Специальное программное обеспечение цифрового USB прибора для измерения вольт-амперных характеристик.

Д.А. Коновалов, н.с. лаб. физического приборостроения КФТИ ФИЦ КазНЦ РАН

В состав программно-управляемого источника-измерителя цифрового USB прибора для измерения вольт-амперных характеристик (ВАХ) входит модуль управляющего контроллера Iskra Nano Pro (Arduino Nano), который обеспечивает следующие функции:

- прием и обработку управляющих директив по последовательному интерфейсу по протоколу RS-232;
- управление модулями ЦАП, АЦП и оптронными ключами P1, P2;
- предварительную цифровую обработку данных, поступающих от измерительных каналов – накопление с усреднением без экстремальных значений, конвертирование отсчетов АЦП в напряжение (мВ) с учетом коэффициента усиления;
- передачу данных в рабочую станцию с прикладным программным обеспечением.

Специальное программное обеспечение (СПО) реализуется как прошивка для микроконтроллера модуля Iskra Nano Pro.

СПО представлено исходным кодом на языке высокого уровня С, предназначенного для формирования исполнимого кода (прошивки) программируемого устройства - микроконтроллера модуля Iskra Nano Pro.

СПО создано и загружено в микроконтроллер в среде разработки Arduino.

Оглавление

1 Описание логической структуры СПО

Структурно СПО состоит из процедур инициализации и конфигурации периферии микроконтроллера и модулей ЦАП и АЦП, выполняемых однократно при запуске СПО, главного рабочего цикла – интерпретатора командной строки, вызывающего по мере необходимости рабочие процедуры (управление модулями ЦАП и АЦП, передача данных) и процедуры чтения данных АЦП, работающей в фоновом режиме по прерыванию.

2 Интерфейс

Контроллер имеет последовательный интерфейс RS-232 и является устройством, отвечающим на команды. Скорость передачи 115200 бит/с без контроля четности, 8 бит данных, 1 стоп-бит. СПО реализует текстовый консольный интерфейс.

3 Команды

Приглашением к вводу команды является символ “>”. Символом завершения ввода и разделителем параметров является пробел.

HELP – выводит список поддерживаемых команд, их синтаксис и допустимые параметры;

ADC_CONFIG? – выводит значения 4-х регистров конфигурации АЦП в шестнадцатеричном формате;

ADC_CONFIG XX XX XX XX – задает значения 4-х регистров конфигурации АЦП. XX – значение регистра в шестнадцатеричном формате;

DAC n DDDD – для канала ЦАП **n** задает значение **DDDD** (**n**: 0-3, **DDDD**: 0-4095);

DAC_CLR – загрузка нулевого кода в регистры всех каналов ЦАП;

GAIN n – задает коэффициент усиления масштабного усилителя АЦП (**n**: 0-7, коэффициент усиления = 2<<n (1, 2, 4, 8, 16, 32, 64, 128);

AUTO_GAIN – пока не реализовано;

MUX n – конфигурирует входной коммутатор АЦП (**n**: 0-5, где 0: AIN0_AIN1, 1: AIN0_AIN2, 2: AIN0_AIN3, 3: AIN1_AIN2, 4: AIN1_AIN3, 5: AIN2_AIN3);

IDAC n – конфигурирует встроенные в АЦП источники тока (**n**: 0-7, где 0: Off, 1: 10uA, 2: 50uA, 3: 100uA, 4: 250uA, 5: 500uA, 6: 1mA, 7: 1.5mA);

I1MUX n – управляет подключением встроенного в АЦП источника тока 1 к входу AIN2 (**n**: 0-1, где 0: Off, 1: AIN2);

I2MUX n – управляет подключением встроенного в АЦП источника тока 2 к входу AIN2 (**n**: 0-1, где 0: Off, 1: AIN2);

DATA? – ждет окончания преобразования АЦП и выводит: результат в мВ с точностью 6 знаков после запятой; отсчет АЦП в шестнадцатеричном формате; значение коэффициента усиления. Разделитель – символ «возврат каретки» (ответ: data_mV<CR> data_hex<CR> gain<CR>);

UI? n m – макрокоманда (**n** GainU:0-7, **m** GainI:0-7). Выполняет последовательность действий по измерению напряжения и тока (ответ: Udata_mV<CR> Idata_mV<CR>).

STREAM – в цикле ждет окончания преобразования АЦП и выводит результат в мВ с точностью 6 знаков после запятой;
OUT n – управляет оптронными ключами, подключающим выходы ИТУН и модуля ЦАП к измерительным клеммам (**n**: 0-2, где 0: Off, 1:Current, 2:Voltage).

4 Алгоритм исполнения программы СПО

При подачи питания запускается СПО. При этом происходит выполнение следующих процедур:

- инициализация внутренних переменных;
- инициализация портов ввода/вывода;
- инициализация модуля ЦАП;
- инициализация модуля АЦП;
- инициализация интерфейса UART.

После этого происходит переход в рабочий цикл. В рабочем цикле реализован интерпретатор командной строки. В начале цикла выводится приглашение “>” и ожидается ввод команды. Принятая последовательность символов сравнивается со списком команд в конструкции if – elseif-elseif-…-else с помощью процедуры сравнения строк без учета регистра. Если принятая последовательность символов не соответствует никакой команде, то выводится сообщение об ошибке и рабочий цикл начинается с начала. Если после команды должен следовать параметр, то ожидается ввода параметра. Принятый параметр проверяется на корректность. В случае неправильного ввода выводится сообщение об ошибке, рабочий цикл начинается с начала. Если команда распознана и все параметры введены корректно, то вызываются необходимые для выполнения команды процедуры и производятся соответствующие манипуляции с переменными и флагами.
Модуль АЦП работает в режиме непрерывных преобразований. После окончания очередного преобразования на выходе $\overline{\mathrm{DRDY}}$ модуля АЦП на короткое время появляется сигнал низкого логического уровня, вызывающий аппаратное прерывание рабочего цикла. Обработка прерывания – считывание отсчета АЦП - производится в процедуре **`ISR_ADS1220.`**

Обмен данными с АЦП осуществляется по интерфейсу SPI. Обращение к модулю АЦП происходит асинхронно в процедуре **`ISR_ADS1220`** и в рабочем цикле при выполнении некоторых консольных команд. Поэтому доступ к интерфейсу SPI контролируется флагом **`SPI_available.`**

СПО использует функции языка С для работы со стандартным вводом/выводом. Стандартная библиотека среды Arduino не поддерживает функции языка С для работы со стандартным вводом/выводом. Использование таких функций как **printf** и **scanf**, делает исходный код программы более компактным и читаемым. Расплатой за такое удобство является увеличение

размера загружаемого кода. Поддержка стандартного ввода/вывода в последовательный порт реализована с использованием библиотеки:
stdinout (https://github.com/AdamCDunlap/arduino/tree/master/libraries/stdinout).

4.1 Алгоритм обработки прерывания и команд

4.1.1 Процедура обработки прерывания ISR_ADS1220

- Ждет освобождения SPI
- Занимает SPI
- Считывает отсчет АЦП
- Освобождает SPI
- Поднимает флаг `new_adc_data`

4.1.2 ADC_CONFIG?

- Ждет освобождения SPI
- Занимает SPI
- Считывает значения регистров конфигурации
- Освобождает SPI
- Выводит значения регистров конфигурации

4.1.3 ADC_CONFIG

- Ждет ввода 4-х параметров и проверяет их корректность
- Ждет освобождения SPI
- Занимает SPI
- Загружает в АЦП значения 4-х регистров конфигурации
- Освобождает SPI

4.1.4 DAC

- Ждет ввода номера канала ЦАП и проверяет его на корректность
- Ждет ввода значения для указанного канала ЦАП и проверяет его на корректность
- Конвертирует значение в формат ЦАП и загружает его в микросхему ЦАП

4.1.5 DAC_CLR

- Осуществляет аппаратный сброс всех каналов ЦАП

4.1.6 GAIN

- Ждет ввода параметра и проверяет его корректность
- Вызывает попрограмму **ads1220_set_pga_gai**

4.1.7 MUX, IDAC, I1MUX, I2MUX

- Ждет ввода параметра и проверяет его корректность
- Вызывает соответствующую подпрограмму: **ads1220_select_mux_channels**, **ads1220_set_idac**, **ads1220_select_i1mux_channel** или **ads1220_select_i2mux_channel**

4.1.8 DATA?

- Вызывает подпрограмму **Wait_Read_ADC_Data**
- Выводит значения отсчета АЦП в mV, в HEX формате и значение переменной `PGA_gain`

4.1.9 UI? n m – макрокоманда

- Ждет ввода двух параметров и проверяет их корректность

- Устанавливает коэффициент усиления **n**, вызвав подпрограмму **ads1220_set_pga_gain**
- Конфигурирует входной коммутатор АЦП: **ads1220_select_mux_channels(MUX_AIN0_AIN1)**
- Вызывает подпрограмму **Wait_Read_ADC_Data**
- Запоминает измеренное значение напряжение в переменную `U_mV`
- Конфигурирует входной коммутатор АЦП: **ads1220_select_mux_channels(MUX_AIN1_AIN0)**
- Вызывает подпрограмму **Wait_Read_ADC_Data**
- Вычисляет напряжение, устранив сдвиг нуля: `U_mV=(U_mV-adc_mV)/2`
- Устанавливает коэффициент усиления **m**, вызвав подпрограмму **ads1220_set_pga_gain**
- Конфигурирует входной коммутатор АЦП: **ads1220_select_mux_channels(MUX_AIN2_AIN3)**
- Вызывает подпрограмму **Wait_Read_ADC_Data**
- Запоминает измеренное значение напряжение в переменную `I_mV`
- Конфигурирует входной коммутатор АЦП: **ads1220_select_mux_channels(MUX_AIN3_AIN2)**
- Вызывает подпрограмму **Wait_Read_ADC_Data**
- Вычисляет напряжение, устранив сдвиг нуля: `I_mV=(I_mV-adc_mV)/2`
- Выводит значения `U_mV` и `I_mV`.

4.1.10 STREAM

- В цикле ждет окончания AD преобразования, проверяя флаг `new_adc_data`
- Вызывает подпрограмму **Wait_Read_ADC_Data**
- Выводит значения отсчета АЦП в mV
- Выход из цикла при появлении байта во входном буфере последовательного порта

4.1.11 OUT

- Ждет ввода параметра и проверяет его корректность
- Осуществляет аппаратное управления оптронными ключами

4.2 Исходный текст специального программного обеспечения цифрового USB прибора для измерения вольт-амперных характеристик (версия 29-01-2021)

```
//////////////////////////////////////////////////////////////////
//
//  Подключение АЦП:
//
//      [Источник_тока] (+/- 40 mA, 24V max)
//     ┌─>>──┤
//     │    [3_МОм]
//  [p-n]    ├──────[AIN0]
//     │   [150кОм]
//     └─>>──┤──────[AIN1, AIN2]
//         [9.1_Ом]
//           │    ┌──[AIN3]
//           ⏚    ⏚

#include "stdinout.h"
#include <Wire.h>
#include <SPI.h>

#define VREF 2048            // Internal reference of 2048 mV
#define FSR (long int)0x7FFFFF

#define ADS1220_CS_PIN    7
#define ADS1220_DRDY_PIN  2
#define PVA_ANODE_PIN 4
#define CPC_ANODE_PIN 5

//ADS1220 SPI commands
#define SPI_MASTER_DUMMY    0xFF
#define RESET               0x06   //Send the RESET command (06h) to make sure the ADS1220 is
properly reset after power-up
#define START               0x08    //Send the START/SYNC command (08h) to start converting in
continuous conversion mode
#define WREG  0x40
#define RREG  0x20
//Config registers
#define CONFIG_REG0_ADDRESS 0x00
#define CONFIG_REG1_ADDRESS 0x01
#define CONFIG_REG2_ADDRESS 0x02
#define CONFIG_REG3_ADDRESS 0x03

#define REG_CONFIG0_PGA_GAIN_MASK 0x0E
#define REG_CONFIG0_MUX_MASK      0xF0
#define REG_CONFIG1_DR_MASK       0xE0
#define REG_CONFIG1_BCS_MASK      0x01
#define REG_CONFIG2_IDAC_MASK     0x07
#define REG_CONFIG3_I1MUX_MASK    0xE0
#define REG_CONFIG3_I2MUX_MASK    0x1C

#define DR_20SPS    0x00
#define DR_45SPS    0x20
#define DR_90SPS    0x40
#define DR_175SPS   0x60
#define DR_330SPS   0x80
#define DR_600SPS   0xA0
#define DR_1000SPS  0xC0

#define PGA_GAIN_1   0x00
#define PGA_GAIN_2   0x02
#define PGA_GAIN_4   0x04
#define PGA_GAIN_8   0x06
#define PGA_GAIN_16  0x08
#define PGA_GAIN_32  0x0A
#define PGA_GAIN_64  0x0C
```

```
#define PGA_GAIN_128 0x0E

#define MUX_AIN0_AIN1   0x00
#define MUX_AIN0_AIN2   0x10
#define MUX_AIN0_AIN3   0x20
#define MUX_AIN1_AIN2   0x30
#define MUX_AIN1_AIN3   0x40
#define MUX_AIN2_AIN3   0x50
#define MUX_AIN1_AIN0   0x60
#define MUX_AIN3_AIN2   0x70

#define IDAC_OFF    0x00
#define IDAC_10uA   0x01
#define IDAC_50uA   0x02
#define IDAC_100uA  0x03
#define IDAC_250uA  0x04
#define IDAC_500uA  0x05
#define IDAC_1000uA 0x06
#define IDAC_1500uA 0x07

#define I1MUX_OFF   0x00
#define I1MUX_AIN0  0x20
#define I1MUX_AIN1  0x40
#define I1MUX_AIN2  0x60
#define I1MUX_AIN3  0x80

#define I2MUX_OFF   0x00
#define I2MUX_AIN0  0x04
#define I2MUX_AIN1  0x08
#define I2MUX_AIN2  0x0C
#define I2MUX_AIN3  0x10

#define DAC_CLR A3

uint8_t m_config_reg0, m_config_reg1, m_config_reg2, m_config_reg3, i;

volatile long adc_data;
volatile long adc_res[8];
volatile uint8_t PGA_gain=1;
volatile float adc_mV, U_mV, I_mV;
volatile bool SPI_available = true, new_adc_data = false, auto_gain = false;

char inpstr[13];
const char err[]="\rUnknown cmd - type HELP";
uint8_t n, GainU=1, GainI=1;
word dddd;
```

```
void i2c_write(uint8_t addr, uint8_t cmd, uint8_t msb, uint8_t lsb){
  Wire.beginTransmission(addr); // transmit to device addr
  Wire.write(cmd);
  Wire.write(msb);              // sends high byte
  Wire.write(lsb);              // sends low byte
  Wire.endTransmission();     // stop transmitting
 }
```

```
void setup() {
  pinMode(PVA_ANODE_PIN, OUTPUT);
  digitalWrite(PVA_ANODE_PIN,LOW); // Отключим выход источника тока
  pinMode(CPC_ANODE_PIN, OUTPUT);
  digitalWrite(CPC_ANODE_PIN,LOW); // Отключим выход источника напряжения
  pinMode(ADS1220_CS_PIN, OUTPUT);
  pinMode(ADS1220_DRDY_PIN, INPUT);
  SPI.begin();
  SPI.setBitOrder(MSBFIRST);
  SPI.setDataMode(SPI_MODE1);
  DAC_begin();
  ads1220_begin();
  attachInterrupt(digitalPinToInterrupt(ADS1220_DRDY_PIN), ISR_ADS1220, LOW);
  ads1220_Start_Conv();  //Start continuous conversion mode
```

```
  Serial.begin(115200); // start serial port at 9600 bps.
}
```

```
void loop() {
  printf("\r>");
  scanf("%13s", inpstr); // ждём ввода не более 12 символов
  if (strcasecmp(inpstr,"HELP")==0){
    printf( // выведем список допустимых команд
    "\rValid commands:\r\r"
    "ADC_CONFIG?\r"
    "ADC_CONFIG XX XX XX XX (XX - value for 0-3 config redisters in HEX)\r"
    "DAC n DDDD (n:0-3, DDDD:0-4095)\r"
    "DAC_CLR\r"
    "GAIN n (n:0-7, Gain = 2<<n)\r"
    "MUX n (n:0-7, where 0:A0-A1, 1:A0-A2, 2:A0-A3, 3:A1-A2, 4:A1-A3, 5:A2-A3, 6:A1-A0, 7:A3-
A2)\r"
    "IDAC n (n:0-7, where 0:Off, 1:10uA, 2:50uA, 3:100uA, 4:250uA, 5:500uA, 6:1mA, 7:1.5mA)\r"
    "I1MUX n (n:0-1, where 0:Off, 1:A2)\r"
    "I2MUX n (n:0-1, where 0:Off, 1:A2)\r"
    "DATA? (answer: data_mV<CR> data_hex<CR> gain)\r"
    "UI? n m (n GainU:0-7, m GainI:0-7; answer: Udata_mV<CR> Idata_mV<CR>)\r"
    "STREAM (scroll show adc_data in mV until recieved char)\r"
    "OUT n (n:0-2, where 0:Off, 1:Current, 2:Voltage)\r"
    );
  }
  else if (strcasecmp(inpstr,"ADC_CONFIG?")==0){
        while (SPI_available == false){} // Ждем освобождения SPI
        SPI_available = false; // Займём SPI
        m_config_reg0 = ads1220_readRegister( CONFIG_REG0_ADDRESS);
        m_config_reg1 = ads1220_readRegister( CONFIG_REG1_ADDRESS);
        m_config_reg2 = ads1220_readRegister( CONFIG_REG2_ADDRESS);
        m_config_reg3 = ads1220_readRegister( CONFIG_REG3_ADDRESS);
        SPI_available = true; // Освободим SPI
        Serial.println(m_config_reg0,HEX);
        Serial.println(m_config_reg1,HEX);
        Serial.println(m_config_reg2,HEX);
        Serial.println(m_config_reg3,HEX);
  }
  else if (strcasecmp(inpstr,"ADC_CONFIG")==0){
        if (scanf("%hhx %hhx %hhx %hhx", &m_config_reg0, &m_config_reg1, &m_config_reg2,
&m_config_reg3) < 4){
          printf(err);
        }
        else {
        while (SPI_available == false){} // Ждем освобождения SPI
        SPI_available = false; // Займём SPI
        ads1220_writeRegister( CONFIG_REG0_ADDRESS , m_config_reg0);
        ads1220_writeRegister( CONFIG_REG1_ADDRESS , m_config_reg1);
        ads1220_writeRegister( CONFIG_REG2_ADDRESS , m_config_reg2);
        ads1220_writeRegister( CONFIG_REG3_ADDRESS , m_config_reg3);
        SPI_available = true; // Освободим SPI
        }
  }
  else if (strcasecmp(inpstr,"DAC")==0){
        //считываем и проверяем номер канала ЦАП
        if (scanf("%2hhu", &n)==0){
          printf(err);
        }
        else if (n>3){
          printf("n=%d n must be 0-3",n);
        }
        else {
          //считываем и проверяем значение для указанного канала ЦАП
          if (scanf("%5u", &dddd)==0){
            printf(err);
          }
          else if (dddd>4095){
            printf("DDDD=%d DDDD must be 0-4095\r",dddd);
          }
```

```
        else{
          dddd<<=4;
          i2c_write(0x10,0x30|n,highByte(dddd),lowByte(dddd));
         }
      }
}
else if (strcasecmp(inpstr,"DAC_CLR")==0){
      digitalWrite(DAC_CLR,LOW);
      digitalWrite(DAC_CLR,HIGH);
}
else if (strcasecmp(inpstr,"GAIN")==0){
      //считываем и проверяем значение GAIN
      if (scanf("%2hhu", &n)==0){
        printf(err);
      }
      else if (n>7){
        printf("n=%d GAIN must be 0-7",n);
      }
      else{
        ads1220_set_pga_gain(n); // Установим GAIN
      }
}
else if (strcasecmp(inpstr,"MUX")==0){
      //считываем и проверяем значение MUX
      if (scanf("%2hhu", &n)==0){
        printf(err);
      }
      else if (n>7){
        printf("n=%d MUX must be 0-7",n);
      }
      else{
        ads1220_select_mux_channels(n<<4); // Выберем каналы
      }
}
else if (strcasecmp(inpstr,"IDAC")==0){
      //считываем и проверяем значение IDAC
      if (scanf("%2hhu", &n)==0){
        printf(err);
      }
      else if (n>7){
        printf("n=%d IDAC must be 0-7",n);
      }
      else{
        ads1220_set_idac(n); // Установим IDAC
      }
}
else if (strcasecmp(inpstr,"I1MUX")==0){
      //считываем и проверяем значение I1MUX
      if (scanf("%2hhu", &n)==0){
        printf(err);
      }
      else if (n>1){
        printf("n=%d I1MUX must be 0-1",n);
      }
      else{
        if (n == 0){
          ads1220_select_i1mux_channel(I1MUX_OFF); // Отключим IDAC1 от AIN2
        }
        else{
          ads1220_select_i1mux_channel(I1MUX_AIN2); // Подключим IDAC1 к AIN2
        }
      }
}
else if (strcasecmp(inpstr,"I2MUX")==0){
      //считываем и проверяем значение I1MUX
      if (scanf("%2hhu", &n)==0){
        printf(err);
      }
      else if (n>1){
```

```
        printf("n=%d I2MUX must be 0-1",n);
      }
      else{
        if (n == 0){
          ads1220_select_i2mux_channel(I2MUX_OFF); // Отключим IDAC2 от AIN2
        }
        else{
          ads1220_select_i2mux_channel(I2MUX_AIN2); // Подключим IDAC2 к AIN2
        }
      }
}
else if (strcasecmp(inpstr,"DATA?")==0){
      Wait_Read_ADC_Data();
      Serial.println(adc_mV,8);
      Serial.println(adc_data,HEX);
      Serial.print(PGA_gain);
}
else if (strcasecmp(inpstr,"UI?")==0){
      if (scanf("%2hhu %2hhu", &GainU, &GainI)==0){
        printf(err);
      }
      else if (GainU>7 || GainI>7){
        printf("GainU=%d GainI=%d must be 0-7",GainU,GainI);
      }
      else{
        ads1220_set_pga_gain(GainU); // Установим GAIN for U channel
        ads1220_select_mux_channels(MUX_AIN0_AIN1);
        Wait_Read_ADC_Data();
        U_mV = adc_mV;
        ads1220_select_mux_channels(MUX_AIN1_AIN0);
        Wait_Read_ADC_Data();
        U_mV=(U_mV-adc_mV)/2;
        ads1220_set_pga_gain(GainI); // Установим GAIN for I channel
        ads1220_select_mux_channels(MUX_AIN2_AIN3);
        Wait_Read_ADC_Data();
        I_mV = adc_mV;
        ads1220_select_mux_channels(MUX_AIN3_AIN2);
        Wait_Read_ADC_Data();
        I_mV=(I_mV-adc_mV)/2;
        Serial.println(U_mV,8);
        Serial.println(I_mV,8);
      }
}
else if (strcasecmp(inpstr,"STREAM")==0){
      while(Serial.available() == 0){
        if (new_adc_data){
          Wait_Read_ADC_Data();
          Serial.println(adc_mV,8);
        }
      }
}
else if (strcasecmp(inpstr,"OUT")==0){
      //считываем и проверяем значение OUT
      if (scanf("%2hhu", &n)==0){
        printf(err);
      }
      else if (n>2){
        printf("n=%d OUT must be 0-1",n);
      }
      else{
        if (n == 0){
          digitalWrite(PVA_ANODE_PIN,LOW); // Отключим выход источника тока
          digitalWrite(CPC_ANODE_PIN,LOW); // Отключим выход источника напряжения
        }
        else if (n == 1){
          digitalWrite(CPC_ANODE_PIN,LOW); // Отключим выход источника напряжения
          digitalWrite(PVA_ANODE_PIN,HIGH); // Подключим выход источника тока
        }
        else {
```

```
            digitalWrite(PVA_ANODE_PIN,LOW); // Отключим выход источника тока
            digitalWrite(CPC_ANODE_PIN,HIGH); // Подключим выход источника напряжения
          }
        }
  }
  else{
    printf(err);
  }
}
```

```
void ISR_ADS1220 (){
  if (SPI_available){
    SPI_available = false;
    adc_data = ads1220_Read_Data();
    SPI_available = true;
    new_adc_data = true;
  }
}
```

```
void Wait_Read_ADC_Data(){
  bool OK = false;
  uint8_t exchange =1;
  long tmp=0;

  for (i=0; i<4; i++) // Сделаем 4 измерений
  {
    while (!new_adc_data){}; // ждем окончания AD преобразования
    adc_res[i]=adc_data;
    new_adc_data = false;
  }
/******* Сортировка массива *** from  AN4073 ST.COM ********/
  while(exchange==1)
  {
   exchange=0;
   for(i=0; i<4-1; i++)
   {
     if( adc_res[i] > adc_res[i+1] )
     {
       tmp = adc_res[i];
       adc_res[i] = adc_res[i+1];
       adc_res[i+1] = tmp;
       exchange=1;
     }
   }
  }
  adc_data=0;// отбросив 1 младших и 1 старших элементов массива,
  for (i=1; i<3; i++) adc_data+=adc_res[i]; // суммируем 2 средних элемента массива с 1 по 2
  adc_data/=2; // усредняем
  adc_mV = (float)((float)adc_data*VREF/FSR/PGA_gain);
}

void DAC_begin(){
  pinMode(DAC_CLR,OUTPUT);
  digitalWrite(DAC_CLR,LOW);
  delay(1);
  digitalWrite(DAC_CLR,HIGH);
  delay(1);
  Wire.begin(); // join i2c bus (address optional for master)
  i2c_write(0x10, 0x77, 0x00, 0x00); // команда REF 4.096V
}

void ads1220_begin()
{
    ads1220_SPI_Command(RESET);
    delay(10);

    m_config_reg0 = 0x00;   //Default settings: AINP=AIN0, AINN=AIN1, Gain 1, PGA enabled
    m_config_reg1 = 0x04;   //Default settings: DR=20 SPS, Mode=Normal, Conv mode=continuous,
Temp Sensor disabled, Burn-out Current Source off
```

```
    m_config_reg2 = 0x00;   //Default settings: internal VREF, no 50 or 60 Hz rejection,
switch open, IDAC off
    m_config_reg3 = 0x00;   //Default settings: IDAC1 disabled, IDAC2 disabled, DRDY pin only

    ads1220_writeRegister( CONFIG_REG0_ADDRESS , m_config_reg0);
    ads1220_writeRegister( CONFIG_REG1_ADDRESS , m_config_reg1);
    ads1220_writeRegister( CONFIG_REG2_ADDRESS , m_config_reg2);
    ads1220_writeRegister( CONFIG_REG3_ADDRESS , m_config_reg3);

}

void ads1220_writeRegister(uint8_t address, uint8_t value)
{
    digitalWrite(ADS1220_CS_PIN,LOW);
    SPI.transfer(WREG|(address<<2));
    SPI.transfer(value);
    digitalWrite(ADS1220_CS_PIN,HIGH);
}

uint8_t ads1220_readRegister(uint8_t address)
{
    uint8_t data;

    digitalWrite(ADS1220_CS_PIN,LOW);
    SPI.transfer(RREG|(address<<2));
    data = SPI.transfer(SPI_MASTER_DUMMY);
    digitalWrite(ADS1220_CS_PIN,HIGH);

    return data;
}

void ads1220_SPI_Command(unsigned char data_in)
{
    digitalWrite(ADS1220_CS_PIN, LOW);
    SPI.transfer(data_in);
    digitalWrite(ADS1220_CS_PIN, HIGH);
}

void Hold_SPI()
{
    while (SPI_available == false){} // Ждем освобождения SPI
    SPI_available = false; // Займём SPI
}

void ads1220_StartConv_Release_SPI()
{
    ads1220_SPI_Command(START);  //ReStart continuous conversion mode
    SPI_available = true; // Освободим SPI
    new_adc_data = false;
}

void ads1220_Start_Conv()
{
    Hold_SPI();
    ads1220_StartConv_Release_SPI();
}

void ads1220_set_data_rate(int datarate)
{
    m_config_reg1 &= ~REG_CONFIG1_DR_MASK;
    m_config_reg1 |= datarate;
    ads1220_writeRegister(CONFIG_REG1_ADDRESS,m_config_reg1);
}

void ads1220_select_mux_channels(int channels_conf)
{
    Hold_SPI();
    m_config_reg0 &= ~REG_CONFIG0_MUX_MASK;
    m_config_reg0 |= channels_conf;
```

```
    ads1220_writeRegister(CONFIG_REG0_ADDRESS,m_config_reg0);
    ads1220_StartConv_Release_SPI();
}

void ads1220_set_pga_gain(int pgagain)
{
    Hold_SPI();
    m_config_reg0 &= ~REG_CONFIG0_PGA_GAIN_MASK;
    m_config_reg0 |= pgagain<<1 ;
    ads1220_writeRegister(CONFIG_REG0_ADDRESS,m_config_reg0);
    ads1220_StartConv_Release_SPI();
    PGA_gain = 1<<pgagain;
}

void ads1220_set_idac(int idac)
{
    Hold_SPI();
    m_config_reg2 &= ~REG_CONFIG2_IDAC_MASK;
    m_config_reg2 |= idac ;
    ads1220_writeRegister(CONFIG_REG2_ADDRESS,m_config_reg2);
    ads1220_StartConv_Release_SPI();
}

void ads1220_select_i1mux_channel(int channel_conf)
{
    Hold_SPI();
    m_config_reg3 &= ~REG_CONFIG3_I1MUX_MASK;
    m_config_reg3 |= channel_conf;
    ads1220_writeRegister(CONFIG_REG3_ADDRESS,m_config_reg3);
    ads1220_StartConv_Release_SPI();
}

void ads1220_select_i2mux_channel(int channel_conf)
{
    Hold_SPI();
    m_config_reg3 &= ~REG_CONFIG3_I2MUX_MASK;
    m_config_reg3 |= channel_conf;
    ads1220_writeRegister(CONFIG_REG3_ADDRESS,m_config_reg3);
    ads1220_StartConv_Release_SPI();
}

int32_t ads1220_Read_Data()
{
    static byte SPI_Buff[3];
    int32_t mResult32=0;
    long int bit24;
    digitalWrite(ADS1220_CS_PIN,LOW);                          //Take CS low
    SPI_Buff[0] = SPI.transfer(SPI_MASTER_DUMMY);
    SPI_Buff[1] = SPI.transfer(SPI_MASTER_DUMMY);
    SPI_Buff[2] = SPI.transfer(SPI_MASTER_DUMMY);
    digitalWrite(ADS1220_CS_PIN,HIGH);                 //  Clear CS to high

    bit24 = SPI_Buff[0];
    bit24 = (bit24 << 8) | SPI_Buff[1];
    bit24 = (bit24 << 8) | SPI_Buff[2];                                // Converting 3 bytes
to a 24 bit int

    bit24= ( bit24 << 8 );
    mResult32 = ( bit24 >> 8 );                       // Converting 24 bit two's complement to
32 bit two's complement

    return mResult32;
}
```