

*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 5.2: Process interface digital 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 5.2: Process interface digital 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 :	<a href="https://www.medis-tempus.eu">https://www.medis-tempus.eu</a>

## 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
1.0	19/09/2014	UPV Team	Final version

# Table of Contents

1	EXECUTIVE SUMMARY.....	1
2	INTRODUCTION.....	1
3	LECTURE .....	1
3.1	OBJECTIVES .....	1
3.2	DIGITAL OUTPUT .....	1
4	SEMINAR: DIGITAL OUTPUT WITH NI USB-6008 DAQ CARD .....	3
4.1	OBJECTIVES .....	3
4.2	NI USB-6008 DAQ DIGITAL OUTPUT.....	3
4.3	NI-DAQMX FUNCTIONS FOR DIGITAL OUTPUT.....	4
5	LAB: DIGITAL OUTPUT .....	5
5.1	OBJECTIVE .....	5
5.2	EQUIPMENT .....	5
5.3	DEPARTING POINT .....	5
5.4	ACTIVITY.....	11
6	MINI-PROJECT: IMPLEMENTATION OF THE PROCESS INTERFACE MODULE: DIGITAL OUTPUT 12	
7	EXTRA: ACTIVITIES.....	12
8	REFERENCES.....	12



## 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 digital output mechanism. The “digital output” mechanism is the generation of signals with two states in the physical lines of the data acquisition system.

## 3 Lecture

### 3.1 Objectives

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

### 3.2 Digital output

Digital output mechanism is similar to digital input. In this case, we write output registers. the data acquisition hardware. In this case, we need to modify groups of bits.

The physical outputs of the data acquisition system may be of different types: TTL, open collector/drain, etc, so the circuitry must be adapted to these characteristics. Note that, in general, the digital outputs are not able to handle more than a few milliamps of current, so most of the conditioning is focused to increasing the driving power.

Similar to digital input, we have the registers which bits will be associated to physical lines and whose value will be associated to the state of the output of the physical line. See figure 14.

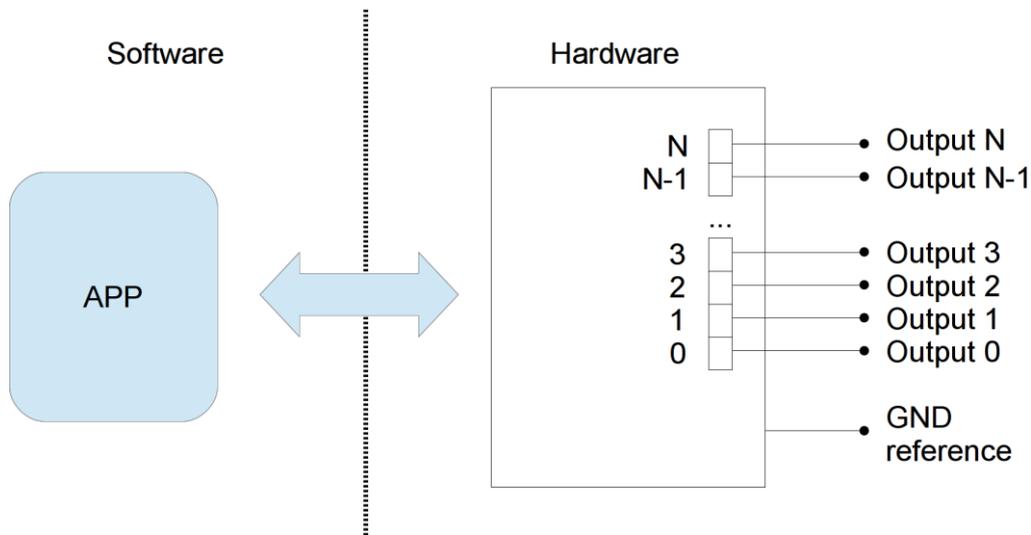


Figure 14: Diagram of the digital output subsystem.

It is common that the output register can only be written at once, affecting all the physical lines. So there is no chance to modify them bit by bit. As digital outputs are used to connect independent elements (valves, relays, ...) it is important to be able to handle that independence at software level.

It is also common that the digital output registers to which the software accesses to modify the output are "read only". That adds an extra complexity at software level that can be solved using auxiliary "mirror" variables. Figure 15 shows a diagram of use of this mirror variable.

The "mirror variable" represents the state of the output register, and the user software is responsible for operating on it for getting the appropriate bits that represents the desired output value; once modified, the mirror variable is copied on the output register.

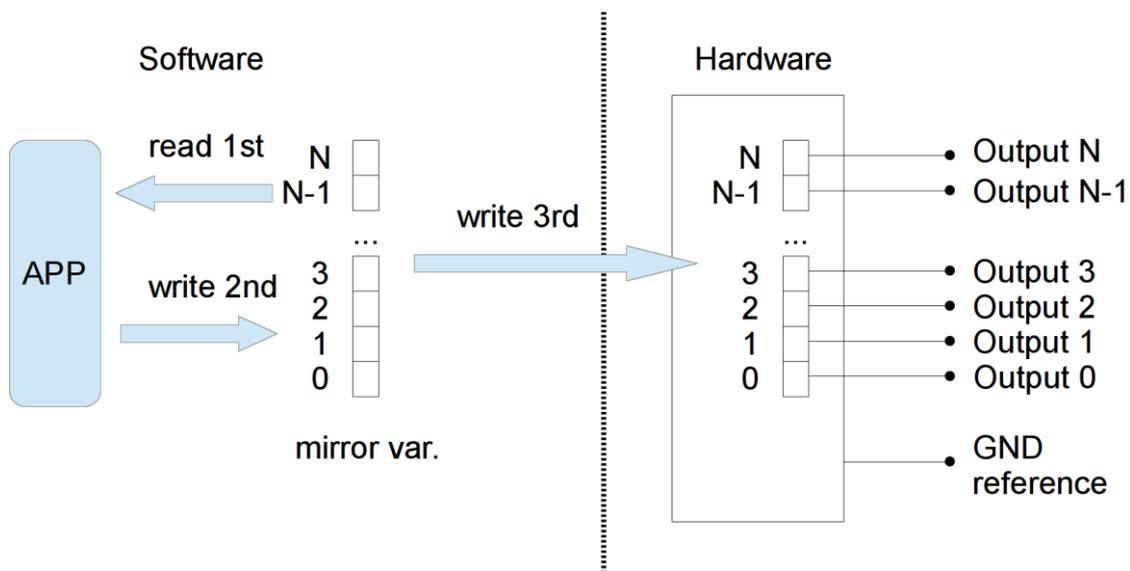


Figure 14: Diagram of the digital output subsystem.

The next C module shows the application of these principles for the electrovalve's digital output of the tank of liquids. Following previous modules, this module hides the underlying hardware and shows an abstract view of the electrovalve's behavior that can be applied to other DAQ systems.

```
/*
 *@file process.cpp
 **/
#include <stdint.h>
#include "process.h"
enum TStatusValve {VALVE_OPENED, VALVE_CLOSED};
#define OUT_REGISTER 0XFEAFEO // a sample register address
uint32_t mirror_digital_output;
void process_Init(void) {

    mirror_digital_output = 0x0000;
    out(OUT_REGISTER, mirror_digital_output);
}
void process_WriteActuatorValve(TStatusValve status) {
    // "1" means open
    // "0" means closed
    // bit for valve is bit 6

    // modify the mirror according "status" variable
    if (TStatusValve == VALVE_CLOSED){
        mirror_digital_output= mirror_digital_output & 0xFFFFFDF;
    } else {

        mirror_digital_output= mirror_digital_output | 0x40;
    }
    // put the mirror in the out() register
    out(OUT_REGISTER, mirror_digital_output);
}
}
```

## 4 Seminar: Digital output with NI USB-6008 DAQ card

### 4.1 Objectives

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

### 4.2 NI USB-6008 DAQ digital output

That diagram of figure shows the physical connection to the card. We are now interested in digital output signals.

**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 digital output.

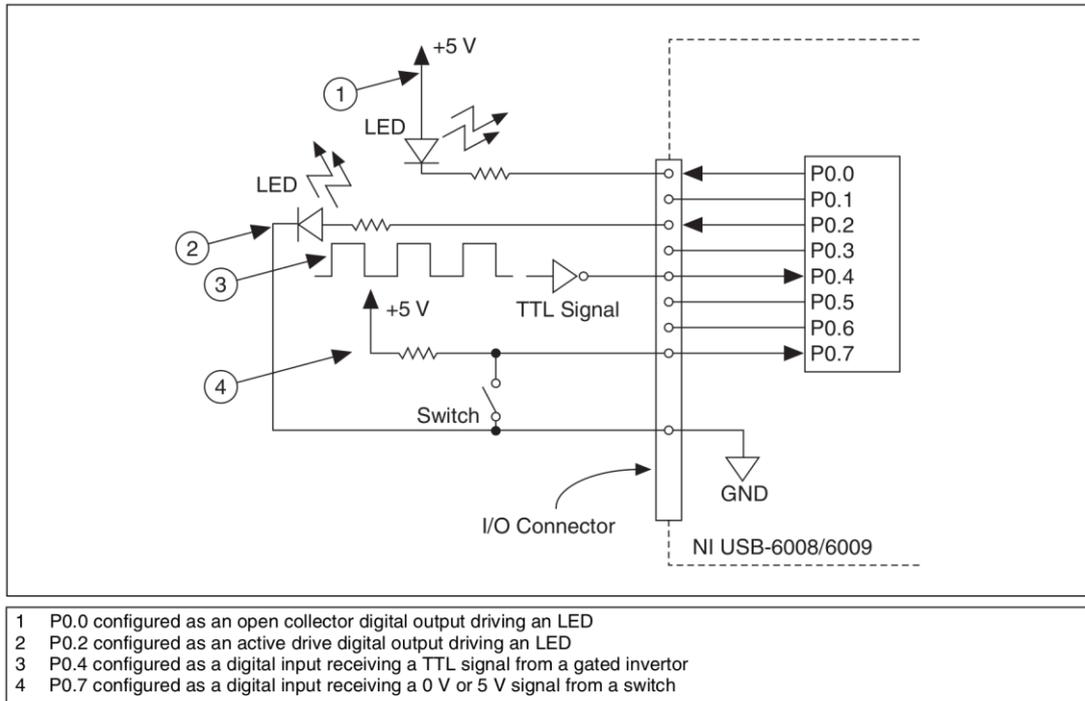


Figure 1: Digital IO connection schematics.

### 4.3 NI-DAQmx functions for digital output

Following the idea of the previous seminar. There are a set of functions of the NI-DAQmx library that deal with digital output signals.

Before writing digital outputs, you need to configure a “digital output” channel. This can be accomplished using the following function:

- DAQmxCreateDOChan()

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

- DAQmxWriteDigitalScalarU32() -> to write the entire port
- DAQmxWriteDigitalLines() -> to write individual lines or groups
- DAQmxWriteDigitalU8() -> to write the entire port

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

## 5 Lab: Digital output

### 5.1 Objective

The goal is to implement the code for handling a digital output of the NI USB-6008 DAQ card to handle a LED that represents the state of a valve.

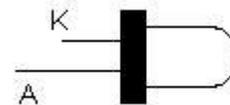
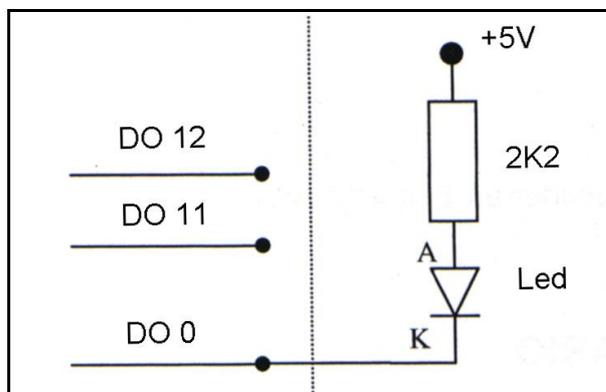
### 5.2 Equipment

- 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 LED and 1 2K2 resistor.
- Copper wires, or male to male Dupont wires
- Screwdriver.
- Breadboard.

### 5.3 Departing point

The aim is to develop an application that activates/deactivates a digital output that represents the electrovalve of the tank of liquids model. Instead of the valve a LED is utilized for representing the open/close state of the valve.

The data acquisition card NI USB-6008 has 12 digital input/output. We elected one of these outputs according to the following mounting configuration:



To handle the LED, the following logic is utilized:

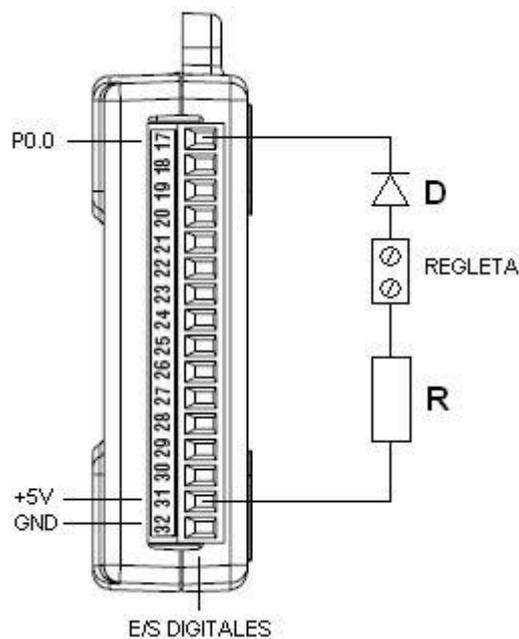
- LED ON -> P0.0 = 0 (0V)

- LED OFF -> P0.0 = 1 (+ 5V)

The reason for using this configuration is sink/source capability of the output. According the card specifications, we have the following digital I/O driving capacity:

- Low Output Voltage: Max. + 0.8V at 8.5 mA
- High Output Voltage: Min + 2V at 0.6 mA.

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 digital output functions.

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

#ifndef PROCESS_H
#define PROCESS_H

void process_Init(void);
```

```

enum TValveStatus {VALVE_OPENED,VALVE_CLOSED};
void process_WriteActuatorValve(TValveStatus state);

#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.c
    @brief   Process interface digital output
*/

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

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

#define DIGITAL_OUTPUT_PORT     "port1"     // port used for digital
output

#define VALVE_ACTUATOR_BIT      0           // line used for valve
actuator

#define DIGITAL_OUPUT_MIRROR_INIT 0x0000    // initial value for the
mirror

// headers -----
----

#include <stdio.h>

// 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_output_task;

UInt32_t digital_output_mirror;

/*****/
/**
    @brief  Inits the DAQ system
    @param   none
    @returns  none

    This function must be called before using any adquisition functions
*/
void process_Init(void)
{

    // digital output peraration
    DAQmxCreateTask("Digital output task",&digital_output_task, __LINE__);
    // configure task for writting a complete port
    DAQmxCreateDOChan(digital_output_task,
        DAQ_NAME/**/"**/DIGITAL_OUTPUT_PORT,
        "", DAQmx_Val_ChannelsForAllLines);

    digital_output_mirror = DIGITAL_OUPUT_MIRROR_INIT; // initial value for
the mirror
}

```

```

/*****/
/**
 @brief Handles/writes the valve
 @param {VALVE_CLOSED, VALVE_OPENED}

 @returns none
 */
void process_WriteActuatorValve(TValveStatus state) {

 // values for the output, 0=VALVE OPEN, 1=VALVE CLOSED
 if (state==VALVE_OPENED) {
     digital_output_mirror = digital_output_mirror | (1 <<
 VALVE_ACTUATOR_BIT);
 } else {
     digital_output_mirror = digital_output_mirror & ~(1 <<
 VALVE_ACTUATOR_BIT);
 }

 // put the "mirror" variable in the outputs
 DAQmxStartTask(digital_output_task);
 DAQmxWriteDigitalScalarU32(digital_output_task,false,0.0,
     digital_output_mirror,NULL);
 DAQmxStopTask(digital_output_task);
 }

```

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

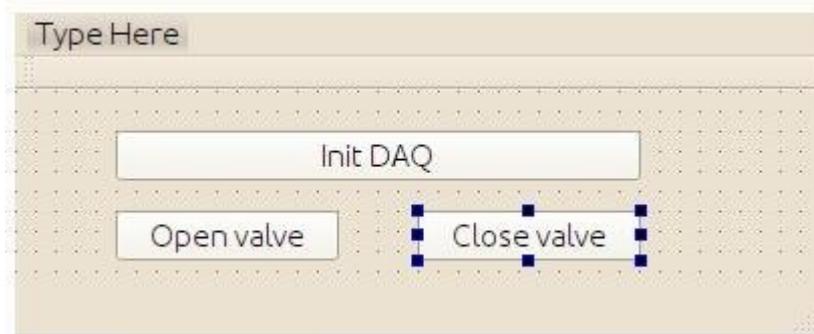
```

```
}

```

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

In the MainForm, add buttons 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_btValveOpen_clicked()
{
    process_WriteActuatorValve (VALVE_OPENED);
}

void MainWindow::on_btValeveClose_clicked()
{
    process_WriteActuatorValve (VALVE_CLOSED);
}

```

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);
}

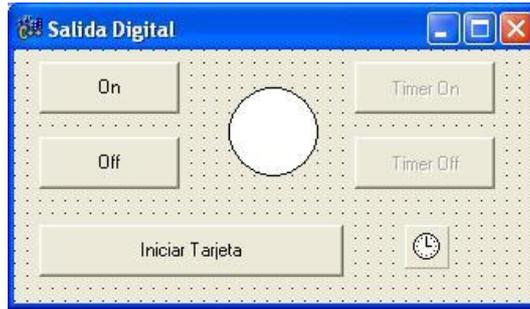
```

## 5.4 Activity

Include a QTimer object that automatically opens and closes the valve every 2 seconds. Add a graphical object tha represents, visually, the state of the valve.

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 output

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

Validate the developed functions creating the appropriate test functions.

## 7 Extra: activities

**ACTIVITY:** Find in Internet a suitable electronic circuit in order to drive an electrovalve using a TTL source.

### ACTIVITY

Assuming that you have the heater connected to bit 2 of the digital output register. Provide the function with prototype

```
void process_WriteActuatorHeater (TStatusHeater status);
```

That follow this logic

“1” means OFF

“0” means ON

## 8 References

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