# main.cpp

#include "mbed.h"

#include "freq\_counter.hpp"

#include "capacitance.hpp"

#include "resistance.hpp"

#include "voltmeter.hpp"

BufferedSerial serial\_port(USBTX, USBRX);

int main() {

while (true) {

char c;

if (serial\_port.read(&c, 1)) {

if (c == 'c') {

capacitance();

} else if (c == 'f') {

freq\_main();

} else if (c == 'v') {

get\_voltage();

} else if (c == 'r') {

printf("about to get the resistance\n");

resistance();

} else {

printf("Invalid Character");

}

}

ThisThread::sleep\_for(100ms);

}

}

# freq\_counter.hpp

float sample();

void freq\_main();

# freq\_counter.cpp

#include "mbed.h"

#include <chrono>

float sample(){

DigitalIn freqIn(D4);

Timer t;

bool running = true;

bool measuring = false;

long long timeus;

float freq;

while(running){

//Measure the digital state of the freqIn pin, and if a sequential

//zero then one is recorded, timer is started

int pinVal = freqIn.read();

if (pinVal == 0 && measuring == false){

pinVal = freqIn.read();

if (pinVal == 1 && measuring == false){

t.start();

measuring = true;

}

}

//If a low is detected on the digital in, and the measuring flag

//is high, the timer is stopped, the elapsed time is microseconds

//cast to timeus, and the measuring and running flags brought low

if (pinVal == 0 && measuring == true){

t.stop();

timeus = std::chrono::duration\_cast<chrono::microseconds>(t.elapsed\_time()).count();

measuring = false;

running = false;

}

}

//Return the time measured by the function

return timeus;

}

void freq\_main(){

float sum = 0;

float i = 0;

//Sampling triggered to occur when less than 10 previous samples have //occurred during same instance of the function call

while (i < 10){

float time = sample();

sum += time;

i += 1;

}

//If the number of samples taken is greater than zero, ie that a signal //has been

//measured, the average time of the pulses is calculated, the frequency //thus found,

//and printed to the serial out. The averageing variables are then set to //zero

if (i > 1){

float average = sum/i;

float freq = 1/((average/1e6)\*2);

if (freq > 1000){

printf("The frequency of the signal is %.2f kHz \n", freq/1000);

}

else{

printf("The frequency of the signal is %.2f Hz \n", freq);

}

sum = 0;

i = 0;

}

}

//printf tends to add at least 2500μs of execution time

# voltmeter.hpp

float get\_voltage();

# voltmeter.cpp

// Voltage Level Shift Voltmeter Design

// This voltmeter design requires the use of one AnalogIn pin and the 5V microcontroller supply.

#include "mbed.h"

void get\_voltage(){

AnalogIn adc(A3); // Setting up an analog pin to read in voltage values

adc.set\_reference\_voltage(3.333333);

//Setting the reference voltage to 3.33333 V, which is the maximum //possible node voltage (at 5V input)

//This maximises the resolution of the pin (all bits are used to represent //a possible value)

float Vread = adc.read\_voltage();

//Returns the voltage as a float, so long as it's below the reference //value

float Vout = 3 \* (Vread - 1.666666);

//Equation for linear relationship between voltage read at node and //voltage input --> see Falstad file for more details

float Vout\_adjusted = Vout - 0.03;

//Adjusting the voltage value for a small measurement error, approx. 30 //mV

printf("Voltage: %.2f", Vout\_adjusted);

}

# capacitance.hpp

// capacitance.hpp

#ifndef CAPACITANCE\_HPP

#define CAPACITANCE\_HPP

void capacitance();

#endif // CAPACITANCE\_HPP

# capacitance.cpp

#include "mbed.h"

#include <chrono>

#include <cstdio>

using namespace std::chrono;

AnalogIn capVoltage(PA\_0);

DigitalOut v(D7);

struct capValues {

double voltage;

double time;

};

capValues runAndFindCapacitance(double duration){

ThisThread::sleep\_for(duration / 10);

Timer time;

v = 1;

time.start();

ThisThread::sleep\_for(duration);

double voltage = capVoltage.read();

time.stop();

v = 0;

auto elapsedTime = duration\_cast<microseconds>(time.elapsed\_time()).count();

double timeSec = elapsedTime \* 1e-6;

return {

voltage,

timeSec

};

}

void capacitance() {

v = 0;

const float RESISTANCE = 4.938e6;

// while (capVoltage.read() > 0.0001){

// printf("capVoltage %f\n", capVoltage.read());

// ;

// }

double dur = 10e-3;

capValues cV = {0, 0};

while (cV.voltage < 0.1 && cV.time < 5) {

printf("dur : %f\n", dur);

cV = runAndFindCapacitance(dur);

dur \*= 10;

}

printf("time : %f\n", cV.time);

printf("voltage: %f\n", cV.voltage);

double capacitanceValue = -cV.time / (RESISTANCE \* log(1 - cV.voltage));

if (capacitanceValue >= 1e-6) {

printf("Capacitance: %f microFarads\n", capacitanceValue \* 1e6);

} else if (capacitanceValue >= 1e-9) {

printf("Capacitance: %f nanoFarads\n", capacitanceValue \* 1e9);

} else {

printf("Capacitance: %f picoFarads\n", capacitanceValue \* 1e12);

}

return;

}

# resistance.hpp

// capacitance.hpp

#ifndef RESTSITANCE\_HPP

#define RESTSITANCE\_HPP

void resistance();

#endif // RESTSITANCE\_HPP

# resistance.cpp

#include "mbed.h"

#include <chrono>

using namespace std::chrono;

AnalogIn resVoltage(PA\_1);

DigitalOut vR(D6);

struct resValues {

double voltage;

double time;

};

resValues runAndFindResistance(double duration){

ThisThread::sleep\_for(duration / 10);

Timer time;

vR = 1;

time.start();

ThisThread::sleep\_for(duration);

double voltage = resVoltage.read();

time.stop();

vR = 0;

auto elapsedTime = duration\_cast<microseconds>(time.elapsed\_time()).count();

double timeSec = elapsedTime \* 1e-6;

return {

voltage,

timeSec

};

}

void resistance() {

// printf("hello there\n");

while (resVoltage.read() > 0.001){

;

};

const float CAPACITANCE = 0.9365e-3;

vR = 0;

double dur = 10e-3;

resValues rV = {

0, 0

};

while (rV.voltage < 0.1 && rV.time < 5){

rV = runAndFindResistance(dur);

// printf("time %f\n", rV.time);

// printf("voltage %f\n", rV.voltage);

// printf("duration : %f\n", dur);

dur\*=10;

};

// printf("time %f\n", rV.time);

// printf("voltage %f\n", rV.voltage);

double resistanceValue = -rV.time / (CAPACITANCE \* log(1 - rV.voltage));

if (resistanceValue >= 1e3) {

printf("Resistance: %f kOhm\n", resistanceValue / 1e3);

} else {

printf("Resistance: %f Ohm\n", resistanceValue);

}

return;

}