MEDIS: A Methodology for the Formation of Highly Qualified Engineers at Masters Level in the Design and Development of Advanced Industrial Informatics Systems

WP2.1 Chapter 5.3: Process interface analog input output

Co-funded by the Tempus Programme of the European Union

544490-TEMPUS-1-2013-1-ES-TEMPUS-JPCR

Authors: Perles, A., Capella, J.V., Albaladejo, J.

MEDIS: A Methodology for the Formation of Highly Qualified Engineers at Masters Level in the Design and Development of Advanced Industrial Informatics Systems

WP2.1 Chapter 5.3: Process interface analog input output

Contract Number: 544490-TEMPUS-1-2013-1-ES-TEMPUS-JPCR

Starting date: 01/12/2013 **Ending date:** 30/11/2016

Deliverable Number: 2.1

Title of the Deliverable: AIISM teaching resources - Industrial Computers

Task/WP related to the Deliverable: Development of the AIISM teaching resources - Industrial Computers

Type (Internal or Restricted or Public): Public

Author(s): Perles, A., Capella, J.V., Albaladejo, J.

Contractual Date of Delivery to the CEC: 30/09/2014

Actual Date of Delivery to the CEC: 30/09/2014

Project Co-ordinator

Company name :	Universitat Politecnica de Valencia (UPV)
Name of representative :	Houcine Hassan
Address :	Camino de Vera, s/n. 46022-Valencia (Spain)
Phone number :	+34 96 387 7578
Fax number :	+34 963877579
E-mail :	husein@upv.es
Project WEB site address :	https://www.medis-tempus.eu

Context

WP 2	Design of the AIISM-PBL methodology
WPLeader	Universitat Politècnica deValència (UPV)
Task 2.1	Development of the AIISM teaching resources - Industrial Computers
Task Leader	UPV
Dependencies	MDU, TUSofia, USTUTT, UP

Author(s)	Perles, A., Capella, J.V., Albaladejo, J.
Reviewer(s)	Domínguez, C., Martínez, J.M., Hassan, H.

History

Version	Date	Author	Comments
0.1	01/03/2014	UPV Team	Initial draft
0.2	04/03/2015	UPV Team	Seminar, lab and miniproject

Table of Contents

1 Executive summary

WP 2.1 details the learning materials of the Advanced Industrial Informatics Specialization Modules (AIISM) related to the Industrial Computers Module.

The contents of this package follow the guidelines presented in the UPV's documentation of the WP 1 (Industrial Computers Module)

- The PBL methodology was presented in WP 1.1
- The list of the module's chapters and the temporal scheduling in WP 1.2
- The required human and material resources in WP 1.3
- The evaluation in WP 1.4

During the development of this WP a separate document has been created for each of the chapters of the Industrial Computers Module (list of chapters in WP1.2).

In each of these documents, section 2 introduces the chapter; sections 3, 4, 5 and 6 details the Lecture, Laboratory, Seminar and Mini-project of the chapter; section 7 lists the bibliography and the references.

2 Introduction

This chapter is dedicated to the analog input and output mechanism. The analog input/output mechanism is able to read and generated signals continuous in a given range. Taking into account the discrete nature of digital computers, the read/generated values will be an approximation.

3 Lecture

3.1 Objectives

- To know basic aspects of data acquisition applied to analog signals.
- To learn to program basic process interfaces for analog input/output.

3.2 Analog output

To generate analog signals (voltage, current, timing, etc.) it is required an electronic device called digital-to-analog converter (DAC). This device generates analog signals departing from discrete integer values.

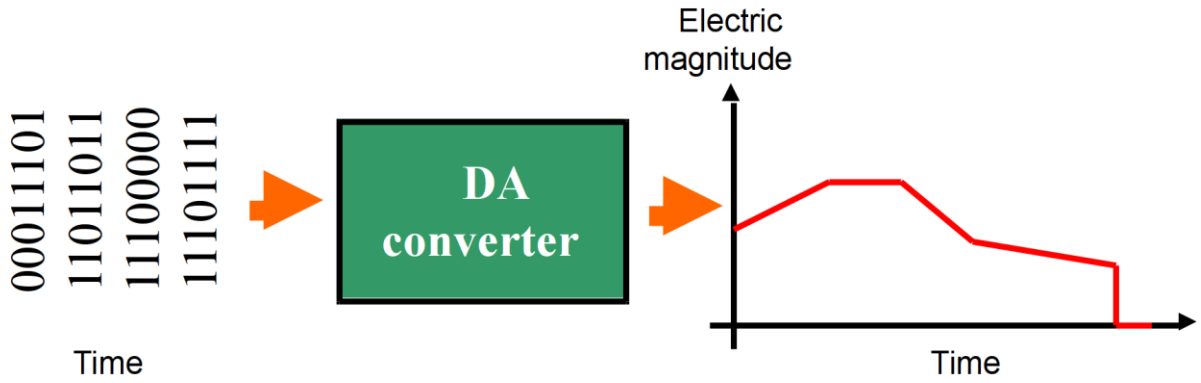


Figure 14: Representation of the digital to analog process conversion.

There are different approaches for the implementation of these devices. One configuration is having an output linearly proportional to a digital value introduced for the digital value that is introduced into the DA block. Figure 2 represents a typical diagram of this configuration.

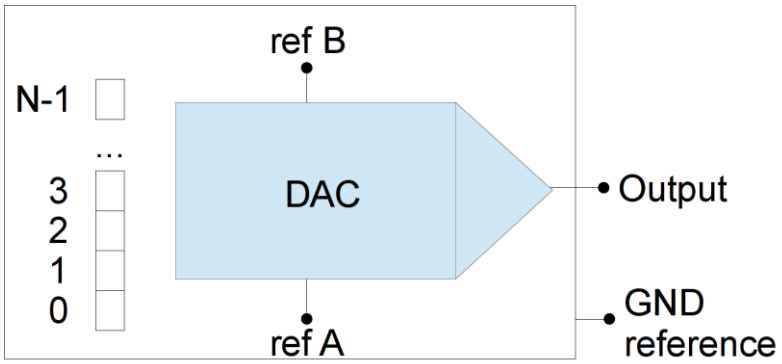


Figure 14: Blocks of a typical DAC converter.

In this configuration, we have a input register of N bits, a two reference signals (voltage in general). The output signal depends on these signals, so the fundamental parameters are:

- The number of bits of the input registers of the DA.
- The voltage or current references to generate the signal (ref A and ref B).

Assuming that both, input and output, are voltages, the formula for calculating the output value is:

$$V_{DAC} = V_{refA} + input\_value * \frac{V_{refB} - V_{refA}}{2^N}$$

Taking into consideration that the input values are integer, it is impossible to get a real continuous signal. Higher number of bits provides better results. This effect is called "resolution", that can be calculated as,

$$resolution = \frac{V_{refB} - V_{refA}}{2^N}$$

The effect of the DAC behavior is shown in figure 3. See the "stairs" effect.

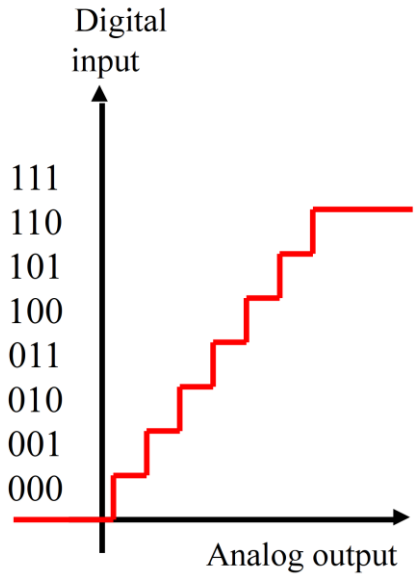


Figure 14: Stair effect of output of the DAC.

For illustrating the DAC use, assume a specific DAC with the characteristics showed below,

- num. of bits: 8
- V_{refA} = 0 V
- V_{refB} = 5 V

Then, the following C function could handle this converter based on an input voltage specified as parameter of the function,

```
#define X_REGISTER 0xFE0FEA // a given controller register

process_WriteAOVoltage(double volts){

    double value;
```

```
    value=volts*(256.0/5.0);
    out(X_REGISTER,value);
}
```

Imagine now that we are interested on handling the pump of the tanks of liquid model. The pump's flow can be controlled by means of a voltage signal.

Continuing with the philosophy of the subject, we are interested in giving a series of services to the rest of the application to handle the pump in the most abstract way possible. Then it is proposed that the pump is managed by using a real number with range from 0 to 100 that represents the % of power or flow that the pump has to provide. This abstraction must be independent of how the pump is internally managed.

An example of code that uses the pump could be:

```
#include <process.h>
int main(void) {

   process_Init();                        //prepare process module
   process_WriteActuatorPump(34.5);       // pump at 34.5% power
   …
}
```

The first step is to calculate the equation that implements the conversion from the abstract value (power in %) to the voltage required by the pump according to the signal table.

For example, assume the following lineal relationship,

- 1 Volt -> 0 % of power
- 3 Volt -> 100 % of power

Then, an appropriate function for implementing this could be,

```
Void process_WriteActuatorPump(double power){

    double value, volts;

    volts = power*((3.0-1.0)/100.0)+1.0;
    value = volts*(256.0/5.0);

    out(X_REGISTER,value);
}
```

3.3 Analog input

Because of the nature of digital systems, the analog signals have to be discretized to be introduced into the computer. This mechanism consists in representing the continuous information by means of a set of discrete values. This is similar to the DAC behavior, but using the inverse path. Figure XX shows, the idea; given a physical magnitude, this is converted to an electrical magnitude (volts, current, modulated signal, …), this electrical

magnitude is attached to an analog-to-digital converter (ADC) the provides a digital output value proportional to the value of the input.

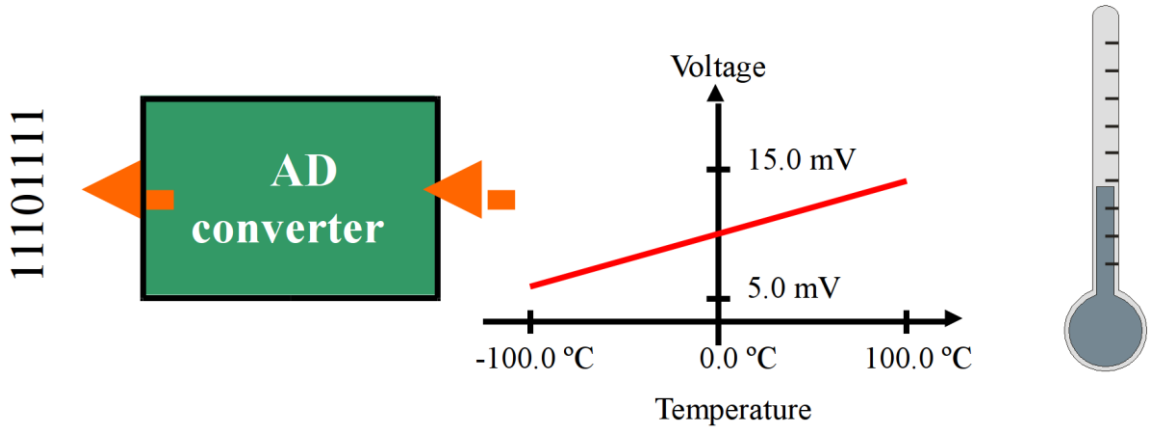


Figure 14: Representation of the analog to digital process conversion.

The ADC can use different approximations for the conversion. A typical configuration is to provide a digital output number of fixed number of bits and linearly proportional to the input voltage, two reference tensions.

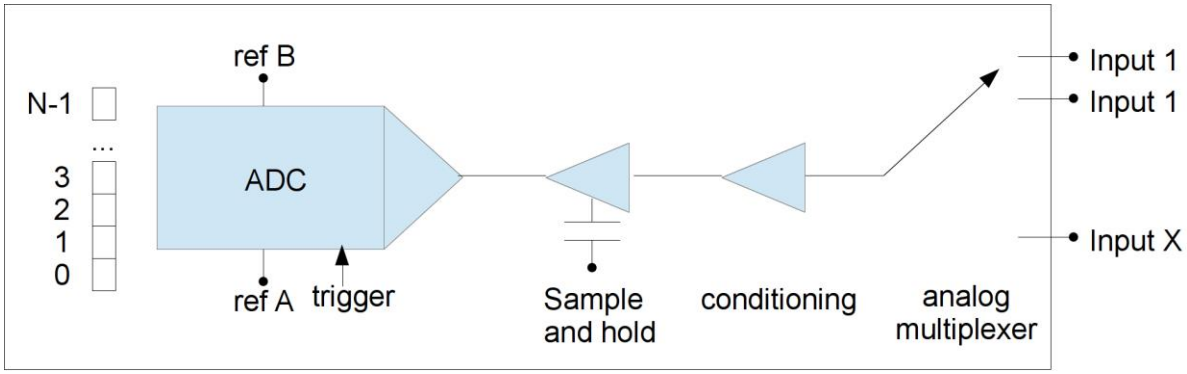


Figure 14: Blocks of a typical ADC conversion system.

Figure XX shows the blocks of typical configurations available on commercial DAQ cards. At the input, there is an analog multiplexer to provide multiple inputs, then the signal can be conditioned using a programmable amplifier, the next stage is a "sample-and-hold" circuits, that retains the signal level during the conversion, finally, the signal arrives to the ADC converter, that needs a trigger signal to start the conversion and, after not negligible time, offers the output in the form of a digital number.

Similar to DAC formula conversion, a typical ADC with linear behavior follows this formula,

$$V_{ADC} = V_{refA} + output\_value * \frac{V_{refB} - V_{refA}}{2^N}$$

Taking into consideration that the output values are integer, there is, almost, a discretization error related to the resolution of the ADC. Similar to DAC, it can be calculated as,

$$resolution = \frac{V_{refB} - V_{refA}}{2^N}$$

An an example of use, assume that we use a popular LM335 integrated temperature sensor. This cdevice behaves like a zener diode whose voltage is directly proportional to the temperature in the relationship 10mV/°K.

We desire a function that provides temperature based on a given ADC converter, an this sensor. An example of code that uses this function could be,

```
#include <process.h>
int main(void) {

   double temperature_cesius;

   process_Init();                        //prepare process module
   temperature_celsius = process_ReadSensorTemperature();
}
```

A generic implementation of the reading function could be similar to this one, assuming 12 bits resolution and reference voltages of 0 V. and 10 V.

```
double process_ReadSensorTemperature (void) {

   int daq_read;
   double volts;
   double degrees;

   ADC_trigger(); // generic trigger
   daq_data = ADC_Read(); //generic read of ADC data register

   volts = 0.0 + daq_data*((10.0 - 0.0)/2048.0);
   degrees = (volts * 100.0) - 273.15;  // LM335 transference function
   return(degrees);
}
```

4 Seminar: Analog input with NI USB-6008 DAQ card

4.1 Objectives

- To understand the physical connection of analog input devices to a real DAQ card
- To develop software to deal with analog input signals on real hardware.

4.2 NI USB-6008 DAQ analog input

The diagram of figure XX shows the working diagram of the analog input section of the DAQ card.

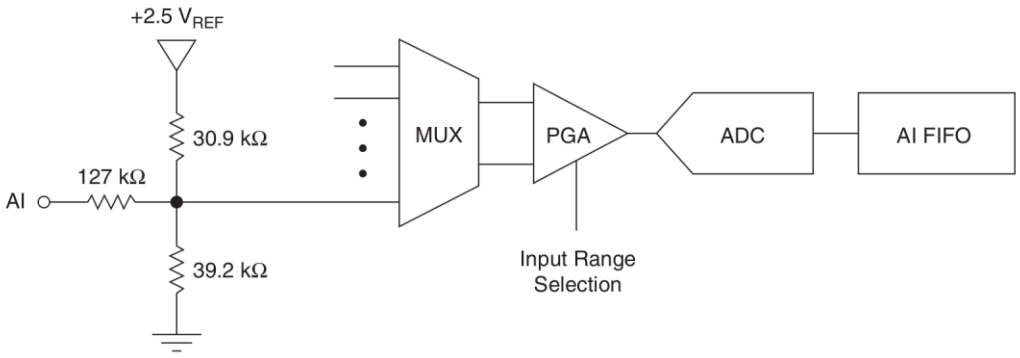


Figure 14: Blocks of the NI USB-6008 analog input section.

ACTIVITY: In groups, access the National Instruments web page and check the details about this card downloading and reading the document named "NI USB-6008/6009 User Guide And Specifications" for analyzing the information related to analog input.

4.3 NI-DAQmx functions for analog input

There are a set of functions of the NI-DAQmx library that deals with analog input.

Before reading analog inputs, you need to configure an "analog input" channel. This can be accomplished using the following function:

- DAQmxCreateAIVoltageChan()
- DAQmxCreateAIStrainGageChan()

Then, we can use specific digital output functions, for example:

- DAQmxReadAnalogScalarF64() -> one reading
- DAQmxReadAnalogF64() -> multiple readings

ACTIVITY: In groups, analyze the DAQmx manual for knowing the parameters of the functions.

5 Lab: analog input

5.1 Objective

The goal is to implement the code for reading a temperature sensor using an analog input of the NI USB-6008 DAQ card.

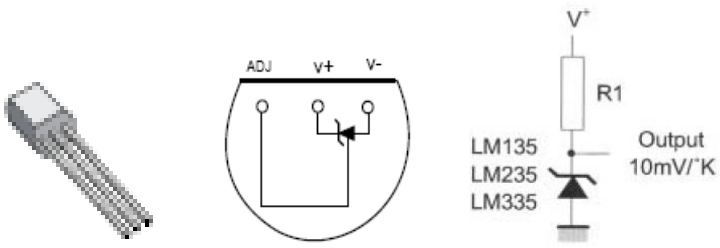
5.2 Equipment

- Personal Computer with Microsoft Windows 7 or superior operating system.
- Open source version of Qt framework for Microsoft Windows.
- Data acquisition card National Instruments USB-6008.
- 1 2K2 resistor.
- LM335 temperature sensor.
- Copper wires, or male to male Dupont wires
- Screwdriver.
- Breadboard.

5.3 Departing point

The aim is to develop an application that reads a sensor and provides the reading the adequate units. For this purpose a low cost popular LM335 temperature sensor has been elected.

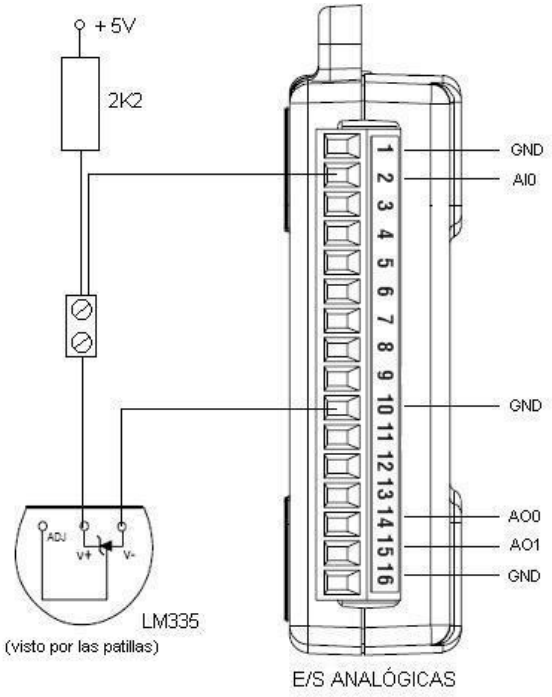
The LM335 integrated circuit is a temperature sensor that behaves like a zener diode whose voltage is directly proportional to the temperature in the relationship 10mV/° K (mV / degree Kelvin).



Given a voltage, the temperature in Celsius degrees can be calculated as,

$$^{o}C = \frac{volts * 1000}{10} - 273.15$$

We can connect the LM335 sensor using the assembly diagram of the components and board as follows,



The + 5V supply can be obtained from the terminal block no. 31 of the digital I/O side of the DAQ card.

From the point of view of the software, you can use the following C header file to provide prototypes and definitions for an initialization function for the DAQ hardware and a digital output functions.

```
/**
    @file    process.h
    @brief   Process interface prototypes

*/

#ifndef PROCESS_H
#define PROCESS_H

void process_Init(void);
double process_ReadSensorTemperature(void);

#endif
// End of file ------------------------------------------
```

For the implementation, the following code is appropriate for handling the DAQ card and the actuator. The name of the card definition must be adapted to the particular configuration in your computer system

```
/**
    @file    process.cpp
    @brief   Process interface analog input
*/

// Here module parameters for easing the configuration of the module

#define DAQ_NAME                    "InfiDAQ"   // DAQ user assigned name

#define TEMPERATURE_SENSOR_CHANNEL   "ai0"      // analog channel for the
temperature sensor

// The following lines area trick for letting to include the header
NIDAQmx.h para NIDAQmx versión 9.1.7
// in the Qt LGPL version. These are compiler dependent.
// See http://www.disca.upv.es/aperles/qt/qt_nidaqmx/qt_nidaqmx.html
#include <QtGlobal>
#ifdef Q_OS_WIN32
  typedef unsigned long long uInt64;
  #define __int64 long long int
#endif
#include <NIDAQmx.h>

#include "process.h"



// Global variables ----------------------------------------------------
TaskHandle temperature_input_task;

/*****************************************************************************/
/**
    @brief  Inits the DAQ system
```

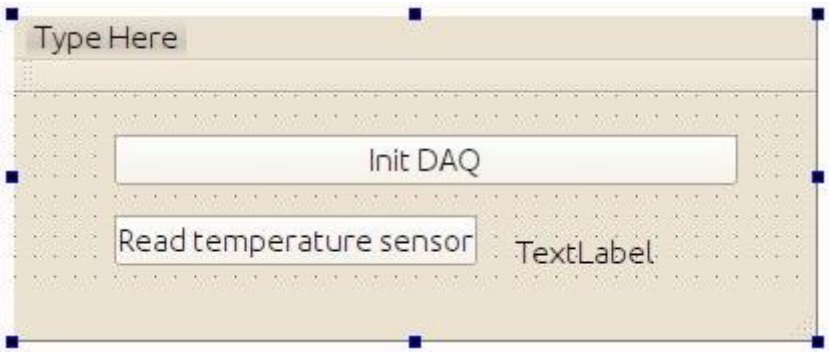
```
    @param      none
    @returns    none

    This function must be called before using any adquisition functions
*/
void process_Init(void)
{
   DAQmxCreateTask("tarea temperatura",&temperature_input_task);
   DAQmxCreateAIVoltageChan (temperature_input_task,
      DAQ_NAME/**/"/"/**/TEMPERATURE_SENSOR_CHANNEL, "",
      DAQmx_Val_RSE, 0.0, 10.0, DAQmx_Val_Volts, NULL);
}



/*********************************************************************/
/**
    @brief  Reads the temperature sensor
    @param     none
    @returnval temperature de Celsius degrees
*/
double process_ReadSensorTemperature(void) {

   float64 volts;
   double degrees;

   DAQmxStartTask(temperature_input_task);
   DAQmxReadAnalogScalarF64 (temperature_input_task, 1.0, &volts, NULL);
   DAQmxStopTask(temperature_input_task);

   degrees = (volts * 100.0) - 273.15;  // LM335
   return(degrees);
}
```

Now, create a Qt application and adjust the .pro file in order to provide information about the NIDAQmx libraries. Adjust to your own system installation

```
# National Instruments NIDAQmx configuration

win32 {

    INCLUDEPATH += "C:/Archivos de programa/National Instruments/NI-
DAQ/DAQmx ANSI C Dev/include"


    LIBS += -L"C:/Archivos de programa/National Instruments/NI-DAQ/DAQmx
ANSI C Dev/lib/msvc"

    LIBS += -lNIDAQmx

}
```

Add this module to a fresh Qt application and proceed to build the code to verify the absence of errors.

In the MainWindow, add buttons and label as follow,



Associate “slots” to the buttons and provide code similar to the following one, (see bold text)

```
#include "mainwindow.h"
#include "ui_mainwindow.h"

#include "process.h"


MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);
}
```

```
MainWindow::~MainWindow()
{
    delete ui;
}


void MainWindow::on_btInit_clicked()
{
    process_Init();
}
void MainWindow::on_btReadTemperatureSensor_clicked()
{
    double degrees;

    degrees = process_ReadSensorTemperature();
    ui->lbTemperature->setNum(degrees);
}
```

Run it to check its operation.

If it does not work properly, try to enhance de process module checking the return value of the DAQ functions. I.e.,

```
int32 daq_error;
 // apply to each DAQ function call
 daq_error=DAQmxCreateTask(…
 if(daq_error != 0) {
     printf("Problems!!!");
     exit(1);
 }
```

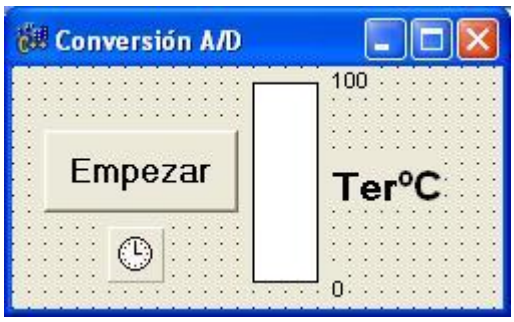
5.4 Activity

Include a QTimer object that automatically reads the state of sensor every 500 milliseconds.

Add two buttons to activate / deactivate the Timer (at the start of the app, the timer should be disabled).

Add a rectangle (for example using a coloured label) that changes its size according the temperature readins.

This could be the aspect of the app,



6 Mini-project: Implementation of the process interface module: Analog input/output

Develop the part of the process module related to analog input/ output.

Validate the developed functions creating the appropriate test functions.

7 Extra activities

ACTIVITY

Find in Internet the typical number of bits of the DA converter of the microphone input sound card in a PC computer/smartphone/tablet.

ACTIVITY

Find in Internet the typical number of bits of the AD converter of the sound outpu card in a PC computer/smartphone/tablet.

8 References

NI USB-6008/6009 User Guide And Specifications. Available at: http://digital.ni.com/manuals.nsf/websearch/CE26701AA052E1F0862579AD0053BE19