**МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ **«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ ИМ. В. Г. ШУХОВА»**

Кафедра программного обеспечения вычислительной техники и автоматизированных систем

**ЛАБОРАТОРНАЯ РАБОТА №4**

**Дисциплина: ЭВМ и периферийные устройства**

**Тема: Изучение принципов обработки прерываний на примере управления встроенными в микроконтроллер таймерами-счетчиками и компаратором**

Выполнил: ст. группы ВТ-31

Подкопаев Антон Валерьевич

Проверил: доцент кафедры ПО и ВТАС

Шамраев Анатолий Анатольевич

**Белгород 2020**

**Цель работы:** изучить принципы разработки процедур обработки прерываний в микроконтроллере MSP430F1xxx, ознакомиться с принципами функционирования встроенных в микроконтроллер 16 – разрядных таймеров-счетчиков и компаратора для измерения сопротивления резистивного датчика.

**Указания по организации самостоятельной работы**

Перед работой необходимо проработать теоретический материал по литературе и конспект лекций, изучить принципы разработки процедур обработки прерываний в микроконтроллере MSP430F1xxx, ознакомиться с возможностями функционирования встроенных в микроконтроллер 16 - разрядных таймеров-счетчиков и компаратора, изучить алгоритмы формирования временных задержек с помощью прерываний от таймеров-счетчиков.

**Порядок проведения работы и указания по ее выполнению**

Перед началом выполнения практической части лабораторной работы проводится экспресс-контроль знаний по принципам функционирования таймеров/счетчиков и компаратора, входящих в состав микроконтроллера MSP430, а также по способам задания и измерения временных интервалов.

**Задание.** Разработать в среде программирования IAR Embedded Workbench программу на языке С для микроконтроллера MSP430, которая обеспечивает измерение сопротивления переменного резистора и выводит рассчитанное значение на ЖКИ. Для решения задачи необходимо использовать встроенный компаратор и таймер А в режиме захвата.

**Порядок выполнения задания:**

– включить лабораторный макет.

– запустить компилятор IAR Embedded Workbench.

– создать пустой проект.

– создать файл ресурса для кода программы и подключить его к проекту.

– ввести код исходного модуля программы измерения сопротивления переменного резистора.

– выполнить компиляцию исходного модуля программы и устранить ошибки, полученные на данном этапе.

– настроить параметры программатора.

– создать загрузочный модуль программы и выполнить программирование микроконтроллера.

– поверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.

Теоретические сведения

**Маскируемые прерывания**

Маскируемые прерывания вызываются периферийными устройствами, имеющими возможность вызова прерываний, включая ситуацию переполнения сторожевого таймера в активном режиме. С помощью индивидуальных битов разрешения прерывания можно отключать источники прерываний как по отдельности, так и все сразу с использованием общего бита разрешения всех прерываний (GIE) в регистре статуса (SR).

**Обработка прерывания**

Если периферийное устройство запросило прерывание и включены биты общего разрешения прерываний GIE и индивидуальный бит разрешения прерывания от этого устройства, будет вызвана процедура обработки прерывания. Для вызова немаскируемого (NMI) прерывания достаточно установки только индивидуального бита разрешения прерывания.

**Получение прерывания**

Время задержки вызова прерывания составляет 6 машинных циклов, с момента приема запроса на прерывание и до начала выполнения первой команды процедуры обработки прерывания. Логика обработки запроса прерывания имеет следующую последовательность:

– любая текущая команда выполняется до конца;

– содержимое программного счетчика PC, указывающего на следующую команду, помещается в стек;

– содержимое регистра статуса SR помещается в стек;

– если поступило несколько прерываний во время выполнения последней команды, обрабатывается прерывание с наивысшим приоритетом, остальные ожидают обслуживания;

– автоматически сбрасывается флаг одного источника прерывания. Флаги запроса остальных прерываний остаются установленными в ожидании обслуживания программным обеспечением.

**16-разрядный таймер-счетчик**

Регистр 16-разрядного таймера/счетчика TAR, инкрементируется или декрементируется (в зависимости от режима работы) с каждым нарастающим фронтом тактового сигнала. TAR может быть программно прочитан и записан. Кроме того, таймер может генерировать прерывание при переполнении. TAR можно очистить установкой бита TACLR. Установка TACLR также очищает делитель тактовой частоты и направление счета для режима вверх/вниз.

**Прерывания Таймера А**

С 16-разрядным модулем таймера А связаны два вектора прерываний:

– вектор прерывания TACCR0 для флага TACCR0 CCIFG;

– вектор прерывания TAIV для всех других флагов CCIFG и TAIFG.

В режиме захвата любой флаг CCIFG устанавливается, когда значение таймера зафиксировано в соответствующем регистре TACCRx. В режиме сравнения устанавливается любой флаг CCIFG, если TAR досчитал до соответствующего значения TACCRx. Программное обеспечение может также устанавливать или очищать любой флаг CCIFG. Все флаги CCIFG запрашивают прерывание, когда установлены их соответствующие биты CCIE и бит GIE.

**Прерывания компаратора А**

С компаратором А связан один флаг прерывания и один вектор прерывания, как показано на рисунке 4.4.14. Флаг прерывания CAIFG устанавливается по любому фронту (нарастающему или спадающему) сигнала на выходе компаратора, что определяется битом CAIES. Если установлены оба бита CAIE и GIE, флаг CAIFG генерирует запрос прерывания. Флаг CAIFG автоматически сбрасывается, когда обрабатывается запрос прерывания или может быть сброшен программно.

Изображение выглядит как снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана

Автоматически созданное описание

Изображение выглядит как снимок экрана

Автоматически созданное описание

**Вывод:** я изучил принцип работы таймеров A и B, а также компаратора в микроконтроллерах семейства MSP430F16xx, а также принцип измерения сопротивления резистора.

*Приложение*

analogsensors.c

// Analog sensors functions

**#include** "function\_prototype.h"

**#include** "sysfunc.h"

**#include** "analogsensors.h"

**const** **float** HIH\_zero\_offset = 0.958; // параметр "начальное смещение" датчика влажности, В

**const** **float** HIH\_slope = 0.03068; // параметр "угол наклона датчика", В / %RH

**const** **float** HIH\_ion = 3.3; // опорное напряжение, В

**const** **float** HIH\_divisor = 1.1; // коэффициент резистивного делителя

**const** **float** INA\_RS = 0.21; // измерительное сопротивление, Ом

**const** **float** INA\_RL = 30.1; // сопротивление нагрузки, Ом

// Получить значение относительной влажности, %RH

**float** **HIH\_get\_hum**()

{

P6SEL |= BIT0; // выбор для ножки P6.0 функции АЦП ADC0, к которому подключен датчик влажности

ADC12CTL1 = SHP + CSTARTADD\_0; // таймер выборки и стартовый адрес преобразования - ADC12MEM0

// выбор опорного напряжения - Vr+ = VеREF+ = 3.3В, Vr- = AVss = 0В

// и входного канала ADC0 для ячейки памяти ADC12MEM0

ADC12MCTL0 = SREF\_3 + INCH\_0;

ADC12CTL0 = ADC12ON; // включение АЦП

ADC12CTL0 |= ENC; // преобразование разрешено

ADC12CTL0 |= ADC12SC; // запуск преобразования

**while** ((ADC12IFG & BIT0)==0); // ожидание результата преобразования

// пересчет результата преобразования АЦП в значение влажности

// с учетом делителя и опорного напряжения

**float** rh = (((ADC12MEM0/4095.0) \* HIH\_ion \* HIH\_divisor) - HIH\_zero\_offset) / HIH\_slope;

ADC12CTL0 = 0; // выключение АЦП

**return** rh;

}

// Получить значение тока потребления системы, А

**float** **INA\_get\_curr**()

{

P6SEL |= BIT1; // выбор АЦП ADC1, к которому подключен датчик тока

ADC12CTL1 = SHP + CSTARTADD\_1; // таймер выборки и стартовый адрес преобразования - ADC12MEM1

// выбор опорного напряжения - Vr+ = VеREF+ = 3.3В, Vr- = AVss = 0В

// и входного канала ADC1 для ячейки памяти ADC12MEM1

ADC12MCTL1 = SREF\_3 + INCH\_1;

ADC12CTL0 = ADC12ON; // включение АЦП

ADC12CTL0 |= ENC; // преобразование разрешено

ADC12CTL0 |= ADC12SC; // запуск преобразования

**while** ((ADC12IFG & BIT1)==0); // ожидание результата преобразования АЦП ADC1

// пересчет результата преобразования АЦП в значение тока потребления системы

// с учетом измерительного сопротивления и сопротивления нагрузки:

**float** curr = (ADC12MEM1\*3.3) / (4095.0 \* INA\_RS \* INA\_RL);

ADC12CTL0 = 0; // выключение АЦП

**return** curr;

}

// Получить значение сопротивления подстроечного резистора R22, Ом

word **R22\_get\_resistance**()

{

P2SEL &= ~(Rref+Rx); // функция ввода-вывода для ножек P2.4 и P2.5

word Nref = res\_measure(Rref); // время разряда через опорный резистор

word Nx = res\_measure(Rx); // время разряда через подстроечный резистор

**return** ((100000\*Nx)/Nref)-10000; // R22 = (100000 \* Nx / Nref) - 10000

}

// Измерение времени разряда конденсатора через resistor (Rref или Rx)

word **res\_measure**(byte Rpin)

{

P2DIR &= ~Rx; // отключить Rx от конденсатора (направление - ввод)

// заряд конденсатора через опорный резистор Rref

CAPD = ~Rref; // отключение аналоговых сигналов от порта компаратора

P2DIR |= Rref; // подключить Rref к конденсатору (направление - на вывод)

P2OUT |= Rref; // установка ножки Rref- заряд кондесатора

TACCR1 = 65000; // время заряда

TACCTL1 = CCIE; // разрешить прерывания

// тактирование от SMCLK, делитель /4, очистка счетчика, непрерывный режим счета

TACTL = TASSEL\_2 + ID\_2 + TACLR + MC\_2;

LPM0; // перейти в режим пониженного потребления и ожидать прерывания

CACTL2 = P2CA0 | CAF; // вход компаратора подключается к CA0, вкл.выходного фильтра

// включение компаратора, опорное напр. 0.25\*Vcc прикладывается к "-"

CACTL1 = CARSEL+CAREF\_1+CAON;

CAPD = ~(Rpin+CA0);

P2DIR &= ~Rref; // отключить Rref от конденсатора (направление - ввод)

P2DIR |= Rpin; // будем разряжать через ножку Rpin

P2OUT &= ~Rpin; // низкий уровень на Rpin - разряд конденсатора

// захват по заднему фронту, входной сигнал - CCI1B, режим захвата, прерывания разрешены

TACCTL1 = CM\_2+CCIS\_1+CAP+CCIE;

TACTL |= TACLR; // сбросить счетчик таймера

LPM0; // перейти в режим пониженного потребления и ожидать прерывания

TACTL = 0x00; // остановить таймер

CACTL1 = 0x00; // отключить компаратор

CAPD = 0; // включить входные буферы компаратора

**return** TACCR1; // возвращаем значение счетчика таймера

}

// обработчик прерываний от таймера

**#pragma** vector=TIMERA1\_VECTOR

**\_\_interrupt** **void** **isrTIMERA**(**void**)

{

LPM0\_EXIT; // выход из LPM0

TACCTL1 &= ~CCIFG; // очистка флага прерывания

}

lcd.c

// LCD-display functions

**#include** "function\_prototype.h"

**#include** "sysfunc.h"

**#include** "lcd.h"

//Таблица киррилицы

**char** LCD\_table[64]={

0x41,0xA0,0x42,0xA1, //0xC0...0xC3 <=> А Б В Г

0xE0,0x45,0xA3,0x33, //0xC4...0xC7 <=> Д Е Ж З

0xA5,0xA6,0x4B,0xA7, //0xC8...0xCB <=> И Й К Л

0x4D,0x48,0x4F,0xA8, //0xCC...0xCF <=> М Н О П

0x50,0x43,0x54,0xA9, //0xD0...0xD4 <=> Р С Т У

0xAA,0x58,0xE1,0xAB, //0xD5...0xD7 <=> Ф Х Ц Ч

0xAC,0xE2,0xAC,0xAE, //0xD8...0xDB <=> Ш Щ Ъ Ы

0x62,0xAF,0xB0,0xB1, //0xDC...0xDF <=> Ь Э Ю Я

0x61,0xB2,0xB3,0xB4, //0xE0...0xE4 <=> а б в г

0xE3,0x65,0xB6,0xB7, //0xE5...0xE7 <=> д е ж з

0xB8,0xA6,0xBA,0xBB, //0xE8...0xEB <=> и й к л

0xBC,0xBD,0x6F,0xBE, //0xEC...0xEF <=> м н о п

0x70,0x63,0xBF,0x79, //0xF0...0xE4 <=> р с т у

0xE4,0xD5,0xE5,0xC0, //0xF5...0xE7 <=> ф х ц ч

0xC1,0xE6,0xC2,0xC3, //0xF8...0xEB <=> ш щ ъ ы

0XC4,0xC5,0xC6,0xC7 //0xFC...0xEF <=> ь э ю я

};

byte LCD\_row, LCD\_col, n;

**void** **LCD\_init**()

{

wait\_1ms(20); // пауза 20 мс после включения модуля

P3DIR |= (D\_nC\_LCD + EN\_LCD); // Настроить порты, которые управляют LCD на вывод

Reset\_EN\_LCD(); // Перевести сигнал "Разрешение обращений к модулю LCD" в неактивное состояние

// Команда Function Set 0 0 1 DL N F \* \*

// установка разрядности интерфейса DL=1 =>8, бит DL=0 =>4 бит

// N=1 => две строки символов, N=0 => одна строка символов

// F=0 => размер шрифта 5х11 точек, F=1 => размер шрифта 5х8 точек

// Выбор режима передачи команд для LCD и вывод байта без ожидание броса влага BF

LCD\_WriteCommand(0x3C);

wait\_1ms(1);

LCD\_WriteCommand(0x3C);

wait\_1ms(1);

// Команда Display ON/OFF control 0 0 0 0 1 D C B

// включает модуль D=1 и выбирает тип курсора (C,D)

// C=0, B=0 - курсора нет, ничего не мигает

// C=0, B=1 - курсора нет, мигает весь символ в позиции курсора

// C=1, B=0 - курсора есть (подчеркивание), ничего не мигает

// C=1, B=1 - курсора есть (подчеркивание), и только он и мигает

LCD\_WriteCommand(0x0C);

LCD\_clear();

// Команда Entry Mode Set 0 0 0 0 0 1 ID SH

// установка направления сдвига курсора ID=0/1 - сдвиг влево/вправо

// и разрешение сдвига дисплея SH=1 при записи в DDRAM

LCD\_WriteCommand(0x06);

}

//Вывод сообщение на LCD дисплей

**void** **LCD\_message**(**const** **char** \* buf)

{

n = 0;

**while** (buf[n])

{

// если выходим за границу строки - переход на следующую

**if** ( (LCD\_row < LCD\_MAXROWS-1) && (LCD\_col >= LCD\_MAXCOLS) )

LCD\_set\_pos(++LCD\_row, 0);

**if** (LCD\_col >= LCD\_MAXCOLS )

LCD\_set\_pos(0,0); // если вышли за границы экрана - начинаем с начала

// break; // или если вышли за границы экрана - перестаем выводить символы

LCD\_WriteData( LCD\_recode(buf[n]) );

LCD\_col++;

n++;

}

}

// Функция очистки экрана

**void** **LCD\_clear**()

{

// Команда Clear Display 0 0 0 0 0 0 0 1

// очищает модуль и помещает курсор в самую левую позицию

LCD\_WriteCommand(0x01);

LCD\_row=0;

LCD\_col=0;

}

// Установка позиции курсора:

// row - номер строки (0...1)

// col - номер столбца (0...15)

**void** **LCD\_set\_pos**(byte row, byte col)

{

**if** (row > LCD\_MAXROWS-1) // проверка на неправильные значения

row = LCD\_MAXROWS-1;

**if** (col > LCD\_MAXCOLS-1) // проверка на неправильные значения

col = LCD\_MAXCOLS-1;

LCD\_row = row;

LCD\_col = col;

LCD\_WriteCommand( BIT7 | ((0x40 \* LCD\_row) + LCD\_col) );

}

byte **LCD\_get\_row**()

{

**return** LCD\_row;

}

byte **LCD\_get\_col**()

{

**return** LCD\_col;

}

// Устновка режима отображения курсора:

// 0 - курсора нет, ничего не мигает

// 1 - курсора нет, мигает весь символ в позиции курсора

// 2 - курсор есть(подчеркивание), ничего не мигает

// 3 - курсор есть(подчеркивание) и только он мигает

**void** **LCD\_set\_cursor**(byte cursor)

{

**if** (cursor > 3) // проверка на неправильные значения

cursor = 2;

LCD\_WriteCommand(cursor | BIT2 | BIT3); // Выполняем команду Display ON/OFF Control

// с нужным режимом отображения курсора

}

**void** **LCD\_WriteCommand**(**char** byte)

{

// Выбор режима передачи команд для LCD и вывод байта

LCD\_WriteByte(byte, 0); //

}

**void** **LCD\_WriteData**(**char** byte)

{

// Выбор режима передачи данных LCD и вывод байта

LCD\_WriteByte(byte, 1);

}

// Вывод байта на индикатор, параметры:

// byte - выводимый байт

// dnc=0 - режим передачи команд, dnc=1 - данных

**void** **LCD\_WriteByte**(**char** byte, **char** D\_nC)

{

DB\_DIR = 0x00; // Шина данных на прием

Set\_MCU\_SEL\_0(); // Выбор модуля LCD MCU\_SEL\_0 = 1

Set\_MCU\_SEL\_1(); // при помощи дешифратора DD7 MCU\_SEL\_0 = 1

// \_

Reset\_D\_nC\_LCD(); // Выбор режима передачи команд для LCD D/C\_LCD = 0

// \_\_ \_\_\_ \_

Set\_nWR\_nRST(); // Cигал WR/RST = 1 => сигнал R/W\_LCD = 1, т.е. в неактивном состоянии

// \_\_

Reset\_nSS(); // Сформировать сигал "OE\_BF\_LCD" SS = 0

Set\_EN\_LCD(); // Сформировать строб данных для LCD EN\_LCD = 1 | |

Set\_EN\_LCD(); // Сформировать строб данных для LCD EN\_LCD = 1 | |

Set\_EN\_LCD(); // Сформировать строб данных для LCD EN\_LCD = 1 \_\_\_\_| |\_\_\_\_\_

**while** (DB\_IN & BIT7); // ожидание сброса флага занятости BUSY

Reset\_EN\_LCD(); // Перевести сигнал "EN\_LCD\_OUT" в неактивное состояние EN\_LCD = 0

// \_\_

Set\_nSS(); // Перевести сигнал "OE\_BF\_LCD" в неактивное состояние SS = 1

**if** (D\_nC) Set\_D\_nC\_LCD(); // Выбрать режим записи данных (D\_nC = 1)

**else** Reset\_D\_nC\_LCD(); // или записи команды (D\_nC = 0)

// \_\_ \_\_\_ \_

Reset\_nWR\_nRST(); // Сформировать сигал WR/RST = 0 => R/W\_LCD = 0

// \_\_

Reset\_nSS(); // Сформировать сигал "OE\_BF\_LCD" SS = 0

DB\_DIR = 0xFF; // Шина данных на выход

DB\_OUT = byte; // Выставить данные на шину данных

// \_\_\_\_\_

Set\_EN\_LCD(); // Сформировать строб данных для LCD EN\_LCD = 1 | |

Set\_EN\_LCD(); // Сформировать строб данных для LCD EN\_LCD = 1 | |

Set\_EN\_LCD(); // Сформировать строб данных для LCD EN\_LCD = 1 \_\_\_\_| |\_\_\_\_\_

Reset\_EN\_LCD(); // Перевести сигнал "EN\_LCD\_OUT" в неактивное состояние EN\_LCD = 0

\_\_

Set\_nSS(); // Перевести сигнал OE\_BF\_LCD =1 в неактивное состояние SS = 1

DB\_DIR = 0x00; // Шина данных на вход

// \_\_ \_\_\_ \_

Set\_nWR\_nRST(); // Cигал WR/RST = 1 => сигнал R/W\_LCD = 1, т.е. в неактивном состоянии

}

//Функция перекодировки символа в киррилицу

**char** **LCD\_recode**(**char** b)

{

**if** (b<192) **return** b;

**else** **return** LCD\_table[b-192];

}

main.c

**#include** <msp430.h>

**#include** "stdio.h"

**#include** "system\_define.h"

**#include** "system\_variable.h"

**#include** "function\_prototype.h"

**#include** "main.h"

**int** **wordToStr**(**char** \*str, **unsigned** num){

**int** i = 0, j;

**do**{

str[i++] = '0' + num%10;

num = num/10;

} **while** (num > 0);

str[i] = '\0';

**char** t;

**for**(j = 0; j < i/2; j++){

t = str[j];

str[j] = str[i-1-j];

str[i-1-j] = t;

}

**return** i;

}

**#include** <msp430.h>

**#include** "stdio.h"

**#include** "system\_define.h"

**#include** "system\_variable.h"

**#include** "function\_prototype.h"

**#include** "main.h"

**void** **main**(**void**) {

WDTCTL = WDTPW|WDTHOLD;

Init\_System\_Clock();

Init\_System();

\_enable\_interrupt();

LCD\_init();

**char** message[32];

**unsigned** res, last\_res = 0;

**while**(1){

res = R22\_get\_resistance();

**if** (res != last\_res){

LCD\_clear();

**sprintf** (message, "%u", res);

LCD\_message(message);

last\_res = res;

}

wait\_1ms(100);

}

}

sysfunc.c

// System functions

**#include** <msp430.h>

**#include** "sysfunc.h"

// инициализация портов системы

**void** **Init\_System**()

{

P1DIR |= (nSS + nWR\_nRST + MCU\_SEL\_0 + MCU\_SEL\_1); // установка направления портов на вывод

DB\_DIR = 0x00; // шина данных настроена на ввод

}

// инициализация системы тактирования

**void** **Init\_System\_Clock**()

{

**volatile** byte i;

BCSCTL1 &= ~XT2OFF; // включение осцилятора XT2

// MCLK = XT2, SMCLK = XT2

**do** // ожидание запуска кварца

{

IFG1 &= ~OFIFG; // Clear OSCFault flag

**for** (i = 0xFF; i > 0; i--); // Time for flag to set

}

**while** ((IFG1 & OFIFG)); // OSCFault flag still set?

BCSCTL2 |= SELM\_2 | SELS; // установка внешнего модуля тактирования

}

// 2do: сделать точную задержку

**void** **wait\_1ms**(word cnt)

{

**for** (wait\_i = 0; wait\_i < cnt; wait\_i++)

**for** (wait\_j = 0; wait\_j < 1000; wait\_j++);

}

**void** **wait\_1mks**(word cnt)

{

**for** (wait\_i = 0; wait\_i < cnt; wait\_i++);

}