**МИНОБРНАУКИ РОССИИ**

ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

**«БЕЛГОРОДСКИЙ ГОСУДАРСТВЕННЫЙ**

**ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ им. В.Г.ШУХОВА»**

**(БГТУ 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-32

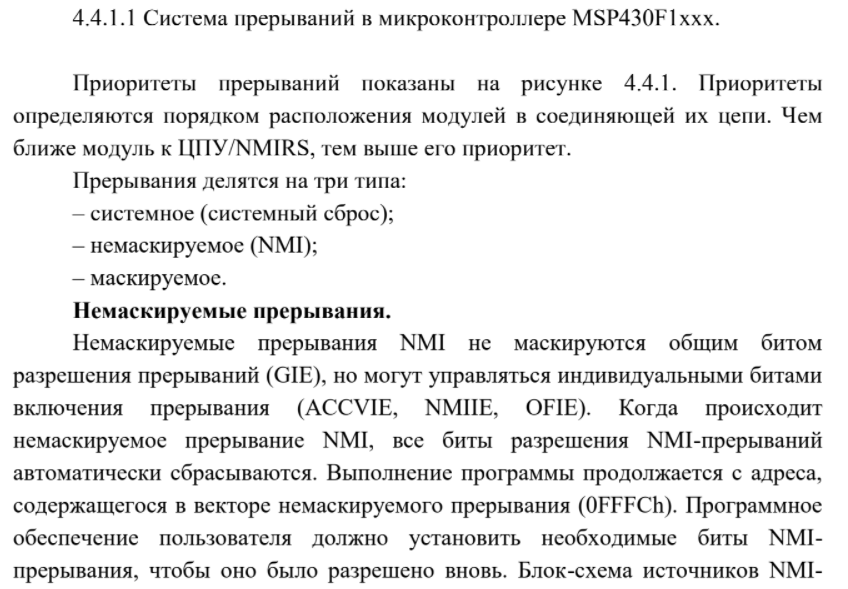
Воскобойников И. С.

Проверил: Шамраев А.А.

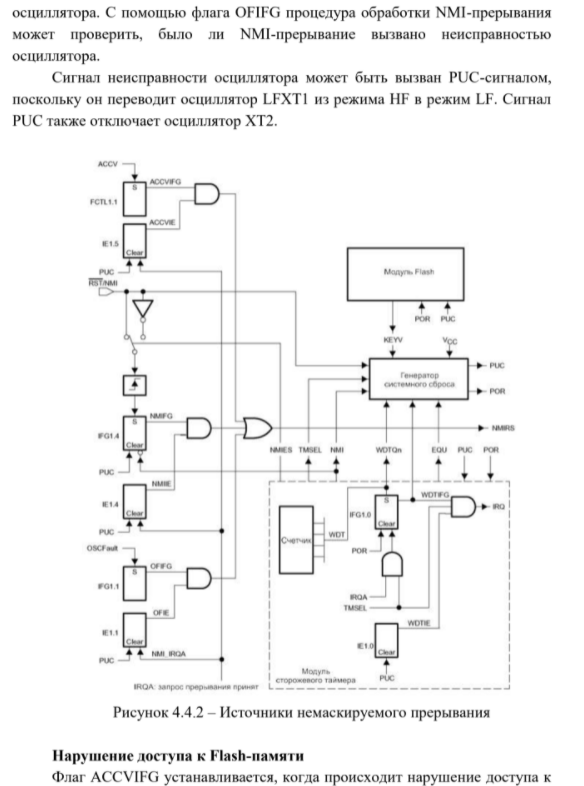
Белгород 2021

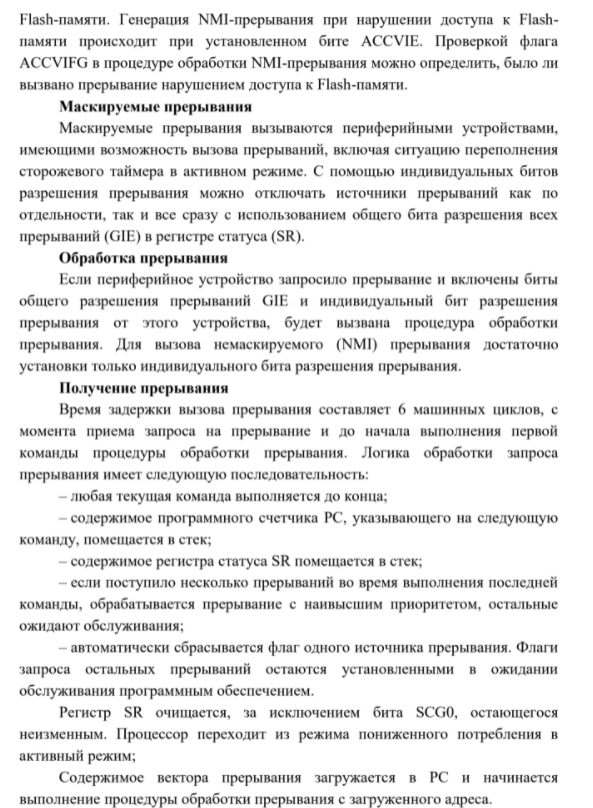
# **Цель работы:** изучить принципы разработки процедур обработки прерываний в микроконтроллере MSP430F1xxx, ознакомиться с принципами функционирования встроенных в микроконтроллер 16 – разрядных таймеров-счетчиков и компаратора для измерения сопротивления резистивного датчика.

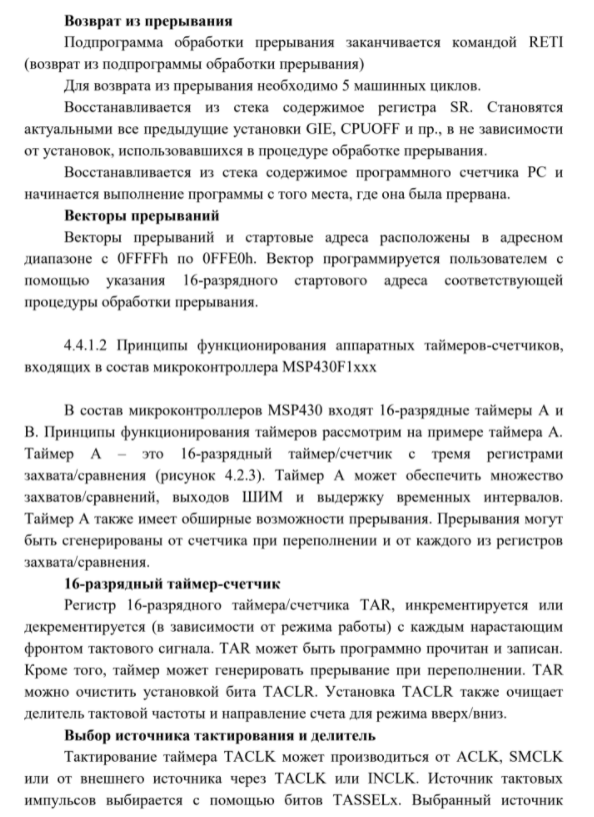
**Краткие теоретические сведения**

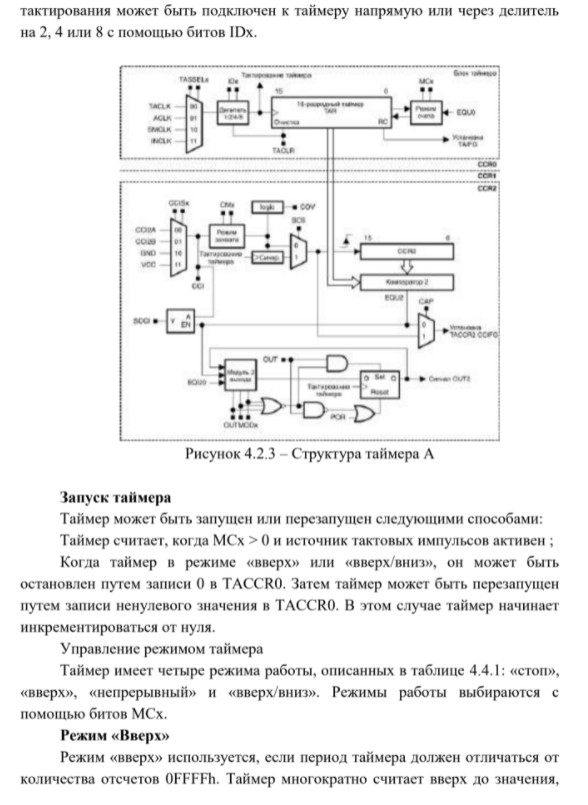


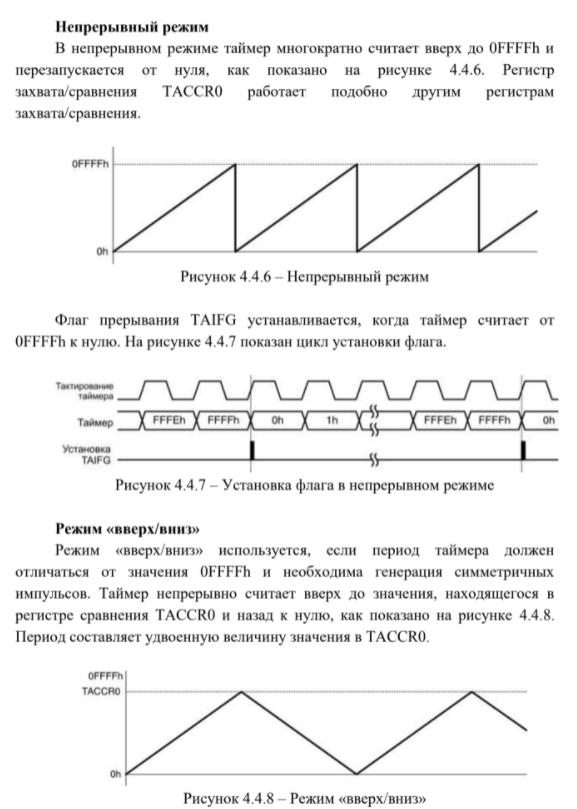


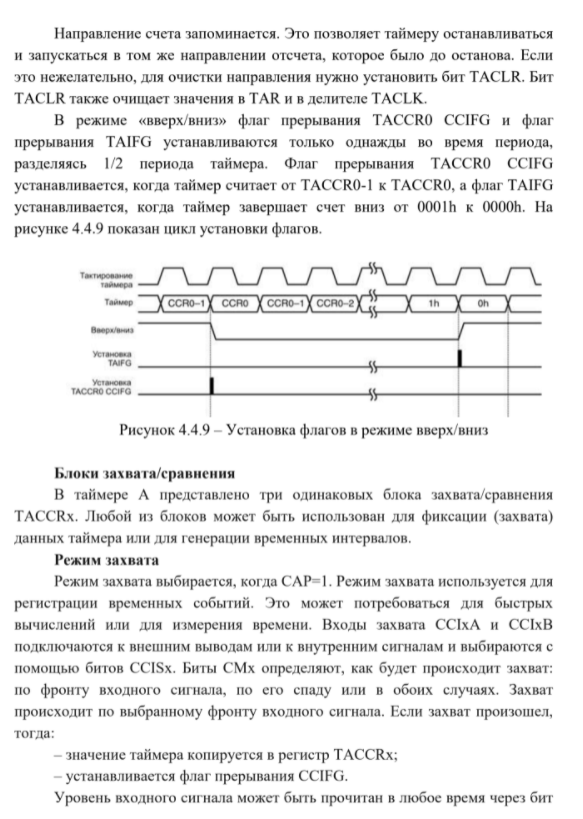






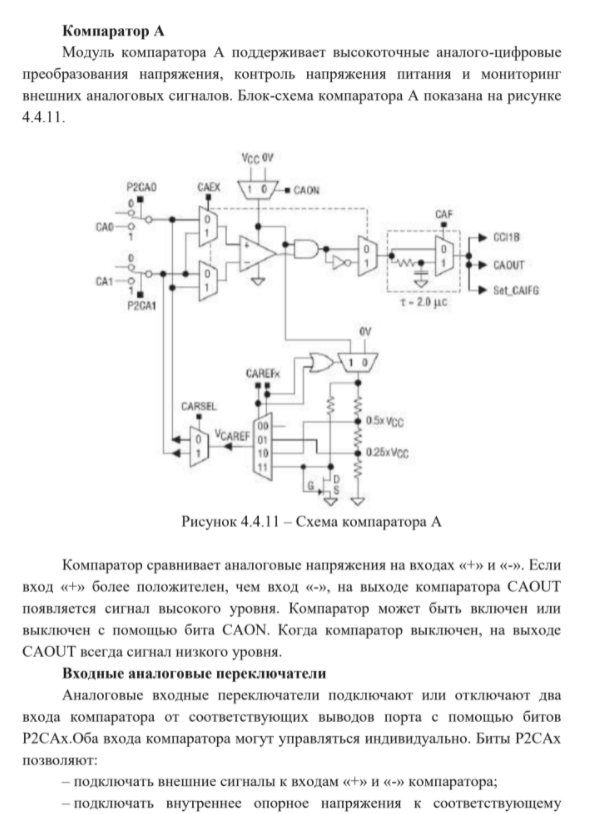


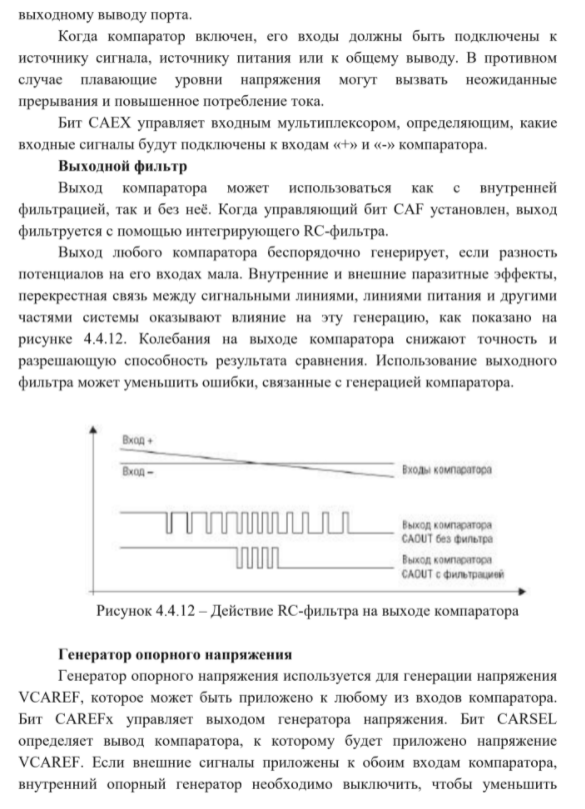


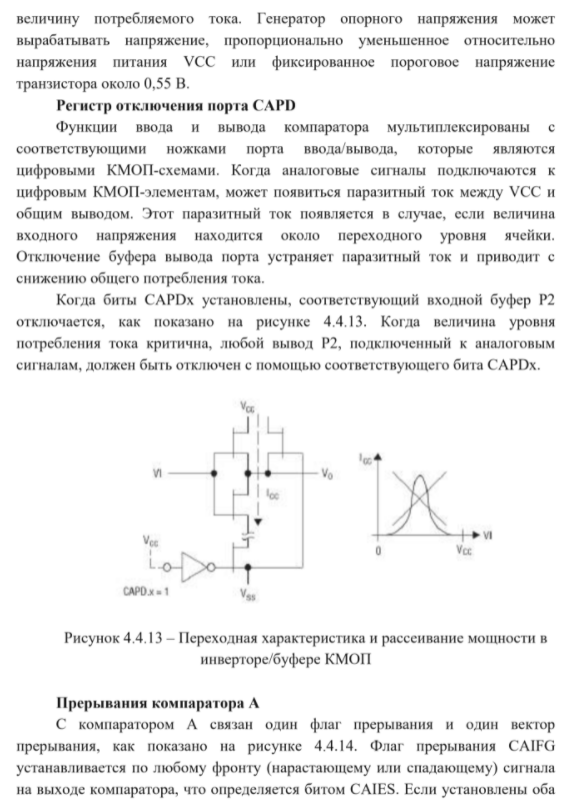


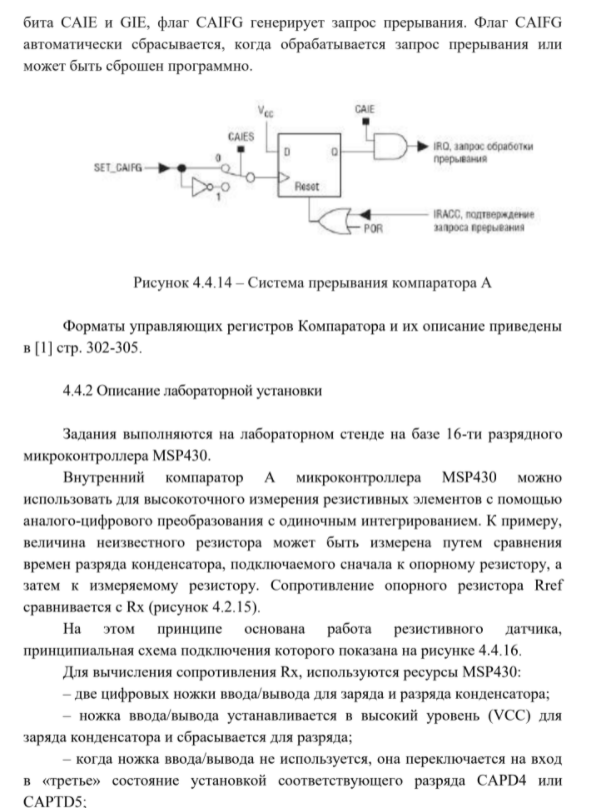
# **C:\Users\500a5\Desktop\01-05-2021 23-22-45.png**

# **C:\Users\500a5\Desktop\01-05-2021 23-22-58.png**











# **C:\Users\500a5\Desktop\01-05-2021 23-25-01.png**

# **C:\Users\500a5\Desktop\01-05-2021 23-25-24.png**

# **C:\Users\500a5\Desktop\01-05-2021 23-25-34.png**

# **C:\Users\500a5\Desktop\01-05-2021 23-25-44.png**

# **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 работы**

– 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.

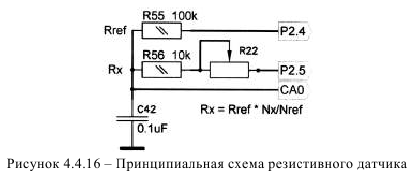
– настроить параметры программатора.

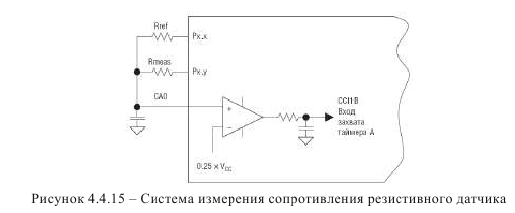
– создать загрузочный модуль программы и выполнить программирование микроконтроллера.

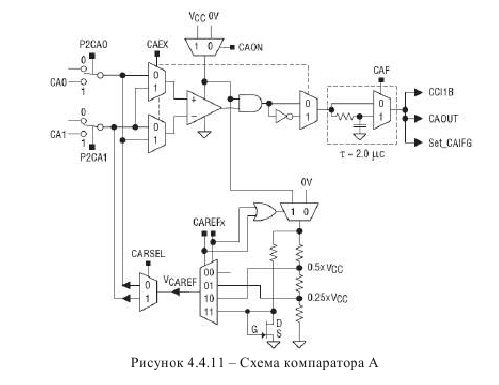
– поверить работоспособность загруженной в микроконтроллер программы и показать результаты работы преподавателю.

В случае некорректной работы разработанной программы, выполнить аппаратный сброс микроконтроллера, провести отладку исходного модуля программы и заново проверить функционирование программы.

**Схема подключения используемых устройств:**







**Текст программы:**

**main.c:**

#include <msp430.h>

#include "stdio.h"

#include "system\_define.h"

#include "system\_variable.h"

#include "function\_prototype.h"

#include "main.h"

/\*

\* main.c

\*/

char message[32] = {0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0};

void main(void) {

WDTCTL = WDTPW+WDTHOLD;

Init\_System\_Clock();

Init\_System();

\_enable\_interrupt();

LCD\_init();

word R = 0;

while (1) {

R = R22\_get\_resistance();

// Записываем в строку сопротивление

sprintf(message, "%u", R);

LCD\_clear();

// выводим строку

LCD\_message(message);

wait\_1ms(1000);

}

}

**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

}

**I2C.c:**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

// I2C function

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

#include "function\_prototype.h"

#include "system\_define.h"

#include "I2C.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

//============================================================================

// Инициализация модуля UART0 для работы в режиме I2C

void Init\_I2C()

{

P3SEL |= 0x0A; // Выбор альтернативной функции для линий порта P3

// в режиме I2C SDA->P3.1, SCL->P3.3

U0CTL |= I2C + SYNC; // Выбрать режим I2C для USART0

U0CTL &= ~I2CEN; // Выключить модуль I2C

// Конфигурация модуля I2C

I2CTCTL=I2CSSEL\_2; // SMCLK

I2CSCLH = 0x26; // High period of SCL

I2CSCLL = 0x26; // Low period of SCL

U0CTL |= I2CEN; // Включить модуль I2C

// формирование строба сброса I2C-регистров PCA9538 - RST\_RG1->P3.1 и RST\_RG2->P3.2

P3DIR |= 0x05; // переключаем эти ножки порта на вывод,

P3SEL &= ~0x05; // выбираем функцию ввода-вывода для них

P3OUT &= ~0x05; // и формируем строб сброса на 1 мс

wait\_1ms(1);

P3OUT |= 0x05;

}

//============================================================================

//============================================================================

// отправка данных по протоколу I2C

void Send\_I2C(unsigned char\* buffer,unsigned int num, unsigned char address)

{

while (I2CBUSY & I2CDCTL); // проверка готовности модуля I2C

BufTptr=buffer;

I2CSA = address; // установка адреса приемнмка

I2CNDAT =num; // количество передаваемых байт

I2CIE = TXRDYIE+ALIE; // разрешение прерываний по окончанию передачи байта и по потере арбитража

U0CTL |= MST; // режим Master

I2CTCTL |= I2CSTT + I2CSTP + I2CTRX; // инициализировать передачу

while ((I2CTCTL & I2CSTP) == 0x02); // ожидание условия СТОП

}

//============================================================================

//============================================================================

// прием данных по протоколу I2C

void Receive\_I2C(unsigned char\* buffer,unsigned int num, unsigned char address)

{

while (I2CBUSY & I2CDCTL); // проверка готовности модуля I2C

BufRptr=buffer;

I2CSA=address;

I2CTCTL&=~I2CTRX; // режим приема

I2CNDAT=num;

I2CIE=RXRDYIE; // резрешение прерывания по окончанию приема байта

U0CTL |= MST;

I2CTCTL |= I2CSTT + I2CSTP; // инициализировать прием

while ((I2CTCTL & I2CSTP) == 0x02); // ожидание условия СТОП

}

//============================================================================

//============================================================================

// отправка байта устройству на шине I2C

void I2C\_SendByte(char data, char i2c\_addr)

{

Tx\_Data[0] = data; // отправляемый байт

Send\_I2C(&Tx\_Data[0], 1, i2c\_addr); // вывод по I2C на устройство

}

//============================================================================

//============================================================================

// запись байта в регистр устройства на шине I2C

void I2C\_WriteByte(char reg, char data, char i2c\_addr)

{

Tx\_Data[0] = reg; // выбираем регистр

Tx\_Data[1] = data; // записываемые данные

Send\_I2C(&Tx\_Data[0], 2, i2c\_addr); // вывод по I2C на устройство

}

//============================================================================

//============================================================================

// чтение байта из регистра устройства на шине I2C

byte I2C\_ReadByte(char reg, char i2c\_addr)

{

Tx\_Data[0] = reg; // выбираем регистр

Send\_I2C(&Tx\_Data[0], 1, i2c\_addr);

Receive\_I2C(&Rx\_Data[0], 1, i2c\_addr); // получаем значение из регистра

return Rx\_Data[0];

}

//============================================================================

//============================================================================

// чтение слова (2 байта) из регистра устройства на шине I2C

int I2C\_ReadWord(char reg, char i2c\_addr)

{

Tx\_Data[0] = reg; // выбираем регистр

Send\_I2C(&Tx\_Data[0], 1, i2c\_addr);

Receive\_I2C(&Rx\_Data[0], 2, i2c\_addr); // получаем 2 байта значение из регистра

return Rx\_Data[0] + (Rx\_Data[1] \* 256);

}

//============================================================================

//============================================================================

//Обработка прерывания от модуля USART0, работающего в режиме I2C

// вектор прерываний для модуля I2C

#pragma vector=USART0TX\_VECTOR

\_\_interrupt void I2C\_ISR()

{

switch(I2CIV)

{

case 0: break; // нет прерывания

case 2: break; // потеря арбитража

case 4: break; // нет подтверждения

case 6: break; // прерывание собственного адреса

case 8: break; // регистр доступен для чтения

case 10: // окончание приема байта

\*BufRptr++=I2CDRB;

break;

case 12: // окончание передачи байта

I2CDRB=\*BufTptr++;

break;

case 14: break; // общий вызов

case 16: break; // обнаружено условие СТАРТ

default : break;

}

}//============================================================================

//---------------------------------------------------------------------------

**keys.c:**

// Keyboard functions

#include "function\_prototype.h"

#include "sysfunc.h"

#include "keys.h"

byte keycol, keyline, KEYS\_last=0;

char table\_keys[12] = {'1', '2', '3', '4', '5', '6', '7', '8', '9', '\*', '0', '#'};

// Проверка нажатия клавиши в текущий момент, результат:

// 0 - клавиша не нажата

// ASCII-код клавиши

char KEYS\_scannow()

{

keyline=0;

// выбираем регистр конфигурации направления (0x03)

// и конфигурируем P4-P7 на вывод - для строба столбцов,

// а P0-P3 на ввод - для опроса строк (1-ввод, 0-вывод)

I2C\_WriteByte(0x03, 0x0F, KEYS\_i2c\_addr);

for (keycol=0; keycol<3; keycol++) {

// последовательно подаем сигнал низкого уровня на столбцы (P4-P7)

I2C\_WriteByte(0x01, ~(1<<keycol<<4) & 0xf0, KEYS\_i2c\_addr);

wait\_1ms(1);

// и опрашиваем строки (P0-P3) на наличие нуля

keyline = ~(I2C\_ReadByte(0x00, KEYS\_i2c\_addr)) & 0x0f;

if (keyline) break;

}

if (!keyline) return 0; // если не была нажата никакая клавиша - возвращаем 0

if (keyline == 4) keyline = 3; // переводим номера разрядов в номер строки

if (keyline == 8) keyline = 4;

KEYS\_last = table\_keys[--keyline\*3+keycol]; // получаем код нажатой клавиши из таблицы

return KEYS\_last;

}

// Возвращает код последней нажатой клавиши, результат:

// 0 - не нажималась никакая клавиша

// ASCII-код клавиши

char KEYS\_lastkey()

{

KEYS\_scannow();

return KEYS\_last;

}

// Очистка последней нажатой клавиши

void KEYS\_clear()

{

KEYS\_last = 0;

wait\_1ms(200);

}

// Ожидание нажатия клавиши, результат - ASCII-код нажатой клавиши

char KEYS\_waitkey()

{

KEYS\_clear(); // очистка последней нажатой клавиши

while (!KEYS\_scannow()) // пока не нажата никакая клавиша,

wait\_1ms(1); // сделать паузу

return KEYS\_last; // вернуть код нажатой клавиши

}

// пауза с циклическим опросом клавиатуры, прерывается если нажата клавиша

void KEYS\_pause(byte cnt)

{

byte i;

KEYS\_clear(); // очистка последней нажатой клавиши

for (i=0; i<cnt; i++)

if (KEYS\_scannow())

break;

}

**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];

}

**leds.c:**

// LED-indicator functions

#include "function\_prototype.h"

#include "sysfunc.h"

#include "leds.h"

char LED\_config=0; // хранится конфигурация светодиодов (вкл/выкл)

void LED\_out(char leds)

{

// регистр конфигурации направления 0x03 конфигурируем на вывод информации (1-ввод, 0-вывод)

I2C\_WriteByte(0x03, 0x00, LED\_i2c\_addr);

I2C\_WriteByte(0x01, leds, LED\_i2c\_addr); // выводим данные в регистр OUTPUT (0x01)

LED\_config=leds; // сохраняем новую конфигурацию

}

// Преобразование номера светодиода в бит, с которым нужно проводить операцию

// 1 = 10000000, 2 = 01000000 ... 8 = 00000001

char LED\_convert(char led)

{

if(led<1)

led=1;

if(led>8)

led=8;

led=9-led;

return (1<<(led-1));

}

// Выключить все светодиоды

void LED\_clear()

{

LED\_out(0x00);

}

// Инвертировать все светодиоды

void LED\_invert()

{

LED\_out(LED\_config ^ 0xff);

}

// Включить светодиод с номером от 1 до 8 (слева направо)

void LED\_set(char led)

{

led=LED\_convert(led);

LED\_config |= led; // устанавливаем соответствующий разряд

LED\_out(LED\_config); // выводим в регистр

}

// Выключить светодиод с номером от 1 до 8 (слева направо)

void LED\_reset(char led)

{

led=LED\_convert(led);

LED\_config &= ~(led); // сбрасываем соответствующий разряд

LED\_out(LED\_config); // выводим в регистр

}

// Сменить состояние светодиода с номером от 1 до 8 (слева направо)

void LED\_change(byte led)

{

led=LED\_convert(led);

LED\_config ^= led; // меняем состояние соответствующего разряда (XOR)

LED\_out(LED\_config); // выводим в регистр

}

void LED\_fx1(int n)

{

LED\_clear();

LED\_set(1);

wait\_1ms(n);

LED\_set(3);

wait\_1ms(n);

LED\_set(5);

wait\_1ms(n);

LED\_set(7);

wait\_1ms(n);

LED\_reset(1);

wait\_1ms(n);

LED\_reset(3);

wait\_1ms(n);

LED\_reset(5);

wait\_1ms(n);

LED\_reset(7);

wait\_1ms(n);

LED\_clear();

}

void LED\_fx2(int n)

{

LED\_clear();

LED\_out(0x81);

wait\_1ms(n);

LED\_out(0x42);

wait\_1ms(n\*8);

LED\_out(0x24);

wait\_1ms(n\*5);

LED\_out(0x18);

wait\_1ms(n\*3);

LED\_out(0x24);

wait\_1ms(n\*5);

LED\_out(0x42);

wait\_1ms(n\*8);

LED\_clear();

}

void LED\_fx3(int n)

{

LED\_clear();

LED\_out(0x01);

wait\_1ms(n\*6);

LED\_out(0x02);

wait\_1ms(n\*4);

LED\_out(0x04);

wait\_1ms(n\*2);

LED\_out(0x08);

wait\_1ms(n\*1);

LED\_out(0x10);

wait\_1ms(n\*1);

LED\_out(0x20);

wait\_1ms(n\*2);

LED\_out(0x40);

wait\_1ms(n\*4);

LED\_out(0x80);

wait\_1ms(n\*6);

LED\_out(0x40);

wait\_1ms(n\*4);

LED\_out(0x20);

wait\_1ms(n\*2);

LED\_out(0x10);

wait\_1ms(n\*1);

LED\_out(0x08);

wait\_1ms(n\*1);

LED\_out(0x04);

wait\_1ms(n\*2);

LED\_out(0x02);

wait\_1ms(n\*4);

LED\_clear();

}

**msp430\_initialize\_unused\_interrrupt\_vectors.c:**

#include <msp430.h>

/\* Initialize non-used ISR vectors with a trap function \*/

/\*

DACDMA\_VECTOR (0 \* 1u) 0xFFE0 DAC/DMA

PORT2\_VECTOR (1 \* 1u) 0xFFE2 Port 2

USART1TX\_VECTOR (2 \* 1u) 0xFFE4 USART 1 Transmit

USART1RX\_VECTOR (3 \* 1u) 0xFFE6 USART 1 Receive

PORT1\_VECTOR (4 \* 1u) 0xFFE8 Port 1

TIMERA1\_VECTOR (5 \* 1u) 0xFFEA Timer A CC1-2, TA // Прерывание используется

TIMERA0\_VECTOR (6 \* 1u) 0xFFEC Timer A CC0

ADC12\_VECTOR (7 \* 1u) 0xFFEE ADC

USART0TX\_VECTOR (8 \* 1u) 0xFFF0 USART 0 Transmit // Прерывание используется

USART0RX\_VECTOR (9 \* 1u) 0xFFF2 USART 0 Receive // Прерывание используется

WDT\_VECTOR (10 \* 1u) 0xFFF4 Watchdog Timer

COMPARATORA\_VECTOR (11 \* 1u) 0xFFF6 Comparator A

TIMERB1\_VECTOR (12 \* 1u) 0xFFF8 Timer B CC1-6, TB

TIMERB0\_VECTOR (13 \* 1u) 0xFFFA Timer B CC0

NMI\_VECTOR (14 \* 1u) 0xFFFC Non-maskable

RESET\_VECTOR (15 \* 1u) 0xFFFE Reset [Highest Priority]

\*/

#pragma vector=DACDMA\_VECTOR,PORT2\_VECTOR,USART1TX\_VECTOR,USART1RX\_VECTOR,PORT1\_VECTOR, \

TIMERA0\_VECTOR,ADC12\_VECTOR,WDT\_VECTOR,COMPARATORA\_VECTOR,TIMERB1\_VECTOR,TIMERB0\_VECTOR, \

NMI\_VECTOR,RESET\_VECTOR

\_\_interrupt void ISR\_trap(void)

{

// Вызов неиспользуемого прерывания приведет к сбросу процессора (PUC reset)

WDTCTL = 0;

}

**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++);

}

**uart.c:**

// UART functions

#include "function\_prototype.h"

#include "sysfunc.h"

#include "uart.h"

// Инициализация режима UART

// speed = 0 - 38400, 1 - 57600, 2 - 115200 - скорость обмена

// databits = 7, 8 - длинна символа

// stopbits = 1, 2 - кол-во передаваемых стоповых бит

// parity = 0 - без контроля четности, 1 - контроль четности, нечетый, 2 - четный

// iface = 0 - USB, 1 - оптика

void UART\_init(byte speed, byte databits, byte stopbits, byte parity, byte iface)

{

P3SEL |= BIT6 | BIT7; // выбор функции USART1

U1CTL = 0; // инициализация состояния USART

ME2 |= UTXE1 + URXE1; // включить приемник и передатчик USART1

if (databits == 7) U1CTL &= ~CHAR; // 7-разрядная длинна символа

if (databits == 8) U1CTL |= CHAR; // 8-разрядная длинна символа

if (stopbits == 1) U1CTL &= ~SPB; // 1 стоповый бит

if (stopbits == 2) U1CTL |= SPB; // 1 стоповых бита

if (parity == 0) U1CTL &= ~PENA; // контроль четности отключен

if (parity == 1) U1CTL = (U1CTL & ~PEV) | PENA; // контроль четности, нечетный

if (parity == 2) U1CTL |= PENA | PEV; // контроль четности, четный

P5DIR |= BIT0; // переключение мультиплексора на USB/оптику

if (iface == 0)

P5OUT |= BIT0;

if (iface == 1)

P5OUT &= ~BIT0;

U1TCTL |= SSEL1; // BRCLK = SMCLK

U1BR0 = 69; // 8Mhz / 115200 = 69.44 (по-умолчанию)

if (speed == 0) U1BR0 = 208; // 8МГц / 38400 = 208.33

if (speed == 1) U1BR0 = 139; // 8МГц / 57600 = 138,89

U1BR1 = 0x00;

U1MCTL = 0x2C; // модуляция

}

// отключение режима UART

void UART\_off()

{

P3SEL |= BIT6 | BIT7; // выбор функции USART1

ME2 &= ~(UTXE1 + URXE1); // выключить приемник и передатчик USART1

U1CTL = SWRST; // отключение USART1

}

// вывод строки символов (символ с кодом 0 - конец строки)

void UART\_message(char \* buf)

{

word i=0;

while (buf[i])

UART\_sendbyte(buf[i++]); // передача сивола

}

// передача байта

void UART\_sendbyte(char byte)

{

while (!(IFG2 & UTXIFG1)); // проверка готовности буфера передачи USART1

U1TXBUF = byte; // передача байта

}

// получение байта

char UART\_getbyte()

{

while (!(IFG2 & URXIFG1)); // проверка готовности буфера приема USART1

return U1RXBUF; // возврат полученного байта

}

**Полученные результаты:** Разработанная программа обеспечивает измерение сопротивления переменного резистора и выводит рассчитанное значение на ЖКИ.

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