**// RingBuffer.hpp**

#pragma once

#include <Arduino.h>

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class RingBuffer<M, N>

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Circular Buffer FIFO data structure, with M arrays of N int elements

\*/

template **<**int M**,** int N**>** class RingBuffer

**{**

public**:**

// Constructor

RingBuffer**()**

**{**

pStart **=** buffer**;**

pHead **=** pStart**;**

pTail **=** pStart**;**

pEnd **=** **&**buffer**[**M**\***N**-**1**];**

size **=** 0**;**

**}**

// Number of elements

int Size**()** **{return** size**;}**

// Operations

void Push**(**int t**[])**

**{**

**if** **(**size **<** M**)**

**{**

size**++;**

**for** **(**int i**=**0**;** i**<**N**;** i**++)** **\***pTail**++** **=** **\***t**++;**

**if** **(**pTail**>**pEnd**)** pTail **=** pStart**;**

**}**

**}**

int Pop**(**int t**[])** // Before test if Size()>0

**{**

size**--;**

**for** **(**int i**=**0**;** i**<**N**;** i**++)** **\***t**++** **=** **\***pHead**++;**

**if** **(**pHead**>**pEnd**)** pHead **=** pStart**;**

**}**

private**:**

int buffer**[**M**\***N**];**

int **\***pHead**,** **\***pTail**,** **\***pStart**,** **\***pEnd**;**

int size**;**

**};**

**// PowerMonitor.h**

#pragma once

#include <Arduino.h>

#include "RingBuffer.hpp"

#define M 100 // Buffer size <256

//#define TESTING

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class PowerMonitor

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Rms voltage, rms current and real power estimations for voltage and current channels

Inspired from :

Robin Emley's Cal\_bothDisplays\_2.ino sketch ( https://www.mk2pvrouter.co.uk )

EmonLib library ( https://github.com/openenergymonitor/EmonLib )

EmonLibCM library ( https://github.com/openenergymonitor/EmonLibCM )

Constructor parameters :

\_period = sampling period for the channels (in us and 105<.<8192)

\_nCycl = number of cycles to construct the estimations (>=1)

\_VPin, \_CPin = Arduino analog pins for the voltage and the current (Ax)

\_VCal, \_CCal = Calculus coefficients to convert the digitized values

\_pha = Phase shift coefficient of the voltage (int)

NB : \_pha=256 => no shift, \_pha<256 => right shift, \_pha>256 => left shift

\*/

enum Polarities **{**POSITIVE**,** NEGATIVE**};** // Alternance polarities

extern "C" void TIMER1\_OVF\_vect**();**

class PowerMonitor

**{**

public**:**

// Constructor

PowerMonitor**(**int \_period**,** int \_nCycl**,** byte \_VPin**,** byte \_CPin**,** double \_VCal**,** double \_CCal**,** int \_pha**);**

void Init**();** // Initialisation (to be called in the setup)

void Process**();** // Process calculus outside the ISR (to be called in the loop()

double RmsVoltage**();** // Vrms (V)

double RmsCurrent**();** // Irms (A)

double RealPower**();** // Pr (W)

#ifdef TESTING

byte MaxSize**();** // Maximum size of the circular buffer, on entering Process()

int NbSamples**();** // Number of samples in estimations

long MaxProcTime**();** // Max time to treat a sample (us)

#endif

private**:**

unsigned int period**;** // Sampling period for the channels

int nCycl**,** iCycl**;** // Number of cycles and index of the cycle

byte VPin**,** CPin**;** // Arduino pins for the voltage and the current

double VCal**,** CCal**;** // Calculus coefficients to convert the digitized vales

int pha**;** // Phase shift coefficients array of the voltage

int iChan**;** // Index of the channel in the digitalization

RingBuffer**<**M**,** 2**>** rbDig**;** // Circular buffer for the digitized values

int tDigISR**[**2**];** // Digitized values array pushed in the ISR

int tDig**[**2**];** // Digitized values array poped in the Process method

long V**,** C**;** // Voltage and current values (<<8 and centered)

Polarities pol**;** // Polarity of the alternance

long offV**,** offC**;** // Offsets for centering

long sumV**,** sumC**;** // Sums of the voltage and current values

long sumVV**,** sumCC**;** // Sums of the squared voltage and current values (>>10)

long savSumVV**,** savSumCC**;** // Saved sum of the squared voltage and current values (>>10)

long sumVC**;** // Sum of the VC products(>>10)

long savSumVC**;** // Saved sum of the VC products(>>10)

long preV**;** // Previous voltage value (>>2)

long phaV**;** // Phase shifted voltage value (>>2)

int nSum**;** // Number (saved) of values in the sums VV, CC and VC

int nSavSum**;** // Number (saved) of values in the sums VV, CC and VC

void FuncISR**();** // ISR function

#ifdef TESTING

byte maxSize**;** // Maximum size of the circular buffer, on entering Process()

long maxProcTime**;** // Maximum of processing time

#endif

friend void TIMER1\_OVF\_vect**();** // To use the private FuncISR() in the ISR

**};**

**// PowerMonitor.cpp**

#include "PowerMonitor.h"

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ISR

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PowerMonitor**\*** pPowerMonitor**;** // As static

ISR**(**TIMER1\_OVF\_vect**)**

**{**

pPowerMonitor**->**FuncISR**();**

**}**

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Class PowerMonitor

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

PowerMonitor**::**PowerMonitor**(**int \_period**,** int \_nCycl**,** byte \_VPin**,** byte \_CPin**,** double \_VCal**,** double \_CCal**,** int \_pha**)**

**{**

pPowerMonitor **=** **this;**

period **=** \_period**;**

nCycl **=** \_nCycl**;**

VPin **=** \_VPin**;**

CPin **=** \_CPin**;**

VCal **=** \_VCal**;**

CCal **=** \_CCal**;**

pha **=** \_pha**;**

**}**

void PowerMonitor**::**Init**()**

**{**

// Set ADC parameters (prescaler, voltage reference), enable and start it with the volatge source

ADCSRA **=** **(**1**<<**ADPS0**)|(**1**<<**ADPS1**)|(**1**<<**ADPS2**);** // Prescaler = 128 (104us/conversion and max accuracy)

ADCSRA **|=** **(**1**<<**ADEN**);** // Enable ADC

ADMUX **=** 0x40**;** // Voltage reference = AVcc with external capacitor at AREF pin

ADMUX **|=** VPin**;** // Set ADPS to the voltage pin

ADCSRA **|=** **(**1**<<**ADSC**);** // Start the conversion

delay**(**1**);** // Wait to perform this conversion

// Set Timer1 interrupts

noInterrupts**();** // Disable all interrupts

TCCR1A **=** 0**;** // Normal port operation (no PWM outputs)

TCCR1B **=** **(**1**<<**WGM13**);** // Phase and frequency correct PWM mode with TOP in ICR1

TCCR1B **|=** **(**1**<<**CS10**);** // No prescaling

ICR1 **=** 8**\***period**;** // TOP

TIMSK1 **=** **(**1**<<**TOIE1**);** // Enable timer OVF interrupt (TOV1 flag set on at BOTTOM)

interrupts**();** // Enable all interrupts

// Init values for members

iCycl **=** 0**;**

iChan **=** 0**;**

nSum **=** 0**;**

nSavSum **=** 0**;**

offV **=** 510L**\***256**;**

sumV **=** 0**;**

sumVV **=** 0**;**

offC **=** 510L**\***256**;**

sumC **=** 0**;**

sumCC **=** 0**;**

sumVC **=** 0**;**

preV **=** 220L**\***64**;**

#ifdef TESTING

maxSize **=** 0**;**

maxProcTime **=** 0**;**

#endif

**}**

void PowerMonitor**::**Process**()**

**{**

#ifdef TESTING

maxSize **=** max**(**maxSize**,** rbDig**.**Size**());**

#endif

**while** **(**rbDig**.**Size**()** **>** 0**)**

**{**

#ifdef TESTING

long startTime **=** micros**();**

#endif

// Pop the digitized values

cli**();**

rbDig**.**Pop**(**tDig**);**

sei**();**

// Voltage and current values (x256 and centering)

V **=** **(((**long**)**tDig**[**0**])<<**8**)-** offV**;**

C **=** **(((**long**)**tDig**[**1**])<<**8**)-** offC**;**

// New half alternance test

**if** **((**V **>=** 0**)** **&&** **(**pol **==** NEGATIVE**))** // Start POSITIVE (new cycle)

**{**

// Polarity new value

pol **=** POSITIVE**;**

iCycl**++;**

**if** **(**iCycl **==** nCycl**)**

**{**

// Save the sums of squared values and VC products, then reinit them

savSumVV **=** sumVV**;**

savSumCC **=** sumCC**;**

savSumVC **=** sumVC**;**

nSavSum **=** nSum**;**

sumVV **=** 0**;**

sumCC **=** 0**;**

sumVC **=** 0**;**

nSum **=** 0**;**

// Reinit iCycl

iCycl **=** 0**;**

**}**

**}**

**else** **if** **((**V **<** 0**)** **&&** **(**pol **==** POSITIVE**))** // Start NEGATIVE

**{**

// Polarity new value

pol **=** NEGATIVE**;**

// Offsets update using low pass filters

offV **+=** **(**sumV**>>**12**);**

offC **+=** **(**sumC**>>**12**);**

sumV **=** 0**;**

sumC **=** 0**;**

**}**

// Sums incrementation

sumV **+=** V**;**

V **=** **(**V**>>**2**);**

sumVV **+=** **((**V**\***V**)>>**6**);**

sumC **+=** C**;**

C **=** **(**C**>>**2**);**

sumCC **+=** **((**C**\***C**)>>**6**);**

phaV **=** preV **+** **(((**V **-** preV**)\***pha**)>>**8**);**

sumVC **+=** **((**phaV**\***C**)>>**6**);**

preV **=** V**;**

nSum**++;**

#ifdef TESTING

maxProcTime **=** max**(**maxProcTime**,** micros**()-**startTime**);**

#endif

**}**

**}**

double PowerMonitor**::**RmsVoltage**()**

**{**

**return** VCal**\***sqrt**(((**double**)**savSumVV**)/(**64.**\***nSavSum**));**

**}**

double PowerMonitor**::**RmsCurrent**()**

**{**

**return** CCal**\***sqrt**(((**double**)**savSumCC**)/(**64.**\***nSavSum**));**

**}**

double PowerMonitor**::**RealPower**()**

**{**

**return** VCal**\***CCal**\*(((**double**)**savSumVC**)/(**64.**\***nSavSum**));**

**}**

#ifdef TESTING

byte PowerMonitor**::**MaxSize**()**

**{**

byte savMaxSize **=** maxSize**;**

maxSize **=** 0**;**

**return** savMaxSize**;**

**}**

int PowerMonitor**::**NbSamples**()**

**{**

**return** nSavSum**;**

**}**

long PowerMonitor**::**MaxProcTime**()**

**{**

long savMaxProcTime **=** maxProcTime**;**

maxProcTime **=** 0**;**

**return** savMaxProcTime**;**

**}**

#endif

void PowerMonitor**::**FuncISR**()**

**{**

// Digitized value

tDigISR**[**iChan**++]** **=** ADC**;**

// Push the digitized values in the buffer

**if(**iChan **==** 2**)**

**{**

iChan **=** 0**;**

ADMUX **=** 0x40 **|** VPin**;**

rbDig**.**Push**(**tDigISR**);**

**}**

// Start the next conversion

ADMUX **=** 0x40 **|** CPin**;**

ADCSRA **|=** **(**1**<<**ADSC**);**

**}**