Программинг микроконтроллеров

индекс 182,92

MSP430, учимся программировать и отлаживать железо (часть 2)

Здравствуй, уважаемый хабрапользователь.
В предыдущей статье мы начали рассматривать программирование под MSP430.
Материал, описанный в данной статье, позволит в общих чертах ознакомиться с **прерываниями** и понять некоторые тонкости MSP430.

Введение

Прерывание (англ. interrupt) — сигнал, сообщающий процессору о наступлении какого-либо события. При этом выполнение текущей последовательности команд приостанавливается и управление передаётся обработчику прерывания, который реагирует на событие и обслуживает его, после чего возвращает управление в прерванный код.
Механизм прерываний создан для обеспечения максимально оперативной реакции программы на определенные события. Это очень важная часть знакомства с любым микроконтроллером.

Прерывания

Начнём с небольшого примера.

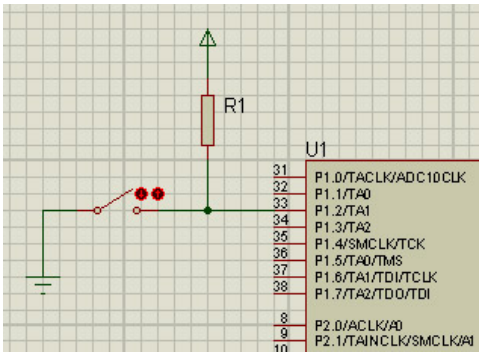
```
1.  #include "msp430f2274.h"
2.
3.  void main()
4.  {
5.    WDTCTL = WDTPW + WDTHOLD;
6.
7.    P1DIR &= ~BIT2;
8.    P1REN |= BIT2;
9.
10.   P1IE |= BIT2; // Разрешение прерываний на P1.2
11.   P1IES |= BIT2; // Прерывание происходит по 1/0 (отпусканию/нажатию)
12.   P1IFG &= ~BIT2; // Очистка флага прерываний для P1.2
13.
14.   P1DIR |= BIT0 + BIT1;
15.   P1OUT |= BIT0;
16.   P1OUT &= ~BIT1;
17.   __bis_SR_register(GIE); // Установка флага глобального разрешения прерыва
      й
18.
19.   while(true);
20. }
```

```
21.
22. #pragma vector=PORT1_VECTOR
23. __interrupt void P1INT() // Обработчик прерывания
24. {
25.   P1IE &= ~BIT2; // Запрет прерываний на P1.2
26.
27.   P1OUT ^= BIT0; // P1.0 меняет своё состояние
28.   P1OUT ^= BIT1; // P1.1 меняет своё состояние
29.
30.   for(volatile unsigned int i = 30000; i != 0; i--); // Задержка
31.   P1IE |= BIT2; // Разрешение прерываний на P1.2
32.   P1IFG &= ~BIT2; // Очистка флага прерываний для P1.2
33. }
34.
```

В данном примере по нажатию на кнопку (P1.2) происходит прерывание, которое меняет состояние двух светодиодов (P1.0 и P1.1). Давайте разберёмся как же это происходит.

PxIE разрешает прерывания для пинов порта Px. Значение «1», записанное в определенный разряд позволяет получать прерывания от определенного пина.

PxIES определяет по какому уровню сигнала будет происходить прерывание. В нашем случае это означает, что единица, помещенная в конкретный разряд данного регистра, позволит получать событие нажатия кнопки. И наоборот, ноль позволит получать событие отпускания кнопки.



Напомню, что кнопка в данном примере «подтянута» до единицы, это означает что нажатая кнопка имеет значение ноль.
На схеме подключения резистор R1 нарисован показательно, на самом деле он находится внутри микроконтроллера и включается регистром P1REN.

PxIFG и есть флаг прерывания. В случае возникновения события нажатия кнопки он будет установлен в единицу, что и вызовет обработчик прерывания (__interrupt void P1INT()). Соответственно, сразу после обработки события флаг необходимо сбросить.
Попробуйте убрать из примера строчку 32 в конце обработчика, и тогда, сразу после первого нажатия кнопки, лампочки будут переключаться так, как будто вы постоянно нажимаете на кнопку.

__bis_SR_register(GIE) устанавливает флаг глобального разрешения прерываний (Global Interrupt Enable) в status register. Фактически это эквивалентно записи SR |= GIE, но мы можем обращаться к регистру SR только по средствам функций __bis_SR_register и __get_SR_register.
Ниже приведена схема битов в SR регистре.

Остановить ACTA в Европе, 11 февраля

Game Development → Visual Studio vNext для разработчика игр

« все лучшие

12.01.2012 → Программинг микроконтроллеров → Digital Metawatch WDS112 оригинальный отладочный комплект от Texas Instruments

21.02.2011 → Программинг микроконтроллеров → Необычный отладочный комплект от Texas Instruments

03.01.2012 → Программинг микроконтроллеров → Комплект разработчика на базе MSP430 от Texas Instruments

08.02.2011 → Железо → Texas Instruments готовит систему-на-кристалле OMAP 5 с поддержкой 3d

14.02.2007 → Гаджеты. Устройства для гиков → Texas Instruments демонстрирует новый мобильный процессор OMAP3430

29.01.2012 → Программинг микроконтроллеров → MSP430, учимся программировать и отлаживать железо

11.09.2007 → Железо → Процессор за $10 от Texas Instruments для мобильных устройств

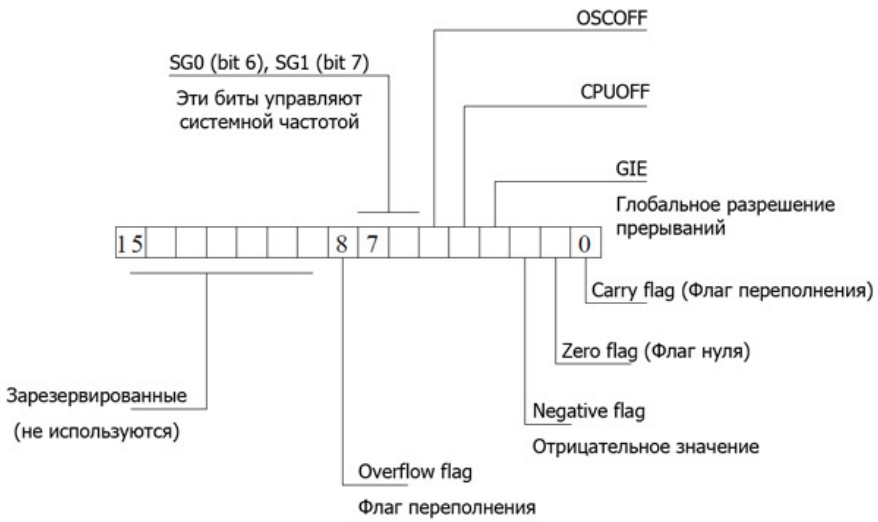
11.01.2007 → Персональные блоги → Texas Instruments сделает мобильник ценой $20

24.01.2012 → DIY или Сделай Сам → Обзор ЛанчПада MSP430 и тестовый проект

18.01.2012 → JAVA → Выпуск №72 — The Art Of Programming [Drinking] Переезд в TEXAS

Kurmunke → Как я был торговцем порно 105

ukko → Пишем обработчик ошибок для



0, 1, 2 и 8 биты — это биты математических операций.
4, 5, 6, 7 — биты управления уровнями энергопотребления, за счёт уменьшения частоты или отключения АЛУ.
Ну и собственно бит **3** — это глобальное разрешение прерываний. Пока этот бит не установлен в единицу, никакие прерывания обрабатываться не будут.

#pragma vector — директива, которая определяет что нижеследующая функция является обработчиком указанного прерывания.
Список всех доступных векторов прерываний можно посмотреть в заголовочном файле для Вашего контроллера. В данном случае это msp430f2274.h.

Дребезг контактов

Про это явление можно подробно прочитать в википедии. Дребезг контактов возникает при нажатии кнопки, благодаря чему обработчик прерываний будет вьывван многократно. В данном примере этого удалось избежать следующим образом:

1. Сразу после вьызова обработчика устанавливается запрет прерываний (строка 25);
2. после переключения состояний светодиодов делается задержка (строка 30);
3. снова устанавливается разрешение прерываний (строка 31).

*Директива **volatile** в 30 строке кода означает, что переменная **i** не будет оптимизироваться при компиляции. Если её убрать, то задержки не будет.*

Заключение

После прочтения предьдущей статьи, коллеги и друзья упрекнули меня в том, что я не уделяю достаточно внимания мелочам. В этот раз я постарался исправить этот недостаток и сделать материал статьи ещё более доступным. Однако это значительно увеличило объем информации.
В следующий раз я постараюсь рассмотреть передачу данных с контроллера на персональный компьютер по средствам программатора и, в зависимости от полученного объема, watchdog.

Я надеюсь, что эта статья оказалась полезна тебе, читатель.

msp430, texas instruments, workbench

— 2 февраля 2012, 00:29 79 kirill89

комментарии (28)

Ekstazi 2 февраля 2012, 03:00 # +1

Спасибо за статью. Вспоминается шуточный код по теме:

```
cli
hlt
```

Прикольная штука, раньше фанател просто от однокристальных микроЭВМ. В одном журнале даже была серия статей про него(не помню названия).

Я так понимаю прерывание возникает при поступлении сигнала на любой из битов любого порта?

Int_13h 2 февраля 2012, 08:29 # ↥ +1

Я так понимаю прерывание возникает при поступлении сигнала на любой из битов любого порта?

Только для лапы 2 порта P1, но можно выбрать при конфигурации и другие лапы и другие порты.

TolTol 2 февраля 2012, 10:00 # ↥ +1

лапа -> вывод

ibnteo 2 февраля 2012, 03:01 # 0

Можете выкладывать содержимое .h файлов из примеров (в данной статье это msp430f2274.h), чтобы можно было повторить программу и в других средах, или даже на ассемблере?

kirill89 2 февраля 2012, 11:35 # ↥ +1

Выложил.
.h файлы для других микроконтроллеров лежат в папке \Program Files\IAR Systems\Embedded Workbench 6.0 Kickstart\430\inc

ibnteo 2 февраля 2012, 17:24 # ↥ 0

Спасибо большое, просто не ставить же этого монстра ради маленького файла, тем более, что он вроде как только в Windows работает. Я изучаю этот МК на ассемблере nasken430asm, компактный быстрый компилятор и отладчик. Просто сам ассемблер нравится, наследник PDP-11, похож на ассемблер БК-0010, на котором я в юности успел попрограммировать.

tipok 2 февраля 2012, 03:20 # 0

Помню, впервые познакомился с этими процессорами, когда искал сверхмаломощные контроллеры. Выбор пал на msp430 после статьи MSP430 Low Power Experiment. Два (2) месяца, или 60 дней контроллер считал на LCD-дисплее цифры от 0 до 9 питаясь от 2х кондёров (ионисторах) по 10Ф.

Ну а «Chronos» может прожить до 2-х лет на одной батарейке, и это при наличии радио модема.

gats 2 февраля 2012, 08:24 # 0

WDTCTL = WDTPW + WDTHOLD;

WTF??

a_v 2 февраля 2012, 11:25 # ↥ +2

Это отключение функции watchdog, которая иначе перезапустит MCU через некоторое время, если за ней не следить. Можно написать так:

```
WDTCTL = WDTPW | WDTHOLD;
```

kirill89 2 февраля 2012, 11:38 # ↥ 0

В следующих статьях обязательно расскажу подробнее про все возможности watchdog. В этот раз не получилось — слишком много материала.

2king2 2 февраля 2012, 09:44 # 0

А почему код примера не на Си? Как я помню поддержка языка была. Си был бы понятен большему кругу людей. ИМХО.

Linol 2 февраля 2012, 10:42 # ↥ +1

Это и есть Си.

AlexLM → Посоветуйте актуальный материал по теме SEO/поискового продвижения 5

IvanFF → Нужн ли цикл статей о финансовой оценке стартапа? 3

Bl00dra1n → Список запрещённых (зарегистрированных) имён пользователей 2

YourChief → Как определить тип NAT, за которым находится собственный комп? 2

« все вопросы

JS-технолог

Консультант для создания логики работы страхового калькулятора

Консультант по SAP MDM

Менеджер интернет проектов

C++ Developer/Team Lead

Senior ASP.NET MVC Developer

PHP программист

IT specialist/ Редактор Интернет-проектов

Project Manager (Москва. от 130 т.р.)

junior C# developer (C# разработчик) - 50-60 т.р. Москва

« все вакансии

Алгоритмы

Ненормальное программирование

Регулярные выражения

Git

Веб-разработка

Sphinx

Разработка

Системное программирование

Game Development

Совершенный код

« все блоги

04 февраля Мастер-класс по разработке приложений для Android

06 февраля QML тренинг в Петрозаводске

07 февраля Начальный курс по разработке сайтов на TPL-шаблонизаторе

07 февраля Семинар по Windows Phone 7

07 февраля Лекция Джона Перри Барлоу, который

kirill89 2 февраля 2012, 11:44 # ↥ 0

Это C++.
Хотя конечно почти никакой специфики C++ в примере нет.
Я очень сильно привык к прелестям ООП и поэтому программы побольше, чем эта, всегда всегда пишу с применением классов.

burjui 2 февраля 2012, 12:51 # ↥ –1

Во-первых, в C давно уже не нужно писать вместо списка аргументов void, если функция не принимает аргументов.

Во-вторых, объявление переменной в цикле

```
for(volatile unsigned int i = 30000; i != 0; i--);
```

есть в стандарте C99.

Во всех остальных частях кода нет даже отдалённого намёка на специфику C++.

kirill89 2 февраля 2012, 13:15 # ↥ 0

Про переменную в цикле я знал, а вот про void — нет. Спасибо.

burjui 2 февраля 2012, 13:45 # ↥ +2

Похоже, я получил незаслуженный плюс. Кто-нибудь, заминусуйте мой верхний коммент, что бы ни дай бог кто-нибудь не воспользовался вредной информацией.

Позор мне за такой пробел в знаниях! Оказывается, если в C оставить список аргументов пустым, то функция может принимать любое число аргументов любых типов. А в C++ это значит, что функция не принимает аргументов. А я уже столько кода написал без void, блин…

Но тогда я не понимаю, как это будет работать? Ведь аргументы кладутся в стек, а компилятор не знает наперёд, сколько аргументов и каких будет в стеке — особенно, если функция будет торчать наружу из какой-нибудь библиотеки.

И главное, как в функции получить доступ к этим аргументам, если макросу **va_start** нужно передать аргумент функции, после которого идёт многоточие?

gats 3 февраля 2012, 08:14 # ↥ 0

Вроде как пустой список аргументов в C задумывался только для прототипирования. В функциях, которым реально нужно обрабатывать переменное количество аргументов таки нужно явно писать первый параметр и многоточие.

KirillovAlex 2 февраля 2012, 13:42 # ↥ +1

Если специфики нет, то надо смотреть какой язык включен в IAR в качестве языка разработки (ALT+F7, далее «General Options->C/C++ Compiler->Language»… там выбор языка), затем нужно, опять же, говорить о компиляторе, а не о языке, ибо компилятор может не полностью соответствовать спецификации, тем более что C++ там поименован как Embedded C++ / extended Embedded.

Следующий вопрос в том, как компилируется код, думаю точнее «свою мысль донести до компилятора» можно на ASM, затем наверно C, потом C++… (Мое личное мнение). Да и вызывать функции класса, чтобы «дрюкнуть» ножкой — как то мне кажется неправильно… Если писать на ASM не сильно продуктивно, и если позволяет память, то, наверное, следует использовать C. То C++ не тот язык, на котором я бы стал писать под MSP430

kirill89 2 февраля 2012, 13:54 # ↥ 0

Когда речь идёт о системах реального времени, в которых необходимо обрабатывать большие объемы данных, то безусловно наилучшем решением является ASM. Ни один компилятор не донесёт до микроконтроллера Ваши мысли лучше Вас самих.
Но если говорить о системах, которые выполняют не большое количество задач, да ещё и с небольшими требованиями к задержкам, то можно использовать любой удобный язык.
В частности C++ в разы повышает скорость разработки. Классы которые я

отписывал раньше, с минимальными изменениями, идут в следующие проекты, при этом не ухудшая читабельность кода. Я не опровергаю мысль о том, что это, может быть, не рационально, но сроки разработки часто играют большую роль.

KirillovAlex 2 февраля 2012, 14:01 # ↥ 0

… также тупо куски кода (без классов) идут в следующие проекты…

kirill89 2 февраля 2012, 14:07 # ↥ 0

С потерей читабельности, а соответственно, сложностями при сопровождении.

KirillovAlex 2 февраля 2012, 16:26 # ↥ 0

Да неее, kirill89, хорошо комментированный код, правильно названные переменные…
Не надо перетягивать одеяло на свою сторону, кому то так удобнее, кому то так…

я вот после 5 лет работы с msp430 уволился с работы,
в другой конторе открыл для себя заново (со времен института) Протеус, ща моделирую в нем… достаточно удобно позволяет при отсутствии железа работать…

Многие нос воротят, а куда деваться, если железки нет…

На прошлой работе всегда железки были… даже вопроса не возникало о симуляторе

Shark 2 февраля 2012, 13:24 # 0

А не могли бы вы как-нибудь таймеры осветить, в частности с режимами вывода на основе таймеров (которые OUTMOD), а то с прерываниями по таймеру разобрался, а это что-то не понимаю.

kirill89 2 февраля 2012, 13:35 # ↥ +1

Хорошо, когда буду писать про таймеры, обязательно включу.

Inquisitor 2 февраля 2012, 13:35 # +1

эм…
1. В разных IDE работа с прерываниями будет разной. Другие макросы или названия обработчиков векторов прерываний.

2. Говоря о прерываеиях обязательно нужно говорить и о сохранении и восстановлении состояний регистров (что делается не всегда автоматически).

3. Прерывания бывают программные.

Ну хотя бы это уж нужно было рассказать)

kirill89 2 февраля 2012, 14:17 # ↥ 0

Действительно, в зависимости от IDE, будут отличия. В прошлой статье я писал о причинах выбора именно этой IDE.

TolTol 3 февраля 2012, 07:51 # 0

> Директива volatile в 30 строке кода означает, что переменная i не будет
> оптимизироваться при компиляции. Если её убрать, то задержки не будет.

По-моему, надо руки обрывать за такое использование volatile.

volatile в первую очередь нужен для глобально-доступных переменных в многопоточной (включая прерывания) среде, при конкурентном доступе.

То, что эта штука у вас сработала не означает, что она правильно использована.

Правильное решение — это сделать функцию на ассемблере, которая будет делать необходимую задержку.

kirill89 3 февраля 2012, 13:25 # ↥ 0

> По-моему, надо руки обрывать за такое использование volatile.

Не будьте так критичны. Я учился работать с MSP430 по официальным примерам от Texas Instruments, и там volatile для организации задержки используется достаточно часто.
Что бы не быть голословным, вот ссылка, откроем, например, файл msp430x22x4_uscia0_irda_01.c, 62 строка:

```
for (i = 1000; i; i--); // Small delay
```

Выше есть объявление (37 строка):

```
volatile unsigned int i;
```