Специальное программное обеспечение цифрового устройства для температурных 4-х зондовых измерений

В состав программно-управляемого источника-измерителя цифрового устройства для температурных 4-х зондовых измерений входит модуль управляющего контроллера ArduinoNano, который обеспечивает следующие функции:

- прием и обработку управляющих директив по последовательному интерфейсу USB по протоколу RS-232;
- управление микросхемами ADS1220 и ADS1220-1;
- предварительную цифровую обработку данных, поступающих от измерительных каналов – накопление с усреднением без экстремальных значений, конвертирование отсчетов АЦП в напряжение (мВ) с учетом коэффициента усиления и т.п.;
- передачу данных по запросу рабочей станции, работающей под управлением прикладного программного обеспечения.

Специальное программное обеспечение (СПО) реализуется как прошивка для микроконтроллера модуля ArduinoNano.

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

СПО создано и загружено в микроконтроллер в среде разработки Arduino.

Данное СПО является общим для устройств:

1. Цифровое устройство для температурных 4-х зондовых измерений (V.3);
2. Цифровое устройство для 4-х зондовых измерений.

В данном разделе описаны алгоритмы и команды, используемые в «1. Цифровое устройства для температурных 4-х зондовых измерений (V.3)».

Оглавление

1 Описание логической структуры СПО

Структурно СПО состоит из процедур инициализации и конфигурации периферии микроконтроллера и микросхем АЦП, выполняемых однократно при запуске СПО, главного рабочего цикла – интерпретатора командной строки, вызывающего по мере необходимости рабочие процедуры (управление микросхемами 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.

Команды делятся на рутинные и макрокоманды. Рутинные команды служат для управления отдельными узлами ADS1220. Такие команды предназначены в основном для отладки. Макрокоманды осуществляют последовательности действий.

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

3.1 Рутинные команды

ADC_CONFIG? – выводит значения 4-х регистров конфигурации ADS1220 в шестнадцатеричном формате;

ADC_CONFIG XX XX XX XX – задает значения 4-х регистров конфигурации ADS1220. XX – значение регистра в шестнадцатеричном формате;

GAIN n – задает коэффициент усиления масштабного усилителя ADS1220 (**n**: 0-7, коэффициент усиления = 2<<n (1, 2, 4, 8, 16, 32, 64, 128);

VREF n – подключает соответствующий источник опорного напряжения (n:0-1, где 0:2.048V, 1:REFP0-REFN0)

MUX n – конфигурирует входной коммутатор ADS1220 (**n**:0-14, где 0:AIN0_AIN1, 1:AIN0_AIN2, 2:AIN0_AIN3, 3:AIN1_AIN2, 4:AIN1_AIN3, 5:AIN2_AIN3, 6:AIN1_AIN0, 7:AIN3_AIN2, 12:REFP_REFN);

IDAC n – конфигурирует встроенные в ADS1220-1 источники тока (**n**: 0-7, где 0: Off, 1: 10uA, 2: 50uA, 3: 100uA, 4: 250uA, 5: 500uA, 6: 1mA, 7: 1.5mA);

IMUX n – управляет подключением встроенных в ADS1220-1 источников тока к выводам ADS1220-1 (**n**: 0-2, где 0:Off, 1:REFP0+REFN0, 2:AIN3);

GND_REFx n – управляет подключением выводов ADS1220-1 REFP0 и REFN0 к общему проводу (**n**:0-2, где 0:OPEN_ALL, 1:GND_REFP0;OPEN_REFN0, 2:GND_REFN0;OPEN_REFP0);

TS? – считывает значение температуры корпуса ADS1220

DATA? – ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов 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>);

3.2 Макрокоманды

TVI? – выполняет последовательность действий по измерению напряжения термопары, температуры корпуса ADS1220, напряжения на измерительном входе ADS1220 с компенсацией нулевого смещения и проверяет режим работы источника тока (ответ: Thermocouple_mV<CR> adc_degrees(C)<CR> Probe_mV<CR> Current_uA==0-1500 if OK, -1 if not stabilized<CR> gain).

RUI? – выполняет последовательность действий по измерению сопротивления RTD, напряжения на измерительном входе ADS1220 с компенсацией нулевого смещения и проверяет режим работы источника тока (ответ: RTD_HEX<CR> Probe_mV<CR> Current_uA<CR> gain)

Перед выполнением макрокоманд необходимо сконфигурировать устройство рутинными командами GAIN n, IDAC n, IMUX 1.

4 Алгоритм исполнения программы СПО

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

- инициализация внутренних переменных;
- инициализация портов ввода/вывода;
- инициализация микросхем ADS1220 и ADS1220-1;
- 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.

Микросхема ADS1220 работает в режиме непрерывных преобразований. После окончания очередного преобразования на выходе ADS1220 $\overline{\mathrm{DRDY}}$ 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 ADS1220 осуществляется по интерфейсу SPI. Обращение к ADS1220 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, команд и макрокоманд

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 GAIN

- Ждет ввода параметра и проверяет его корректность
- Вызывает процедуру **ads1220_set_pga_gain**
- Модифицирует значение переменной **PGA_gain**

4.1.5 VREF n

- Ждет ввода параметра и проверяет его корректность
- Вызывает процедуру **ads1220_set_vref**

4.1.6 MUX, IDAC, IMUX

- Ждет ввода параметра и проверяет его корректность
- Модифицирует значение соответствующего регистра конфигурации АЦП, вызвав процедуру **ads1220_select_mux_channels**, **ads1220_set_idac** или **ads1220_1_imux**

4.1.7 GND_REFx

- Ждет ввода параметра и проверяет его корректность
- В зависимости от значения параметра подключает выводы микроконтроллера 17 и 19 к входу АЦП микроконтроллера или к общему проводу
- Посылает команду на перезапуск преобразования АЦП ADS1220

4.1.8 TS?

- Подключает к входу АЦП ADS1220 внутренний температурный датчик
- Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов
- Осуществляет преобразование отсчетов АЦП в градусы по Цельсию
- Отключает внутренний температурный датчик
- Выводит температуру корпуса ADS1220 в градусах по Цельсию

4.1.9 DATA?

- Ждет окончания AD преобразования, проверяя флаг **new_adc_data**
- Пересчитывает значение отсчета АЦП в напряжение с учетом значения переменной **PGA_gain**
- Сбрасывает флаг **new_adc_data**
- Выводит значения отсчета АЦП в mV, в HEX формате и значение переменной **PGA_gain**

4.1.10 TVI?

- Измеряет температуру корпуса ADS1220
    - Подключает к входу АЦП ADS1220 внутренний температурный датчик
    - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов
    - Осуществляет преобразование отсчетов АЦП в градусы по Цельсию
    - Отключает внутренний температурный датчик
- Измеряет напряжение на термопаре
    - Конфигурирует входной коммутатор ADS1220 в состояние AIN0_AIN1
    - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.

    - Конфигурирует входной коммутатор ADS1220 в состояние AIN1_AIN0
    - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
    - Из первого результата вычитает второй и делит на два.
- Измеряет напряжение между зондами 2 и 3
    - Подключает вывод ADS1220-1 REFP0 к общему проводу.
    - Подключает вывод ADS1220-1 REFN0 к входу АЦП микроконтроллера ArduinoNano.
        - Конфигурирует входной коммутатор ADS1220 в состояние AIN2_AIN3
        - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
        - Переключает полярность измерительного входа ADS1220.
        - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
        - Из первого результата вычитает второй и делит на два.
    - Подключает вывод ADS1220-1 REFN0 к общему проводу.
    - Подключает вывод ADS1220-1 REFP0 к входу АЦП микроконтроллера ArduinoNano.
        - Конфигурирует входной коммутатор ADS1220 в состояние AIN3_AIN2
        - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
        - Переключает полярность измерительного входа ADS1220.
        - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
        - Из первого результата вычитает второй и делит на два.
    - Из первого измеренного напряжения вычитает второе измеренное напряжение и делит на два.
- Передает в последовательный порт напряжение термопары, температуру корпуса ADS1220 и напряжение между зондами 2 и3.
- Считывает отсчет АЦП микроконтроллера ArduinoNano, подключенного к выходу источника тока ADS1220-1 REFP0.
- Если отсчет АЦП превышает значение 3.7 В, то передает в последовательный порт «-1», иначе передает в последовательный порт значение, заданного командой IDAC, тока в мкА.

4.1.11 RUI?

- Измеряет напряжение, падающее на датчике RTD
    - Подключает к выводу ADS1220-1 AIN3 внутренний источник тока и задает ток 10 uA
    - В качестве опорного напряжения ADS1220 использует напряжение между выводами ADS1220 REFP0 и REFN0, к которым подключен резистор R_{REF} токовой цепи датчика RTD.
    - Устанавливает коэффициент усиления равным 32.
    - Конфигурирует входной коммутатор ADS1220 в состояние AIN0_AIN1
    - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
    - Конфигурирует входной коммутатор ADS1220 в состояние AIN1_AIN0
    - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
    - Из первого результата вычитает второй и делит на два.
- Передает в последовательный порт измеренное напряжение.

- Измеряет напряжение между зондами 2 и 3
    - Подключает источник тока микросхемы ADS1220-1 к выводам REFP0 и REFN0
    - В качестве опорного напряжения ADS1220 использует внутреннее 2.048 В.
    - Устанавливает ток стабилизации, заданный ранее командой IDAC.
    - Устанавливает коэффициент усиления, заданный ранее командной GAIN.
    - Подключает вывод ADS1220-1 REFP0 к общему проводу.
    - Подключает вывод ADS1220-1 REFN0 к входу АЦП микроконтроллера ArduinoNano.
        - Конфигурирует входной коммутатор ADS1220 в состояние AIN2_AIN3
        - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
        - Переключает полярность измерительного входа ADS1220.
        - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
        - Из первого результата вычитает второй и делит на два.
    - Подключает вывод ADS1220-1 REFN0 к общему проводу.
    - Подключает вывод ADS1220-1 REFP0 к входу АЦП микроконтроллера ArduinoNano.
        - Конфигурирует входной коммутатор ADS1220 в состояние AIN3_AIN2
        - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
        - Переключает полярность измерительного входа ADS1220.
        - Ждет окончания 8-ми преобразований ADS1220, отбрасывает два экстремума, усредняет оставшиеся 6 результатов.
        - Из первого результата вычитает второй и делит на два.
    - Из первого измеренного напряжения вычитает второе измеренное напряжение и делит на два.
- Передает в последовательный порт измеренное напряжение.
- Считывает отсчет АЦП микроконтроллера ArduinoNano, подключенного к выходу источника тока ADS1220-1 REFP0.
- Если отсчет АЦП превышает значение 3.7 В, то передает в последовательный порт «-1», иначе передает в последовательный порт значение, заданного командой IDAC, тока в мкА.

4.2 Исходный текст специального программного обеспечения цифрового устройства для температурных 4-х зондовых измерений (V.3)

```
////////////////////////////////////////////////////////////////////
//
//  Подключение АЦП:
//
//     ┌─>>──────[ads1:AIN3] <─ [ads1:IDAC1]
//     ├─>>──────[ads0:AIN0]
//   [RTD]
//     ├─>>──────[ads0:AIN1]
//     ├──────────[ads0:REFP0]
//   [75k]
//     ├──────────[ads0:REFN0]
//     ┴
//                ┌──────  [Arduino A3 (17)]
//                │
//     I1 ─>>─────┴──[ads1:REFP0] <─ [ads1:IDAC1]
//     U1 ─>>──────[ads0:AIN2]
//     U2 ─>>──────[ads0:AIN3]
//     I2 ─>>─────┬──[ads1:REFN0] <─ [ads1:IDAC2]
//                │
//                └──────  [Arduino A5 (19)]

#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_1_CS_PIN  6
#define ADS1220_DRDY_PIN  2

//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_CONFIG1_TS_MASK       0x02
#define REG_CONFIG2_VREF_MASK     0xC0
#define REG_CONFIG2_IDAC_MASK     0x07
#define REG_CONFIG3_I1MUX_MASK    0xE0
#define REG_CONFIG3_I2MUX_MASK    0x1C

#define TS_ENABLE   0x02
#define TS_DISABLE  0x00

#define VREF_2048   0x00
#define VREF_REFx0  0x40

#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 MUX_AIN0_AVSS   0x80
#define MUX_AIN1_AVSS   0x90
#define MUX_AIN2_AVSS   0xA0
#define MUX_AIN3_AVSS   0xB0
#define MUX_REFP_REFN   0xC0 //(V(REFPx) - V(REFNx)) / 4
#define MUX_AVDD_AVSS   0xD0 //{AVDD - AVSS) / 2
#define MUX_SHORT_AIN   0xE0 // Short AINp, AINn to (AVDD+AVSS)/2

#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 I1MUX_REFP0 0xA0
#define I1MUX_REFN0 0xC0

#define I2MUX_OFF   0x00
#define I2MUX_AIN0  0x04
#define I2MUX_AIN1  0x08
#define I2MUX_AIN2  0x0C
#define I2MUX_AIN3  0x10
#define I2MUX_REFP0 0x14
#define I2MUX_REFN0 0x18

const int16_t idac[] = {0, 10, 50, 100, 250, 500, 1000, 1500};

uint8_t m_config_reg0, m_config_reg1, m_config_reg2, m_config_reg3, i;

volatile long adc_data, adc_data1;
long adc_res[8];
uint8_t ads_idac, gain=0, PGA_gain=1;
float adc_mV, adc_mV_1, probe_mV, adc_degrees, thermocouple_mV;
volatile bool SPI_available = true, new_adc_data = false;

char inpstr[13];
const char err[]="\rUnknown cmd - type HELP";
```

```
uint8_t n;
word dddd;
```

```
void setup() {
  pinMode(ADS1220_CS_PIN, OUTPUT);
  pinMode(ADS1220_1_CS_PIN, OUTPUT);
  pinMode(ADS1220_DRDY_PIN, INPUT);
  SPI.begin();
  SPI.setBitOrder(MSBFIRST);
  SPI.setDataMode(SPI_MODE1);
  SPI.setClockDivider(SPI_CLOCK_DIV128);
  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.
  digitalWrite(17, LOW); // Подготовим вывод A3 к заземлению
  digitalWrite(19, LOW); // Подготовим вывод A5 к заземлению
}
```

```
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"
    "GAIN n (n:0-7, and the real Gain = 2<<n (1,2,4,8,16,32,64,128)\r"
    "VREF n (n:0-1, where 0:2.048V, 1:REFP0-REFN0)\r"
    "MUX n (n:0-14, where 0:AIN0_AIN1, 1:AIN0_AIN2, 2:AIN0_AIN3, 3:AIN1_AIN2, 4:AIN1_AIN3,
5:AIN2_AIN3, 6:AIN1_AIN0, 7:AIN3_AIN2, 12:REFP_REFN)\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"
    "IMUX n (n:0-2, where 0:Off, 1:REFP0+REFN0, 2:AIN3)\r"
    "GND_REFx n (n:0-2, where 0:OPEN_ALL, 1:GND_REFP0;OPEN_REFN0, 2:GND_REFN0;OPEN_REFP0)\r"
    "TS? (answer: adc_degrees (C))\r"
    "DATA? (answer: data_mV<CR> data_hex<CR> gain)\r"
    "TVI? (answer: Thermocouple_mV<CR> adc_degrees(C)<CR> Probe_mV<CR> Current_uA==0-1500 if
OK, -1 if not stabilized<CR> gain)\r"
    "RUI? (answer: RTD_HEX<CR> Probe_mV<CR> Current_uA<CR> gain)\r"
    "UI? (answer: Probe_mV<CR> Current_uA<CR> gain)\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_CS_PIN);
        ads1220_writeRegister( CONFIG_REG1_ADDRESS , m_config_reg1, ADS1220_CS_PIN);
        ads1220_writeRegister( CONFIG_REG2_ADDRESS , m_config_reg2, ADS1220_CS_PIN);
        ads1220_writeRegister( CONFIG_REG3_ADDRESS , m_config_reg3, ADS1220_CS_PIN);
        SPI_available = true; // Освободим SPI
        }
```

```
}
else if (strcasecmp(inpstr,"GAIN")==0){
      //считываем и проверяем значение GAIN
      if (scanf("%2hhu", &gain)==0){
        printf(err);
      }
      else if (gain>7){
        printf(err);
      }
      else{
        ads1220_set_pga_gain(gain<<1); // Установим GAIN
        PGA_gain = 1<<gain;
      }
}
else if (strcasecmp(inpstr,"VREF")==0){
      //считываем и проверяем значение VREF
      if (scanf("%2hhu", &n)==0){
        printf(err);
      }
      else if (n>1){
        printf(err);
      }
      else{
        ads1220_set_vref(n<<6); // Установим источник VREF
      }
}
else if (strcasecmp(inpstr,"MUX")==0){
      //считываем и проверяем значение MUX
      if (scanf("%2hhu", &n)==0){
        printf(err);
      }
      else if (n>14){
        printf(err);
      }
      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(err);
      }
      else{
        ads1220_set_idac(n); // Установим IDAC
        ads_idac = n;
      }
}
else if (strcasecmp(inpstr,"IMUX")==0){
      //считываем и проверяем значение IMUX
      if (scanf("%2hhu", &n)==0){
        printf(err);
      }
      else if (n>2){
        printf(err);
      }
      else{
          ads1220_1_imux(n); // MUX IDAC
      }
}
else if (strcasecmp(inpstr,"GND_REFx")==0){
      //считываем и проверяем значение I1MUX
      if (scanf("%2hhu", &n)==0){
        printf(err);
      }
      else if (n>2){
```

```
        printf(err);
      }
      else{
       if (n == 0){
         pinMode(17,INPUT);
         pinMode(19,INPUT);
       }
       else if (n == 1){
         pinMode(17,OUTPUT);
         pinMode(19,INPUT);
       }
       else{
         pinMode(19,OUTPUT);
         pinMode(17,INPUT);
       }
       ads1220_Start_Conv();  //ReStart continuous conversion mode
      }
}
else if (strcasecmp(inpstr,"TS?")==0){
      ads1220_set_TS(TS_ENABLE); // Подключим температурный датчик
      Wait_Read_ADC_Data();
      adc_data&=0xFFFFFF;
      adc_data>>=10;
      adc_degrees=adc_data * 0.03125;
      ads1220_set_TS(TS_DISABLE); // Отключим температурный датчик
      Serial.println(adc_degrees,4);
}
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,"TVI?")==0){
  /**** Температура корпуса ADC *****/
      ads1220_set_TS(TS_ENABLE); // Подключим температурный датчик
      Wait_Read_ADC_Data();
      adc_data&=0xFFFFFF;
      adc_data>>=10;
      adc_degrees=adc_data * 0.03125;
      ads1220_set_TS(TS_DISABLE); // Отключим температурный датчик
  /**** Термопара ****/
      ads1220_select_mux_channels(MUX_AIN0_AIN1);
      Wait_Read_ADC_Data();
      thermocouple_mV=adc_mV;
      ads1220_select_mux_channels(MUX_AIN1_AIN0);
      Wait_Read_ADC_Data();
      thermocouple_mV=(thermocouple_mV-adc_mV)/2;
  /**** Напряжение между 2 и 3 зондами ****/
      pinMode(17,OUTPUT);
      pinMode(19,INPUT);
      probe_mV=get_probe_mV();
      pinMode(19,OUTPUT);
      pinMode(17,INPUT);
      probe_mV=(probe_mV-get_probe_mV())/2;

      Serial.println(thermocouple_mV,8);
      Serial.println(adc_degrees);
      Serial.println(probe_mV,8);
  /**** Проверка: IDAC стабилизирует? ****/// 0-1500 if OK, -1 if not stabilized
      if (analogRead(17)<800){
       Serial.println(idac[ads_idac]);
      }
      else {
       Serial.println("-1");
      }
      Serial.println(PGA_gain);
}
else if (strcasecmp(inpstr,"RUI?")==0){
```

```
    /**** RTD ****/
        ads1220_1_imux(2); // ads1:AIN3 <- 10uA
        ads1220_set_vref(VREF_REFx0); // 75k * 10uA = 0.75V
        delay(1);
        ads1220_set_pga_gain(5<<1); // gain=32, при Жидком Гелии: 1400 Ом * 10uA = 0.014V;
0.014 * 32 = 0.448 V.
        ads1220_select_mux_channels(MUX_AIN0_AIN1);
        Wait_Read_ADC_Data();
        adc_data1=adc_data;
        ads1220_select_mux_channels(MUX_AIN1_AIN0);
        Wait_Read_ADC_Data();
        adc_data=(adc_data1-adc_data)/2;
        Serial.println(adc_data,HEX);
    /**** Напряжение между 2 и 3 зондами ****/
        ads1220_1_imux(1); // ads1:REFx0 <- ads_idac
        ads1220_set_vref(VREF_2048);
        ads1220_set_idac(ads_idac); // Установим IDAC
        delay(1);
        ads1220_set_pga_gain(gain<<1); // Установим GAIN
        pinMode(17,OUTPUT);
        pinMode(19,INPUT);
        probe_mV=get_probe_mV();
        pinMode(19,OUTPUT);
        pinMode(17,INPUT);
        probe_mV=(probe_mV-get_probe_mV())/2;
        Serial.println(probe_mV,8);
    /**** Проверка: IDAC стабилизирует? ****/// 0-1500 if OK, -1 if not stabilized
        if (analogRead(17)<800){
         Serial.println(idac[ads_idac]);
        }
        else {
         Serial.println("-1");
        }
        Serial.println(PGA_gain);
  }
  else if (strcasecmp(inpstr,"UI?")==0){
        pinMode(17,OUTPUT);
        pinMode(19,INPUT);
        probe_mV=get_probe_mV();
        pinMode(19,OUTPUT);
        pinMode(17,INPUT);
        probe_mV=(probe_mV-get_probe_mV())/2;
        Serial.println(probe_mV,8);
    /**** Проверка: IDAC стабилизирует? ****/// 0-1500 if OK, -1 if not stabilized
        if (analogRead(17)<800){
         Serial.println(idac[ads_idac]);
        }
        else {
         Serial.println("-1");
        }
        Serial.println(PGA_gain);
  }
  else{
    printf(err);
  }
}

float get_probe_mV (){
  ads1220_select_mux_channels(MUX_AIN2_AIN3);
  Wait_Read_ADC_Data();
  adc_mV_1=adc_mV;
  ads1220_select_mux_channels(MUX_AIN3_AIN2);
  Wait_Read_ADC_Data();
  adc_mV_1=(adc_mV_1 - adc_mV)/2;
  return adc_mV_1;
}

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<8; i++) // Сделаем 8 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<(8-1); 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<7; i++) adc_data+=adc_res[i]; // суммируем 6 средних элемента массива с 2 по 7
  adc_data/=6; // усредняем
  adc_mV = adc_data;
  adc_mV *= VREF;
  adc_mV /=FSR;
  adc_mV /=PGA_gain;
}

void ads1220_begin()
{
    ads1220_SPI_Command(RESET, ADS1220_CS_PIN);
    ads1220_SPI_Command(RESET, ADS1220_1_CS_PIN);
    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 = 0x20;   //Default settings: internal VREF, 50 Hz rejection only, 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_CS_PIN);
    ads1220_writeRegister( CONFIG_REG1_ADDRESS , m_config_reg1, ADS1220_CS_PIN);
    ads1220_writeRegister( CONFIG_REG2_ADDRESS , m_config_reg2, ADS1220_CS_PIN);
    ads1220_writeRegister( CONFIG_REG3_ADDRESS , m_config_reg3, ADS1220_CS_PIN);

}

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

void ads1220_Start_Conv()
{
    while (SPI_available == false){} // Ждем освобождения SPI
    SPI_available = false; // Займём SPI
    ads1220_SPI_Command(START, ADS1220_CS_PIN);
    SPI_available = true; // Освободим SPI
    new_adc_data = false;
}

void ads1220_set_TS(int TS)
{
    while (SPI_available == false){} // Ждем освобождения SPI
    SPI_available = false; // Займём SPI
    m_config_reg1 &= ~REG_CONFIG1_TS_MASK;
    m_config_reg1 |= TS;
    ads1220_writeRegister(CONFIG_REG1_ADDRESS,m_config_reg1, ADS1220_CS_PIN);
    ads1220_SPI_Command(START, ADS1220_CS_PIN);  //ReStart continuous conversion mode
    SPI_available = true; // Освободим SPI
    new_adc_data = false;
}

void ads1220_select_mux_channels(int channels_conf)
{
    while (SPI_available == false){} // Ждем освобождения SPI
    SPI_available = false; // Займём SPI
    m_config_reg0 &= ~REG_CONFIG0_MUX_MASK;
    m_config_reg0 |= channels_conf;
    ads1220_writeRegister(CONFIG_REG0_ADDRESS,m_config_reg0, ADS1220_CS_PIN);
    ads1220_SPI_Command(START, ADS1220_CS_PIN);  //ReStart continuous conversion mode
    SPI_available = true; // Освободим SPI
    new_adc_data = false;
}

void ads1220_set_pga_gain(int pgagain)
{
    while (SPI_available == false){} // Ждем освобождения SPI
    SPI_available = false; // Займём SPI
    m_config_reg0 &= ~REG_CONFIG0_PGA_GAIN_MASK;
    m_config_reg0 |= pgagain ;
    ads1220_writeRegister(CONFIG_REG0_ADDRESS,m_config_reg0, ADS1220_CS_PIN);
    ads1220_SPI_Command(START, ADS1220_CS_PIN);  //ReStart continuous conversion mode
    SPI_available = true; // Освободим SPI
    new_adc_data = false;
}

void ads1220_set_vref(int vref)
{
    while (SPI_available == false){} // Ждем освобождения SPI
    SPI_available = false; // Займём SPI
```

```
    m_config_reg2 &= ~REG_CONFIG2_VREF_MASK;
    m_config_reg2 |= vref ;
    ads1220_writeRegister(CONFIG_REG2_ADDRESS,m_config_reg2, ADS1220_CS_PIN);
    ads1220_SPI_Command(START, ADS1220_CS_PIN);  //ReStart continuous conversion mode
    SPI_available = true; // Освободим SPI
    new_adc_data = false;
}

void ads1220_set_idac(int idac)
{
    while (SPI_available == false){} // Ждем освобождения SPI
    SPI_available = false; // Займём SPI
//    m_config_reg2 &= ~REG_CONFIG2_IDAC_MASK;
//    m_config_reg2 |= idac ;
    ads1220_writeRegister(CONFIG_REG2_ADDRESS,idac, ADS1220_1_CS_PIN);
    ads1220_SPI_Command(START, ADS1220_CS_PIN);  //ReStart continuous conversion mode
    SPI_available = true; // Освободим SPI
    new_adc_data = false;
}

void ads1220_1_imux(uint8_t nn)
{
    while (SPI_available == false){} // Ждем освобождения SPI
    SPI_available = false; // Займём SPI
    if (nn == 0){
      ads1220_writeRegister(CONFIG_REG3_ADDRESS, 0x00, ADS1220_1_CS_PIN);
      }
    else if (nn == 1){
      ads1220_writeRegister(CONFIG_REG3_ADDRESS, 0xB8, ADS1220_1_CS_PIN);
      }
    else {
      ads1220_writeRegister(CONFIG_REG2_ADDRESS,IDAC_10uA, ADS1220_1_CS_PIN); // Больше 10 uA
нельзя!!!
      ads1220_writeRegister(CONFIG_REG3_ADDRESS, 0x80, ADS1220_1_CS_PIN);
      }
    ads1220_SPI_Command(START, ADS1220_CS_PIN);  //ReStart continuous conversion mode
    SPI_available = true; // Освободим SPI
    new_adc_data = false;
}

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;
}
```