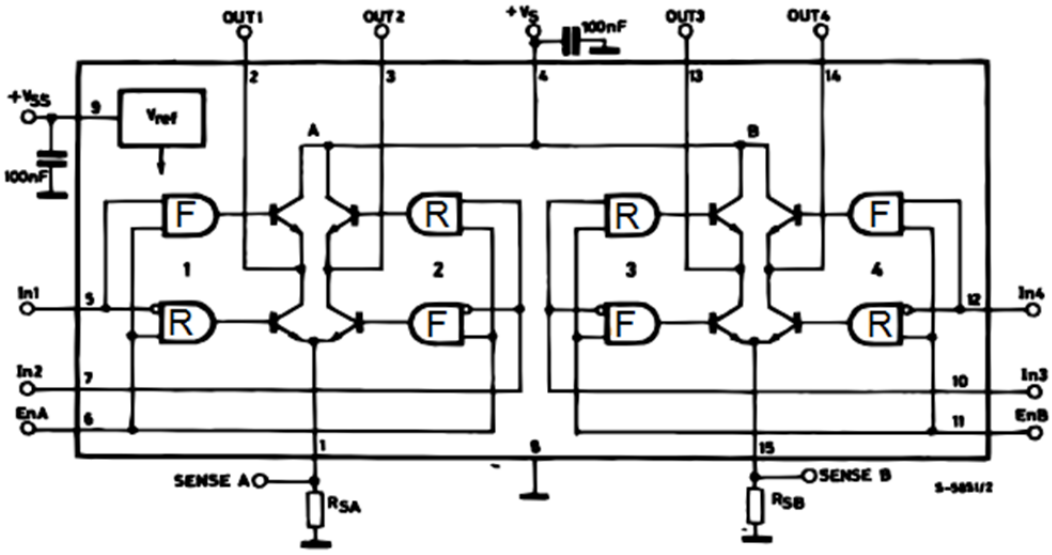

Рис. 17. Внутреннее строение драйвера L298N

5. При помощи мультиметра в режиме «прозвонки» необходимо найти соответствующие необходимые выводы на коннекторе «Входы драйвера» (рис. 18 и 19).

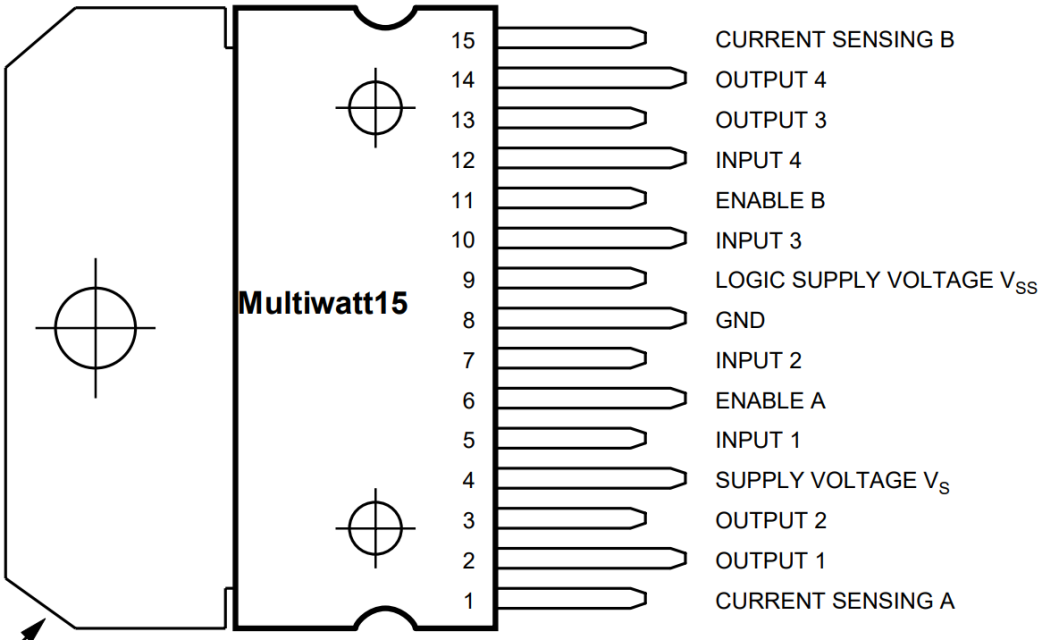

Рис. 18. L298N – описание выводов

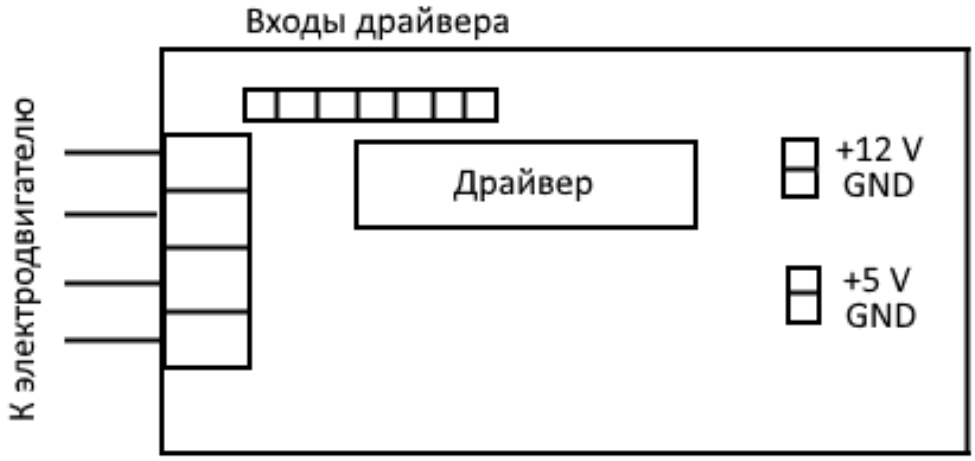

Рис. 19. Схема драйвера электропривода

6. Назначьте каждому из *in 1, in 2, enA, in 3, in 4, enB* вывод на контроллере (P1). Соедините при помощи проводов контроллер с платой драйвера.

7. Подайте питание для электродвигателя +12 В, и для микроконтроллера +5 В.

8. Напишите программу. Основой для программы может являться листинг, реализующий логику работы трехцветного светофора из Л.р. № 3. Используйте директивы *#define*, функции и прототипы функций для написания программы (Листинг 10).

Листинг 10

```
01  #include <msp430.h>
02  #define in1 (1<<0)
03  void Rforward(void);
04  int i=0;
05  int main(void)
06  {
07  WDTCTL = (WDTPW | WDTHOLD);
08  P1DIR |= in1;
09
10  __bis_SR_register(LPM0_bits | GIE);
11  while(1);
12  }
13  void Rforward(void)
14  {
15  P1OUT |= in1|enA;
16  P1OUT &=~ in2;
17  }
18
19  #pragma vector=TIMER0_A0_VECTOR
20  __interrupt void isr_timer_a0(void)
21  {
22  i++;
23  switch(i)
24  {
25  case 1:
26  Rforward();
27  Lforward();
28  break;
29  }
```

Контрольные вопросы

1. Что такое ШИМ? Перечислите режимы работы ШИМ. Какие строки листинга отвечают за инициализацию ШИМ?
2. Каким образом осуществляется объявление функции? Что такое прототип функции?
3. Поясните структуру директивы #define.

ЛАБОРАТОРНАЯ РАБОТА № 4

Аналого-цифровой преобразователь

Цель работы

Изучение основных характеристик аналого-цифрового преобразователя. Изучение основных приемов программирования микроконтроллера MSP430 с использованием АЦП.

Краткие теоретические сведения

Аналогово-цифровой преобразователь – это периферийное устройство, позволяющее переводить непрерывный сигнал в последовательность дискретных значений с определенной частотой дискретизации и разрешением. Разрешение определяет количество уровней квантования на одну выборку сигнала. Как правило, с помощью АЦП измеряется напряжение на входе микроконтроллера. Например, для размаха в 3,6 В и разрешения в 10 бит на одну выборку существует 1024 уровня квантования, что равно разрешению в 3,5 мВ. В реальных приложениях редко удается достичь заявленного разрешения по причине шума.

Принято считать, что для большинства неспециализированных АЦП без прецизионных источников опорного напряжения как минимум младший бит содержит шум. С ним можно бороться программным (фильтрация, например усреднение) или аппаратными (использование внешнего стабильного и точного источника опорного напряжения) средствами, либо их комбинацией. Также можно просто отбрасывать (обнулять) биты, содержащие шум, снижая тем самым разрешение. Выбор конкретного метода зависит от назначения встраиваемой системы, требований, предъявляемых к ней, а также возможностей микроконтроллера.

Существует несколько разновидностей АЦП: последовательные прямого преобразования, последовательного приближения, последовательные с сигма-дельта модуляцией, параллельные одноступенчатые, параллельные двух- и более ступенчатые (конвейерные).

В микроконтроллерах с архитектурой MSP430 реализован АЦП последовательного приближения. Принцип его работы достаточно прост. Вначале измеряемое напряжение сравнивается с половинным

опорным напряжением посредством встроенного одноразрядного компаратора. Как известно, компаратор представляет собой устройство, принимающее на вход два сигнала и выдающее высокий логический уровень, если один из них больше другого по величине, а низкий, если наоборот. Таким образом, когда исследуемое напряжение больше половины опорного, старший бит результата дискретизации устанавливается в единицу, если меньше – в ноль.

Затем опорное напряжение устанавливается на 1/4 или 3/4 от базового в зависимости от предыдущего результата и вновь сравнивается с исследуемым сигналом. При этом вычисляется следующий бит результата оцифровки. Используя такой метод, можно получить цифровое значение любой разрядности в зависимости от того, сколько измерений было сделано. В случае десятибитного аналогово-цифрового преобразователя процессоров семейства MSP430G2 проводится десять таких измерений. Максимальная скорость преобразования достигает 200 тысяч выборок в секунду.

В данной лабораторной работе будет рассмотрена реализация оцифровки сигнала от встроенного в микроконтроллер температурного сенсора.

Порядок выполнения работы:

Шаг 1. Модифицируйте код программы в соответствии с листингом 11.

Листинг 11

```
01        #include <msp430.h>
02        #define ADC_DELTA       3
03        static unsigned int first_adc_val;
04        int main(void)
05        {
06        WDTCTL = (WDTPW | WDTHOLD);
07        ADC10CTL1 = (ADC10DIV_3 | INCH_10 |
                    SHS_1 | CONSEQ_2);
08        ADC10CTL0 = (SREF_1 | ADC10SHT_3 |
          REF2_5V |
                    ADC10IE | REFON | ADC10ON);
09        __enable_interrupt();
10        TACCR0 = 30;
11        TACCTL0 |= CCIE;
```

```
12          TACTL = (TASSEL_2 | MC_1);
13          LPM0;
14          TACCTL0 &= ~CCIE;
15          __disable_interrupt();
16          ADC10CTL0 |= ENC;
17          TACCTL1 = OUTMOD_4;
18          TACTL = (TASSEL_2 | MC_2);
19          while (!(ADC10IFG & ADC10CTL0));
20          first_adc_val = ADC10MEM;
21          P1OUT = BIT0;
22          P1DIR |= BIT0;
23          __bis_SR_register(LPM0_bits | GIE);
24          }
25          #pragma vector=ADC10_VECTOR
26          __interrupt void isr_adc10(void)
27          {
28          if (ADC10MEM >= first_adc_val +
            ADC_DELTA)
29          P1OUT |= BIT0;
30          else
31          P1OUT &= ~BIT0;
32          }
33          #pragma vector=TIMER0_A0_VECTOR
34          __interrupt void isr_ta0(void)
35          {
36          TACTL = 0;
37          LPM0_EXIT;
38          }
```

В листинге представлена реализация анализатора градиента температуры. Если за определенный период времени, температура, регистрируемая датчиком, увеличивается больше чем на 2 °С по сравнению с первым измерением, загорается красный индикаторный светодиод. После возвращения температуры в изначальный диапазон, светодиод гаснет.

Для управления периферийным блоком АЦП используется два регистра управления: **ADC10CTL0** и **ADC10CTL1** [7, с. 477]. В строке 7 задаются параметры регистра **ADC10CTL1**. Выбирается десятый

канал преобразования – **INCH_10**, всего есть 8 внешних каналов и 4 внутренних, десятый соединен с внутренним датчиком температуры.

Флаг **CONSEQ_2** задает циклический одноканальный режим преобразования. Также существует однократный одноканальный (**CONSEQ_0**), однократный последовательный (CONSEQ_1) и циклический последовательный (**CONSEQ_3**) режимы преобразования.

ADC10DIV_3 задает делитель тактового сигнала АЦП, в данном случае – 8. **SHS_1** – источник сигнала запуска выборки/преобразования, в нашем случае используется модуль вывода 1 таймера А.

Для регистра **ADC10CTL0** задаются следующие флаги. **SREF_1** – положительное опорное напряжение задается внутренним источником, отрицательное – потенциалом земли микроконтроллера. **ADC10SHT_3** – время выборки, равное 64 тактам сигнала **ADC10CLK**, по умолчанию внутренний генератор модуля АЦП.

REF2_5V – величина опорного напряжения 2,5 В. **ADC10IE** – разрешить прерывания от модуля АЦП. **REFON** – включить генератор опорного напряжения. **ADC10ON** – включить модуль АЦП.

В строке 9 вызывается функция **__enable_interrupt**, разрешающая прерывания без перехода в режим пониженного энергопотребления. В строке 10 задается период для таймера А. В строках 11 и 12 разрешаются прерывания от этого таймера, после чего он запускается. В строке 13 контроллер переводится в первый режим пониженного энергопотребления (с прерываниями, которые уже были разрешены ранее).

Логика работы здесь следующая: отсчет таймера задает небольшую задержку при старте микроконтроллера, которая позволяет установиться значению опорного источники напряжения. По достижению счетчиком таймера значения **CCR0** происходит прерывание, управление передается соответствующему обработчику (строки 33–38). В обработчике таймер останавливается (в регистр **TACTL** записывается 0), а микроконтроллер выходит из режима пониженного энергопотребления. После чего мы запрещаем прерывание от регистра **CCR0** (строка 14) и опускаем флаг общего разрешения прерываний (строка 15).

В строке 16 мы разрешаем начать преобразования, устанавливаем режим переключения для **CCR1**, так как выше мы задали вывод 1 таймера А в качестве источника сигнала запуска выборки/преобразования (строка 17), а затем запускаем таймер А в режим непрерывного счета (строка 18).

В строке 19 ждем первого сигнала от блока АЦП, то есть значения первого преобразования и записываем его в переменную **first_adc_val** (строка 20). Обратите внимание на то, что в этот момент опущен флаг общего разрешения прерываний, значит, хотя периферийный модуль АЦП и подал сигнал о произошедшем прерывании, оно не будет обработано, а мы не перейдем в функцию обработчика.

После записи первого значения преобразования мы настраиваем вывод светодиода на выход, обнуляем его (светодиод не должен гореть на старте) и переходим в режим пониженного энергопотребления с прерываниями.

В обработчике прерывания от модуля АЦП (строки 25–32) мы сравниваем текущее значение преобразования **ADC10MEM** с первоначальным значением (переменная **first_adc_val**) плюс **ADC_DELTA**, задающее приращение температуры по сравнению с изначальным. Значение 3 примерно равно двум градусам Цельсия.

Шаг 2. Запустите программу на исполнение. Прикоснитесь пальцем к корпусу микроконтроллера так, чтобы площадь соприкосновения была максимальной или поднесите к чипу любой источник тепла (ни в коем случае не используйте открытый огонь!). Дождитесь, пока температура установится, то есть светодиод перестанет мигать.

Мигание демонстрирует упомянутые выше шумы, когда в процессе одного измерения из-за погрешностей оцифровки температура уже выше порогового значения, а на следующем – нет. Уберите источник тепла, дождитесь выключения светодиода.

Задание для самостоятельной работы. Напишите программу, которая линейно изменяет яркость встроенного светодиода при изменении положения подключенного к микроконтроллеру потенциометра. Для достижения линейного изменения яркости используйте код из предыдущей работы. Для выполнения работы соберите следующую электрическую схему (рис. 20).

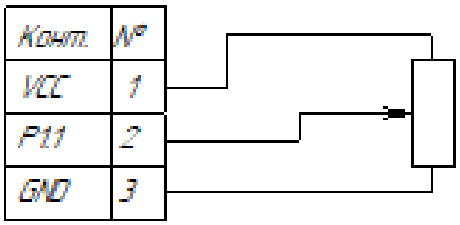

Рис. 20. Схема подключения потенциометра

Контрольные вопросы

1. Что такое АЦП? Перечислите режимы работы АЦП. Какие строки листинга отвечают за инициализацию АЦП?
2. Какие регистры отвечают за управления периферийным блоком АЦП?
3. Для чего необходима функция **`__enable_interrupt?`**

Заключение

Лабораторный практикум позволяет получить студентам современное представление о программировании микропроцессорных устройств, а также закрепить и углубить теоретические знания по дисциплине. Тематика лабораторного практикума охватывает основную часть дисциплины «Микропроцессорные устройства в электроэнергетических объектах», в частности ознакомление с основными принципами и методами работы с программным продуктом, интерфейсами Code Composer Studio. В лабораторном практикуме используются современные технологии проектирования и средства автоматизации для разработки электроэнергетических и электротехнических систем. В теоретической части приведены принципы построения и функционирования микроЭВМ, структурная организация микроконтроллеров; определение методам адресации и системам команд, основам проектирования программного обеспечения встроенных систем; так же содержатся примеры интерфейсов ввода-вывода данных.

Список литературы

1. MSP430x2xx Family. User's Guide. URL: http://www.ti.com/lit/ug/slau144j/slau144j.pdf (дата обращения: 29.06.2021)

2. MSP430G2x53, MSP430G2x13 Mixed Signal Microcontroller (Rev. J). URL: http://www.ti.com/lit/ds/symlink/msp430g2553.pdf (дата обращения: 29.06.2021)

3. Семейство микроконтроллеров MSP430x2xx. Архитектура, программирование, разработка приложений / пер. с англ. А. В. Евстифеева. М.: Додэка-XXI, 2010. 544 с.