Politecnico di Torino

Testing and Certification

Third Laboratory Session

Implementation of a Capacitance meter using an Arduino UNO Board

Authors:
D'Addato Mauro
Fini Simone
Marmo Sebastiano
Triarico Antonio

Professor:
Carullo Alessio

20-27 November 2017

POLITECNICO
DI TORINO

1 Introduction

In this laboratory session the objective was to create a capacitance meter which was able to measure two values of capacitance (specifically, $1\mu F$ and $150nF$).
This report will describe both the hardware and software implementation of the acquisition board and the theoretical approach used to project it. In addition, it will present different ways used to calculate the value of the capacitance:

- two-sample technique, implemented via software;
- multiple reading technique;
- linear regression.

1.1 Instrumentation

The instruments used to implement the aformentioned project are:

- Arduino UNO Board;
- Diode 1N4007;
- $1\mu F$ capacitor;
- $150nF$ capacitor;
- $2.2k\Omega$ resistor;
- HP 34401A Multimeter (to measure the correct value of the components).

In this case it is important to highlight that the accuracy considered to compute the error which affected the reading of the real R value is $\pm(0.010 + 0.001)$, expressed as $\pm(\%\ of\ reading + \%\ of\ range)$.

Function	Range [3]	Test Current or Burden Voltage	24 Hour [2] 23°C ± 1°C	90 Day 23°C ± 5°C	1 Year 23°C ± 5°C	Temperature Coefficient /°C 0°C – 18°C 28°C – 55°C
Resistance [4]	100.0000 Ω	1 mA	0.0030 + 0.0030	0.008 + 0.004	0.010 + 0.004	0.0006 + 0.0005
	1.000000 kΩ	1 mA	0.0020 + 0.0005	0.008 + 0.001	0.010 + 0.001	0.0006 + 0.0001
	10.00000 kΩ	100 µA	0.0020 + 0.0005	0.008 + 0.001	0.010 + 0.001	0.0006 + 0.0001
	100.0000 kΩ	10 µA	0.0020 + 0.0005	0.008 + 0.001	0.010 + 0.001	0.0006 + 0.0001
	1.000000 MΩ	5 µA	0.002 + 0.001	0.008 + 0.001	0.010 + 0.001	0.0010 + 0.0002
	10.00000 MΩ	500 nA	0.015 + 0.001	0.020 + 0.001	0.040 + 0.001	0.0030 + 0.0004
	100.0000 MΩ	500 nA || 10 MΩ	0.300 + 0.010	0.800 + 0.010	0.800 + 0.010	0.1500 + 0.0002

Fig. 1: Specifications provided by the DMM datasheet

2 Theoretical Design

The capacitance was measured in an indirect way, based on the discharge of the electric component. In particular, the dropping voltage across the capacitor is stored inside the board memory (in specific time intervals, which will be explained later) and the computation of C is provided by the division between τ and the known resistor R.

$$\tau = RC \Rightarrow C = \frac{\tau}{R} \tag{1}$$

Of course, the time constant τ must have been estimated; in order to do this, it was used the following formula:

$$\tau = \frac{T}{\ln \frac{V_1}{V_2}}, \tag{2}$$

where

- T is the acquisition time
- V_1 is the value of the first sample
- V_2 is the value of the last sample.

The problem was that the acquisition period had not to be nor too short (to avoid sampling the voltage too closely) neither too long (to avoid too low voltage measurements). So, to minimize the measurement uncertainty, the correct duration of the acquisition interval had to be estimated; basing on the calculations provided by the professor, it came up that the optimal interval was:

$$T = \tau. \tag{3}$$

Having computed T, the second issue was about setting the correct value of the prescaler so as to reduce the sampling frequency.

$$f_s = \frac{f_{clock}}{13 \cdot prescalervalue}, \tag{4}$$

where the factor 13 was included because a complete conversion of the input signal takes 13 clock cycles.// Indeed, the entire acquisition process was based on converting the analog input without using the Arduino native function *analogRead()*; in such a way the control over the sampling frequency would have been better and the result more precise. A way to develop this kind of strategy was to create a variable which would have counted up to a value (calculated by hand) indicating that the time interval equals to τ was passed, and therefore the acquisition must have been interrupted.
Due to the fact that using the 2.2 $k\Omega$ resistor the values of τ for the $150nF$ and $1\mu F$ capacitors are, respectively, $0.33\mu s$ and $2.2\mu s$, the first attempt was to compute the value of the prescaler having decided in advance the number of samples which corresponded to a time interval of τ. But in this way obtaining an integer value was tricky; so it was decided to set the prescaler value first and then calculate the number of samples, according to the formula

$$n = f_s \cdot \tau. \tag{5}$$

Specifically, with a prescaler value of 16, the computed n for the $1\mu F$ and 150 nF capacitors are, respectively:

$$n_{1\mu F} = f_s \cdot \tau_{1\mu F} = \frac{16MHz}{13 \cdot 16} \cdot RC_{1\mu F} \simeq 169 \tag{6}$$

$$n_{150nF} = f_s \cdot \tau_{150nF} = \frac{16MHz}{13 \cdot 16} \cdot RC_{150nF} \simeq 25 \tag{7}$$

3 Firmware Implementation

3.1 Setup() Section

```
#define C150
//#define C1

double R;             //expressed in kOhm
double C;            //expressed in nF
double C_computed;
double tau_computed;
int stopsample;
double readings[256];       //array to store samples
int index;
double timepassed;

void setup(){
  Serial.begin(9600);
  pinMode(A0, INPUT);
  pinMode(8, OUTPUT);
  digitalWrite(8,HIGH);     //sets the digitalOutput to HIGH

  ADMUX  = 0b01000000;
  ADCSRA = 0b10111100;
  ADCSRB = 0x00;

  R=2.14832;
  #ifdef C1
    C=1000;
    stopsample=169;
  #else
    C=150;
    stopsample=25;
  #endif

  index=0;
  Serial.println("Setup completed");
  delay(2000);
}
```

Fig. 2: Variables declaration and Setup section

First of all, a *#define* construct is used, in order to be able to change both the value of C and the one of *stopsample* just by commenting a single line of code.
Secondly, an set of variables is declared:

- R and C: the values of C and R. The former is just printed during the execution to be compared to the value of the computed capacitance, while the latter represents the real value of the resistor, measured by using the DMM; its value (assigned in the *Setup()* is $2.14832 \pm 0.03148\ k\Omega$;
- *C_computed*: the experimental value of the capacitance;
- *tau_computed*: the experimental value of τ;
- *stopsample*: the number of samples; this was used to stop the acquisition and compute the results;
- *readings[256]*: the array in which the samples are stored;
- *index*: the sample index;
- *timepassed*: time passed from the beginning of the acquisition.

Turning to the *Setup()* section, the USB connection to the PC is initialized at a baud rate of 9600 by using the Arduino class *Serial*; then, the analog pin *A0* is set as an INPUT (to acquire the analog voltage) and the digital one *D8* is set as an OUTPUT and HIGH. Subsequently, three Arduino registers are set.

3.1.1 ADMUX: ADC Multiplexer Selection Register

Bit	7	6	5	4	3	2	1	0
	REFS1	REFS0	ADLAR		MUX3	MUX2	MUX1	MUX0
Access	R/W	R/W	R/W		R/W	R/W	R/W	R/W
Reset	0	0	0		0	0	0	0

Fig. 3: Description of the content of ADMUX register taken by the ATmega328/P datasheet

- Bits 7 and 6 select the voltage reference of the ADC. Since in this case it was taken externally, the aformentoned bits are 01;
- Bit 5 affects the presentation of the ADC conversion result in the ADC Data Register. If 0, the result is right adjusted (adopted solution), if 1 left adjusted;
- Bit 4 has no meaning;
- Bits from 3 to 0 select which analog inputs are connected to the ADC. Since the used analog pin was A0, they were set to 0000.

3.1.2 ADCSRA: ADC Control and Status Register A

Bit	7	6	5	4	3	2	1	0
	ADEN	ADSC	ADATE	ADIF	ADIE	ADPS2	ADPS1	ADPS0
Access	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W
Reset	0	0	0	0	0	0	0	0

Fig. 4: Description of the content of ADCSRA register taken by the ATmega328/P datasheet

- Bit 7 enables the ADC; writing 1 means turn the ADC on, writing 0 turn it off;
- Bit 6 starts the first conversion;
- Bit 5 enables Auto Triggering mode;
- Bit 4 is set when an ADC conversion is completed and the Data Registers are updated;
- Bit 3 enables the ADC interrupt;
- Bits 2 to 0 determine the correct prescaler value. Since it was decided to store a value of 16, these 3 bits were set to 100, which refers to the value of the 2 power, as explained below.

$$100_{binary} = 4 \Rightarrow 2^4 = 16 \tag{8}$$

3.1.3 ADCSRB: ADC Control and Status Register B

Bit	7	6	5	4	3	2	1	0
		ACME				ADTS2	ADTS1	ADTS0
Access		R/W				R/W	R/W	R/W
Reset		0				0	0	0

Fig. 5: Description of the content of ADCSRB register taken by the ATmega328/P datasheet

- Bit 7,5,4, and 3 do not have meaning;
- Bit 6 has no meaning in this experiment;
- Bits 2 to 0 select the trigger source. In particular, in order to select the Free Running mode, the value 000 was written.

```
ADMUX  = 0b01000000;
ADCSRA = 0b10111100;
ADCSRB = 0x00;
```

Finally, the correct values of R, C and the number of samples are stored (according to which of the two capacitor was about to be measured), the index was set to 0 and a string indicating that the *Setup()* section is finished is printed on terminal.

3.2 Loop() Section

```
void loop(){

  if(index==0){
    bitSet(ADCSRA,ADEN);       //switches on the ADC
    bitSet(ADCSRA,ADSC);       //starts ADC measurements
    digitalWrite(8,LOW);
  }

  if(index==stopsample){
    timepassed=1.0/(16.0e6/16.0/13.0)*1e3*(stopsample-1);
    Serial.println("Acquisition finished...computing the result");
    Serial.println("READING 0");
    Serial.println(readings[0]);
    Serial.println("LAST READING");
    Serial.println(readings[stopsample-1]);
    Serial.println("TIMEPASSED");
    Serial.println(timepassed,5);

    tau_computed=timepassed/log(readings[0]/readings[stopsample-1]);
    C_computed=tau_computed/R;
    Serial.println(C_computed,5);

    for(int i=0; i<stopsample; i++)
      Serial.println(readings[i]);

    delay(2000);
    index=0;
  }
}
```

Fig. 6: Loop code

At the beginning of the loop, the variable *index* is checked and, if it is equal to 0 (i.e. the program has just start running), the bit number 7 (ADEN) of the ADCSRA is set to 1; this is not important for the first execution of the loop, since the ADC is switched on during the setup, but for the following ones: indeed, as it will be explained in subsection 3.3, at the end of the acquisition the Analog-to-Digital converter is switched off; so, if multiple

acquisition processes need to be executed, this method allows to do it without resetting the entire system.
Then the bit number 6 (ADSC) is set to 1, starting the conversion procedure, and the digital voltage is swapped to the low logical value to make the capacitor discharge process begin.
At the end of the acquisition process (i.e. when the index is equal to the prefixed number of samples), the passed time (expressed in *mus*) is computed, using the formula

$$passed\,time = \frac{number\,of\,samples}{\frac{16MHz}{13 \cdot 16}}, \tag{9}$$

and the value of the experimental τ is calculated, using the formula (2). It is important to notice that, due to the way the code is written, the first sample is always stored in the first element of *readings* and the last in the last one (*readings[stopsample-1]*).
At the end, the experimental value of C is computed and printed, and the index is reset to 0.
Important: the *for* cycle is used just to be able to transfer every sample value from the memory of the board to an Excel sheet to implement the linear regression, as well as the several *println* instructions.

3.3 Interrupt Service Routine

```
ISR(ADC_vect){

  readings[index]=ADCW;
  index++;
  if(index==stopsample){
    bitClear(ADCSRA,ADEN);      //switches off the ADC
    digitalWrite(8,HIGH);
  }
```

Fig. 7: Code inside the ISR

As soon as the conversion of the analog input is done and the content of the Data Registers ADCH and ADCL is updated, the Interrupt Service Routine is executed, and firstly the value inside the ADCW register (which is the 16bit register divided into ADCH and ADCL) is stored into the array *readings* and the index is incremented.
Only if the index is equal to the prefixed number of samples to be taken, the ADEN bit of the register ADCSRA is set to 0 by means of the function *bitClear* and the digital output is reset to the high logical value, in order to let the capacitor charge again.

4 Results of the Two-Sample Technique

At the end of the acquisition process, data was elaborated and the subsequent results were showed:

Important: data below refers to the third acquisition, since it was decided not to take into account the first two of them because of a possible system stabilization.

Passed Time (μs)	τ Computed (μs)	C Computed (μF)
0.312	0.326	0.15153

Table 1: Results regards the 150nF Capacitor

The measure was of course affected by an uncertainty, that was computed by means of the following formula:

$$\epsilon_C = \epsilon_\tau + \epsilon_R, \ where \tag{10}$$

$$\epsilon_\tau = \frac{\Delta T}{T} + \frac{1}{\ln(\frac{V_1}{V_2})}\left(\frac{\Delta V_1}{V_1} + \frac{\Delta V_2}{V_2}\right), \tag{11}$$

$$\frac{\Delta T}{T} = \frac{13 \cdot 16 \cdot stopsample}{f_{clock}^2} \cdot \Delta f_{clock} \tag{12}$$

and ϵ_R is the relative uncertainty or the resistor measured by the DMM.
On account of the fact that:

- the absolute uncertainty of the ADC is $\pm 2LSB$;
- the number of bits is 10;
- Arduino Uno uses a ceramic resonator (as it is showed by Fig.7) whose frequency tolerance is $\pm 0.5\%$ (as described by Fig.8 taken by the component datasheet);

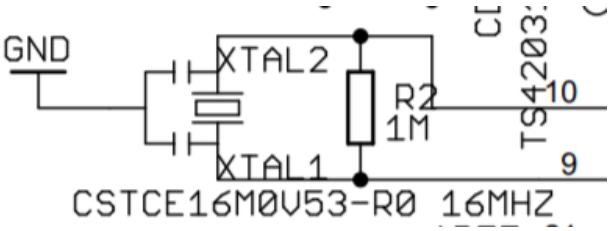

Fig. 8: Schematics of the oscillator circuit implemented on the UNO board

Item / Part Number	Frequency Range (MHz)	Initial Tolerance of Oscillation Frequency	Temperature Stability of Oscillation Frequency (-20 to +80°C)	Oscillating Frequency Aging
CSTCE□V	14.00—20.00	±0.5%	±0.3%	±0.3%

Fig. 9: Specifications of the CSTCE16M0V53-R0 ceramic resonator

$$\epsilon_\tau = 4.6235 \cdot 10^{-5} \tag{13}$$

$$\epsilon_R = 1.4653 * \cdot 0^{-2} \tag{14}$$

$$\epsilon_C = 1.4699 \cdot 10^{-2}. \tag{15}$$

Finally, the complete measure of the $150nF$ capacitor is:

$$C_{150nF} = 0.15153 \pm 0.00222\ \mu F \tag{16}$$

Passed Time (μs)	τ Computed (μs)	C Computed (μF)
2.184	2.144	0.99842

Table 2: Results regards the $1\mu F$ Capacitor

In the same way as before,

$$\epsilon_\tau = 5.2431 \cdot 10^{-5} \tag{17}$$

$$\epsilon_R = 1.4653 \cdot 10^{-2} \tag{18}$$

$$\epsilon_C = 1.4705 \cdot 10^{-2}. \tag{19}$$

At the end, the computed value of the $1\mu F$ capacitor is:

$$C_{1\mu F} = 0.99842 \pm 0.01468\ \mu F \tag{20}$$

5 Multiple Reading Technique

Having calculated the values of the capacitors using the two-sample technique, a second method, the multiple reading technique, was implemented.
Specifically, the samples of 10 successive readings were taken and saved, and at the end of the acquisition the value of C was computed by means of the average of the aformentioned data.
Important: since reporting every measurement of the 10 processes was reputed not very relevant (in total would have been 250 and 1690 samples, indeed), the tables just refer to the results of the calculations.

	READING #1	READING #2	READING #3	READING #4	READING #5	READING #6	READING #7	READING #8	READING #9	READING #10
TAU COMPUTED (us)	0.32661	0.32661	0.32553	0.32553	0.32553	0.32553	0.32553	0.32661	0.32620	0.32769
C COMPUTED (uF)	0.15203	0.15203	0.15153	0.15153	0.15153	0.15153	0.15153	0.15203	0.15184	0.15253
ABSOLUTE UNCERTAINTY ON C (uF)	0.00223	0.00223	0.00223	0.00223	0.00223	0.00223	0.00223	0.00223	0.00223	0.00224
C AVERAGE (uF)	0.15181									
ABSOLUTE UNCERTAINTY AVERAGE (uF)	0.00223									

Fig. 10: Results of the multiple reading technique applied to the 150nF capacitor

	READING #1	READING #2	READING #3	READING #4	READING #5	READING #6	READING #7	READING #8	READING #9	READING #10
TAU COMPUTED (us)	2.13159	2.14735	2.14493	2.13825	2.12921	2.14065	2.14735	2.13159	2.13825	2.13398
C COMPUTED (uF)	0.99221	0.99955	0.99842	0.99531	0.99110	0.99643	0.99955	0.99221	0.99531	0.99333
ABSOLUTE UNCERTAINTY ON C (uF)	0.01459	0.01470	0.01468	0.01464	0.01457	0.01465	0.01470	0.01459	0.01464	0.01461
C AVERAGE (uF)	0.99534									
ABSOLUTE UNCERTAINTY AVERAGE (uF)	0.01464									

Fig. 11: Results of the multiple reading technique applied to the 1uF capacitor

To sum up, it can be clearly seen that:

- the measure of the$150nF$is equal to $0.15181 \pm 0.00223nF$, which is very close compared to the one obtained by the two-sample technique;
- the measure of the $1\mu F$ is equal to $0.99534 \pm 0.01464\mu F$, which is very similar to the one obtained by the two-sample technique again.

6 Linear Regression

The last task to accomplish was to estimate the value of the two capacitors using an off-line processing: the linear regression

$$log(ADC_{value}) = -\frac{1}{\tau}t + log(ADC_{maxvalue}). \tag{21}$$

In order to obtain the linear regression parameters of the red straight line, Microsoft Excel was used and the following formula was implemented:

$$[m, q] = (U^T \cdot U)^{-1} \cdot U^T \cdot y, where \tag{22}$$

- m is the angular coefficient equals to $\frac{1}{\tau}$;
- q is the offset f the line;
- y is a vector containing the codes of the ADC expressed as logarithms;
- U is a matrix defined as $\begin{bmatrix} \Delta t & 1 \\ 2\Delta t & 1 \\ ... & ... \\ N\Delta t & 1 \end{bmatrix}$.

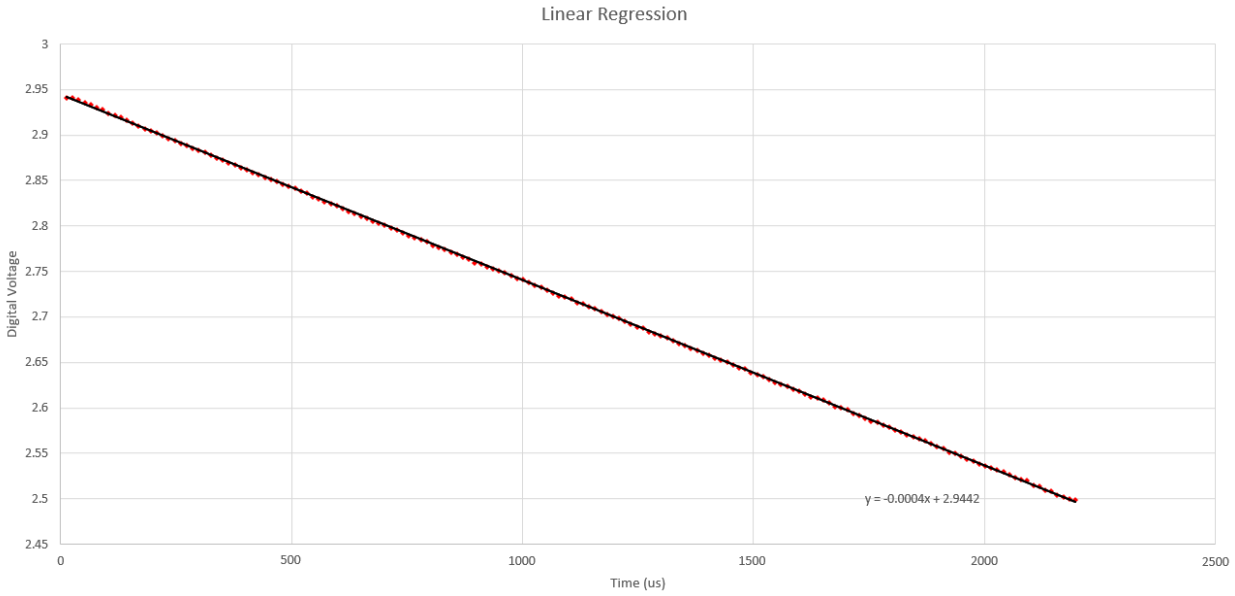

Fig. 12: Linear regression for the 1uF capacitor

τ Computed (μs)	C Computed (μF)	δC (μF)
2.500	1.164	0.013

Table 3: Results regards the $1\mu F$ Capacitor

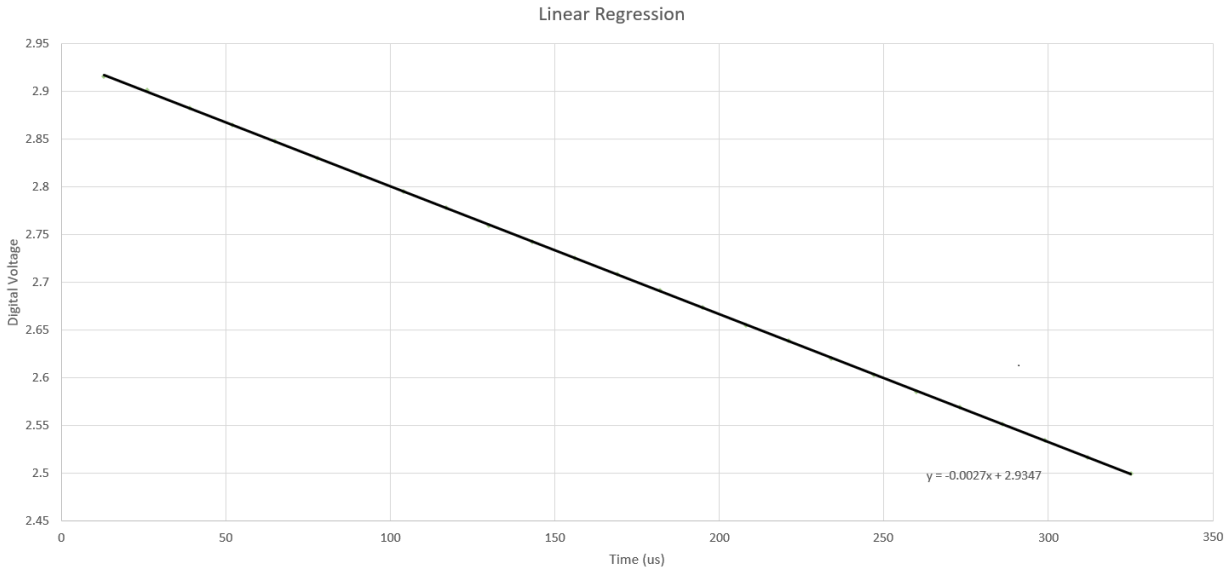

Fig. 13: Linear regression for the 150nF capacitor

τ Computed (μs)	C Computed (μF)	δC (μF)
0.370	0.172	0.019

Table 4: Results regards the $150nF$ Capacitor