БГУИР

Кафедра ЭВМ

Отчет по лабораторной работе № 2
Тема: «Прерывания. Таймеры»
Вариант №8

Выполнил:
студент группы 150503 Ходосевич М.А.

Проверил:
ассистент каф. ЭВМ _____ Шеменков В.В.

Минск
2024

1. Постановка задачи

Написать программу, используя таймеры и прерывания, в соответствии с заданием варианта. Задание 8 варианта: реализовать режим потухания с использованием LED1-LED3, S2 и WDT. При нажатии на кнопку все диоды включаются, при отпускании - гаснут друг за другом с небольшой задержкой.

2. Теоретические сведения

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).

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)
```

MSP430F5529 содержит 32-разрядный сторожевой таймер WDT (базовый адрес 015Ch), 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 В 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.

3. Листинг программы

```
#include <msp430.h>

volatile int button_pressed = 0;  // Флаг нажатия кнопки
volatile int led_off_step = 0;    // Шаг выключения светодиодов
volatile int delay_counter = 0;    // Счетчик для увеличения задержки

void setup_watchdog_timer() {
    WDTCTL = WDTPW | WDTTMSEL | WDTCNTCL | WDT_MDLY_32; // Таймер WDT с
интервалом 32 мс
    SFRIE1 |= WDTIE; // Разрешаем прерывания от WDT
}

void stop_watchdog_timer() {
    SFRIE1 &= ~WDTIE;  // Отключаем прерывание WDT
    WDTCTL = WDTPW | WDTHOLD;  // Останавливаем WDT
}

void main(void) {
    WDTCTL = WDTPW | WDTHOLD;

    P1DIR |= BIT0;
    P8DIR |= BIT1;
    P8DIR |= BIT2;

    // Настройка кнопки
    P2DIR &= ~BIT2;
    P2REN |= BIT2;
    P2OUT |= BIT2;
    P2IE |= BIT2;   // Разрешаем прерывание по кнопке
    P2IES |= BIT2;  // Прерывание по спадающему фронту (нажатие)
    P2IFG &= ~BIT2; // Сбрасываем флаг прерывания

    __enable_interrupt(); // Разрешаем глобальные прерывания

    // Начальное состояние светодиодов
    P1OUT &= ~BIT0;
    P8OUT &= ~BIT1;
    P8OUT &= ~BIT2;

    while (1) {
        // Основной цикл пустой, так как вся работа идет в прерываниях
        __low_power_mode_3(); // Включаем режим энергосбережения
    }
```

```
    }

    // Прерывание по нажатию или отпусканию кнопки
    #pragma vector=PORT2_VECTOR
    __interrupt void PORT2_ISR(void) {
        if (!(P2IN & BIT2)) {  // Кнопка нажата (спадающий фронт)
            button_pressed = 1;      // Устанавливаем флаг нажатия кнопки
            led_off_step = 0;        // Сбрасываем шаг гашения светодиодов
            delay_counter = 0;       // Сбрасываем счетчик задержки
            // Включаем все светодиоды
            P1OUT |= BIT0;
            P8OUT |= BIT1;
            P8OUT |= BIT2;
            stop_watchdog_timer();   // Останавливаем WDT, если был запущен
            P2IES &= ~BIT2;          // Меняем на прерывание по
возрастающему фронту (отпускание)
        } else {                     // Кнопка отпущена (возрастающий фронт)
            button_pressed = 0;      // Сбрасываем флаг нажатия кнопки
            led_off_step = 0;        // Сбрасываем шаг гашения светодиодов
            delay_counter = 0;       // Сбрасываем счетчик задержки
            setup_watchdog_timer();  // Запускаем WDT для последовательного
гашения светодиодов
            P2IES |= BIT2;           // Меняем на прерывание по спадающему
фронту (нажатие)
        }

        P2IFG &= ~BIT2; // Сбрасываем флаг прерывания
    }

    // Прерывание от таймера WDT
    #pragma vector=WDT_VECTOR
    __interrupt void WDT_ISR(void) {
        if (!button_pressed) {  // Если кнопка отпущена
            delay_counter++;    // Увеличиваем счетчик задержки

            if (delay_counter >= 10) { // Каждые 10 раз по 32 мс = 1 секунда
                switch (led_off_step) {
                case 0:
                    P1OUT &= ~BIT0; // Выключаем LED1
                    led_off_step++;
                    break;
                case 1:
                    P8OUT &= ~BIT1; // Выключаем LED2
                    led_off_step++;
                    break;
                case 2:
                    P8OUT &= ~BIT2; // Выключаем LED3
                    led_off_step++;
                    stop_watchdog_timer();  // Останавливаем WDT после
завершения
                    break;
                default:
                    stop_watchdog_timer();  // Останавливаем WDT, если шаг
превышает 2
                    break;
                }
                delay_counter = 0; // Сбрасываем счетчик после каждого шага
гашения
            }
        }
    }
```

4. Заключение

В ходе выполнения лабораторной работы удалось ознакомиться с работой прерываний и таймерами микроконтроллераа MSP-EXP430F5529. Удалось написать программу с использованием таймеров и прерываний в соответствии с вариантом №8.