```
#include <Adafruit_PWMServoDriver.h>
#include <Wire.h>
#include <SPI.h>
#include "ecgRespirationAlgo.h"
#include "MeanFilterLib.h"
#include <stdarg.h>

Adafruit_PWMServoDriver servos = Adafruit_PWMServoDriver(0x40);
ecg_respiration_algorithm ECG_RESPIRATION_ALGORITHM;
MeanFilter<long> meanFilter(4);

unsigned int pos0 = 90;  // ancho de pulso para pocicion 0°
unsigned int pos90 = 290;// ancho de pulso para la pocicion 90°
unsigned int pos180 = 490;// ancho de pulso para la pocicion 180°

int32_t Data_canal[2];  //almacena los datos de cada uno de los canales
int16_t buffer_filtro[2], salida_filtro[2]; //almacena datos de filtro
int mean[2]; //almacena salida de filtro media móvil

boolean read_ads_data = true;   //True para leer ADS1292
boolean mensaje       = true;   //True para mostrar mensajes en pantalla
boolean datos_canal   = true;  //True para imprimir datos de salida de ADS1292
int datos_a_imprimir  = 1;      //Numero de canal a imprimir sus datos

/* Pines Esp32
Mosi   - 23   ADS_DIN
Miso   - 19   ADS_DOUT
SCK    - 18   ADS_SCLK
SS     - 5    ADS_CS
*/

//DEFINICIÓN DE PINES ESP32
#define PIN_CS      5
#define PIN_RESET   16
#define PIN_START   17
#define PIN_DRDY    4
#define PIN_LED     3

//DEFINICIÓN COMANDOS SPI
//Comandos del sistema
#define WAKEUP    0x02
#define STANDBY   0x04
#define RESET     0x06
#define START     0x08
#define STOP      0x0A

//Comandos de lectura
#define RDATAC    0x10
#define SDATAC    0x11
#define RDATA     0x12

// comandos de registro
#define READ  0x20
#define WRITE  0x40

//DEFINICIÓN DIRECCIÓN DE REGISTROS
//configuracion del dispositivo
#define ID 0x00

//configuraciones globales
#define CONFIG1 0x01
#define CONFIG2 0x02
#define LOFF  0x03
```

```
//Configuracion de canales
#define CH1SET  0X04
#define CH2SET  0X05

//Estado de lead off
#define RLD_SENS  0x06
#define LOFF_SENS 0x07
#define LOFF_STAT 0x08

//Otros
#define GPIO1 0x0B
#define RESP1 0x09
#define RESP2 0x0A

//Funcion que convierte de hex a char para mostrarlo en la pantalla

String hex_a_char(int hexa) {
  int precision = 2;
  char tmp[16];
  char formato[128];
  sprintf(formato, "0x%%.%dX", precision);
  sprintf(tmp, formato, hexa);
  return (String(tmp));
}

//Función que escribe un byte en un registro dada la direccion

void escribir_reg(int direccion_reg, int valor_hexa) {

  digitalWrite(PIN_CS, LOW);
  delayMicroseconds(5);
  SPI.transfer(0x40 | direccion_reg);
  delayMicroseconds(5);
  SPI.transfer(0x00);   //numero de registros a leer/escribir
  delayMicroseconds(5);
  SPI.transfer(valor_hexa);
  delayMicroseconds(10);
  digitalWrite(PIN_CS, HIGH);

  if (mensaje) {
    Serial.print( "Registro a escribir: " + hex_a_char(direccion_reg) );
    Serial.print( "\t Escritura: " + hex_a_char(valor_hexa) + "\t" );
  }
  int lectura = leer_reg(direccion_reg);
}

//Funcion que lee un registro dada la direccion

int leer_reg(int direccion_reg) {
  int out = 0;
  digitalWrite(PIN_CS, LOW);
  SPI.transfer(0x20 | direccion_reg);
  delayMicroseconds(5);
  SPI.transfer(0x00); // numero de registros a leer/escribir
  delayMicroseconds(5);
  out = SPI.transfer(0x00);
  delayMicroseconds(1);
  digitalWrite(PIN_CS, HIGH);

  if (mensaje) {
    Serial.print( "Registro a leer: " + hex_a_char(direccion_reg) );
    Serial.println( "\tLectura: " + hex_a_char(out) );
  }
  return (out);
}
```

```
//Funcion que envia un comando

void enviar_cmd(uint8_t cmd) {
  digitalWrite(PIN_CS, LOW);
  delayMicroseconds(5);
  SPI.transfer(cmd);
  delayMicroseconds(10);
  digitalWrite(PIN_CS, HIGH);

  if (mensaje) {
    Serial.println( "Comando enviado \t" + hex_a_char(cmd) );
  }
}

//Función que lee datos de salida del ADS1292
int leer_canales() {

  uint8_t tmp;
  int i = 0;
  int j = 0;

  digitalWrite(PIN_CS, LOW);
  delayMicroseconds(1);

  // Data por canal (24 status bits + 24 bits CH1 + 24 bits CH2) = 72 bits
  tmp = SPI.transfer(0x00); // obtiene byte 1 de status bits
  tmp = SPI.transfer(0x00); // obtiene byte 2 de status bits
  tmp = SPI.transfer(0x00); // obtiene byte 3 de status bits
  //delayMicroseconds(1);

  //obtiene data de cada canal
  for (int i = 0; i < 2; i++) {

    Data_canal[i] = 0;

    for (int j = 0; j < 3; j++)
    {
      tmp = SPI.transfer(0x00);
      Data_canal[i] = ((Data_canal[i]) << 8) | tmp;
    }

    // Ignora los 8 los bits menos significativos de cada canal
    buffer_filtro[i] = (int16_t)(Data_canal[i] >> 8);
    // Filtra la señal con filtro pasabajos de fc = 40Hz
    ECG_RESPIRATION_ALGORITHM.ECG_ProcessCurrSample(&buffer_filtro[i], &salida_filtro[i]);
    // Aplica filtro de media móvil
    mean[i] = meanFilter.AddValue(salida_filtro[i]);
  }

  delayMicroseconds(1);
  digitalWrite(PIN_CS, HIGH);
  //return mean[n_canal];
  return mean[0],mean[1];
}

//Funcion que inicializa los pines

void init_pines() {

  pinMode(PIN_CS,    OUTPUT);
  pinMode(PIN_RESET, OUTPUT);
  pinMode(PIN_START, OUTPUT);
  pinMode(PIN_DRDY, INPUT);
  digitalWrite(PIN_CS, HIGH);
```

```
  digitalWrite(PIN_START, LOW);
  delay(1);
}

//Funcion que inicializa el ADS1292

void init_ads1292() {
  int chSet;

  //Pulso de reseteo
  digitalWrite(PIN_RESET, HIGH);
  delay(1000);
  digitalWrite(PIN_RESET, LOW);
  delay(1000);
  digitalWrite(PIN_RESET, HIGH);
  delay(100);

  //Resetear conversiones
  digitalWrite(PIN_START, HIGH);
  delay(1000);
  digitalWrite(PIN_START, LOW);
  delay(1000);

  //Resetear communicación
  digitalWrite(PIN_CS, LOW);
  delay(1000);
  digitalWrite(PIN_CS, HIGH);

  delay(10);
  enviar_cmd(START);
  delay(10);
  enviar_cmd(STOP);

  //Esperar a que el chip inicie
  delay(500);

  //Envio de comando SDATAC
  enviar_cmd(SDATAC);
  delay(10);

  //Lectura de registro de ID
  chSet = leer_reg(READ | ID);
  if (mensaje) {
    Serial.println("ID de ADS1292:\t" + String(chSet));
  }

  //INICIALIZACIÓN DE ESCRITURA DE REGISTROS

  //config 1: Establecer frecuencia de muestreo en 125 SPS
  escribir_reg(CONFIG1, 0x00);

  //config 2: Para una entrada con electrodos (0xA0).
  //Para activar la señal de prueba (0xA3)
  escribir_reg(CONFIG2, 0xA0);
  //escribir_reg(CONFIG2, 0xA3);

  //Loff: Lead off detection desactivada
  escribir_reg(LOFF, 0x10);

  //CH1 y CH2: Activa cada canal y configura con G=12 (0x60).
  //Al usar la señal de prueba usar (0x05)
  escribir_reg(CH1SET, 0x60);
  escribir_reg(CH2SET, 0x60);
  //escribir_reg(CH1SET, 0x05);
  //escribir_reg(CH2SET, 0x05);
```

```
  //RLD_Sens: Activa buffer RLD. Conecta cada canal al RLD.
  escribir_reg(RLD_SENS, 0x2F);

  //Loff_sens: Desactiva todas las configuraciones de Loff sense
  escribir_reg(LOFF_SENS, 0x00);

  //Loff_stat: fMOD = fCLK / 4
  escribir_reg(LOFF_STAT, 0x00);

  //Resp1: Para ADS1292 escribir 0x02
  escribir_reg(RESP1, 0x02);

  //Resp2: Calibración desactivada.
  //Señal RLDREF(AVDD – AVSS) / 2 generada internamente
  escribir_reg(RESP2, 0x07);

  //GPIO
  escribir_reg(GPIO1, 0x00);

  //Start
  digitalWrite(PIN_START, HIGH);
  delay(150);

  //Inicia lectura
  enviar_cmd(RDATAC);
}

//Inicializacion de todas las funciones

void setup() {
  delay(3000);

  //INICIALIZA UART
  Serial.begin(115200);
  Serial.flush();
  delayMicroseconds(100);

  //INICIALIZA SPI
  SPI.begin();
  SPI.beginTransaction (SPISettings (1000000, MSBFIRST, SPI_MODE1));  // 1 MHz clock, MSB first, mode
1
  delay(1);

  if(mensaje){
    Serial.println("SPI y UART encendidos");
  }

  //INICIALIZA PCA9685
  servos.begin();
  servos.setPWMFreq(50); //Frecuecia PWM de 50Hz o T=20ms

  //INICIALIZA PINES ESP32 Y ADS
  init_pines();
  init_ads1292();
}

void loop() {

  if (read_ads_data) {

    if (digitalRead(PIN_DRDY) == LOW) {


      int dato_nuevo_1,dato_nuevo_2 = leer_canales();
```

```
      /*
      int dato_nuevo_1 = leer_canales();
      */
      int nuevo_pwm_1 = pos90 + 6.9*dato_nuevo_1;
      int nuevo_pwm_2 = pos90 + 13.1*dato_nuevo_2;
      servos.setPWM(0,0,nuevo_pwm_1);
      servos.setPWM(1,0,nuevo_pwm_2);
      delayMicroseconds(1);
      if (datos_canal) {
        //Serial.println(int32_t(Data_canal[datos_a_imprimir]));
        Serial.println( dato_nuevo_1 );
      }
    }
  }
}
```