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 5.1: Process interface basics and digital input

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 5.1: Process interface basics and digital input

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)	Martínez, J.M., Hassan, H.

History

Version	Date	Author	Comments
0.1	01/03/2014	UPV Team	Initial draft
1.0	19/09/2014	UPV Team	Final version

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 basics of process interface. In order to know the state of an industrial process and to achieve effective control actions, is necessary to establish a communication mechanism between the physical and the virtual world inside the controlling computer.

In this context, we name "process interface" to the set of circuits and programs in charge of establishing that communication.

3 Lecture

3.1 Objectives

- To understand the concept of process interface.
- To know basic aspects of data acquisition.
- To learn to program basic process interfaces.

3.2 Basic concepts

Figure 1 represents graphically this piece in the Industrial Informatics System.

This lecture will describe basic concepts related to the process interface and generic procedures to deal with this aspect of a computer based control.

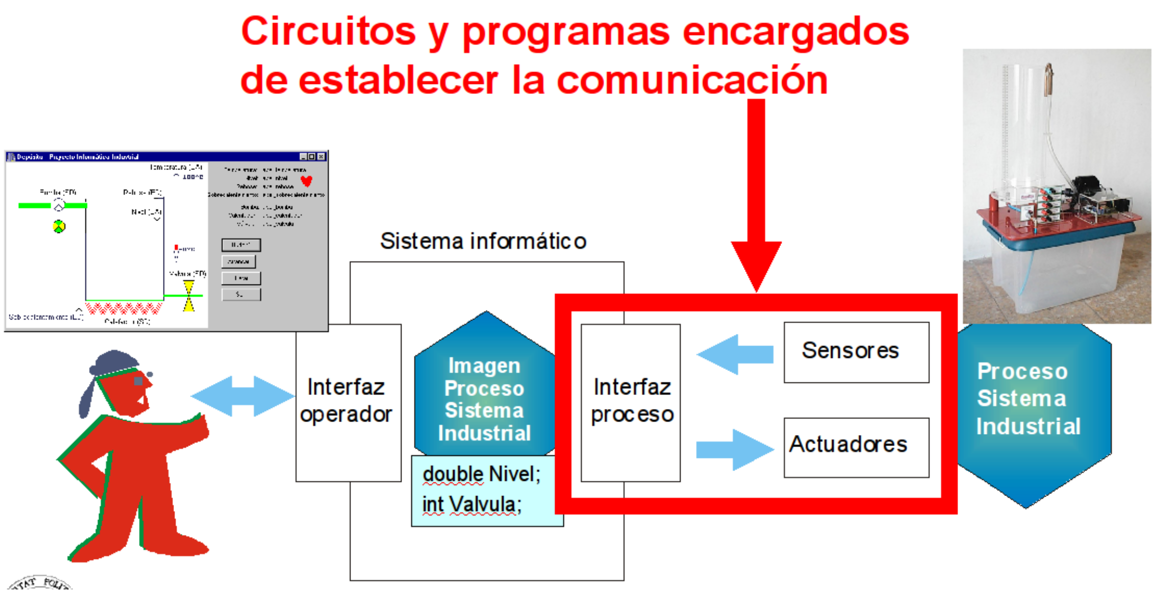


Figure 1: Idea of the interface with the process.

3.2.1 Sensors and actuators

The sensors and actuators are the only elements that are in physical contact with the industrial process. They allow to know the physical state of the process and to modify it.

A sensor is a transducer that translates a physical magnitude into an electrical magnitude. This electrical magnitude describes the value of a variable (position, temperature, pressure ...).

An actuator is a transducer that allows us to modify the state of a physical process by means of an electrical magnitude.

Figure 2 (a) shows an example of a temperature sensor and an electrovalve actuator (b). These elements are appropriate for the liquids tank model.

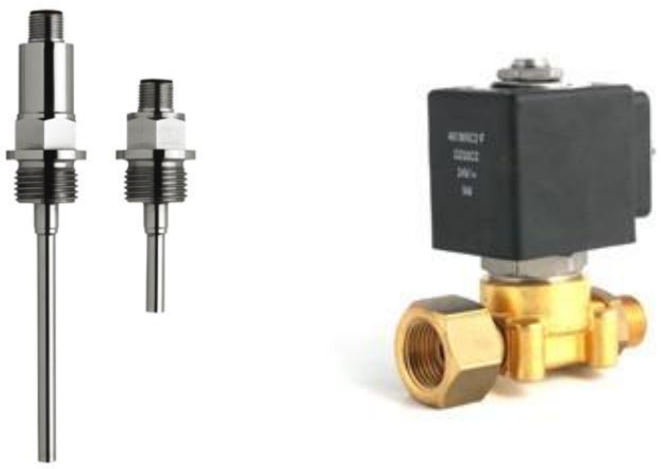


Figure 2: Temperature sensor (a) and electrovalve actuator (b).

3.2.2 Sense of communication

The information between the physical world and the abstract image inside the computer can flow in both senses.

Defining the sense of the signals uses as a reference the controlling system (e.g. a computer, PLC, microcontroller, …). For sensors, the signal is called an input, for actuators, the signal is called output. Figure 3 represents this concept.

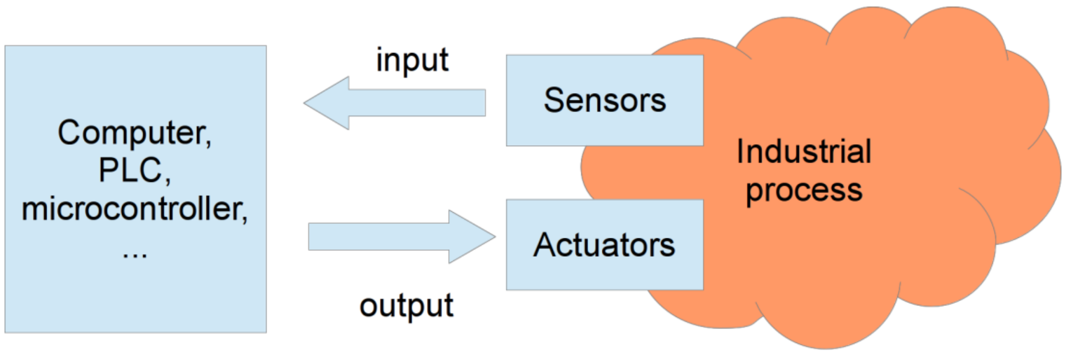


Figure 3: Signals sense in a control system.

In the most general case the sense of communication is bidirectional, so the information flows both ways (from the sensors to the internal image and from that image to the actuators). This case is called a closed loop control system because there is information feedback. Figure 4 shows a temperature regulator that uses an input for measuring temperature and an output for handling a heather.

Figure 4: Temperature regulator.

When there is only output signal to actuators, the system is called open loop In that case; the information about the current state of the process is just estimated For example, a stepper motor control is an output only system. Figure 5 shows an example of a typical actuator used on open loop systems, a digital stepper motor.

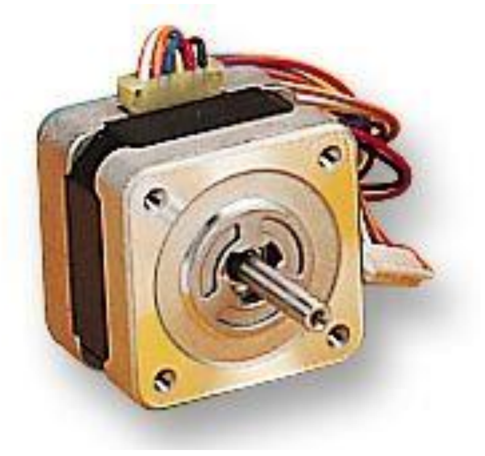


Figure 5: Stepper motor.

On the other side, monitoring systems are just the contrary, the information that flows through the interface is just "input", from the sensors, as there is no intention of controlling the process. For example, a "data-logger" is a typical monitoring system. Figure 6 shows an example of monitor that reads information of the human pulse using a photosensor.

Figure 6: Pulse monitor.

3.2.3 Types of signals

There are two very different types of signals to deal with: digital and analog. The nature of these types of signals is so different that it is necessary to provide different hardware and software mechanisms to adapt the characteristics of the electrical signals involved.

The information or signals that flow between the IIS and the process can be of analog or digital nature.

3.2.3.1 Digital signals

If the physical magnitude of interest can be represented using values in a discrete set, then we can say that the electrical signal is digital.

It is usual that the signals represent two possible values (active/inactive, normal/alarm, etc.). For example, figure 7 shows the possible state of a digital valve, that can be either open or closed. In that case it is possible to represent the value of the interface using a single bit, which value represent the open or close state, this procedure is called encoding. This single bit corresponds to a single digital input line.

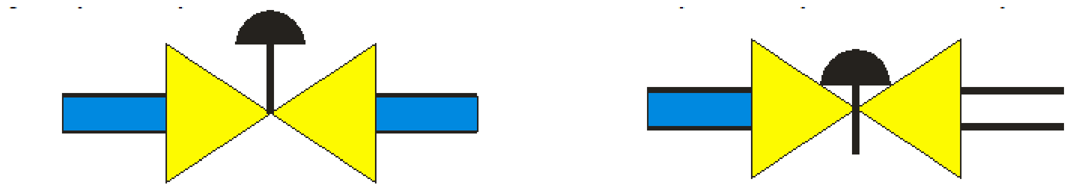


Figure 7: Representation of a digital valve in its two different states.

Others cases include more than two values. As an example, next table contains the discrete set of possible Euro coins that could be inserted in a vending machine.

Type of coin
1 cent
2 cents
5 cents
10 cents
2 cents
50 cents
1 euro
2 euros

In order to design the digital interface between the real set of coins and the computer, it is necessary to “codify” the possible set values using binary numbers. For the coins example, 3 bits would be enough to make the coding in the following way.

Type of coin	Binary encoding
1 cent	000
2 cents	001

5 cents	010
10 cents	011
2 cents	100
50 cents	101
1 euro	110
2 euros	111

The number of bits for the digital representation depends on the encoding technique. Al least, a set of "n" values requires a minimum of ceiling(log2(n)) bits for the encoding.

3.2.3.2 Analog signals

If the physical variable of interest can take values in a continuous range, then the signal is called analog. Typical analog signals in a process can be temperature, speed, pressure, etc.

In these cases, analog sensors/actuators are utilized. These devices have a transfer function that relates the physical magnitude with an electrical magnitude. For example, the figure 8 shows the transfer function of a thermocouple.

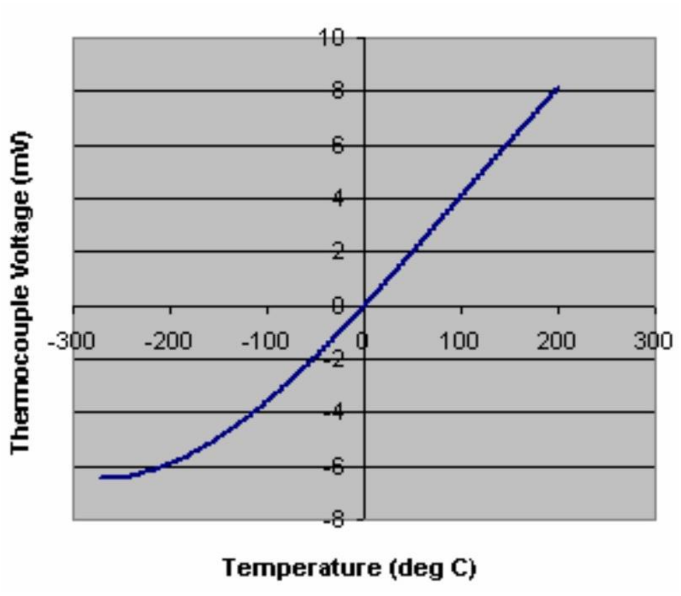


Figure 8: Transfer function of a thermocouple.

3.3 Data acquisition systems

3.3.1 Basic concepts

A data acquisition system (DAQ) is a requirement to allow the flow between the controlling computer system and the physical world. The most basic approach for this is to incorporate specific hardware in the IO subsystem of the computer. A typical IO mechanism in the form get the input signals in the computer system. As an example, figure 9 shows a typical DAQ card for personal computer that is inserted in the IO subsystem of the computer.

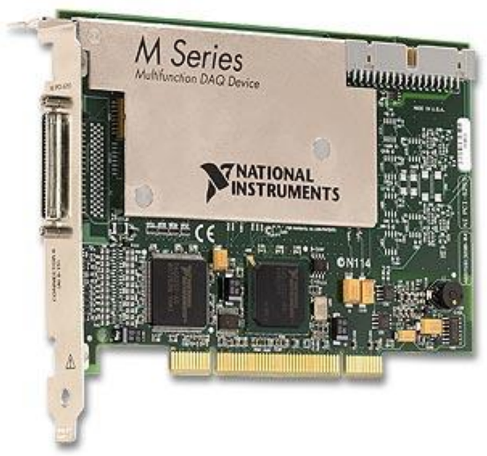


Figure 9: Typical DAQ card for personal computers.

Figure 10 shows a simplified the diagram of the DAQ hardware. Viewed from the outside world to the inside of hardware IO subsystem of the computer, there is adaption circuitry responsible for conditioning the electrics levels, then electrical levels are converted to digital information and, finally, information flows to/from IO registers (data and control).

Ficar (dibuix *3 ##BUSCAR) i explicar-ho un poc

Figure 10: Simplified diagram of a DAQ subsystem.

Depending on the type of architecture of the computer these registers will be seen form the software side as memory addresses in the main memory (for example in the ARM architecture) or as ports addresses (for example in the x86 processors in PCs).

As an example, next fragment of code lets to read a device's register of the x86 architecture.

```
#include <stdint.h>
#define DEVICE_ADDRESS 0xFEA7
uint32_t data;
data = inport(DEVICE_ADDRESS);
```

These registers will contain a copy of binary data that is moving from/to the computer. The size of those registers will depend on the hardware, but, in general, they will be in groups of 8, 16 or 32 bits.

There are more complex approaches for the design of DAQ subsystems. It is interesting to note the addition of some networking facilities to promote flexibility and/or to reach remote signals. For figure 11 shows an USB-based DAQ card.

Figure 11: USB-based DAQ card.

In any case, the internal design of a remote DAQ follows the same architecture and the same programming principles.

Next sections explain the basics of treatment of each type of signal: digitals and analog.

3.3.2 Digital input

By means of the "digital input" mechanism, we obtain the information from the physical signals that can be in two possible states (High/Low, 5V/0V, ...) and translate to digital bits (1/0). Figure 12 shows a block diagram for the digital input mechanism.

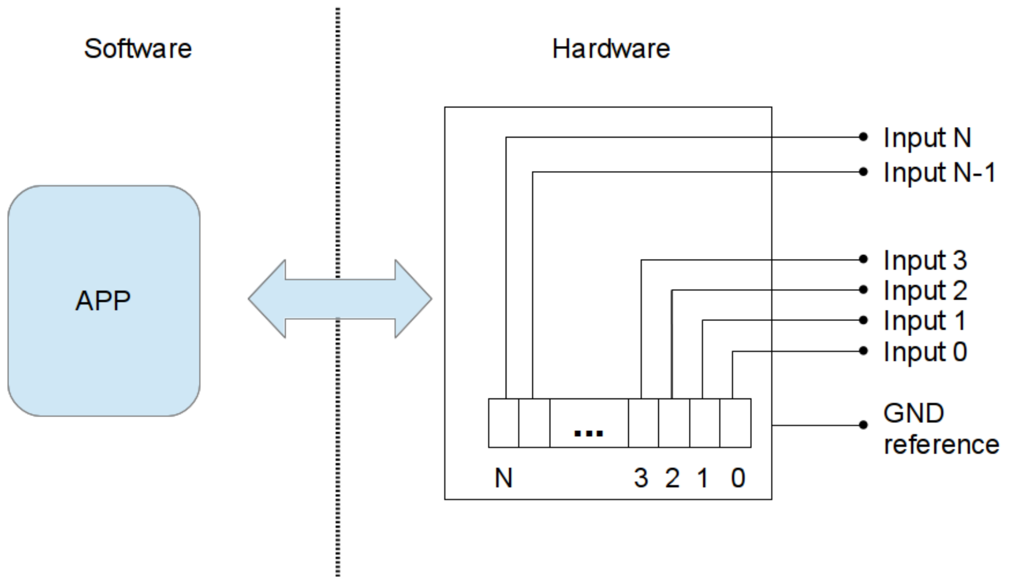


Figure 12: Diagram of the digital input subsystem.

In the right side of the diagram, we have physical digital lines that will be connected to adequate sensors. The lines can be in two possible states depending on the state that the digital sensor generates. The interface electronics will be in charge of translating these electrical levels into contents in corresponding bits inside the registers.

The type of digital signal applied to the line and its translation to a binary value will depend on the design of the hardware. For example, if the lines are TTL compatible, then:

- For 5 volts we have a “1” logic value.
- For 0 volts we have a “0” logic value.

From the software programming perspective, reading a digital input is based on reading the specific data register associated with the data acquisition hardware, so it is needed to learn how to handle correctly the bits inside these registers. In general these registers are read in block and it is not possible to read a single line individually.

The next step is to elect to which line each sensor is connected to. For example, we could connect an overflow sensor to the bit 0 of the register of the digital input, and we could assign the bits 3, 2 and 1 of a digital coin reader. See figure 13.

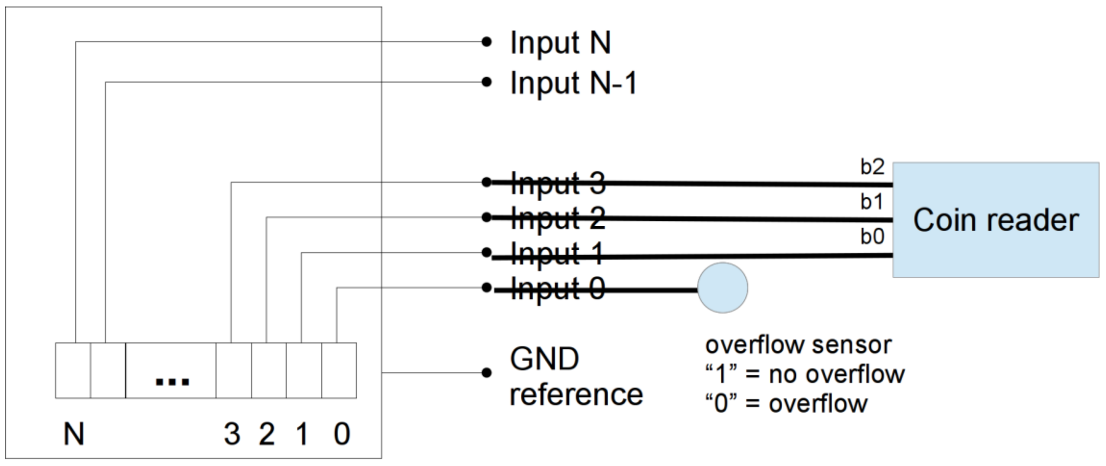


Figure 13: Example of pin assignment.

Taking into consideration that we read the complete input register, we need to apply the appropriate mask to discriminate the adequate bits an interpret it. The use of the logical AND operator (& in C language) is necessary here.

For example, to know the state of the sensor, we need to apply the following mask to the read value of the register.

```
       bN ... b6 b5 b4 b3 b2 b1 b0

AND  0        0  0  0  0  0  0  1

-------------------------------

       0        0  0  0  0  0  0  b0
```

A minimal fragment of C code to deal with this sensor could be:

```
    uint32_t data;

    data = inport(???);
    if ((data & 0x01) != 0) {
        printf ("Upss! Tank overflowww!!!");
    } else {
        printf ("No overflow.");
    }
  }
```

When developing the code you have to try to deliver the information from the data acquisition subsystem in the most abstract and clear manner possible to the rest of the application. This simple code can is the basis for the following example that provides the adequate function for reading the overflow sensor of the tank. See how the enumeration values are utilized for providing an abstraction of the electronic world.

```
  enum TOverflowStatus { OVERFLOW_YES, OVERFLOW_NO};


  TOverflowStatus process_ReadOverflowSensor(void){

    uint32_t data;

    data = inport(???);
    if ((data & 0x1) ! =0) {
        return OVERFLOW_YES;
    } else {
        return OVERFLOW_NO;
    }
  }
```

A similar approach can be applied to the coin reader. The following fragment of code creates an abstraction of the underlying hardware.

```
  enum TCoin { COIN_1_CENT, COIN_2_CENTS, … , COIN_2_EURO};

  TCoin CoinRead(void) {

    uint32_t data;
    uint32_t coin_data;

    data = inport(???);
  ..coin_data = (data & 0x0E) >> 1;


    switch (coin_data) {
      case 0x0:
        return COIN_1_CENT;
        break;
  …
      case = 0x7:
        return COIN_2_EURO;
        break;
}
```

4 Seminar: DAQ application with National Instruments devices

4.1 Objectives

- To understand the application of the National Instruments USB-6008 DAQ card.
- To install, configure and use a commercial DAQ Card.
-

4.2 The NI USB-6008 DAQ

The National Instruments USB-6008 DAQ card will utilized for applying the terethical concepts presented in the lecturer session. These are their main advantages:

- It is a low-cost, general purpose card.
- It is ideal for teaching but is very limited when applying it to projects with more strict requirements.
- In the case of a more serious project, another card is recommended.

Figure 1: Package contents for the NI USB-6008 DAQ

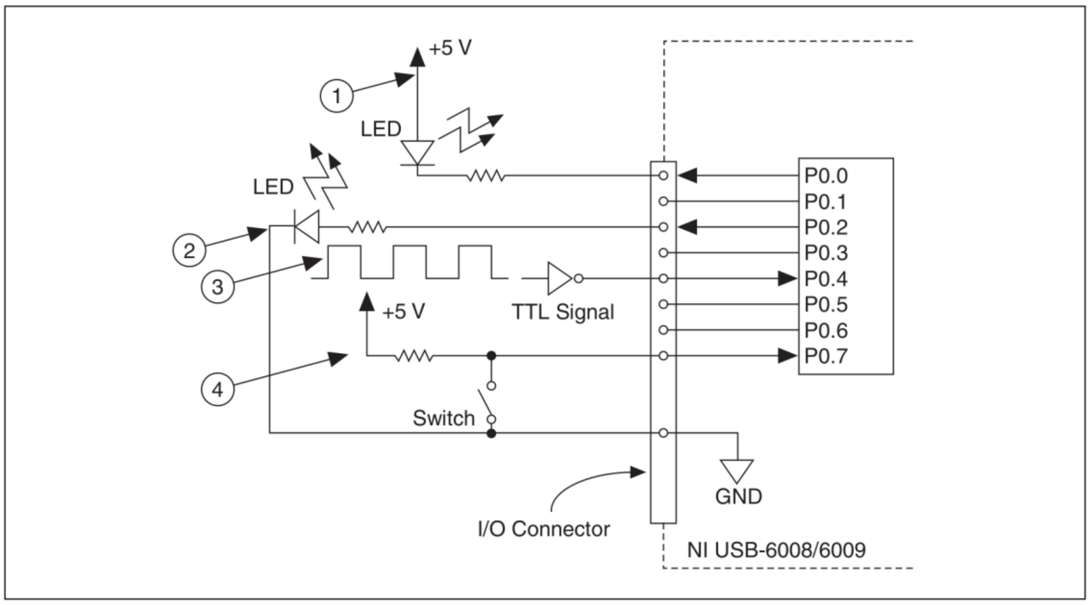
Characteristics:

- Cheap (about €150).
- USB connection interface with the computer.
- 12 digital input/output TTL compatible channels.
- 8 mono-polar or 4 differential inputs as analog input of 12 bits.
- Variable gain in differential input mode.
- 2 analog output channels of 12 bits.
- 1 32 bit digital counter.

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”.

4.3 Digital input/output connections

In this seminar, we are interested on digital input. This is the diagram of physical connection to the card.



1	P0.0 configured as an open collector digital output driving an LED
2	P0.2 configured as an active drive digital output driving an LED
3	P0.4 configured as a digital input receiving a TTL signal from a gated invertor
4	P0.7 configured as a digital input receiving a 0 V or 5 V signal from a switch

Figure 2: Digital IO connection schematics.

ACTIVITY: In groups, read the document named “NI USB-6008/6009 User Guide And Specifications” and analyze the section related to IO connections.

4.4 Software libraries: NI-DAQmx

The best option for developing software using the C/C++ language is to uses the NI-DAQmx libraries provided by the manufacturer. These libraries offer a common interface for all data acquisition cards.

This is very important because, in general, a change of the model and/or manufacturer of the card brings a change of libraries and API, so the applications must be rewritten.

NIDAQmx libraries are available for the Microsoft Windows, MacOX and Linux platforms.

The main advantages of NI-DAQmx are:

- The interface is common for all cards. The programs need no change when changing another card
- Multiplatform: Windows, Linux Mac OSX.
- High level functions: direct reading of thermocouples, accelerometers, multi-thread programming, etc.
- NI Labview, NI Labwindows/CVI, Microsoft Visual Basic, Microsoft Visual C++ and ANSI C interfaces.

In order to use these libraries for the Qt Framework Windows Open Source version, follow the indications available at http://www.disca.upv.es/aperles/qt/qt_nidaqmx/qt_nidaqmx.html.

Based, reading the National Instruments documents about the library can be cumbersome, so some explanation is included here before starting to work.

The working philosophy of the library is:

- Create acquisition tasks associated to "channels".
- Use a task when we need an acquisition.

About tasks, saying that:

- Conceptually, a task represents a measure or a signal generation that is going to be done.
- In this case it is a set of one or more virtual channels, with timing, trigger and other properties.
- All the channels included in a task must be of the same type, like analog input, digital output, counter output....
- To make a measure or generate a signal with a task these step have to be followed:
  1. Create a task or load an existent one. Function DAQmxCreateTask().
  2. Configure the channel and, if necessary, timing and trigger.
  3. Read or write or samples.
  4. Delete the taskHandle. DAQmxClearTask().
- If the task is going to repeat over time (e.g. a loop), step 3 is repeated. You can use functions such as DAQmxStartTask() and DAQmxStopTask().

A task needs "channels". Channels are:

- Two types: physical and virtual.
- A physical channel is a terminal or pin through which we can measure or generate a signal (analog or digital). Each physical channel has a basic format "name of device/name of channel", for example Dev2/ao5 or Dev6/ctr3.
- The virtual channels are software entities that encapsulate the physical channel with another information specific for it (range, terminal configuration, scale, etc.) , that formats the data.
- Different types of virtual channels can be crated depending on the type of signal and its direction:
- Analog input channels
- Analog output channels
- Digital I/O channels (based on lines or in ports)
- Counter I/O channels

It is required the creation of a channel associated with physical lines before starting the acquisition of digital signals. This can be accomplished using the function:

- DAQmxCreateDIChan()

To use a task configurated with a digital input channel, we can use the following functions:

- DAQmxReadDigitalScalarU32() -> to read the entire port
- DAQmxReadDigitalLines() -> to read individual lines or groups
- DAQmxReadDigitalU8() -> to read the entire port

5 Lab: Digital input

5.1 Objective

The goal is to implement the code for handling a digital input of the NI USB-6008 DAQ card to read the state of a digital sensor.

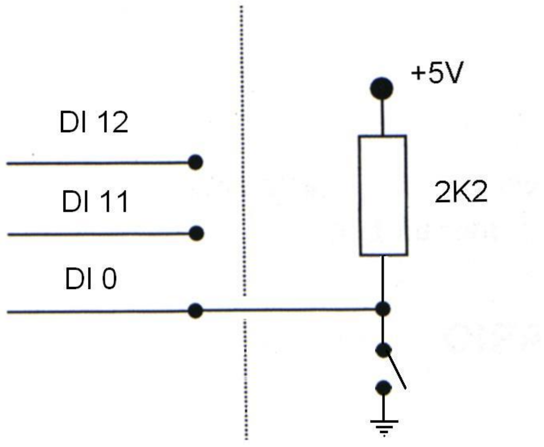
5.2 Equipment

- 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 pushbutton / switch and one 2K2 resistor.
- Copper wires, or male to male Dupont wires
- Screwdriver.
- Breadboard.

5.3 Departing point

The main objective of the application is to read the state of a digital input. To provide the electronic state to the input of the DAQ card, a simple push-button or switch can simulate the operation of the liquid overflow sensor, that will apply a logic signal (1/0) to the corresponding TTL compatible input.

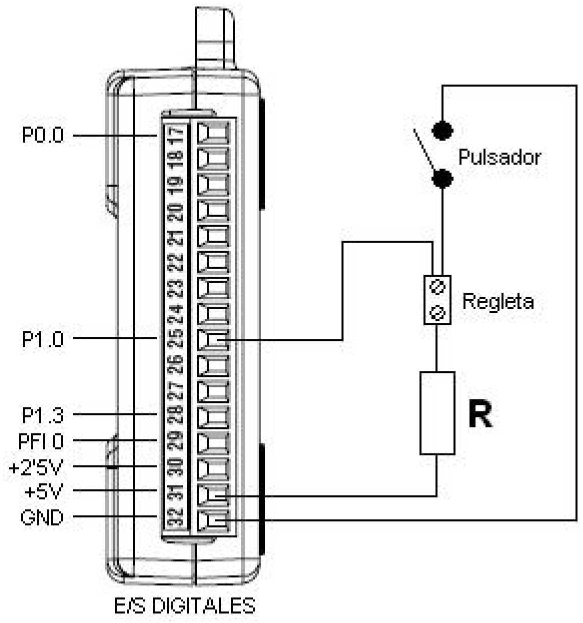
The configuration is as follows,



The overflow sensor uses the following logic:

- 0 (0V) → There are overflow
- 1 (+5 V) → No overflow

The assembly diagram of the components and board is as follows,



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 read function.

```
/**


    @file    process.h

    @brief   Process interface prototypes for GELASA project
```

```
*/

#ifndef PROCESS_H
#define PROCESS_H

void process_Init(void);


enum TOverflowStatus {OVERFLOW_YES, OVERFLOW_NO};
TOverflowStatus process_ReadSensorOverflow(void);

#endif
```

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

```
/**
    @file    process.c
    @brief   Process interface implementation for GELASA project

    This module implements the functions for a National Instruments USB-
6008 DAQ

*/

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

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

#define DIGITAL_INPUT_PORT          "port0"     // port used for digital
input
#define OVERFLOW_SENSOR_BIT         0           // line used for the
overflow 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 digital_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)
{
   // digital input preparation
   // first the task creation
   DAQmxCreateTask("Digital input task",&digital_input_task);
   // configure the task for reading a complete port

DAQmxCreateDIChan(digital_input_task,DAQ_NAME/**/"/"/**/DIGITAL_INPUT_PORT,
"", DAQmx_Val_ChanForAllLines);
}

/***************************************************************************
****/
```

```
/**
    @brief  Reads the overflow sensor
    @param  none

    @retval OVERFLOW_YES or OVERFLOW_NO
*/
TOverflowStatus process_ReadSensorOverflow(void)
{

   uInt32 data;

   DAQmxStartTask(digital_input_task);
   DAQmxReadDigitalScalarU32(digital_input_task,0.0,&data,NULL);
   DAQmxStopTask(digital_input_task);

   // test the bit, 0=YES, 1=NO
   if ((data & (1<<OVERFLOW_SENSOR_BIT)) == 0) {
      return(OVERFLOW_YES);
   } else {
      return(OVERFLOW_NO);
   }
}
```

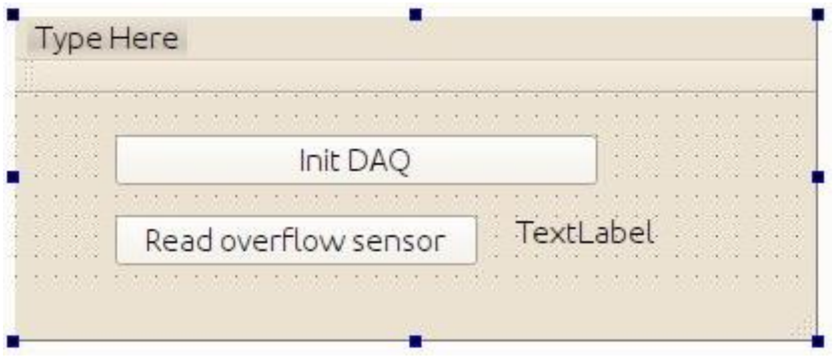
Now, create a Qt application and adjust the .pro file in order to provide information about the NIDAQmx libraries. Adjust to your own system installation

```
# 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 MainForm, add to buttons and a 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_btReadOverflowSensor_clicked()
{
    if (process_ReadSensorOverflow() == OVERFLOW_YES) {
        ui->lbOverflowState->setText("DANGER Overflow!!!");
    } else {
        ui->lbOverflowState->setText("No overflow");
    }
}
```

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.,

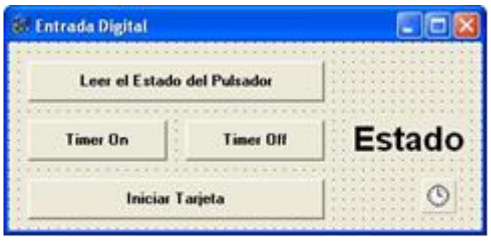
```
   int32 daq_error;
    // apply to each DAQ function call
    daq_error=DAQmxCreateTask(…
    if(daq_error != 0) {
        printf("Problems!!!");
        exit(1);
    }
```

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).

This could be the aspect of the app,



6 Mini-project : Implementation of the process interface module: Digital input

Develop the part of the process module related to digital input.

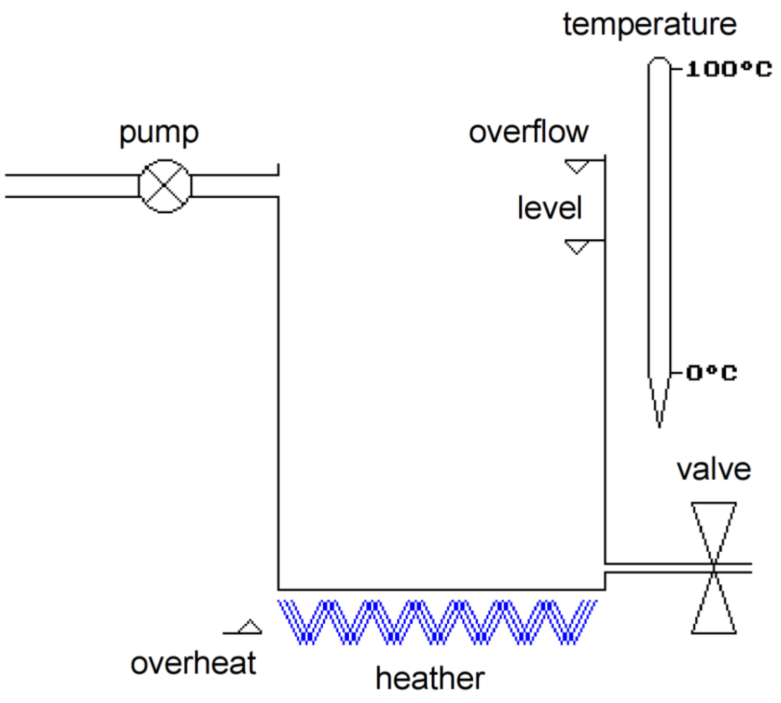
Validate the developed functions creating the appropriate test functions.

7 Extra: activities

ACTIVITY**: **Signals table of the tank of liquids model

When electing the sensors and actuators for a given IIS project it is fundamental to document the type of sensor and its connections. A simple table can be used to gather that information.

Given the following diagram of the tank of liquids problem, decide the nature and sense of the sensors and actuators connected to the tank model. Take into consideration that the election depends on the problem to be solved: the same design could use analog or digital devices based on the requisites.



And fill in the following table.

Signal name	Type	Sense	Connection	Description

overflow	Not now	Model, range, transfer function
overheat	Not now	
electrovalve	Not now	
heater	Not now	
temperature	Not now	
level	Not now	
pump	Not now	

ACTIVITY: Liquids level sensor.

Locate a liquids level sensor in Internet and present it to your colleagues.

ACTIVITY: Implement functions for handling the overheat sensor.

8 References

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