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 №5
Дисциплина: ЭВМ и периферийные устройства
Тема: Изучение принципов работы со встроенным в микроконтроллер аналогово-цифровым преобразователем на примере измерения относительной влажности воздуха и потребляемого стендом тока

Выполнил: ст. группы ВТ-31
Подкопаев Антон Валерьевич
Проверил: доцент кафедры ПО и ВТАС
Шамраев Анатолий Анатольевич

Белгород 2020

Цель работы: изучить принципы функционирования встроенного в микроконтроллер MSP430F1611 АЦП и методику измерения относительной влажности и потребляемого тока с помощью датчиков влажности и тока.

Указания по организации самостоятельной работы

Перед работой необходимо проработать теоретический материал по литературе [1–4] и конспекту лекций, ознакомиться с основными возможностями и принципами функционирования АЦП, изучить принципы работы датчиков относительной влажности и тока.

Порядок проведения работы и указания по ее выполнению

Перед началом выполнения практической части лабораторной работы проводится экспресс-контроль знаний по принципам функционирования модуля АЦП, входящего в состав микроконтроллера MSP430F1611. При подготовке к лабораторной работе необходимо составить предварительный вариант листинга программы, в соответствие с индивидуальным заданием.

Задание: разработать в среде программирования IAR Embedded Workbench программу на языке С для измерения значений влажности и тока потребления в соответствие с параметрами режима работы.

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 для считывания данных с модуля АЦП.

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

– настроить параметры программатора.

– создать загрузочный модуль программы и выполнить программирование микроконтроллера.

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

Вариант 9

Разработать программу, выполняющую измерение относительной влажности в режиме одиночного преобразования (делитель частоты равен 2) и отображающую результат измерений на ЖКИ.

Теоретические сведения

Двенадцатиразрядный АЦП12

В состав микроконтроллеров MSP430x13x, MSP430x14x, MSP430x15x и MSP430x16x входит 12-разрядный аналого-цифровой преобразователь (АЦП). Модуль АЦП12 обеспечивает быстрые 12-разрядные аналого-цифровые преобразования. Модуль имеет 12-разрядное ядро SAR, схему выборки, опорный генератор и буфер преобразования и управления объемом 16 слов. Буфер преобразования и управления позволяет получать и сохранять до 16 независимых выборок АЦП без вмешательства ЦПУ. Ядро АЦП преобразует аналоговый входной сигнал в 12-разрядное цифровое представление и сохраняет результат в памяти преобразований.

Ядро использует два программируемых/выбираемых уровня напряжения (VR+ и VR) для задания верхнего и нижнего пределов преобразования. Когда входной сигнал равен или выше VR+ на цифровом выходе (NADC) формируется значение 0FFFh, и ноль, когда входной сигнал равен или ниже VR-. Входной канал и опорные уровни напряжения (VR+ и VR-) задаются в памяти управления преобразованиями.

Ядро АЦП12 конфигурируется двумя управляющими регистрами: ADC12CTL0 и ADC12CTL1. Ядро включается битом ADC12ON. За некоторыми исключениями биты управления АЦП12 могут быть модифицированы, только когда ENC=0. ENC должен быть установлен в 1 перед выполнением любого преобразования.

Генератор опорного напряжения

Модуль АЦП12 содержит встроенный генератор опорного напряжения с двумя выбираемыми уровнями напряжения: 1,5 В и 2,5 В. Любое из этих опорных напряжений может быть использовано внутренне или внешне на выводе VREF+. Чтобы включить внутренний опорный источник необходимо установить бит REFON=1. Если REF2_5V=1, то внутреннее опорное напряжение равно 2,5 В, а при REF2_5V=0 опорное напряжение равно 1,5 В. Для правильной работы внутреннего генератора опорного напряжения необходимо использовать внешний конденсатор, подключенный между VREF+ и AVSS. Рекомендуется использовать комбинацию из включенных параллельно конденсаторов на 10 мкФ и 0,1 мкФ. После включения в течение максимум 17 мс необходимо дать возможность генератору опорного напряжения зарядить конденсаторы. Внешние опорные источники могут быть задействованы для VR+ и VR- через выводы VeREF+ и VREF-/VeREF- соответственно.

Память преобразований

Результаты преобразований сохраняются в 16-ти регистрах памяти преобразований ADC12MEMx. Каждый регистр ADC12MEMx конфигурируется соответствующим управляющим регистром ADC12MCTLx. Биты SREFx устанавливают опорное напряжение, а биты INCHx задают входной канал. Бит EOS определяет конец последовательности, если используется последовательный режим преобразования. Если бит EOS в ADC12MCTL15 не установлен, то результаты

преобразования последовательно сохраняются в регистрах с ADC12MEM15 по ADC12MEM0.

Если выбран режим преобразования «последовательность каналов» или «повторяющаяся последовательность каналов», CSTARTADDx указывают на расположение первого регистра ADC12MCTLx, используемого в последовательности. Программно невидимый указатель автоматически инкрементируется после каждого преобразования. Последовательность продолжается пока не будет обнаружен установленный бит EOS в ADC12MCTLx. При записи результата преобразования в выбранный регистр ADC12MEMx устанавливается соответствующий флаг в регистре ADC12IFGx.

Прерывания АЦП12

АЦП12 имеет 18 источников прерывания:

- ADC12IFG0

- ADC12IFG15;

- ADC12OV, переполнение AD12MEMx;

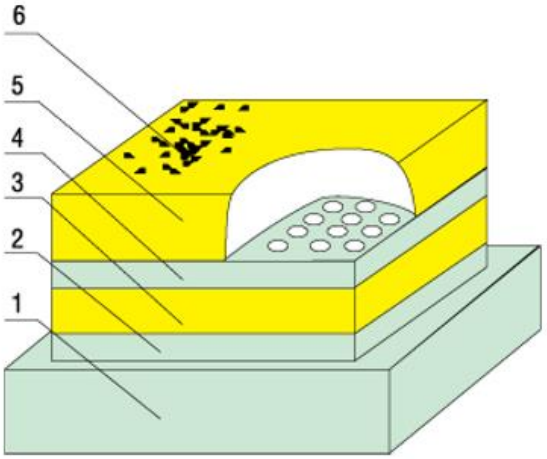
- ADC12TOV, переполнение времени преобразования АЦП12.

Биты ADC12IFGx устанавливаются, когда в их соответствующие регистры памяти ADC12MEMx загружается результат преобразования. Если соответствующий бит ADC12IEx и бит GIE установлены, генерируется запрос прерывания. Состояние ADC12OV появляется, когда результат нового преобразования записывается в любой регистр ADC12MEMx до прочтения предыдущего результата. Состояние ADC12TOV генерируется, когда до завершения текущего преобразования затребована другая выборка преобразование.

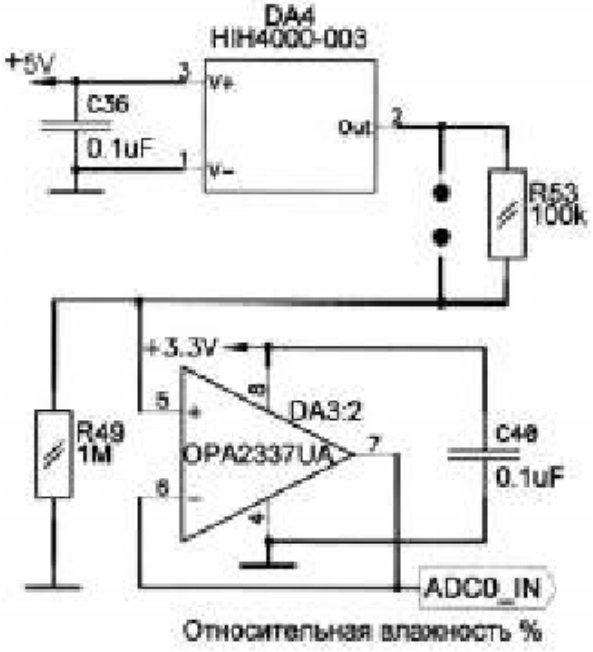
Подключение датчика влажности

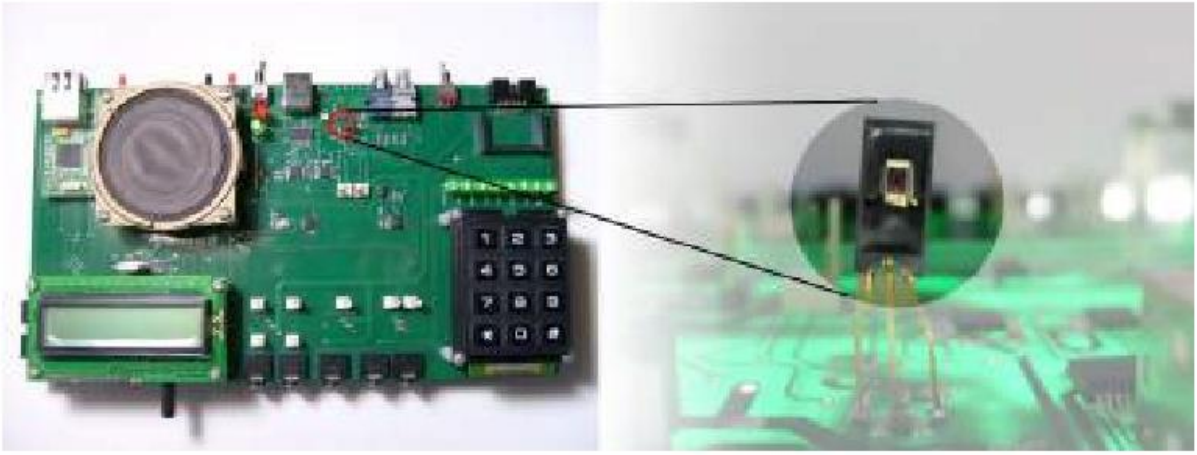
Для измерения влажности используются датчики, основанные на различных физических принципах и выполненные по различным технологиям. Можно выделить основные четыре типа датчиков: емкостные, резистивные, на основе оксида олова и на основе оксида алюминия. Рассмотрим кратко особенности каждого типа.

Из представленных четырех основных типов для измерения влажности, оптимальным по совокупности параметров является емкостной. Он обеспечивает широкий диапазон измерений, высокую надежность и низкую стоимость при использовании микроэлектронной технологии. Благодаря чему мы имеем миниатюрные габариты чувствительного элемента, возможность имплементации на кристалле специализированной интегральной схемы обработки сигнала. Технологичность и высокий выход годных кристаллов обеспечивают малую стоимость продукции данного типа. Итак, для измерения влажности емкостной метод является лучшим.



1– кремниевая подложка; 2, 4 – платиновый электрод;
3, 5 – термореактивный полимер; 6 – пыль, грязь, масло.





lcd.c

```
#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 5x11 точек, F=1 => размер шрифта 5x8 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();     // Сигал 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();       // Сигал 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];
}
```

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+ = VeREF+ = 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+ = VeREF+ = 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
}
```

main.c

```
#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];
    float hum, last_hum = 0;
    while(1){
        hum = HIH_get_hum();
        if (hum != last_hum){
            LCD_clear();
            sprintf (message, "%5.3f", hum);
            LCD_message(message);
            last_hum = hum;
        }
        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++);
}
```