БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе № 2

Прерывания. Таймеры

Вариант №2

Выполнил:

студент группы 950502 Сякачёв П.В.

Проверил:

Богдан Е.В.

Минск 2022

**1 ЦЕЛЬ РАБОТЫ**

Ознакомиться с работой подсистемы прерываний и таймерами микроконтроллера MSP430F5529.

**2 ИСХОДНЫЕ ДАННЫЕ**

В соответствии с вариантом, используя прерывания и таймеры, запрограммировать кнопки и светодиоды. Для работы с кнопками использовать только прерывания. Не использовать опросы флагов состояния в цикле и циклы задержки (активное ожидание). Не допускается использовать иные заголовочные файлы, кроме msp430, не допускается также использовать высокоуровневые библиотеки. При выполнении задания особое внимание уделить грамотному выбору режима работы таймера. Комментарии в тексте программы обязательны, они должны пояснять что именно делает данные фрагмент.

Задание по варианту:





**3 ТЕОРЕТИЧЕСКИЕ СВЕДЕНИЯ**

**3.1 Прерывания**

Различают системные немаскируемые (SMNI), пользовательские немаскируемые (UNMI) и маскируемые прерывания. К системным немаскируемым относятся: сигнал RST/NMI в режиме NMI, сбой генератора, ошибка доступа Flash памяти. К пользовательским немаскируемым – сбой напряжения питания (от подсистемы PMM), доступ к несуществующей (vacant) памяти, события с буфером (mailslot) JTAG интерфейса. Маскируемые прерывания могут быть отключены (замаскированы) индивидуально или все сразу (бит GIE регистра состояния SR).

Кратко рассмотрим, как происходит обработка прерывания. Задержка от возникновения запроса на прерывание до начала выполнения обработчика составляет 6 циклов. При этом заканчивается выполнение текущей инструкции, счетчик команд PC сохраняется в стеке (указывает на следующую команду), регистр состояния SR сохраняется в стеке, выбирается прерывание с максимальным приоритетом (если поступило несколько запросов), автоматически сбрасывается флаг запроса от отдельного прерывания (сброс общего флага запроса должен осуществляться программно). Далее, все биты SR сбрасываются, за исключением SCG0, так как останавливаются все режимы с низким питанием. Так как бит GIO при этом устанавливается в 0, все прерывания запрещаются. Наконец, вектор (адрес обработчика) загружается в PC.

Из-за конвейерной архитектуры процессора, команда, следующая за EINT (разрешение прерывания), всегда выполняется, даже если запрос на прерывание возник до его разрешения. Если за EINT сразу следует DINT, прерывание, ожидающее обработки может быть не обслужено. Команды, следующие за DINT, в этом случае могут сработать некорректно. Аналогичные последствия вызываются альтернативными командами, которые устанавливают и сразу сбрасывают флаг GIE регистра состояний. Рекомендуется вставлять хотя бы одну команду между EINT и DINT. Возврат из прерывания выполняется командой RETI, которая выполняется за 5 циклов и загружает из стека SR, PC. Таблица векторов прерываний располагается по адресам 0FFFFh – 0FF80h и содержит 64 вектора. Бит SYSRIVECT регистра SYSCTL позволяет определить альтернативную таблицу векторов, в старших адресах RAM. По сигналу сброса этот бит автоматически сбрасывается. За прерывания отвечают ряд системных регистров.

Пользовательские маскируемые прерывания рассматриваются отдельно при обсуждении соответствующего функционального узла архитектуры микроконтроллера, в частности, ранее уже рассматривались регистры для работы с прерываниями от цифровых портов ввода-вывода.

Работа с прерываниями достаточно проста. Вначале необходимо разрешить соответствующее прерывание, например, P1IE |= BIT7; - разрешает прерывание по входу 7 вывода порта 1, в экспериментальной плате к нему подключена кнопка S1. После того, как режим инициализирован, хорошим тоном считается перевод контроллера в режим пониженного энергопотребления. Сделать это можно, используя вызов \_\_bis\_SR\_register, например, следующий фрагмент переводит контроллер в режим LPM0 с разрешением прерываний: \_\_bis\_SR\_register(LPM0\_bits + GIE);

Еще одной особенностью запуска в среде отладки Code Composer Studio

является необходимость вызова \_\_no\_operation() перед завершением функции main, если она не использует некоторого цикла. Без этого вызова с завершением функции main завершится и выполнение кода в оболочке.

Собственно обработчик прерывания описывается с использованием директивы #pragma vector. Например, фрагмент кода ниже описывает обработчик прерывания от порта ввода-вывода 1:

#pragma vector=PORT1\_VECTOR

\_\_interrupt void PORT1\_ISR(void)

**3.2 Таймеры**

MSP430F5529 содержит 32-разрядный сторожевой таймер WDT (базовый адрес 015Сh), 3 таймера TAx (базовые адреса соответственно 0340h, 0380h, 0400h), таймер TBx (базовый адрес 03C0h) и таймер часов реального времени RTC\_A (базовый адрес 04A0h).

Основная функция сторожевого таймера WDT – генерация сигнала сброса при программном сбое, например, зацикливании: если заданный интервал времени истек, генерируется сигнал сброса. WDT может быть сконфигурирован как интервальный и генерировать сигналы прерываний по истечении заданного промежутка времени.

Таймер А – это 16-разрядный таймер/счетчик с широкими возможностями по использованию прерываний, которые могут генерироваться счетчиком в случае переполнения и от каждого регистра захвата/сравнения.

Таймер А обладает следующими возможностями:

* асинхронный 16-битный таймер/счетчик с четырьмя рабочими режимами;
* выбираемый и конфигурируемый источник счетного импульса;
* три конфигурируемых регистра захвата/сравнения (в таймере TA0 их 5);
* возможность множественного захвата/сравнения;
* конфигурируемые выводы с возможностью широтно-импульсной

модуляции;

* асинхронная фиксация (защелка) входа и выхода;
* счет по фронту тактового импульса;
* возможность генерации прерываний при переполнении;
* регистр вектора прерываний для быстрого декодирования всех
* прерываний таймера А.

Источниками входного импульса для таймера А могут быть следующие тактовые сигналы: ACLK, SMCLK, внешние CAxCLK, INCLK. На входе имеется программно доступный делитель частоты, который позволяет снижать частоту в 2,3,4,5,6,7,8 раз. Режимы работы таймера: остановка, прямой счет (до уровня TAxCCR0) (Up Mode), непрерывный режим (Continuous Mode), реверсивный счет (Up/Down mode).

Таймер B имеет ряд отличий от таймера А:

* 7 регистров захвата/сравнения;
* разрядность счетчика программируется (8, 10, 12, 16 бит);
* регистр TBxCCRn с двойной буферизацией и может быть сгруппирован;
* все выходы имеют высокоимпедансное состояние;
* не поддерживается бит SCCI.

Таймер часов реального времени RTC\_A представляет собой конфигурируемые часы реального времени с функцией календаря и счетчика общего назначения. Поддерживает выбор формата BCD или двоичный в режиме часов реального времени, имеет программируемый будильник, подстройку коррекции времени, возможность прерываний. Рассмотрим подробнее работу со сторожевым таймером WDT. Он имеет 8 программно выбираемых временных интервалов, поддерживает сторожевой и интервальный режимы, обеспечивает защиту доступа к управляющему регистру, может отключаться для экономии энергии. Важным свойством WDT является отказоустойчивый сигнал (источник счетного сигнала не может быть отключен, пока таймер в сторожевом режиме). Это может не позволить перейти в режим пониженного потребления энергии (LPM).

Регистр счетчика WDT непосредственно программно не доступен. Сигнал на счетный вход может подаваться с тактовых линий SMCLK, ACLK, VLOCLK либо X\_CLK от некоторых устройств. После сброса сторожевой таймер настроен на сторожевой режим, входным выбран сигнал от SMCLK. Поэтому необходимо остановить, установить либо сбросить таймер до истечения установленного интервала, иначе будет сгенерирован сигнал сброса PUC. Флаг запроса на прерывание сбрасывается автоматически после обслуживания, также может быть сброшен программно. Адреса обработчиков в сторожевом и интервальном режиме различны. Разрешение прерываний WDT осуществляется битом WDTIE регистра SFRIE1, флаг прерывания — бит WDTIFG в регистре SFRIFG1.

Рассмотрим особенности работы с таймером А, таймеры В и RTC\_A подробно рассматривать не будем. Структура таймера изображена на рис. 2.1. В последующих таблицах представлены регистры таймера и некоторые поля регистров таймера.

Первым этапом выполняется инициализация таймера TАx с помощью регистров TAxCTL, TAxCCRn и TAxCCTLn. В регистре TAxCTL рекомендуется выбрать в качестве источника тактирования SMCLK с выходной частотой тактирования 1МГц, режим счета, коэффициент деления и установить бит TACLR. В регистре счета/сравнения TAxCCTLn необходимо разрешить прерывания. В 16-битном регистре TAxCCRn указывается значение счетчика, при достижении которого в режиме прямого или реверсивного счета генерируется прерывание. При захвате значения для его сохранения также используется данный регистр. Сброс состояния таймера осуществляется путем записи нулевого значения в конфигурационные регистры. TAxIV – 16-разрядный регистр вектора прерывания. Биты 0-2 регистраTAxEX0 (поле TAIDEX) устанавливают параметры расширенного делителя входа (от деления на 1 при 000b до деления на 8 при 111b).

В режиме прямого счета таймер считает от 0 до значения, установленного в регистре TAxCCR0. При достижении установленного значения таймер продолжает счет с 0. Количество тактовых импульсов в периоде равно TAxCCR0+1. Флаг прерывания TAxCCR0 CCIFG устанавливается, когда счетчик досчитал до значения TAxCCR0. Флаг прерывания TAxCTL TAIFG устанавливается, когда счетчик переходит от TAxCCR0 к 0.

В непрерывном режиме таймер считает от 0 до 0FFFFh. Регистр захвата/сравнения TAxCCR0 работает аналогично остальным регистрам захвата/сравнения. Флаг прерывания TAxCTL TAIFG устанавливается, когда счетчик переходит от 0FFFFh к 0.

Непрерывный режим можно использовать для генерирования независимых выходных интервалов и временных частот. При окончании любого из интервалов, генерируется прерывание. Следующий временной интервал добавляется к TAxCCRn обработчиком прерываний.

В реверсивном режиме таймер считает от 0 до значения, установленного в регистре TAxCCR0. При достижении установленного значения таймер продолжает счет в обратном направлении к 0. Период счета равен удвоенному значению TAxCCR0. Направление счета запоминается, что позволяет выполнять остановку таймера, а затем продолжить счет с прерванного места. Флаг прерывания TAxCCR0 CCIFG устанавливается, когда счетчик досчитал до значения TAxCCR0. Флаг прерывания TAxCTL TAIFG устанавливается, когда счетчик досчитал в обратном направлении от TAxCCR0 до 0.

Реверсивный режим позволяет поддерживать пустые интервалы (Dead Time) между выходными сигналами, когда ни один из них не активен. Регистры TAxCCRn не имеют буфера, поэтому они изменяются сразу после записи.

Режим захвата выбирается, когда CUP = 1. Используется для записи времени какого-либо события. В качестве сигналов на входы захвата CCIxA и CCIxB могут быть поданы сигналы с внешних выводов или внутренние сигналы. Источник выбирается CCIS битами. Биты CM определяют, будет ли захват происходить по фронту сигнала, по спаду, либо и по фронту, и по спаду. При наступлении соответствующего события значение счетчика копируется в регистр TAxCCRn и устанавливается флаг прерываний CCIFG. Уровень входного сигнала может быть прочитан в любое время из бита CCI. Поскольку сигнал на входе не синхронизирован с тактовыми импульсами, могут возникать гонки. Поэтому рекомендуется устанавливать бит SCS, чтобы захват происходил с началом очередного тактового импульса. Захват может быть выполнен программно. Режим сравнения выбирается, когда CUP = 0. Используется для генерации на выходе ШИМ-сигнала или прерывания через заданный временной интервал. Когда счетчик достигает значения TAxCCRn, устанавливается флаг прерывания CCIFG, внутренний сигнал EQUn устанавливается в 1, EQUn влияет на выход в соответствии с режимом, а входной сигнал CCI защелкивается в регистре SCCI.

Каждый блок захвата/сравнения содержит выходной модуль, который формирует выходной сигнал на основе EQU0 и EQUn сигналов в зависимости от установленного режима выхода. Биты OUTMOD позволяют задать один из 8 режимов. Сигнал OUTn изменяется по переднему фронту синхросигнала, за исключением режима 0. Режимы 2, 3, 6 и 7 не пригодны для использования с выходным блоком 0, поскольку EQUn = EQU0.

Режимы выхода:

000 — Значение бита OUT. Сигнал OUTn изменяется сразу же с изменением бита OUT;

001 — Установка. Однократная установка при достижении заданного значения TAxCCRn;

010 — Переключение/сброс. Выход меняется при достижении значения TAxCCRn, сбрасывается при достижении TAxCCR0;

011 — Установка/сброс. Выход устанавливается при достижении значения TAxCCRn, сбрасывается при достижении TAxCCR0;

100 — Переключение. Выход меняется при достижении значения TAxCCRn;

101 — Сброс. Однократный сброс при достижении заданного значения TAxCCRn;

110 — Переключение/установка. Выход меняется при достижении значения TAxCCRn, устанавливается при достижении TAxCCR0;

111 — Сброс/установка. Выход сбрасывается при достижении значения TAxCCRn, устанавливается при достижении TAxCCR0.

При использовании констант (msp430f5529.h) не стоит забывать принципы их именования:

* константа, соответствующая биту поля-флага, именуется по имен поля,
* например, полю CPUOFF регистра состояния процессора SR (бит 4)
* соответствует константа CPUOFF;
* константа, соответствующая биту n в поле NNN, именуется NNNn;
* константа, соответствующая номеру x выбранного варианта для поля
* NNN именуется NNN\_x;
* константа, соответствующая выбранному режиму zz для поля NNN
* именуется NNN\_\_zz.

Так, например, для 3-битного поля SELA, константа, соответствующая 0 биту поля, именована SELA0, вариант выбора 0 (SELA = 000) именован SELA\_0, а режим, соответствующий данному варианту именован SELA\_\_XT1CLK. В некоторых случая поля задают делители либо множители, соответствующие степени двойки. Тут надо быть особо внимательным и не спутать похожие мнемоники, например, NN4 (четвертый бит, т.е. 10000), NN\_4(четвертый вариант, т.е. 00100), NN\_\_4 (режим деления на 4, т.е. 00011).

**4 ВЫПОЛНЕНИЕ РАБОТЫ**

#include <msp430.h>

const int interval = 3;

int index = -1, count=0, st1=0, st2=0, st3=0,st4=0, st5 = 0;

int buttonState = 0;

int limit = 0, time = 0;

int main(void)

{

WDTCTL = WDT\_ADLY\_1000;

P1OUT = 0;

P1DIR |= BIT1;

P1DIR |= BIT2;

P1DIR |= BIT3;

P1DIR |= BIT4;

P1DIR |= BIT5;

P1REN |= BIT7;

P1OUT |= BIT7;

SFRIE1 |= WDTIE;

P1IE |= BIT7;

\_\_bis\_SR\_register(GIE);

P1IFG = 0;

P1IES |= BIT7;

P1OUT &= ~BIT1;

index = 0;

\_\_no\_operation();

return 0;

}

#pragma vector = WDT\_VECTOR

\_\_interrupt void WDTINT()

{

count++;

if(count >= st1 && st1 != 0)

{

P1OUT &= ~BIT1;

st1 = 0;

}

if(count >= st2 && st2 != 0)

{

P1OUT &= ~BIT2;

st2 = 0;

}

if(count >= st3 && st3 != 0)

{

P1OUT &= ~BIT3;

st3 = 0;

}

if(count >= st4 && st4 != 0)

{

P1OUT &= ~BIT4;

st4 = 0;

}

if(count >= st5 && st5 != 0)

{

P1OUT &= ~BIT5;

st5 = 0;

}

}

#pragma vector=TIMER1\_A0\_VECTOR

\_\_interrupt void SINT()

{

if(limit >= 2)

{

time = 0;

limit = 0;

TA1CCTL0 = CCIE;

}

++limit;

}

#pragma vector = PORT1\_VECTOR

\_\_interrupt void P1INT()

{

if(time == 0)

{

TA1CTL = TASSEL\_2 + MC\_1 + ID\_1 + TACLR;

TA1CCR0 = 10000;

TA1CCTL0 = CCIE;

++time;

++index;

if(index == 1)

{

P1OUT |= BIT1;

st1 = count + interval;

}

else if(index == 2)

{

P1OUT |= BIT2;

st2 = count + interval;

}

else if(index == 3)

{

P1OUT |= BIT3;

st3 = count + interval;

}

else if(index == 4)

{

st4 = count + interval;

P1OUT |= BIT4;

}

else if(index == 5)

{

st5 = count + interval;

P1OUT |= BIT5;

}

if(index > 4)

{

index = 0;

}

P1IES ^= BIT7;

}

P1IFG = 0;

}

**5 ВЫВОДЫ**

В ходе выполнения лабораторной работы была написана программа с использованием таймеров и прерываний в соответствии с заданием варианта.