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

WP4 Deliverable 4.1 Example of Lecture

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

Authors: Domínguez, C., Hassan, H., Martínez, J.M.

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

WP4 Deliverable 4.1 Example of Lecture

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

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

Deliverable Number: 4.1

Title of the Deliverable: Advanced Industrial Informatics Specialization Modules

Task/WP related to the Deliverable: Industrial Computers Module

Type (Internal or Restricted or Public): Public

Author(s): Domínguez, C., Hassan, H., Martínez, J.M.

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 4	Advanced Industrial Informatics Specialization Modules
WPLeader	Universitat Politècnica deValència (UPV)
Task 4.1	Industrial Computers Module
Task Leader	UPV
Dependencies	MDU, TUSofia, USTUTT, UP

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

History

Version	Date	Author	Comments
0.1	01/09/2014	UPV Team	Initial draft
1.0	29/09/2014	UPV Team	Final version

Table of Contents

Deliverable 2.1 UPV:


UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA
medis

List of contents

Deliverable 2.1 UPV:

- Chapter 1 - Introduction to Industrial Informatics
- Chapter 2 – Computer
- Chapter 3 - Project Planning
- Chapter 4 - Programming + Data
- **Chapter 5 - Process Interface**
- Chapter 6 - User Interface
- Chapter 7 - Tasks
- Chapter 8 - Regulation
- Chapter 9 - Project (2) Ending

MEDIS Project
Deliverable 2.1 UPV
Chapter 5.2

Process Interface digital output

Lecture: 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 fort digital output.

Lecture: Digital output

- Digital output mechanism is similar to digital output. In this case, we write output registers. the data acquisition hardware. In this case, we need to modify groups of bits.
- The "digital output" mechanism is the generation of signals with two states in the physical lines of the data acquisition system.
- 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.
- 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.

Lecture: Digital output (Cont.)

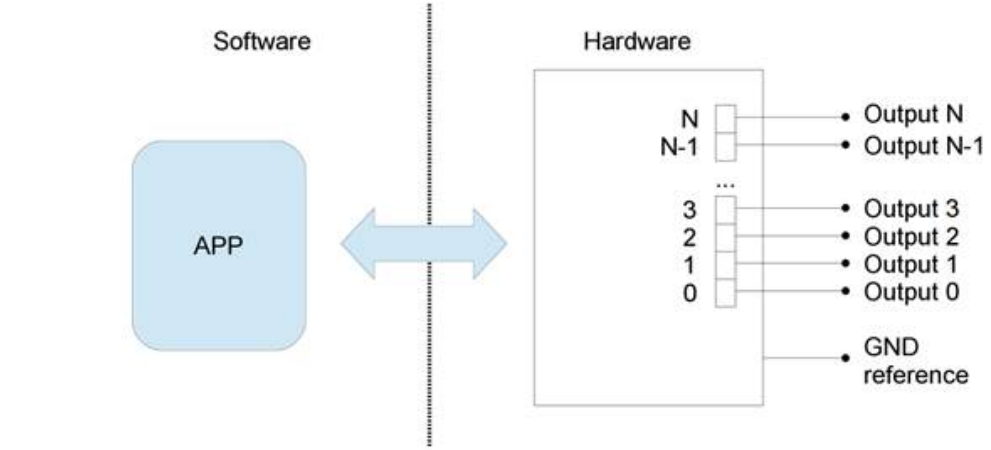
I

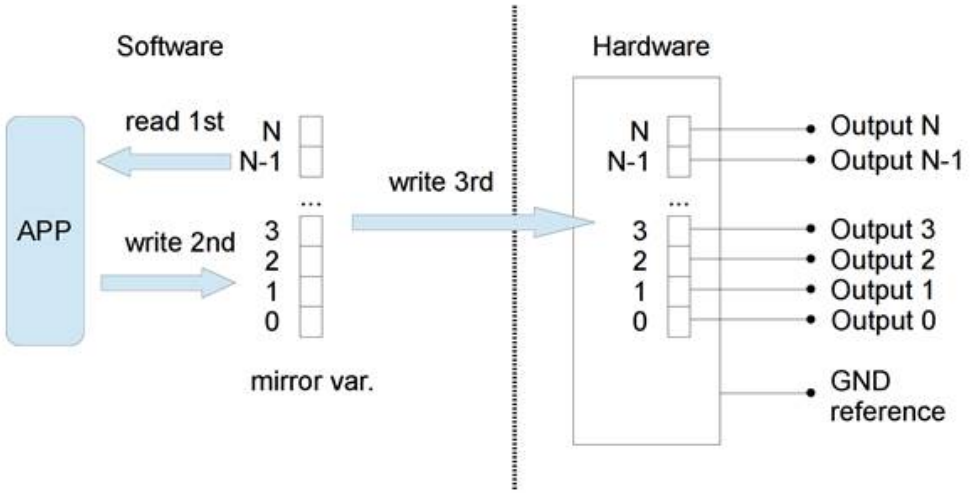
- t is common that the output register can only be written at once, affecting all the physical lines. So there is no chance to modify them bit by bit. As digital outputs are used to connect independent elements (valves, relays, …) it is important to be able to handle that independence at software level.

- 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 al software level that can be solved using auxiliary "mirror" variables.

- 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.

Lecture: Digital output (Cont.)





Lecture: Digital output (Cont.)

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_OPENNED, VALVE_CLOSED};
#define OUT_REGISTER 0XFEAFE0  // 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 & 0xFFFFFFDF;
    } else {

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

Seminar: Digital output with NI USB-6008 DAQ card

Objectives:

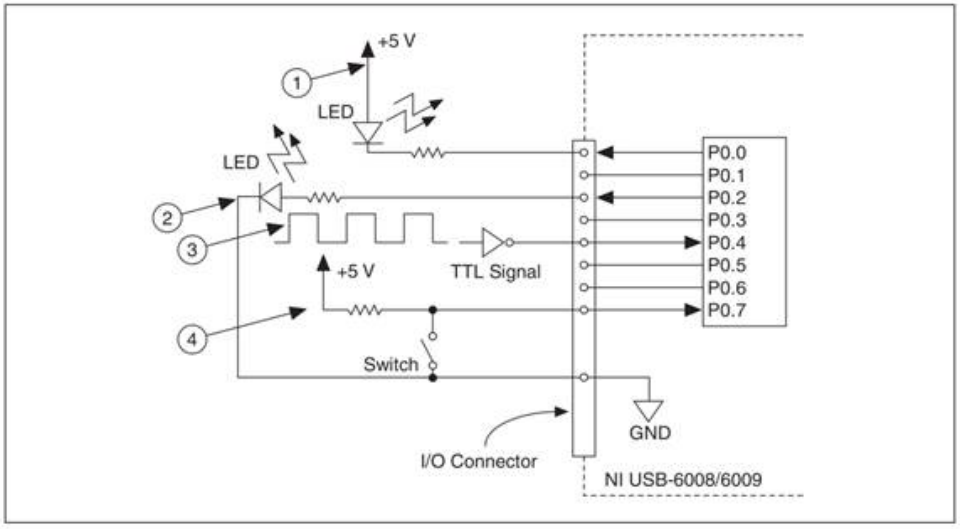
- To understand the physical connection of digital output devices to a real DAQ card
- To develop software to deal with digital output signals on real hardware.

Seminar: Digital output with NI USB-6008 DAQ card

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.



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

Seminar: Digital output with NI USB-6008 DAQ card

NI-DAQmx functions for digital output

The are a set of functions of the NI-DAQmx library that deals 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.

Lab: Digital output

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.

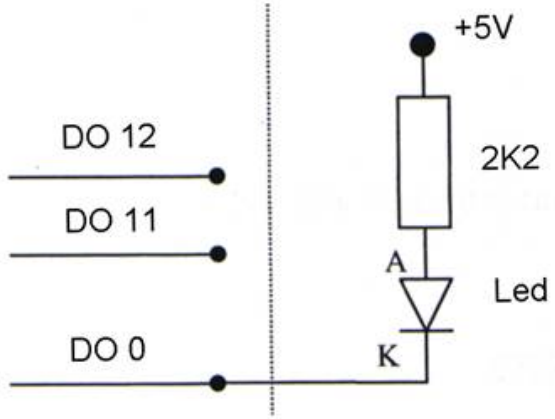
Equipment:

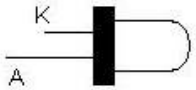
- Personal Computer with Microsoft Windows 7 or superior operating system.
- Open source version of Qt framework for Microsoft Windows.
- Data acquisition card National Instruments USB-6008.
- 1 LED and 1 2K2 resistor.
- Copper wires, or male to male Dupont wires
- Screwdriver.
- Breadboard.

Lab: Digital output

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:

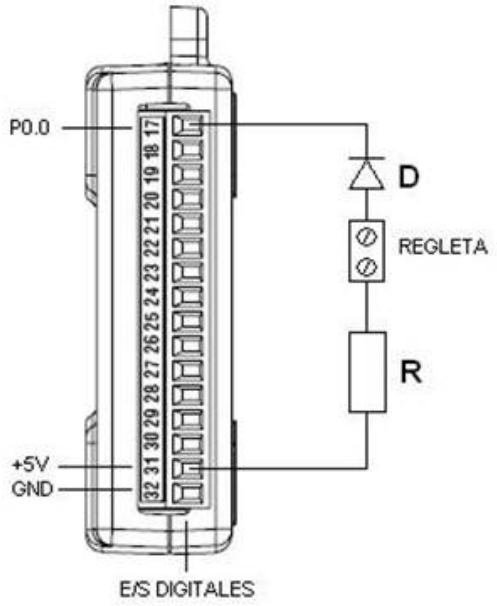
LED ON -> P0.0 = 0 (0V)
LED OFF -> P0.0 = 1 (+ 5V)

Lab: Digital output

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:



Lab: Digital output (Cont.)

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_OPENNED,VALVE_CLOSED};
void process_WriteActuatorValve(TValveStatus state);

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

Lab: Digital output (Cont.)

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_ChanForAllLines);

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

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

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

   // values for the output, 0=VALVE OPEN, 1=VALVE CLOSED
   if (state==VALVE_OPENNED) {
      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);
}
```

Lab: Digital output (Cont.)

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

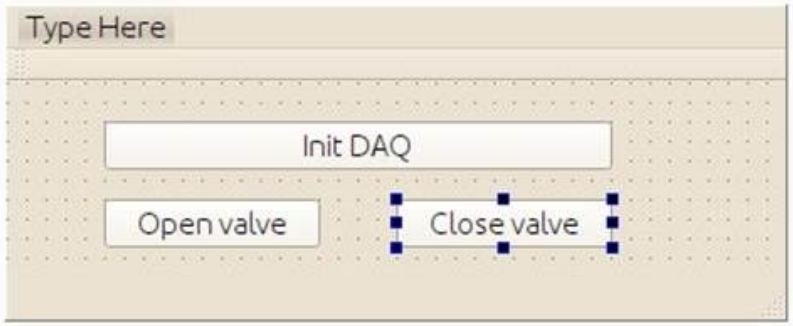
```
# National Instruments NIDAQmx configuration
win32 {
    INCLUDEPATH += "C:/Archivos de programa/National Instruments/NI-DAQ/DAQmx
ANSI C Dev/include"

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

Lab: Digital output (Cont.)

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,



Lab: Digital output (Cont.)

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_OPENNED);
}

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.,

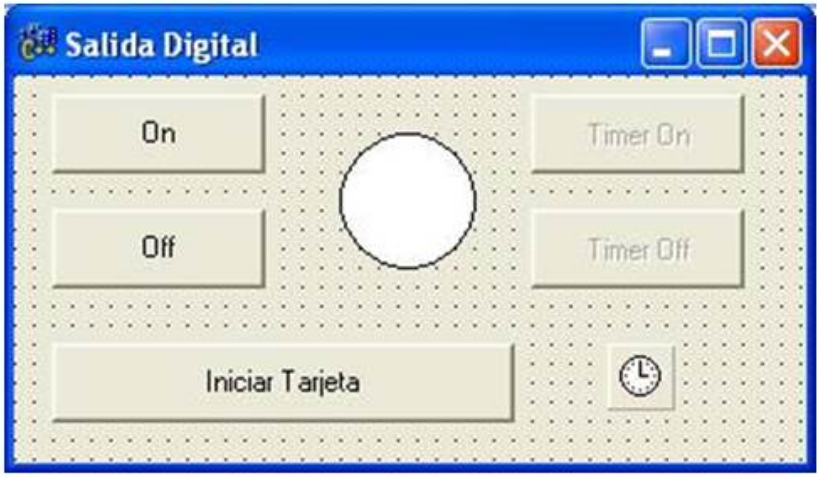
```
int32 daq_error;
   // apply to each DAQ function call
   daq_error=DAQmxCreateTask(…
   if(daq_error != 0) {
       printf("Problems!!!");
       exit(1);
   }
```

Lab: Digital output

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,



Lab: Digital output

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.

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

```
voit process_WriteActuatorHeater {TStatusHeater status);
```

That follow this logic

"1" means OFF
"0" means ON

Thanks!


UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA
www.upv.es