

# The Answer is 42!!

Electronics, Robotics and Computing Tutorials for Everyone!

## About Me

Alexander Lang

[View my complete profile](#)

Sunday, 17 September 2017

## Pulse Oximeter Functionality for a Medical Device

It has again been some time since I did any work on the medical device I have been developing, I've as usual, been busy! The previous post is here in case it is of interest:

<http://langster1980.blogspot.co.uk/2017/06/myoware-muscle-sensor-circuits-from.html>

The gentlemen for whom I'm developing this hardware for has requested some additional functionality. The additional functionality requested is a Pulse Oximetry measurement. Pulse Oximetry is the measurement of a person's pulse along with how much oxygen is present within their blood. It is a common measurement made by medical practitioners to ensure their patients are in good health. I suspect for the medical device, this information will be correlated with a person's breathing to assess how well a person's lungs are working and how much oxygen from the air is getting into their blood.

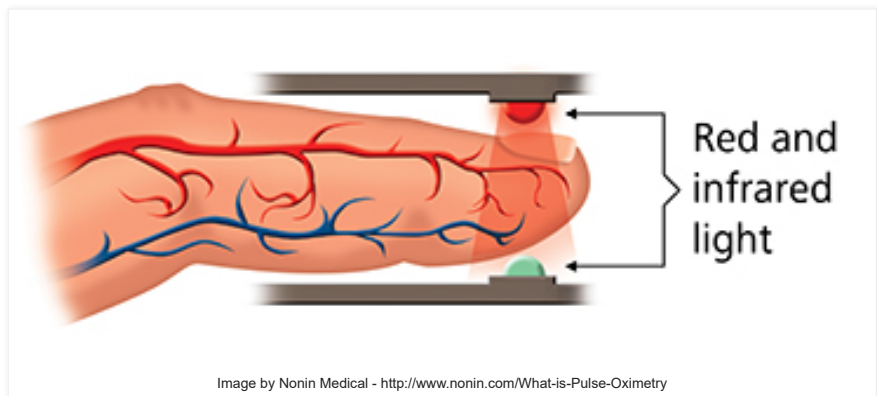


Image by Nonin Medical - <http://www.nonin.com/What-is-Pulse-Oximetry>

Here is an article by Nonin Medical on Pulse Oximetry:

<http://www.nonin.com/What-is-Pulse-Oximetry>

Here is the wikipedia article on Pulse Oximetry:

[https://en.wikipedia.org/wiki/Pulse\\_oximetry](https://en.wikipedia.org/wiki/Pulse_oximetry)

Here is another article on Pulse Oximetry:

[https://www.howequipmentworks.com/pulse\\_oximeter/](https://www.howequipmentworks.com/pulse_oximeter/)

The measurement works by shining a bright light through the a suitable point on the patient's skin and detecting the diffused light and from that the pulse and oxygen levels can be calculated. The units used are SpO<sub>2</sub> which stands for Peripheral Oxygen Saturation where:

Sp = Peripheral saturation

O<sub>2</sub> = Chemical symbol for an Oxygen molecule

There is a formula associated with Pulse Oximetry:

$$SpO_2 = \frac{HbO_2}{HbO_2 + Hb}$$

Where HbO<sub>2</sub> is Oxygenated Haemoglobin and Hb is Deoxygenated Haemoglobin\*

SpO<sub>2</sub> is presented as a percentage and most people in normal conditions exhibit 94 to 99% saturation of oxygen in their blood. If a person were unwell due to breathing difficulty then their SpO<sub>2</sub> is lower. If a person's SpO<sub>2</sub> is below 65% then their ability to think clearly is impaired and if below 55% most people have lost consciousness.

\*Haemoglobin is the red coloured metallo-protein within human blood which contains iron and carries oxygen around the body.

So that is the theory! Lets talk about how one might design an electronic circuit to measure a person's SpO<sub>2</sub> level. We need to shine a light through an exposed part of the patient's skin say on the top of their index finger and measure the amount of light present at the bottom of the finger after the light has travelled through their skin. At the same time the light signal needs to be converted to an electrical signal and then filtered. It then will probably need to be amplified and finally converted from an analogue signal to a digital signal for processing by a microcontroller and finally displaying the

resulting measurement.

It is not a particularly difficult circuit to design and plenty of other people have already done so. I have put some of the sites I used as research below for interest:

<https://www.swharden.com/wp/2013-04-14-simple-diy-ecg-pulse-oximeter-version-2/>

<http://www.instructables.com/id/Simple-DIY-Pulse-Sensor/>

<http://www.elektroda.pl/rtvforum/topic1655378.html?l=en>

I may yet design my own implementation as it could be one way to reduce costs. There is an Arduino R3 shield available though and it's open source. It is made by ProtoCentral and is called the Proto Central AFE4490 Pulse Oximeter Shield V2. It cost £71.12 which I actually think is a bit expensive however it does look to be well designed and implemented.

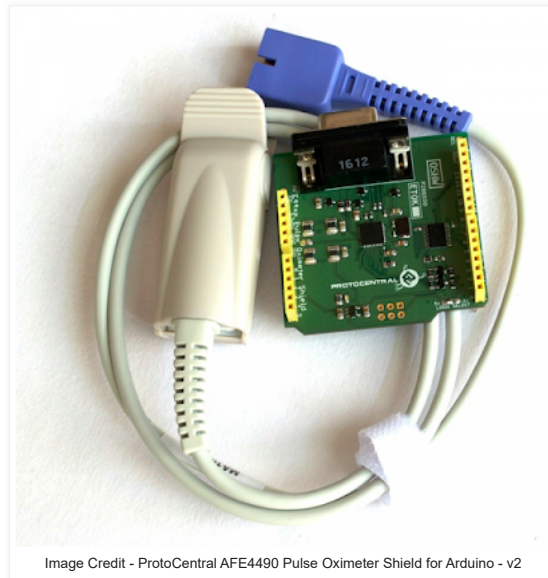
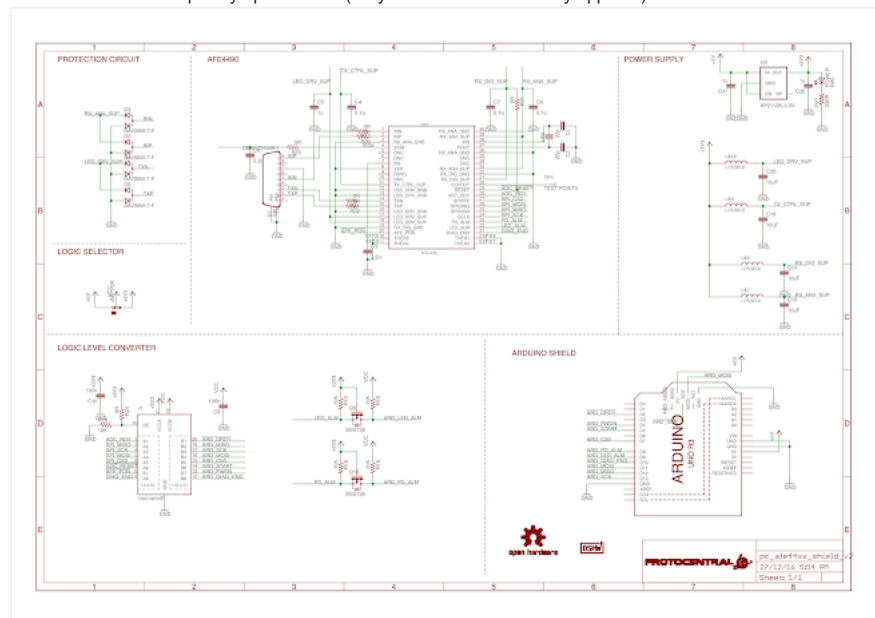


Image Credit - ProtoCentral AFE4490 Pulse Oximeter Shield for Arduino - v2

It can be bought from their website below:

[Pulse Oximeter Arduino Shield](#)

Because the device is completely open source (Very cool - I whole heartedly approve) the schematic is below:



The whole circuit is based around a very clever integrated circuit by Texas Instruments - the AFE4490. The datasheet for this device is here:

<http://www.ti.com/lit/ds/sbas602h/sbas602h.pdf>

Basically the optical signal from the supplied finger clip connects to the nine way 'D type' connector on the Arduino shield. The signal output is protected from being too high by the the circuit labelled 'Protection circuit' The diodes ensure that the electrical signals passed to and from the clip can never be greater than 3.3 Vdc which ensures that no damage can occur.

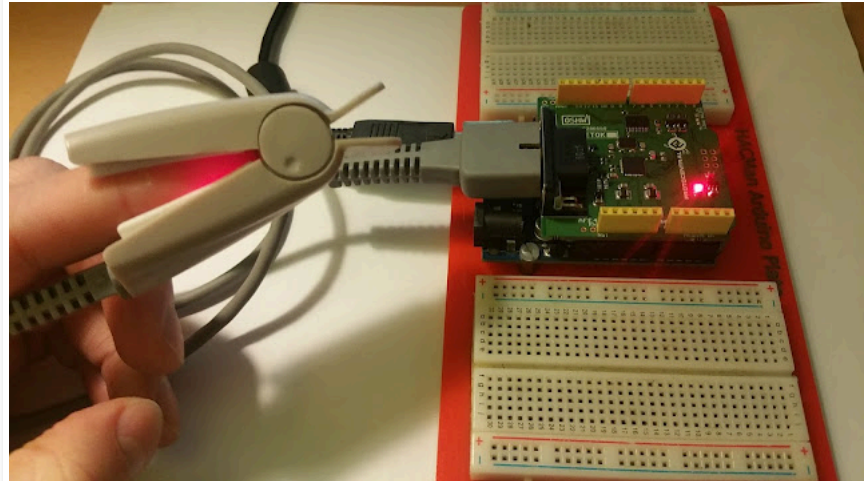
The analogue signal sent from the clip is processed by the AD4490 device which is a complete analogue front end device for pulse oximetry. The device provides the electrical signals to drive the sensor clip, a 22 bit analogue to digital converter to change the analogue signal output from the sensor to a form which can be easily read into a microcontroller using the Serial Peripheral Interface (SPI) protocol.

A logic level converter device is present to ensure that the 3.3 Vdc signals from the AD4490 device are turned into 5 Vdc so that they are correctly read into the arduino and processed appropriately. There are two FET logic level converter circuits present to convert the 3.3 Vdc LED diagnostic signals to 5 Vdc for processing by the Arduino.

The circuit looks to be exactly how one would use the AD4490 having read the datasheet in it's entirety. Reading and understanding or interpreting datasheets is a true skill needed for aspiring electronics design engineers.

[https://github.com/Protocentral/AFE4490\\_Oximeter/tree/master/arduino](https://github.com/Protocentral/AFE4490_Oximeter/tree/master/arduino)

Lets see the circuit in action!



COMS (Arduino/Genuino Uno)

Intilization AFE44mx..

intilization is done

calculating spO2... SpO2 : 95A ,Pulse rate :157 bpm

calculating spO2... SpO2 : 95A ,Pulse rate :160 bpm

calculating spO2... SpO2 : 95A ,Pulse rate :157 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :171 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :171 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :157 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :157 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :157 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :168 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :157 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :157 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :178 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :175 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :175 bpm

calculating spO2... SpO2 : 96A ,Pulse rate :175 bpm

calculating spO2... SpO2 : 94A ,Pulse rate :166 bpm

Probe error!!!!

Probe error!!!!

Autoscroll

No line ending

57600 baud

<http://farrellf.com/TelemetryViewer/>

```
//  
// AFE44xx Arduino Firmware  
//
```

```

// Copyright (c) 2016 ProtoCentral
//
// SpO2 computation based on original code from Maxim Integrated
//
// This software is licensed under the MIT License(http://opensource.org/licenses/MIT).
//
// THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT
// NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT
// IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY
// WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
// SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
//
// For information on how to use the HealthyPi, visit https://github.com/ProtoCentral/afe44xx\_Oximeter
//
#include <string.h>
#include <SPI.h>
#include <math.h>
//afe44xx Register definition
#define CONTROL0 0x00
#define LED2STC 0x01
#define LED2ENDC 0x02
#define LED2LEDSTC 0x03
#define LED2LEDENDC 0x04
#define ALED2STC 0x05
#define ALED2ENDC 0x06
#define LED1STC 0x07
#define LED1ENDC 0x08
#define LED1LEDSTC 0x09
#define LED1LEDENDC 0x0a
#define ALED1STC 0x0b
#define ALED1ENDC 0x0c
#define LED2CONVST 0x0d
#define LED2CONVEND 0x0e
#define ALED2CONVST 0x0f
#define ALED2CONVEND 0x10
#define LED1CONVST 0x11
#define LED1CONVEND 0x12
#define ALED1CONVST 0x13
#define ALED1CONVEND 0x14
#define ADCRSTCNT0 0x15
#define ADCRSTENDCT0 0x16
#define ADCRSTCNT1 0x17
#define ADCRSTENDCT1 0x18
#define ADCRSTCNT2 0x19
#define ADCRSTENDCT2 0x1a
#define ADCRSTCNT3 0x1b
#define ADCRSTENDCT3 0x1c
#define PRPCOUNT 0x1d
#define CONTROL1 0x1e
#define SPARE1 0x1f
#define TIAGAIN 0x20
#define TIA_AMB_GAIN 0x21
#define LEDCNTRL 0x22
#define CONTROL2 0x23
#define SPARE2 0x24
#define SPARE3 0x25
#define SPARE4 0x26
#define SPARE4 0x26
#define RESERVED1 0x27
#define RESERVED2 0x28
#define ALARM 0x29
#define LED2VAL 0x2a
#define ALED2VAL 0x2b
#define LED1VAL 0x2c
#define ALED1VAL 0x2d
#define LED2ABSVAL 0x2e
#define LED1ABSVAL 0x2f
#define DIAG 0x30
#define count 60
#define CES_CMDIF_PKT_START_1 0x0A
#define CES_CMDIF_PKT_START_2 0xFA
#define CES_CMDIF_TYPE_DATA 0x02
#define CES_CMDIF_PKT_STOP 0x0B
//int IRheartsignal[count];
//int Redheartsignal[count];
int IRdc[count];
int Reddc[count];
double difIRheartsig_dc;

```

```

double difREDheartsig_dc;
double powdifIR;
double powdifRed;
double IRac;
double Redac;
double SpOpercentage;
double Ratio;
const int SPISTE = 7; // chip select
const int SPIDRDY = 2; // data ready pin
volatile int drdy_trigger = LOW;
void afe44xxInit (void);
void afe44xxWrite (uint8_t address, uint32_t data);
uint32_t afe44xxRead (uint8_t address);
signed long average_BPM( signed long );
volatile char DataPacketHeader[6];
volatile char DataPacket[10];
volatile char DataPacketFooter[2];
int datalen = 0x09;
unsigned long time;
volatile static int SPI_RX_Buff_Count = 0;
volatile char *SPI_RX_Buff_Ptr;
volatile int afe44xx_data_ready = false;
volatile unsigned int pckt =0, buff=0,t=0;
unsigned long ueegtemp = 0, ueegtemp2 = 0;
unsigned long IRtemp,REDtemp;
signed long seegtemp=0, seegtemp2=0;
volatile int i;
uint16_t aun_ir_buffer[100]; //infrared LED sensor data
uint16_t aun_red_buffer[100]; //red LED sensor data
#define FS      25 //sampling frequency
#define BUFFER_SIZE (FS*4)
#define MA4_SIZE 4 // DONOT CHANGE
#define min(x,y) ((x) < (y) ? (x) : (y))
const uint8_t uch_spo2_table[184]={ 95, 95, 95, 96, 96, 96, 97, 97, 97, 97, 97, 98, 98, 98, 98, 98, 99,
    99, 99, 99, 99, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100, 100,
    100, 100, 100, 100, 99, 99, 99, 99, 99, 99, 99, 98, 98, 98, 98, 98, 97, 97,
    97, 97, 96, 96, 96, 96, 96, 95, 95, 95, 94, 94, 94, 93, 93, 93, 92, 92, 92, 91, 91,
    90, 90, 89, 89, 89, 88, 88, 87, 87, 86, 86, 85, 85, 84, 84, 83, 82, 82, 81, 81,
    80, 80, 79, 78, 78, 77, 76, 76, 75, 74, 74, 73, 72, 72, 71, 70, 69, 69, 68, 67,
    66, 66, 65, 64, 63, 62, 62, 61, 60, 59, 58, 57, 56, 56, 55, 54, 53, 52, 51, 50,
    49, 48, 47, 46, 45, 44, 43, 42, 41, 40, 39, 38, 37, 36, 35, 34, 33, 31, 30, 29,
    28, 27, 26, 25, 23, 22, 21, 20, 19, 17, 16, 15, 14, 12, 11, 10, 9, 7, 6, 5,
    3, 2, 1 } ;
static int32_t an_x[ BUFFER_SIZE];
static int32_t an_y[ BUFFER_SIZE];
volatile int8_t n_buffer_count; //data length
int32_t n_spo2; //SPO2 value
int8_t ch_spo2_valid; //indicator to show if the SPO2 calculation is valid
int32_t n_heart_rate; //heart rate value
int8_t ch_hr_valid; //indicator to show if the heart rate calculation is valid
long status_byte=0;
uint8_t LeadStatus=0;
boolean leadoff_detetted = true;
uint8_t spo2_probe_open = false;
int dec=0;
void setup()
{
    Serial.begin(57600);
    Serial.println("Initialising AFE44xx.. ");
    delay(2000) ; // pause for a moment
    SPI.begin();
    // set the directions
    pinMode (SPISTE,OUTPUT);//Slave Select
    pinMode (SPIDRDY,INPUT);// data ready
    attachInterrupt(0, afe44xx_drdy_event, RISING ); // Digital2 is attached to Data ready pin of AFE is i
    // set SPI transmission
    SPI.setClockDivider (SPI_CLOCK_DIV8); // set Speed as 2MHz , 16MHz/ClockDiv
    SPI.setDataMode (SPI_MODE0); //Set SPI mode as 0
    SPI.setBitOrder (MSBFIRST); //MSB first
    // Packet structure
    DataPacketHeader[0] = CES_CMDIF_PKT_START_1; //packet header1 0x0A
    DataPacketHeader[1] = CES_CMDIF_PKT_START_2; //packet header2 0xFA
    DataPacketHeader[2] = datalen; // data length 9
    DataPacketHeader[3] = (uint8_t)(datalen >> 8);
    DataPacketHeader[4] = CES_CMDIF_TYPE_DATA;
    DataPacketFooter[0] = 0x00;
    DataPacketFooter[1] = CES_CMDIF_PKT_STOP;
    afe44xxInit ();

```

```

Serial.println("Initialisation Complete!");
}
////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void loop()
{
    if (drdy_trigger == HIGH)
    {
        detachInterrupt(0);
        afe44xxWrite(CONTROL0,0x000001);
        IRtemp = afe44xxRead(LED1VAL);
        afe44xxWrite(CONTROL0,0x000001);
        REDtemp = afe44xxRead(LED2VAL);
        afe44xx_data_ready = true;
    }
    if(afe44xx_data_ready == true)
    {
        IRtemp = (unsigned long) (IRtemp<<10);
        seegtemp = (signed long) (IRtemp);
        seegtemp = (signed long) (seegtemp>>10);
        REDtemp = (unsigned long) (REDtemp<<10);
        seegtemp2 = (signed long) (REDtemp);
        seegtemp2 = (signed long) (seegtemp2>>10);
        if(dec==20)
        {
            aun_ir_buffer[n_buffer_count]=(uint16_t) (seegtemp>>4);
            aun_red_buffer[n_buffer_count]=(uint16_t) (seegtemp2>>4);
            n_buffer_count++;
            dec=0;
        }
        dec++;
        if(n_buffer_count>99)
        {
            estimate_spo2(aun_ir_buffer, 100, aun_red_buffer, &n_spo2, &ch_spo2_valid,&n_heart_rate, &ch_hr_vali
            if(n_spo2 == -999)
                Serial.println("Probe error!!!!");
            else
            {
                // removed by AL to allow live graphing by telemetry viewer
                //Serial.print("calculating sp02...");
                //Serial.print(" Sp02 : ");
                //Serial.print(n_spo2);
                //Serial.print("% ,");
                //Serial.print("Pulse rate :");
                //Serial.print(n_heart_rate);
                //Serial.println(" bpm");
                Serial.print("SP02 (%), Pulse Rate (BPM)");
                Serial.print(n_spo2);
                Serial.print(",");
                Serial.print(n_heart_rate);
                Serial.print(",");
                Serial.println();
            }
            n_buffer_count=0;
        }
        afe44xx_data_ready = false;
        drdy_trigger = LOW;
        attachInterrupt(0, afe44xx_drdy_event, RISING );
    }
}

////////// Gets Fired on DRDY event////////////////////////////////////////
void afe44xx_drdy_event()
{
    drdy_trigger = HIGH;
}

//////////////////////////////////AFE44xx initialization//////////////////////////////////
void afe44xxInit (void)
{
    // Serial.println("afe44xx Initialization Starts");
    afe44xxWrite(CONTROL0,0x000000);
    afe44xxWrite(CONTROL0,0x000008);
    afe44xxWrite(TIAGAIN,0x000000); // CF = 5pF, RF = 500kR
    afe44xxWrite(TIA_AMB_GAIN,0x000001);
    afe44xxWrite(LEDCTRL,0x001414);
    afe44xxWrite(CONTROL2,0x000000); // LED_RANGE=100mA, LED=50mA
    afe44xxWrite(CONTROL1,0x010707); // Timers ON, average 3 samples
    afe44xxWrite(PRPCOUNT, 0X001F3F);
    afe44xxWrite(LED2STC, 0X001770);
    afe44xxWrite(LED2ENDC,0X001F3E);
}

```

```
afe44xxWrite(LED2LEDSTC,0X001770);
afe44xxWrite(LED2LEDEND,0X001F3F);
afe44xxWrite(ALED2STC, 0X000000);
afe44xxWrite(ALED2END, 0X0007CE);
afe44xxWrite(LED2CONVST,0X000002);
afe44xxWrite(LED2CONVEND, 0X0007CF);
afe44xxWrite(ALED2CONVST, 0X0007D2);
afe44xxWrite(ALED2CONVEND,0X000F9F);
afe44xxWrite(LED1STC, 0X0007D0);
afe44xxWrite(LED1END, 0X000F9E);
afe44xxWrite(LED1LEDSTC, 0X0007D0);
afe44xxWrite(LED1LEDEND, 0X000F9F);
afe44xxWrite(ALED1STC, 0X000FA0);
afe44xxWrite(ALED1END, 0X00176E);
afe44xxWrite(LED1CONVST, 0X000FA2);
afe44xxWrite(LED1CONVEND, 0X00176F);
afe44xxWrite(ALED1CONVST, 0X001772);
afe44xxWrite(ALED1CONVEND, 0X001F3F);
afe44xxWrite(ADCRSTCNT0, 0X000000);
afe44xxWrite(ADCRSTENDCT0,0X000000);
afe44xxWrite(ADCRSTCNT1, 0X0007D0);
afe44xxWrite(ADCRSTENDCT1, 0X0007D0);
afe44xxWrite(ADCRSTCNT2, 0X000FA0);
afe44xxWrite(ADCRSTENDCT2, 0X000FA0);
afe44xxWrite(ADCRSTCNT3, 0X001770);
afe44xxWrite(ADCRSTENDCT3, 0X001770);
delay(1000);
// Serial.println("afe44xx Initialization Done");
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void afe44xxWrite (uint8_t address, uint32_t data)
{
    digitalWrite (SPISTE, LOW); // enable device
    SPI.transfer (address); // send address to device
    SPI.transfer ((data >> 16) & 0xFF); // write top 8 bits
    SPI.transfer ((data >> 8) & 0xFF); // write middle 8 bits
    SPI.transfer (data & 0xFF); // write bottom 8 bits
    digitalWrite (SPISTE, HIGH); // disable device
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
unsigned long afe44xxRead (uint8_t address)
{
    unsigned long data=0;
    digitalWrite (SPISTE, LOW); // enable device
    SPI.transfer (address); // send address to device
    //SPI.transfer (data);
    data |= ((unsigned long)SPI.transfer (0)<<16); // read top 8 bits data
    data |= ((unsigned long)SPI.transfer (0)<<8); // read middle 8 bits data
    data |= SPI.transfer (0); // read bottom 8 bits data
    digitalWrite (SPISTE, HIGH); // disable device
    return data; // return with 24 bits of read data
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void estimate_spo2(uint16_t *pun_ir_buffer, int32_t n_ir_buffer_length, uint16_t *pun_red_buffer, int32_t
{
    uint32_t un_ir_mean,un_only_once ;
    int32_t k, n_i_ratio_count;
    int32_t i, s, m, n_exact_ir_valley_locs_count, n_middle_idx;
    int32_t n_th1, n_npk, n_c_min;
    int32_t an_ir_valley_locs[15] ;
    int32_t n_peak_interval_sum;
    int32_t n_y_ac, n_x_ac;
    int32_t n_spo2_calc;
    int32_t n_y_dc_max, n_x_dc_max;
    int32_t n_y_dc_max_idx, n_x_dc_max_idx;
    int32_t an_ratio[5], n_ratio_average;
    int32_t n_num, n_denom ;
    // calculates DC mean and subtract DC from ir
    un_ir_mean =0;
    for (k=0 ; k<n_ir_buffer_length ; k++ ) un_ir_mean += pun_ir_buffer[k] ;
    un_ir_mean =un_ir_mean/n_ir_buffer_length ;
    // remove DC and invert signal so that we can use peak detector as valley detector
    for (k=0 ; k<n_ir_buffer_length ; k++ )
        an_x[k] = -1*(pun_ir_buffer[k] - un_ir_mean) ;
    // 4 pt Moving Average
    for(k=0; k< BUFFER_SIZE-MA4_SIZE; k++){
        an_x[k]=( an_x[k]+an_x[k+1]+ an_x[k+2]+ an_x[k+3])/(int)4;
    }
}
```

```

// calculate threshold
n_th1=0;
for ( k=0 ; k<BUFFER_SIZE ;k++){
    n_th1 += an_x[k];
}
n_th1= n_th1/ ( BUFFER_SIZE);
if( n_th1<30) n_th1=30; // min allowed
if( n_th1>60) n_th1=60; // max allowed
for ( k=0 ; k<15;k++) an_ir_valley_locs[k]=0;
// since we flipped signal, we use peak detector as valley detector
find_peak( an_ir_valley_locs, &n_npk, an_x, BUFFER_SIZE, n_th1, 4, 15 );//peak_height, peak_distance,
n_peak_interval_sum =0;
if (n_npk>=2){
    for (k=1; k<n_npk; k++) n_peak_interval_sum += (an_ir_valley_locs[k] -an_ir_valley_locs[k -1] ) ;
    n_peak_interval_sum =n_peak_interval_sum/(n_npk-1);
    *pn_heart_rate =(int32_t)( (FS*60)/ n_peak_interval_sum );
    *pch_hr_valid = 1;
}
else {
    *pn_heart_rate = -999; // unable to calculate because # of peaks are too small
    *pch_hr_valid = 0;
}
// load raw value again for SPO2 calculation : RED(=y) and IR(=X)
for (k=0 ; k<n_ir_buffer_length ; k++ ) {
    an_x[k] = pun_ir_buffer[k] ;
    an_y[k] = pun_red_buffer[k] ;
}
// find precise min near an_ir_valley_locs
n_exact_ir_valley_locs_count =n_npk;
//using exact_ir_valley_locs , find ir-red DC and ir-red AC for SPO2 calibration an_ratio
//finding AC/DC maximum of raw
n_ratio_average =0;
n_i_ratio_count = 0;
for(k=0; k< 5; k++) an_ratio[k]=0;
for (k=0; k< n_exact_ir_valley_locs_count; k++){
    if (an_ir_valley_locs[k] > BUFFER_SIZE ){
        *pn_spo2 = -999 ; // do not use SPO2 since valley loc is out of range
        *pch_spo2_valid = 0;
        return;
    }
}
// find max between two valley locations
// and use an_ratio between AC compoent of Ir & Red and DC compoent of Ir & Red for SPO2
for (k=0; k< n_exact_ir_valley_locs_count-1; k++){
    n_y_dc_max= -16777216 ;
    n_x_dc_max= -16777216;
    if (an_ir_valley_locs[k+1]-an_ir_valley_locs[k] >3){
        for (i=an_ir_valley_locs[k]; i< an_ir_valley_locs[k+1]; i++){
            if (an_x[i]> n_x_dc_max) {n_x_dc_max =an_x[i]; n_x_dc_max_idx=i;}
            if (an_y[i]> n_y_dc_max) {n_y_dc_max =an_y[i]; n_y_dc_max_idx=i;}
        }
        n_y_ac= (an_y[an_ir_valley_locs[k+1]] - an_y[an_ir_valley_locs[k] ] )*(n_y_dc_max_idx -an_ir_valley_locs[k]);
        n_y_ac= an_y[an_ir_valley_locs[k+1]] + n_y_ac/ (an_ir_valley_locs[k+1] - an_ir_valley_locs[k] ) ;
        n_y_ac= an_y[n_y_dc_max_idx] - n_y_ac; // subtracting linear DC compoentents from raw
        n_x_ac= (an_x[an_ir_valley_locs[k+1]] - an_x[an_ir_valley_locs[k] ] )*(n_x_dc_max_idx -an_ir_valley_locs[k]);
        n_x_ac= an_x[an_ir_valley_locs[k+1]] + n_x_ac/ (an_ir_valley_locs[k+1] - an_ir_valley_locs[k]);
        n_x_ac= an_x[n_y_dc_max_idx] - n_x_ac; // subtracting linear DC compoentents from raw
        n_num=( n_y_ac *n_x_dc_max)>>7 ; //prepare X100 to preserve floating value
        n_denom= ( n_x_ac *n_y_dc_max)>>7;
        if (n_denom>0 && n_i_ratio_count <5 && n_num != 0)
        {
            an_ratio[n_i_ratio_count]= (n_num*100)/n_denom ; //formular is ( n_y_ac *n_x_dc_max) / ( n_x_ac *n_y_dc_max)
            n_i_ratio_count++;
        }
    }
}
// choose median value since PPG signal may varies from beat to beat
sort_ascend(an_ratio, n_i_ratio_count);
n_middle_idx= n_i_ratio_count/2;
if (n_middle_idx >1)
    n_ratio_average =( an_ratio[n_middle_idx-1] +an_ratio[n_middle_idx])/2; // use median
else
    n_ratio_average = an_ratio[n_middle_idx ];
if( n_ratio_average>2 && n_ratio_average <184){
    n_spo2_calc= uch_spo2_table[n_ratio_average] ;
    *pn_spo2 = n_spo2_calc ;
    *pch_spo2_valid = 1; // float_SPO2 = -45.060*n_ratio_average* n_ratio_average/10000 + 30.354 *n_ratio_a
}

```



```

else{
    *pn_spo2 = -999 ; // do not use SPO2 since signal an_ratio is out of range
    *pch_spo2_valid = 0;
}
}

/////////////////////////////////////////////////////////////////
void find_peak( int32_t *pn_locs, int32_t *n_npk, int32_t *pn_x, int32_t n_size, int32_t n_min_height,
/**
 * \brief    Find peaks
 * \par      Details
 *           Find at most MAX_NUM peaks above MIN_HEIGHT separated by at least MIN_DISTANCE
 *
 * \retval    None
 */
{
    find_peak_above( pn_locs, n_npk, pn_x, n_size, n_min_height );
    remove_close_peaks( pn_locs, n_npk, pn_x, n_min_distance );
    *n_npk = min( *n_npk, n_max_num );
}

/////////////////////////////////////////////////////////////////
void find_peak_above( int32_t *pn_locs, int32_t *n_npk, int32_t *pn_x, int32_t n_size, int32_t n_min_height
/**
 * \brief    Find peaks above n_min_height
 * \par      Details
 *           Find all peaks above MIN_HEIGHT
 *
 * \retval    None
 */
{
    int32_t i = 1, n_width;
    *n_npk = 0;
    while (i < n_size-1){
        if (pn_x[i] > n_min_height && pn_x[i] > pn_x[i-1]){ // find left edge of potential peaks
            n_width = 1;
            while (i+n_width < n_size && pn_x[i] == pn_x[i+n_width]) // find flat peaks
                n_width++;
            if (pn_x[i] > pn_x[i+n_width] && (*n_npk) < 15 ){ // find right edge of peaks
                pn_locs[( *n_npk )++] = i;
                // for flat peaks, peak location is left edge
                i += n_width+1;
            }
            else
                i += n_width;
        }
        else
            i++;
        // Serial.println("beat");
    }
}

/////////////////////////////////////////////////////////////////
void remove_close_peaks(int32_t *pn_locs, int32_t *pn_npk, int32_t *pn_x, int32_t n_min_distance)
/**
 * \brief    Remove peaks
 * \par      Details
 *           Remove peaks separated by less than MIN_DISTANCE
 *
 * \retval    None
 */
{
    int32_t i, j, n_old_npk, n_dist;
    /* Order peaks from large to small */
    sort_indices_descend( pn_x, pn_locs, *pn_npk );
    for ( i = -1; i < *pn_npk; i++){
        n_old_npk = *pn_npk;
        *pn_npk = i+1;
        for ( j = i+1; j < n_old_npk; j++){
            n_dist = pn_locs[j] - ( i == -1 ? -1 : pn_locs[i] ); // lag-zero peak of autocorr is at index -1
            if ( n_dist > n_min_distance || n_dist < -n_min_distance )
                pn_locs[( *pn_npk )++] = pn_locs[j];
        }
    }
    // Resort indices int32_to ascending order
    sort_ascend( pn_locs, *pn_npk );
}

/////////////////////////////////////////////////////////////////
void sort_ascend(int32_t *pn_x, int32_t n_size)
/**
 * \brief    Sort array

```

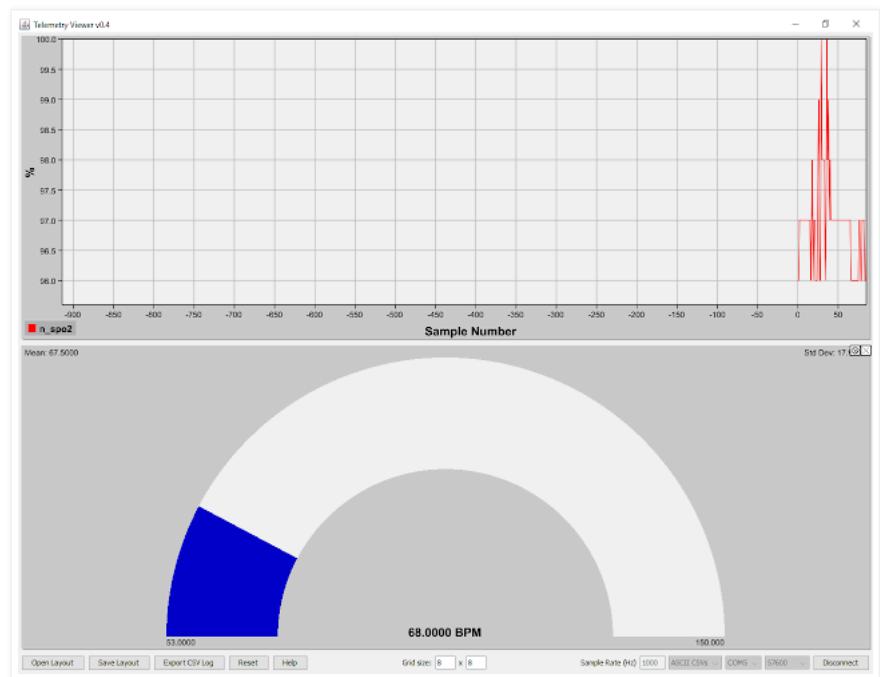
```

* \par    Details
*         Sort array in ascending order (insertion sort algorithm)
*
* \retval  None
*/
{
    int32_t i, j, n_temp;
    for (i = 1; i < n_size; i++) {
        n_temp = pn_x[i];
        for (j = i; j > 0 && n_temp < pn_x[j-1]; j--)
            pn_x[j] = pn_x[j-1];
        pn_x[j] = n_temp;
    }
}

////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////////
void sort_indices_descend( int32_t *pn_x, int32_t *pn_indx, int32_t n_size)
/**
* \brief   Sort indices
* \par     Details
*         Sort indices according to descending order (insertion sort algorithm)
*
* \retval  None
*/
{
    int32_t i, j, n_temp;
    for (i = 1; i < n_size; i++) {
        n_temp = pn_indx[i];
        for (j = i; j > 0 && pn_x[pn_indx[n_temp]] > pn_x[pn_indx[j-1]]; j--)
            pn_indx[j] = pn_indx[j-1];
        pn_indx[j] = n_temp;
    }
}

```

Here is the output from Telemetry Viewer 0.4:



I think this is actually a very well implemented product and if I can use it as part of my project it will save me months of development...my only concern is cost and that it updates slowly and it also uses most of the program memory available on an ATME1 328P Microcontroller. One final comment I might make is that the example code provided is very long and complex. It would be better if the initialisation functions and the measurement calculation functions were moved to a library. It will make reading and writing code much simpler.

Whether I use this arduino shield or not is up for debate...I will probably discuss things further and come to a decision later.

Well that's all for now - Langster!

Posted by [Alexander Lang](#) at 16:24

Labels: [Arduino](#) , [Medical Device](#) , [Pulse Oximeter](#) , [tutorial](#)

6 comments :



**SantaMedical** 26 March 2019 at 09:40

This comment has been removed by a blog administrator.

[Reply](#)



**SantaMedical** 19 June 2019 at 10:05

This comment has been removed by the author.

[Reply](#)



**SantaMedical** 4 July 2019 at 10:08

This comment has been removed by a blog administrator.

[Reply](#)



**santagurin** 23 July 2019 at 10:34

This comment has been removed by a blog administrator.

[Reply](#)



**Unknown** 22 October 2019 at 03:32

This comment has been removed by a blog administrator.

[Reply](#)



**Max99** 19 July 2020 at 15:20

Why don't work my SpO2 value?

[Reply](#)

To leave a comment, click the button below to sign in with Google.

SIGN IN WITH GOOGLE

[Newer Post](#)

[Home](#)

[Older Post](#)

Subscribe to: [Post Comments \( Atom \)](#)