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 2: Computer Architecture

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

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

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 2: Computer Architecture

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

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

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 content of this package follows 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 describes the computer architecture from the point of view of the different units: central processing unit, memory and input/output.

The desired approach is to address concepts from a single view and eliminating the complexity of details and commercial models.

Moreover also addresses the use and programming of libraries with their application to informatics industrial systems. These libraries are both addressed when they are static or when are dynamic.

Besides is presented the formal specification of the mini-project that will be developed among the course.

Finally some bibliography is proposed to expand and deepen concepts in this area.

3 Lecture

3.1.1 Objectives

- To know the philosophy and structure of microprocessor based systems
- To understand the different Functional Units.

- To understand the concept of Input/Output and how to use it

3.1.2 Introduction

The Industrial Informatics Systems are based on the concept of "**Programmable Machine**" that gives the flexibility for the adaptation of the system to a different industrial process and situations. The "Programmable Machine" is a general concept which has led to other terms as Computer or Informatics.

The great advantage of the "Programmable Machine" is the possibility that a program can manage the device or devices and only changing the program the device can have another behavior. The physical part is called hardware, while the program and data is called software.

Figure 1 shows a scheme for the integration of the software and hardware into the programmable machine.

Figure 1: Components of a programmable machine

3.1.3 Functional Units of the Computer

Most of the computers used in industrial informatics systems used the structure shown in Figure 2, they are based on 4 functional units:

- Central Process Unit (CPU)
- Memory
- Input/Output

For the interconnection of the different functional units are used the buses.

The CPU is like the brain and controls the system by the interpretation of the programs that uses the data. Both, programs and data are located into the memory. The data can enter or leave the computer by using the input/output unit. This unit is also in charge to control the different peripherals connected into the system. The buses are the way to circulate the information among the different units.

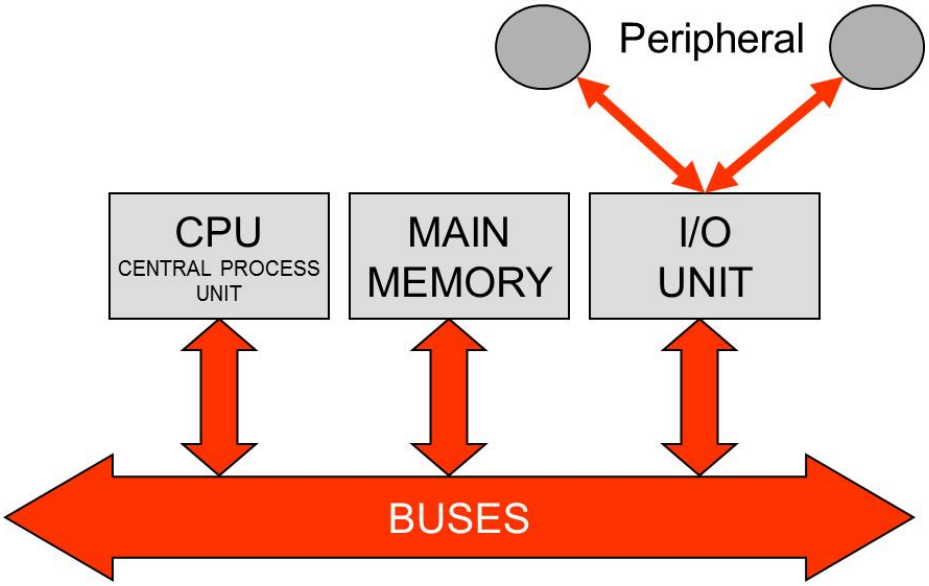

Figure 2: Computer Structure

All the information processed by a computer is in a digital format using binary digits o bits. Bit can only be “0” or “1”, thus the programs and data on which the computer is working will be a sequence of numbers composed by a combination of “0” and “1”. The reason that computers work with “0” and “1” is because it is easy to manufacture electronic circuits representing different values of voltages to each of these figures. For example, can be usually common represent with “1” the value of +5 Volts, and with “0” the value of 0 Volts (GND).

3.1.4 The Memory Unit

The memory is the unit in charge of store all the information used by the system. Inside this unit are stored all the instructions and data necessary to control a particular process.

It can be distinguish between two of the most important memory types inside the computer:

- External or secondary memory: is permanent (keeps the information even without power) and stores all the programs and data that are not being used at any given time. It is used to store large amounts of programs/data those who do not need continuous access. Examples of this type of memory are hard disks, CD-ROM, DVD, USB disc, etc.
- Central or main memory: also is called simply as "memory", contains the programs and data that are being used at a given time. In this section we will refer to this type of memory

The main memory is the unit to store binary information in electronic semiconductor cells. Each cell can store the equivalent to a one bit, so the content may be exclusively “0” or “1”. It is usual that the cells are organized in groups distinguished among them by their **position or memory address.**

As indicated, the memory will be used to store information, which means that can save (*write*) and retrieve (*read*) a binary data into a specific position of memory, and the number of bits of this binary data corresponds with the size of each group of cells.

To use the memory will be needed:

- The memory position (**Address**) to which access is needed, which is a binary number. For example, if we have a memory with 8 positions, (with 3 bits can be addressed 8 binary numbers, $2^3 = 8$) a number will be used between 000_2 and 111_2 to access any of the positions.
- Indicate the memory if the operation is for **read** or **write** information. For example, the operation of read can have associated the bit "1" and "0" for write.
- A signal for the activation of the memory (**Chip Select, CS**) to let it know we want to work with it. In the section on buses will expand about it.
- Also the data (program instruction or data information) will be needed.

The Figure 3 shows a simplified scheme for a four positions memory and one byte of storing capacity per position, in which there is illustrated how to read the position 1. For this it is necessary to put "01_2" (address number 1) in the address entry, indicate with "1" in the entry of Read/Write (R/W) that we need to perform a read and activate the memory. After a time which is called access time, the data will appear in the data input/output lines. Should be noted that in most of the memories the data enter and exit through the same lines.

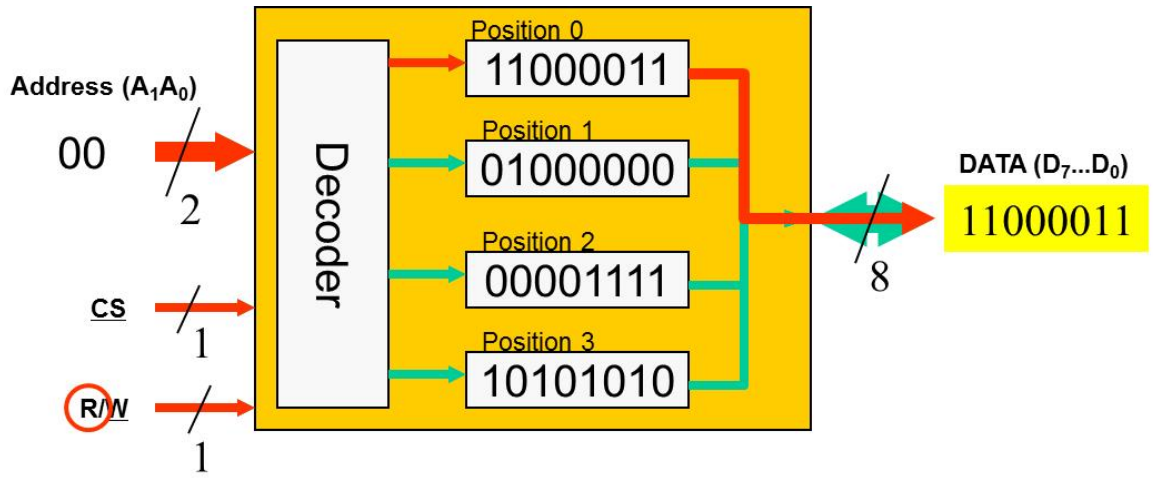

Figure 3: Logic scheme of a memory

The content of a memory can be represented using a memory map, for this is used a matrix with the number of rows equivalent to the addresses and in each row containing the bits of each memory position. For example Figure 4 shows the table for a memory of N positions (address from 0 to N-1) and 8 bits per address.

Address	b7	b6	b5	b4	b3	b2	b1	b0
0	1	0	1	1	1	1	0	1
1	0	1	1	1	0	0	0	1
...								
N-1	0	0	0	0	1	1	1	1

Figure 4: Example of a N positions Memory and 8 bits per address

The capacity of a memory usually is given in the notation *PxB*, where *P* is the number of memory positions and *B* is the number of bits per position. For example, a memory of 2048x8 has 2048 positions and each has can store 8 bits. Also it is possible indicate the amount of bytes in memory usually expressed in multiples of $2^{10.}$ Table 1 summarizes the most common multiples:

1 Kbyte	1.024 bytes
1 Mbyte	1.024 Kbytes = 1.048.576 bytes
1 Gbyte	1.024 Mbytes = 1.048.576 Kbytes = 1.073.741.824 bytes
1 Tbyte	1.024 Gbytes = 1.048.576 Mbytes = 1.073.741.824 Kbytes= 1.099.511.627.776 bytes

Table 1: Common multiples of the amount of memory

By the way of storing the information are two main groups of memory:

- ***ROM*** (Read Only Memory): The information only can be read and it is not possible to modify the information because it is out of the system previously stored. The information remains stored even if there is no power supply (Nonvolatile Memory). The programs stored permanently in ROM are called *Firmware*
- ***RAM*** (Random Access Memory): The information can be written or read as many times as the system may need it. The information stored is lost when the memory is no longer with power supply (Volatile Memory)

3.1.5 The Central Process Unit

The Central Process Unit (also called CPU, Central Processor, and Microprocessor) is the real brains of the system. It mission is to control, coordinate and perform all the system operations following a defined sequence. Extracts program instructions from the memory and analyzes in order to run that the instruction indicates. The instructions that can be executed by a processor are very simple. Figure 5 shows an example of what may be an instruction (in assembler language: MOV AL,12h)

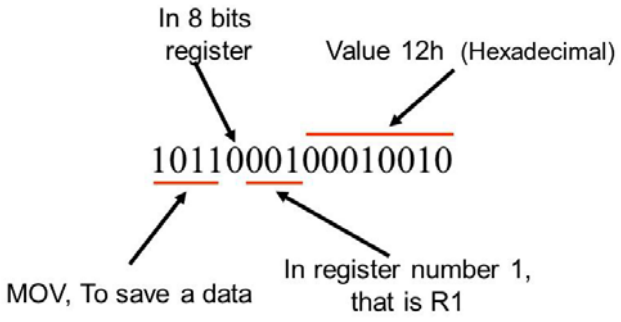

Figure 5: Codification of the assembler instruction MOV AL,12h

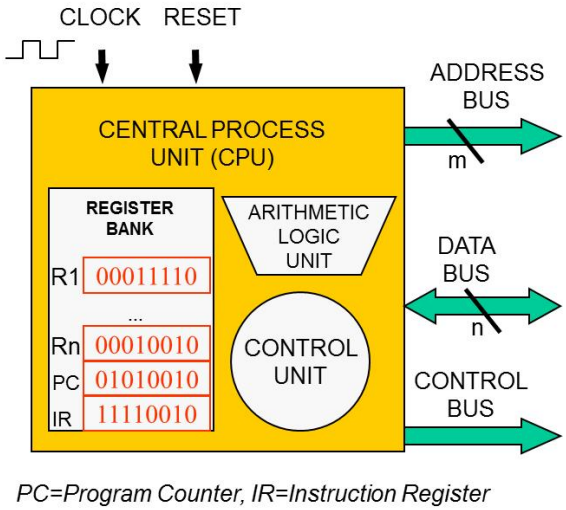

Figure 6: CPU structure

Figure 6 shows the basic CPU structure with tree main parts: **Control Unit**, **Arithmetic-Logic Unit** (ALU) and **Register Bank**.

The **Control CPU Unit** generates the signals necessary for the execution of instructions and from it control and govern all the operations in an organized way according to a fixed automatic behavior such as shown in Figure 7:

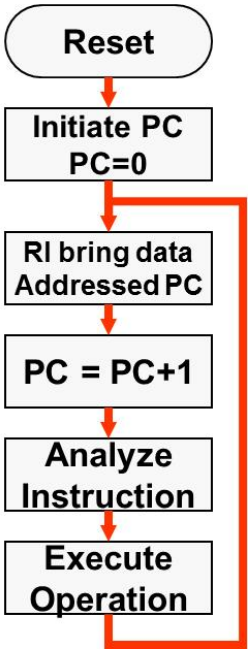

Figure 7: Instruction Execution Cycle

The **Arithmetic-Logic Unit** (ALU) is responsible for performing mathematical operation among data. The basic operations that can perform are: add, subtract, multiply and divide between integer numbers of the register size. Also, logic operations as: AND, OR, NOT, Comparisons, etc. The Control Unit is

what makes getting the data to the ALU and carries the result to their destination. Figure 8 shows a simplified diagram of the ALU.

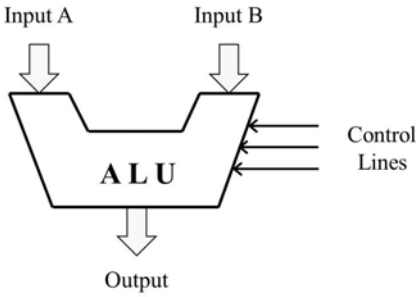

Figure 8: Arithmetic-Logic Unit

The **Register Bank** is a small memory that usually operates the ALU and transfer data between the Main Memory and the CPU. Each register is identified by a special name and stores one data. Remarkable registers are the **PC** register (Program Counter) that stores a data which is the number that the Control Unit uses as the memory address in order to localize the next instruction; the **IR** register (Instruction Register) that stores the instruction/data that is interpreting (executing). Also there are some general purpose registers (**Rn...R0**).

The CPU works with a rate set by a square signal called **clock** (CLK). Faster clock will be higher instruction execution speed. As the CPU is governed by an automaton that is the Control Unit, it is essential to ensure an initial state of the automaton. This is achieved by the **Reset** signal which, among other things, starts the PC register, for example setting it to 0.

The CPU gets the instructions and data and governs the whole system using the buses to connect to the other functional units.

3.1.6 The Input/Output Unit

A computer without the possibility to be connected outside does not have much use. The Input/Output (I/O) unit has the mechanism that allows the system the connection with the **peripheral devices**. The I/O can receive/send information from/to the different devices as hard disks, USB disks, graphical cards, printers, plotters, network cards, measurement instruments, oscilloscopes, data acquisition cards, etc.

From our point of view is important to know the mechanisms used to transfer information using I/O, because through it will be accessed to the different characteristic providing such the data acquisition cards.

The access to the I/O devices is by using their address, similarly as done with the memories, called in this case Input/Output **ports**. There are two forms of present this addresses to the CPU:

- To locate the devices in specific positions of the address space of the main memory, so to read or write a device will be the same as read or write in a specific memory position.
- To have two address spaces, one for the main memory and the other one for I/O. In this case a special signal will be needed, with this signal the CPU can know if the access is to the memory or to I/O. Also the CPU needs a specific set of instructions to access I/O.

Figure 9 shows the different necessary elements to use peripheral I/O devices:

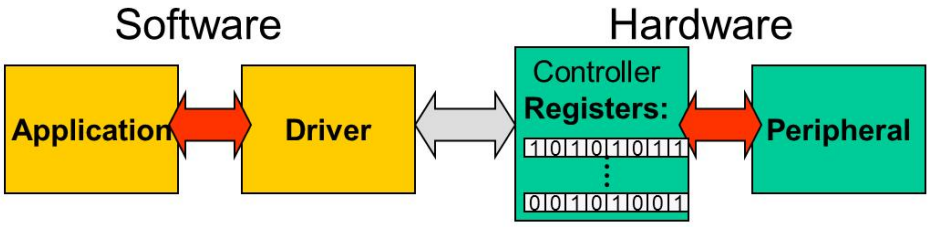

Figure 9: Elements involved in the I/O system

A **device controller** is needed which is a hardware that really connects the peripheral. The CPU can access to this controller using their ports, corresponding the ports with the controller registers (similar to the CPU registers).

Because of the complexity of the devices are designed specific software called **device manager** or **driver** and is in charge of the device management through the corresponding controllers. Each device has their own driver with the main function is to be responsible for receiving requests from "higher levels" and turn them into a series of commands to the device driver. For example, if a program requests to the disk driver some of the information stored in the USB disk, the driver will verify that the USB disk is in working conditions, and will start with all the commands in order to read the information.

The variety and complexity of I/O devices necessitates a number of mechanisms for management. Considering that the devices through digital information (binary numbers) flows and the CPU is the coordinator of all, there are two points to note in this transfer of information. The first one is how the CPU knows when the information should be gathered by the I/O? This is called **synchronization**. The second question is how to perform the data transfer? This is called **data transfer.**

3.1.7 Input/Output Synchronization techniques

The synchronization is the technique that permits to know if a specific I/O device needs the CPU services to send/receive information. For the synchronization there are two main techniques:

- **Polling**: is a status inquiry that CPU read specific I/O registers to know if there are information ready to be transferred. This technique is inefficient and is used in slow devices that do not need too much attention. Figure 10 shows how polling will work for the attention of tree peripheral:

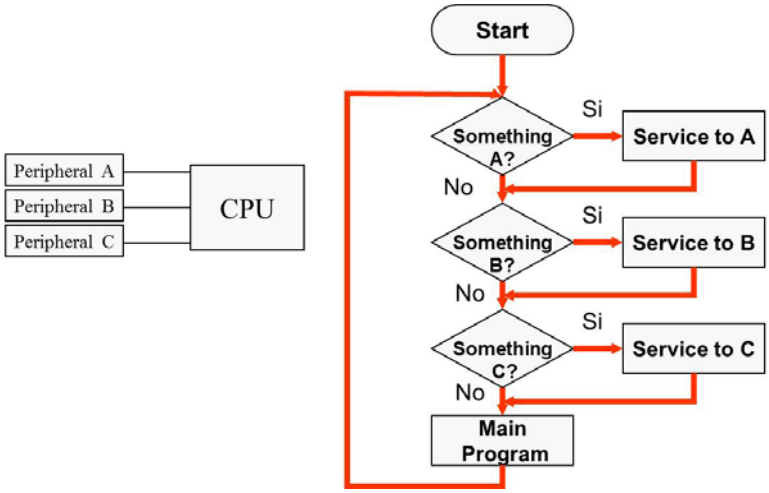

Figure 10: Polling between CPU and three Peripherals

- **Interruptions**: When a peripheral requires the CPU service causes an interruption and the CPU leaves the current program execution and serves the peripheral. This technique allows the CPU only invests time on a device when requested. When an interrupt occurs, the CPU saves the state of the current program running, and starts the execution of a program that manages the interrupt and called Interrupt Service Routine or Service to the Interruption. Once the service to the interruption finish, the CPU continues with the execution of the program, Figure 11 shows this behavior

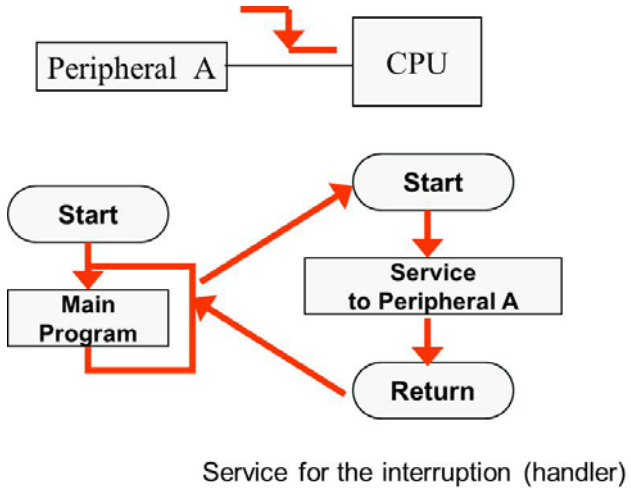

Figure 11: Synchronization by interruption

3.1.8 Techniques for data transfer

For the transfer of information there are two main techniques:

- The **transfer using a program**, based on continuous read/write into the data registers until the transference finish (synchronization by Polling). Normally a loop is used and not allows or difficult for the CPU to do other tasks (busy waiting)
- The **Direct Memory Access (DMA)** that uses special circuits capable of transferring blocks of data from/to memory to/from devices directly, without CPU intervention. The CPU only programs the DMA to indicate how much data has to be transferred and where to place it. Suitable for devices that need to transfer a lot of information in a short time. For example the hard disk, network card, video card, etc. Figure 12 shows the relation between DMA system and the rest of functional units in the computer.

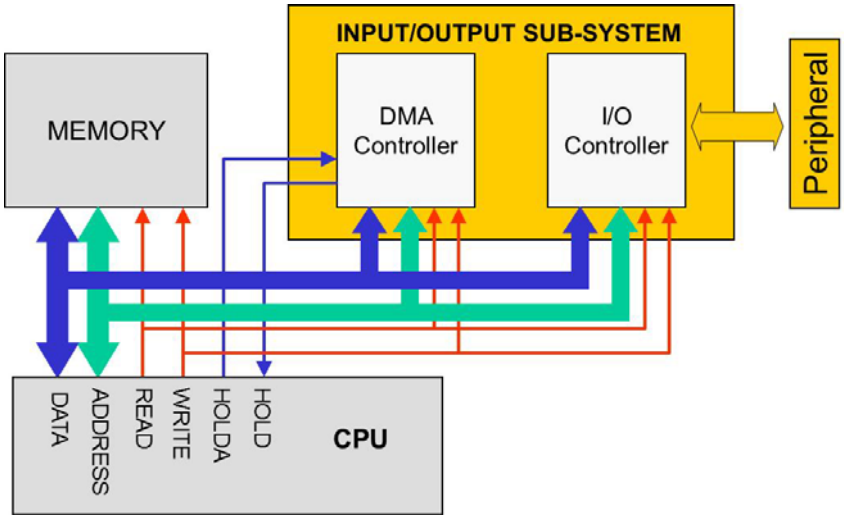

Figure 12: Direct Memory Access (DMA) System

3.1.9 The Buses

In the computer the buses are not a functional unit, buses are a set of electric lines that interconnect the different functional units (CPU with Memory and Input/Output) by transferring them parallel binary signals according to certain protocol, being often the CPU that decides who and how used.

The Buses can by classified in tree groups:

- The **Data Bus (D_n...D_0)**, along which data are transferred between the different elements. Thus, eight electric lines allow the transfer of one byte; sixteen lines allow the transfer of a 16 bits (two bytes) data. The higher the "bus width" (number of lines) greater amount of information can be transfer but the complexity on the interconnection elements will be higher
- The **Address Bus (A_m...A_0)**, along which binary numbers are transferred that represent one memory position or an I/O port. The total number of lines of this bus indicates the maximum number of addressable memory. For example with 16 lines (A15...A0) a total of 65.536 locations can be addressed (2^{16}= 65536=64K).
- The **Control Bus**, is the lines that allows the connected elements indicate among other things, if is needed a reading or writing, and if the memory references are made to the memory or I/O system, temporizations, etc. We did not see this bus as a whole, but each individual line has it specific function. For example the lines that can be in the control bus are:
  - Clock (CLK)
  - Memory and I/O Control (R/W, MEMRQ, IORQ)
  - Interrupts (INT, NMI)
  - Reset (RST)
  - Power supply (Vcc, GND)
  - Direct memory access (DMA, HOLD, HOLDA)
  - Hold (HLD)
  - Etc.

4 Lab

4.1 Identification of the Functional Units

In this lab the main objective is to identify the different functional units of the PC used at laboratory. This will allow us to apply the theoretical concepts covered in the previous section. We intend to see general concepts that apply to any business model PC

This first thing to do is open and removes the lid of the PC using the right tools and screwdrivers. The PC will be in advance disconnected from the power supply and all the wires removed.

Then we will see the **mainboard**: This is where the processor, RAM card, video card, network card, sound card, etc. are assembled. Some mainboards have built some of the cards such as video, sound and network. The mainboard must be compatible with the processor model that will be used, the speed of

the RAM cards or the interface of the video card that will be used. Figure 13 shows an example of mainboard.

Figure 13: ASUS mainboard

With the open PC and the teacher's help, we will identify the different functional units, after having identified each of the functional units will take a photo that will attach to the practice bulletin. Additionally, we will identify the reference of each of the following devices subsequently on internet we will look for technical characteristics that will also include to the practice report:

- The Central Process Unit (CPU): The CPU is based on a chip called microprocessor is the brain of the PC. We will identify this chip on the mainboard with their commercial reference. This microprocessor is responsible for processing all information both program and data. Today there are processors that have more than one processing core and this result in better performance when performing multiple tasks at once. The processor speeds are now measured in Gigahertz and indicate the frequency of the signal from the master of the central processing unit clock. The faster the computer processor, the faster it will respond to any request process. Figure 14 shows an example of a AMD microprocessor.

Figure 14: AMD Microprocessor

- The Main Memory: It plays a key role in establishing the operating speed of the PC. In the **RAM** are all the programs and data that are currently in use. If the computer does not have enough RAM to store all programs that run the computer at one time, the system starts to use the hard disk (HD) to temporarily load the programs to be run, which results in a slower in the reaction of the programs. The amount of RAM that can be installed into the computer is limited by the capacity that can hold the mainboard and its use is limited by the operating system installed. The RAM comes in different speeds which are measured in MHz. The faster the speed of the faster RAM can change the execution between programs, i.e., someone may be using a Web browser and quickly switch to editor images without altering its performance and responsiveness. Figure 15 shows an example of a commercial RAM.

Figure 15: Kingston RAM

- The **BIOS** (Basic Input-Output System): It is a small chip built into the mainboard. Its purpose is to maintain some basic information on computer startup. This information can be setup on your hard drive, date time of system boot priority, boot from the network etc. When it is installed a new disk drive or CD-ROM stores the BIOS settings charge it after in RAM at boot the computer, so in these cases it is advisable to access the BIOS and check that it has been correctly recognized. Figure 16 shows an example of a commercial BIOS

Figure 16: Vinbond BIOS

- In the Input/Output Unit all peripherals are included. Identify the following with its commercial model and consult its features using internet. Also include a picture in lab report:
    - **Hard Disk** (HD): This is where the files, programs and computer operating system are stored. Hard drives come in different capacities; the larger capacity hard drive has more files it can be stored. Today you can find several TB hard drives capacity. Also, HD has different interfaces, which means that there are hard disks with different connectors that are integrated into the mainboard, these interfaces can be: IDE, SATA, SAS, SCSI, FC and USB. The data transfer rate of a hard disk depends on its interface, type and speed of rotation. There are two types of hard disk, traditional mechanicals that are made within several discs that rotate and store information and SSDs using SSD or state non-volatile memory such as flash memory. Figure 17 shows an example of a commercial mechanical HD.

Figure 17: Hitachi HD

- **Video Card**: Has a microchip video processing and video memory. To choose the best video card it is necessary to know the characteristics and performance of video chips and also know the type and amount of video memory that comes with the card. Figure 18 shows an example of a commercial video card

Figure 18: NVIDIA Video Card

- **Sound Card**: Most mainboards now have integrated a sound card, but if it's needed more functionality and power always is the possibility to add an extra sound card features. Sound cards may have different interfaces, which mean that they can be integrated into the computer in different ways. These interfaces include: PCI, ISA, PCMCIA or USB. Figure 19 shows an example of a sound card

Figure 19: Sound Card

- **Network Card**: Currently the network card is already built into most mainboards sold in the market. This card allows connecting a network cable with RJ-45 to another network device such as a router. Network cards can also be wireless and they have different kinds. These cards are known as network cards WI-FI and the class depends on the communication standard support. These standards are: IEEE 802.11b, IEEE 802.11g and IEEE 802.11n each offering different data transfer rates ranging from 11Mbps, 300Mbps and 54Mpbs respectively. So also increase your range of action. Figure 20 shows an example of a network card

Figure 20: Network Card

- **Power Supply**: It is a metal box that goes inside the case and is responsible for distributing the electrical power system components used. The power supply is vital as it should have enough power, stability and reliability to power the electronic

components of the computer. Some damage to the internal components of the computer are due to a bad power supply, so it is best to invest in a good power source any brand recognized expertise that has enough power to support all devices that incorporated in the system. Figure 21 shows an example of a power supply. Please, identify the power used in the power supply of your PC lab.

Figure 21: Power Supply

- The **Expansion Slots**: They are the slots where the different cards are connected in the system, such as video, audio, network. Slots are the most common:
  - ISA are ancient but nowadays not used.
  - PCI are common in today's computers.
  - AGP is for the video card.

Figure 22: Example of Expansion Slots

As noted above, laboratory activity will be to identify in the PC laboratory the elements described above, with their commercial reference and accessing the internet for its technical specifications. Also, for each component take a picture that will be incorporated in the lab report.

4.2 Using Libraries in C

Libraries are "packages" or collections of classes and functions already defined in the language, which provides solutions to common problems and generally require generic actions, for example, the function *sqrt* in C language calculates the square root of a number this function is useful for performing mathematical calculations and can be used according to the application being developed or the purpose that is sought (hence the word generic). This function is in the library "*match.h*" which has many more useful features like: *pow*, *sin*, *floor*, among other.

When we are programming in C language we can find **static libraries** that are already installed into the programming environment itself and usually are defined in the basic language and included in standards such as ISO C90. Some examples are:

stdio.h: Used for standard input/standard output

complex.h: Used for complex numbers

math.h: Contains the common mathematical functions

string.h: For manipulating strings

time.h: For treatment and conversion between formats of date and time

...

A **dynamic link library** (*DLL*) is an executable file that acts as a shared library of functions. Dynamic linking provides a way for processes to call a function that is not part of the executable code. The executable code for the function is in a *DLL*, which contains one or more functions that are compiled, linked, and stored separately from the processes that use them. *DLLs* also facilitate the sharing of data and resources. Different applications can simultaneously access the contents of a single copy of a *DLL* in memory.

Dynamic linking differs from static linking in that it allows an executable module (either a .dll or .exe file) includes only the information needed to locate the executable code of a *DLL* function at runtime. In static linking, the linker gets all the functions that are referenced from the static link library and places them into executable code.

The use of dynamic linking, instead of static linking, offers several advantages. DLLs save memory, reduce page flipping, save disk space, easy upgrades, provide post-sales support, providing a mechanism to extend the MFC library classes, provide support multilanguage programs and facilitate the creation of international versions.

As an example we will conduct a practical exercise on the use of a dynamic library in the environment of the Qt. To do this, we are going to use the library provided by National Instruments for to use the *data acquisition card NI USB6008*

First we are going to execute Qt application and create a new project (Figure 23):

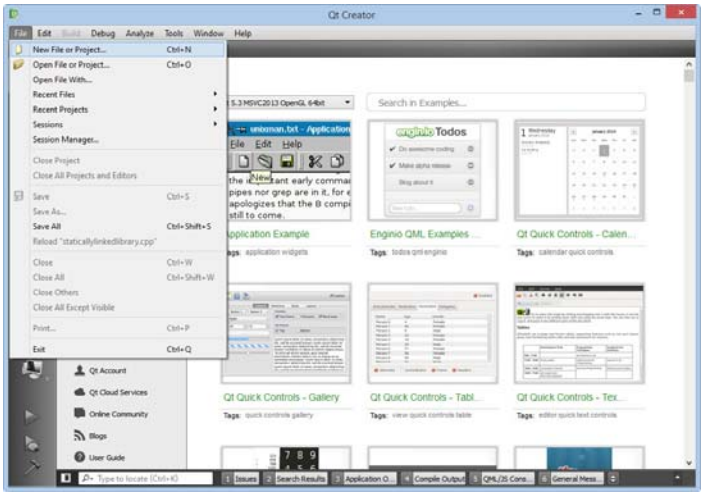

Figure 23: New project on Qt Creator

On the next screen we are going to select in the "*Applications*" the "*Qt Console Application*" as shown in Figure 24:

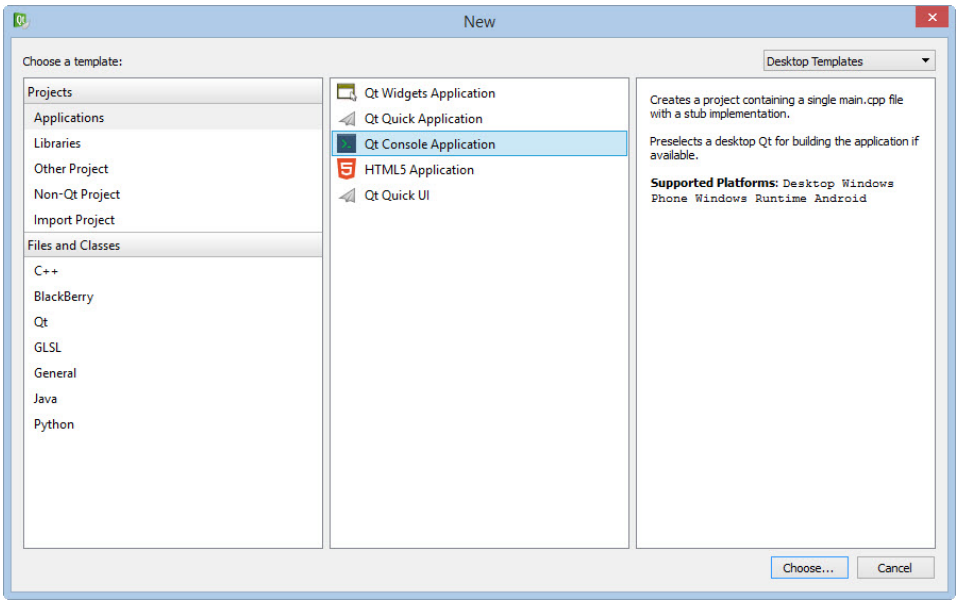

Figure 24: Selection of Qt Console Application

In the next step we are going to select the name and the project location (Figure 25)

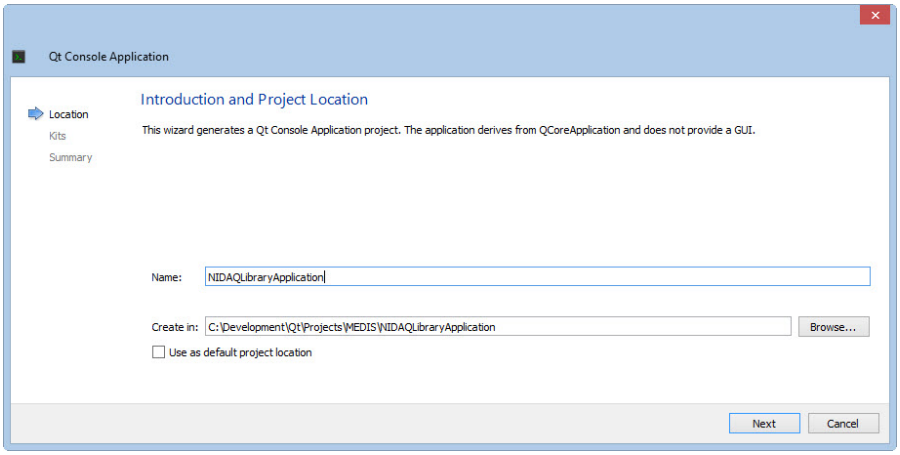

Figure 25: Introduction and Project Location

Concerning the kit selection we will choose a console application: "*Desktop Qt 5.3 MinGW 32 bit*", the appropriate option for our operating system (Figure 26):

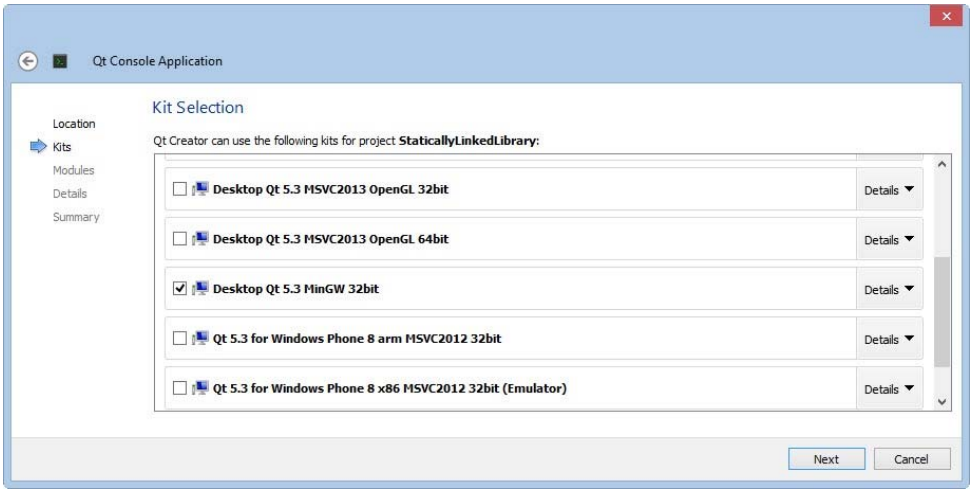

Figure 26: Kit Selection

The next screen is for entering information concerning project management. In our case we select "*None*" as shown in Figure 27:

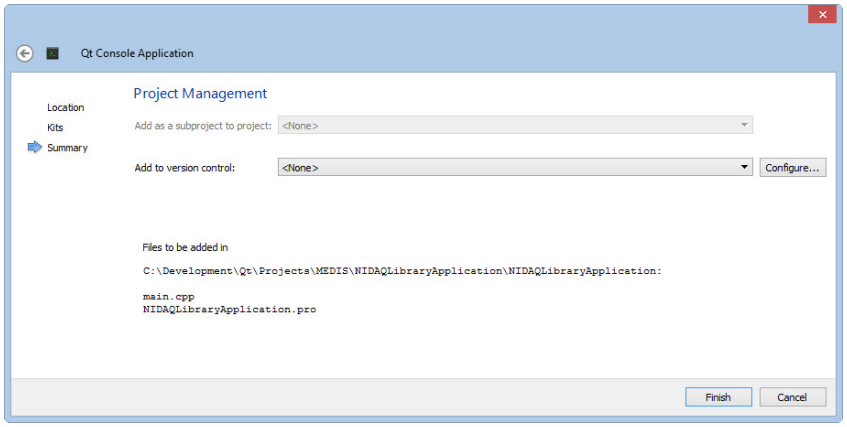

Figure 27: Project Management

Now we have the Template generated (Figure 28):

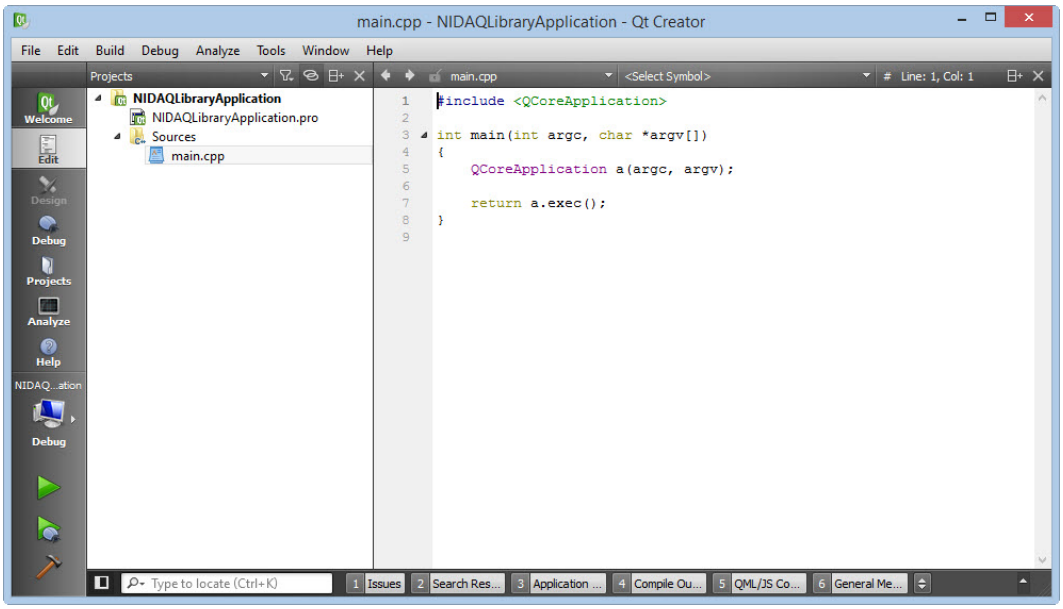

Figure 28: Template generated *main.cpp*

We can inspect the content of the *".pro"* (Figure 29):

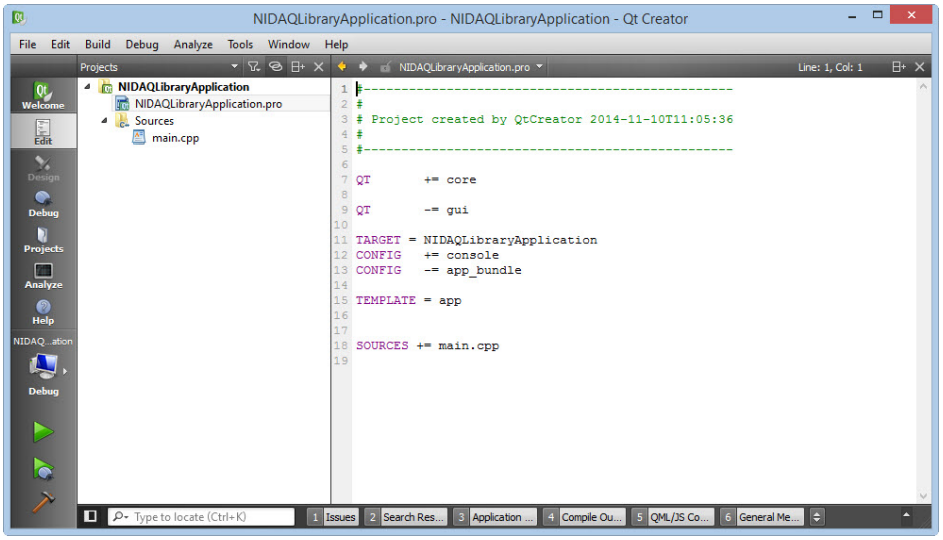

Figure 29: Template *.pro*

With the right button we can get the popup menu and add the library as shown in Figure 30:

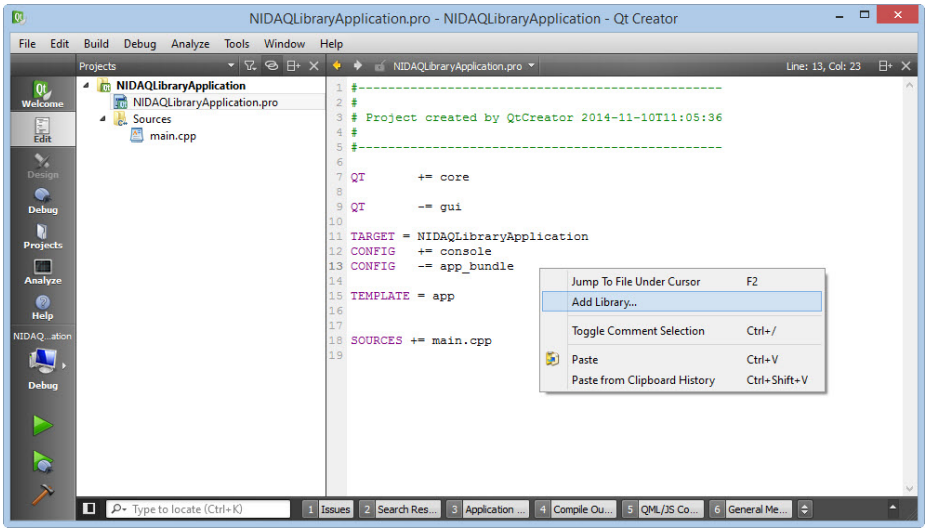

Figure 30: How to add a Library

In this case we are going to use an "*External Library*" (Figure 31):

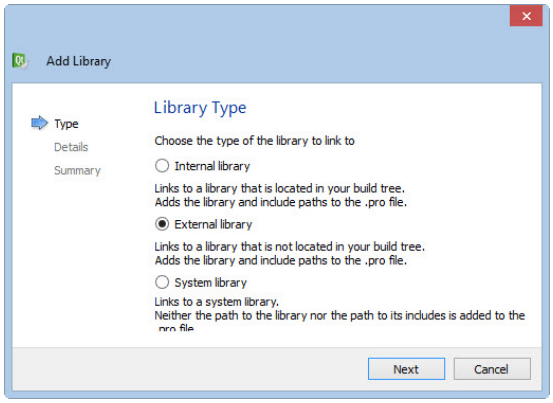

Figure 31: Using External Library

After that it is necessary to include information for the localization of the library (Figure 32):

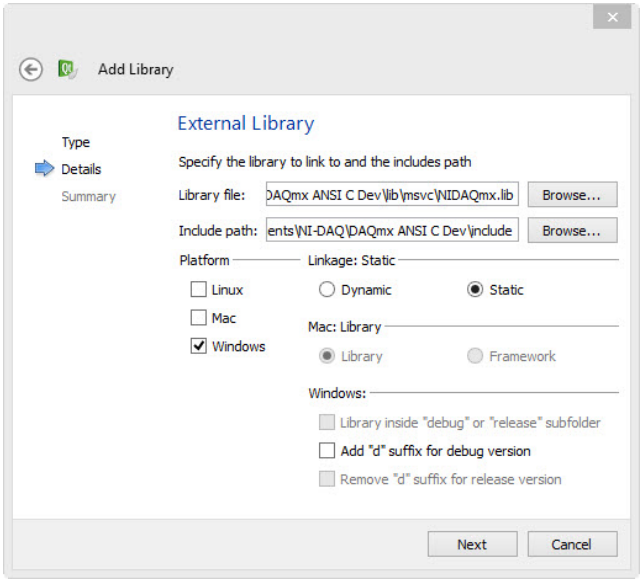

Figure 32: External Library Information

If we select the option "Summary" the Qt will show information about the library (Figure 33):

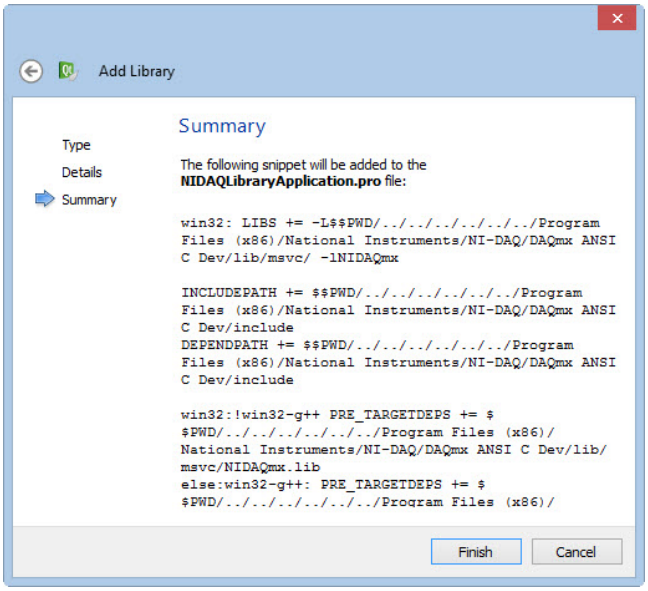

Figure 33: Summary of the Library

Now we can see that the information related to the new library is included in our "*.pro*" template (Figure 34):

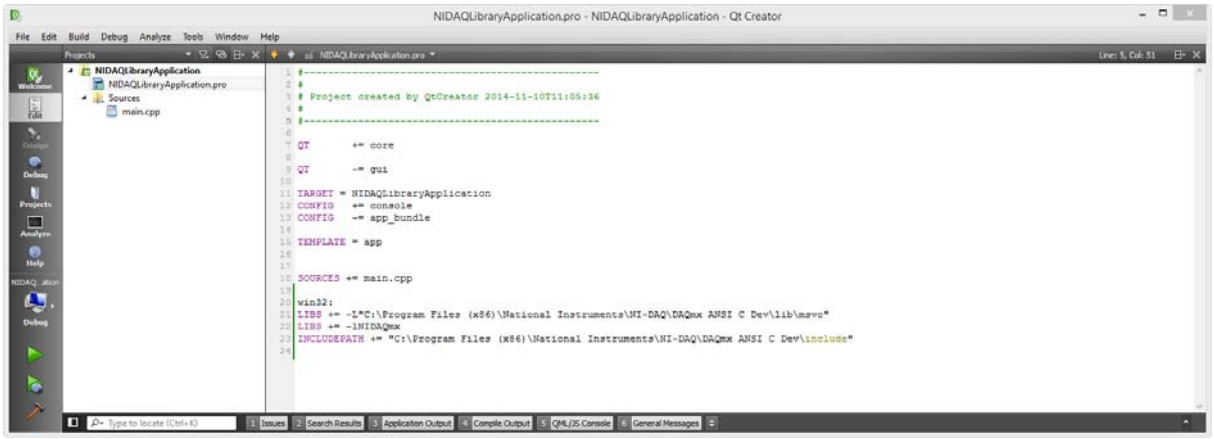

Figure 34: Template *.pro*

Everything is ready now to compile and build the application (Figure 35):

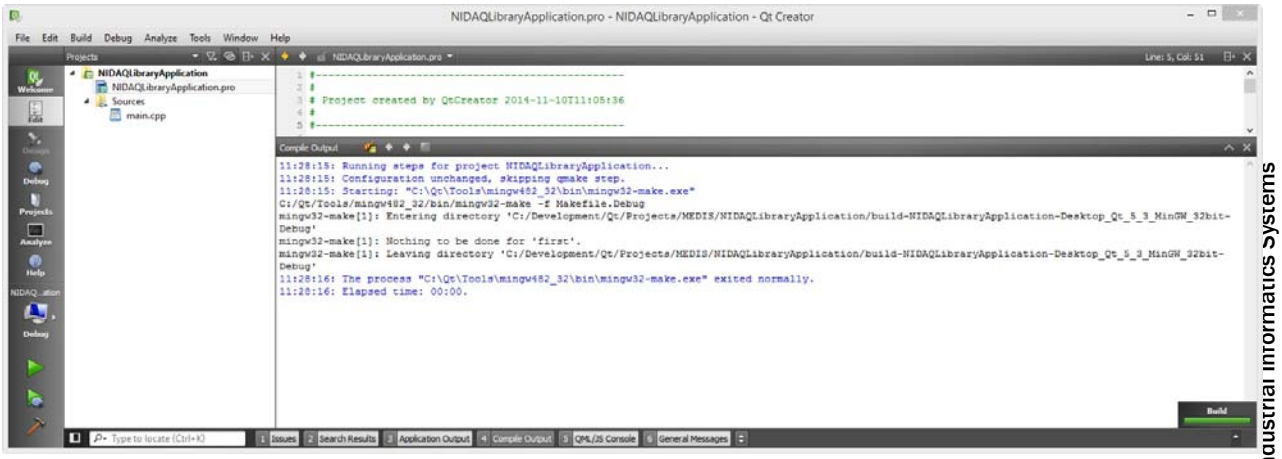

Figure 35: Compile Output

After all this, we can now develop applications that use the functions offered by the library (Figure 36)

Figure 36: Example using the library

5 Seminar

5.1 Computer Architecture

This seminar will be to work on business models used to develop PC-based industrial systems using **Industrial PCs**. With internet we will collect the following information:

- Some companies that sell industrial PC.
- List of some business models of Industrial PCs of each of the brands
- Main features of each proposed model (i.e.: cost, computing power, memory, interconnectivity, cooling characteristics, insulation, etc.).
- Web address of the corresponding datasheet of each proposed industrial PC.
- Comparative review of each of the models.

- Model would be chosen as the most suitable for the realization of an industrial computer system medium duty.

5.2 C Programming (3). Libraries

In this part of the seminar we are going to work on the programation and use of an example of static library in the Qt environment. We will use the Qt environment to develop a library that allows us to read a generic sensor. The sensor works by providing a proportional voltage to the measured physical magnitude. The sensor shall have a transfer function that allows us to obtain the value of the physical magnitude in terms of volts delivered by the sensor itself. An example of the generated c++ code to create this static library in Qt for a given generic sensor can be:

```
#include <math.h>
#include "staticallylinkedlibrary.h"

StaticallyLinkedLibrary::StaticallyLinkedLibrary()
{
    NumberOfBits = 12;
    MinVoltage   = 0.0;
    MaxVoltage   = 5.0;
    MinMagnitude = 10.0;
    MaxMagnitude = 60.0;
}
bool StaticallyLinkedLibrary::SetNumberOfBits(unsigned char num_bits)
{
    if((num_bits < MIN_NUM_BITS) || (num_bits > MAX_NUM_BITS))
    {
        return false;
    }
    NumberOfBits = num_bits;
    return true;
}
bool StaticallyLinkedLibrary::SetVoltage(double min_voltage, double
max_voltage)
{
```

```
    if(min_voltage > (max_voltage - MIN_DIFF_VOLTAGE))
    {
        return false;
    }
    MinVoltage = min_voltage;
    MaxVoltage = max_voltage;
    return true;
}
bool StaticallyLinkedLibrary::SetMinVoltage(double min_voltage)
{
    if(min_voltage > (MaxVoltage - MIN_DIFF_VOLTAGE))
    {
        return false;
    }
    MinVoltage = min_voltage;
    return true;
}
bool StaticallyLinkedLibrary::SetMaxVoltage(double max_voltage)
{
    if(max_voltage < (MinVoltage + MIN_DIFF_VOLTAGE))
    {
        return false;
    }
    MaxVoltage = max_voltage;
    return true;
}
bool StaticallyLinkedLibrary::SetMagnitude(double min_magnitude, double
max_magnitude)
{
    if(min_magnitude > (max_magnitude - MIN_DIFF_MAGNITUDE))
    {
        return false;
    }
```

```
    MinMagnitude = min_magnitude;
    MaxMagnitude = max_magnitude;
    return true;
}
bool StaticallyLinkedLibrary::SetMinMagnitude(double min_magnitude)
{
    if(min_magnitude > (MaxVoltage - MIN_DIFF_MAGNITUDE))
    {
        return false;
    }
    MinMagnitude = min_magnitude;
    return true;
}
bool StaticallyLinkedLibrary::SetMaxMagnitude(double max_magnitude)
{
    if(max_magnitude < (MinVoltage + MIN_DIFF_MAGNITUDE))
    {
        return false;
    }
    MaxMagnitude = max_magnitude;
    return true;
}
unsigned int StaticallyLinkedLibrary::GetNumBits(void)
{
    return NumberOfBits;
}
double StaticallyLinkedLibrary::GetMinVoltage(void)
{
    return MinVoltage;
}

double StaticallyLinkedLibrary::GetMaxVoltage(void)
{
```

```
    return MaxVoltage;
}
double StaticallyLinkedLibrary::GetMinMagnitude(void)
{
    return MinMagnitude;
}
double StaticallyLinkedLibrary::GetMaxMagnitude(void)
{
    return MaxMagnitude;
}
double StaticallyLinkedLibrary::MagnitudeForQuantification(unsigned int
quantification)
{
    return MinMagnitude + (MaxMagnitude - MinMagnitude) *
(VoltageForQuantification(quantification) - MinVoltage) / (MaxVoltage -
MinVoltage);
}
double StaticallyLinkedLibrary::MagnitudeForVoltage(double voltage)
{
    return MinMagnitude + (MaxMagnitude - MinMagnitude) * (voltage -
MinVoltage) / (MaxVoltage - MinVoltage);
}
double StaticallyLinkedLibrary::VoltageForQuantification(unsigned int
quantification)
{
    return MinVoltage + (MaxVoltage - MinVoltage) * quantification / pow(2,
NumberOfBits);
}
double StaticallyLinkedLibrary::VoltageForMagnitude(double magnitude)
{
    return MinVoltage + (MaxVoltage - MinVoltage) * (magnitude -
MinMagnitude) / (MaxMagnitude - MinMagnitude);
}
unsigned int StaticallyLinkedLibrary::QuantificationForVoltage(double
voltage)
{
```

```
    return (voltage - MinVoltage) * pow(2, NumberOfBits) / (MaxVoltage -
MinVoltage);

}

unsigned int StaticallyLinkedLibrary::QuantificationForMagnitude(double
magnitude)

{

    return(VoltageForMagnitude(magnitude)-
MinVoltage)*pow(2,NumberOfBits)/(MaxVoltage-MinVoltage);
```

Concerning the code that will be included on the header file (.h):

```
#ifndef STATICALLYLINKEDLIBRARY_H

#define STATICALLYLINKEDLIBRARY_H


#define MIN_NUM_BITS       1

#define MAX_NUM_BITS       16

#define MIN_DIFF_VOLTAGE   0.1

#define MIN_DIFF_MAGNITUDE 0.1


class StaticallyLinkedLibrary

{

public:


    StaticallyLinkedLibrary();


    bool SetNumberOfBits(unsigned char num_bits);

    bool SetVoltage     (double min_voltage, double max_voltage);

    bool SetMinVoltage  (double min_voltage);

    bool SetMaxVoltage  (double max_voltage);

    bool SetMagnitude   (double min_magnitude, double max_magnitude);

    bool SetMinMagnitude(double min_magnitude);

    bool SetMaxMagnitude(double max_magnitude);


    unsigned int GetNumBits     (void);
```

```
    double       GetMinVoltage  (void);
    double       GetMaxVoltage  (void);
    double       GetMinMagnitude(void);
    double       GetMaxMagnitude(void);

    double MagnitudeForQuantification(unsigned int quantification);
    double MagnitudeForVoltage       (double voltage);

    double VoltageForQuantification(unsigned int quantification);
    double VoltageForMagnitude     (double magnitude);

    unsigned int QuantificationForVoltage  (double voltage);
    unsigned int QuantificationForMagnitude(double magnitude);

private:
    unsigned char NumberOfBits;
    double MinVoltage, MaxVoltage, MinMagnitude, MaxMagnitude;

};

#endif // STATICALLYLINKEDLIBRARY_H
```

To better understand the operation of creating static libraries, then first we propose to analyze the sample of library code (*.cpp* and *.h*) shown and to understand its functionality and second to proceed with the implementation in Qt.

First, execute Qt application and create a new project as shown in Figure 37 (*File -> New File or Project...*):

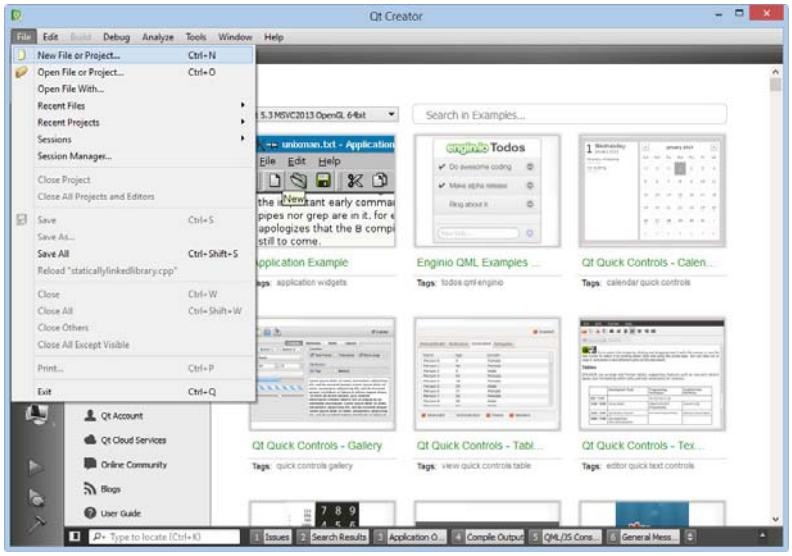

Figure 37: New Project in Qt

Then choose option Libraries with C ++ Library as shown in Figure 38:

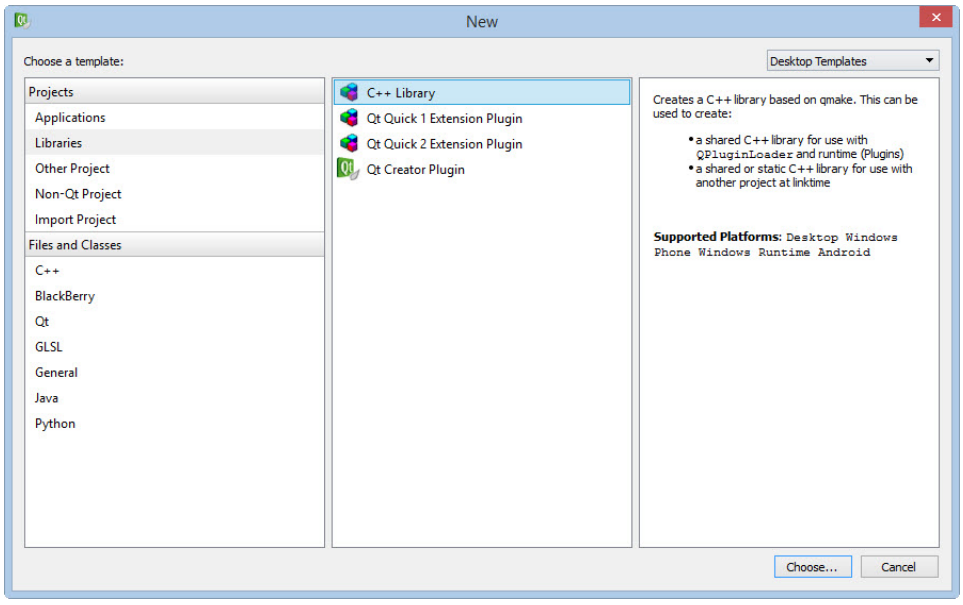

Figure 38: The Project is to develop a Static C++ Library

With regard to the location we choose "Statically Linked Library" also the Name that we want for the library and the directory to store it, as is shown in Figure 39:

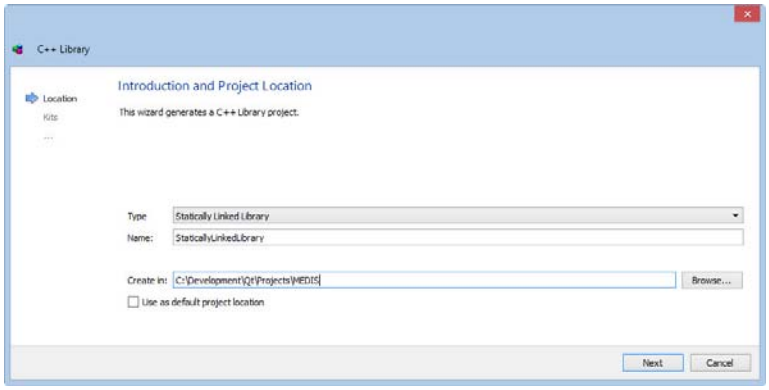

Figure 39: Introduction and Project Location

Regarding the *Kit Selection,* the appropriate option for our operating system will be to choose "*Desktop Qt 5.3 MinGW 32 bit*", as shown in Figure 40

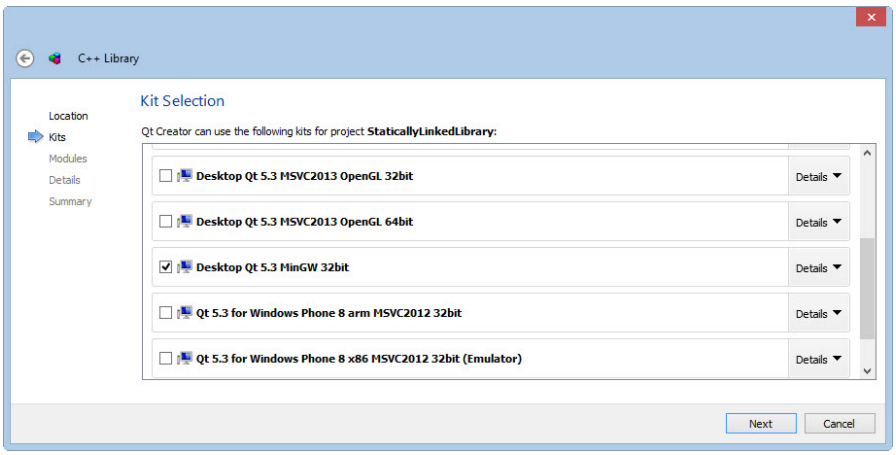

Figure 40: Kit Selection

On the next screen we will choose the modules that we need to integrate in our library, must choose "*QtCore*" as shown in Figure 41:

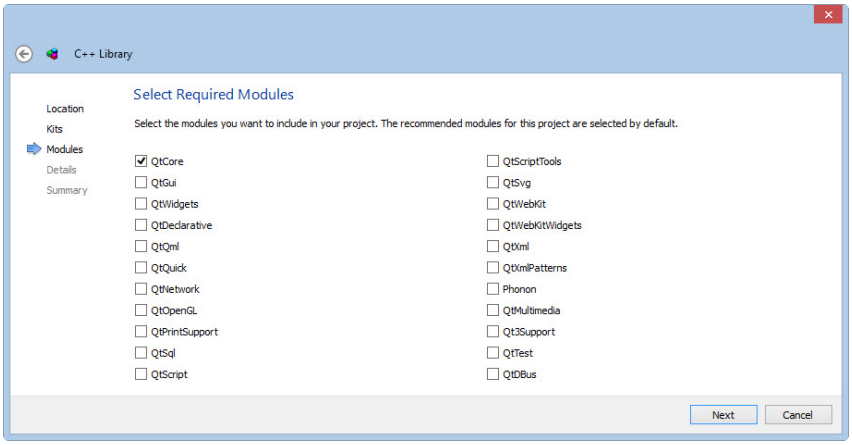

Figure 41: Selection of Required Modules

Now we select the information to the class ("*Class Information*"). We will introduce the class name and the name of the files *".h*" and "*.cpp*", as shown in Figure 42:

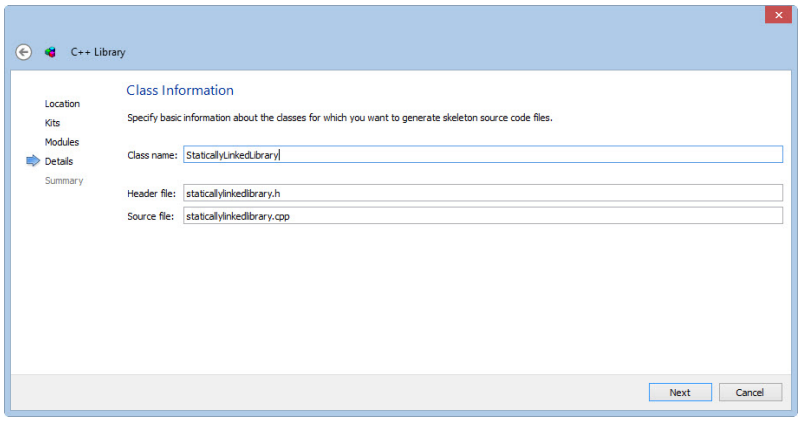

Figure 42: Class Information

The next screen is for entering information concerning project management. In our case we select "*None*" as shown in Figure 43:

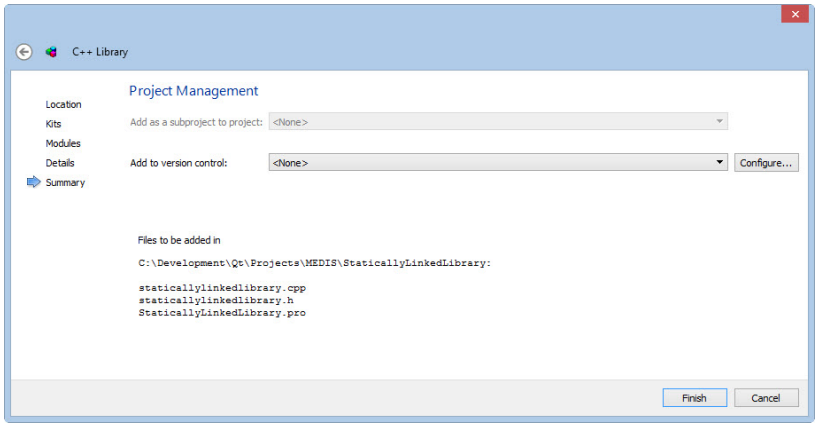

Figure 43: Project Management

Once completed the information and finish Qt will create the Template to enter the c++ code of the library as we have presented above.

Template format is shown in Figure 44:

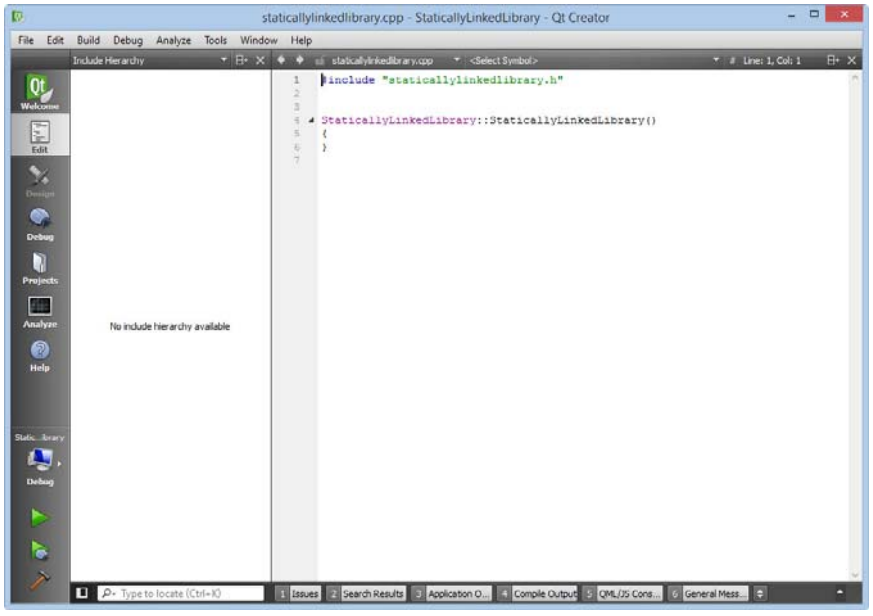

Figure 44: Template format

You can check the contents included in the "*StaticallyLinkedLibrary.pro*", as shown in Figure 45:

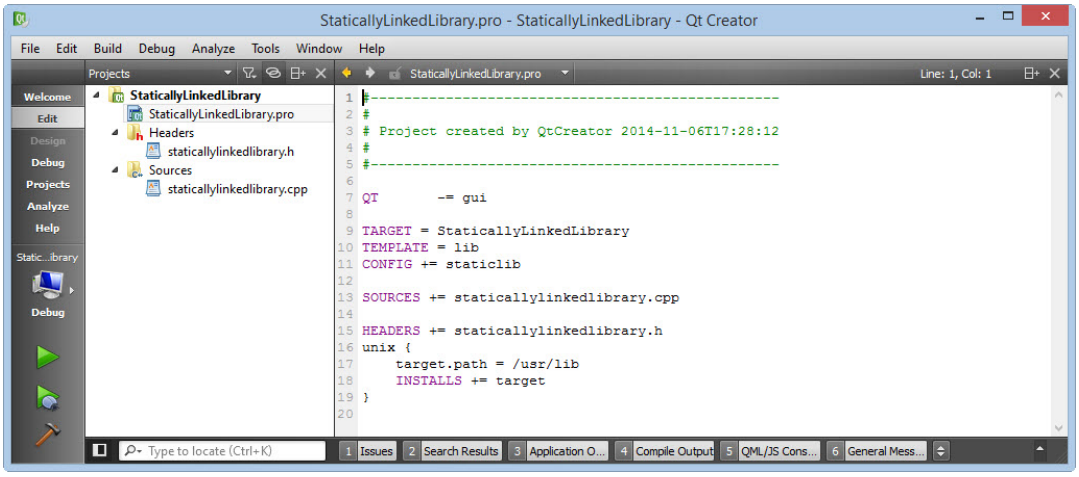

Figure 45: .pro Code

In the section of "*Sources*" we can write the code in c ++ of the library as shown above. In Figure 46 you can see the illustration:

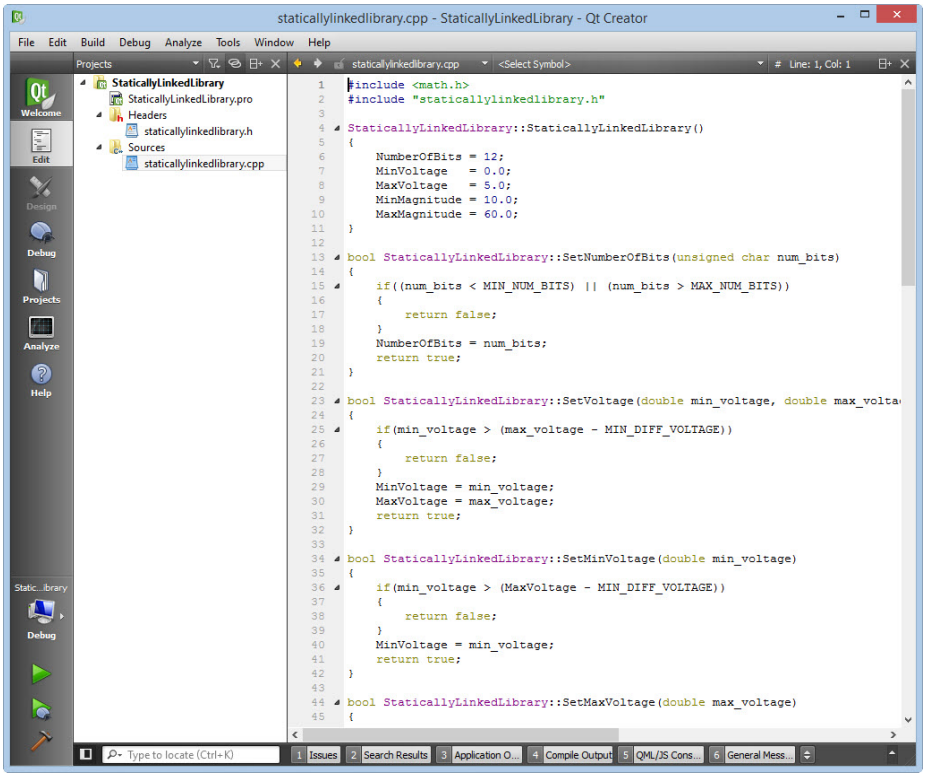

Figure 46: Library c++ code

In the section "Headers" we enter code in c ++ of the header file "*.h*" as shown above. In Figure 47 you can see the illustration:

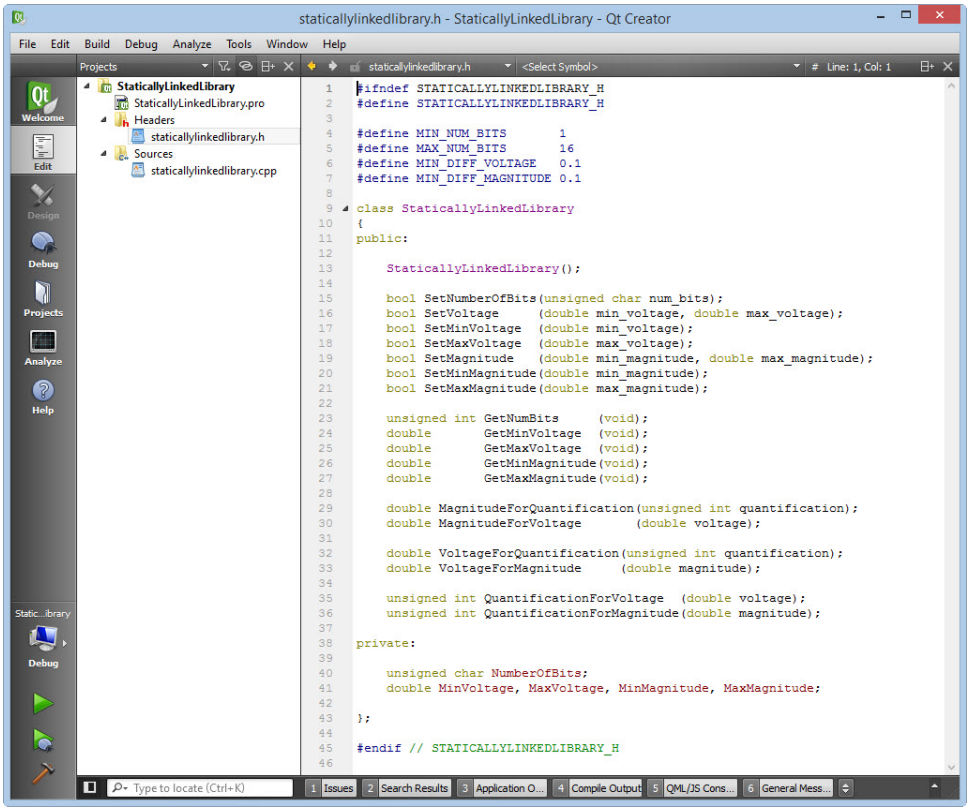

Figure 47: C++ Code in *.h*

The next step will consist in compile and built the library and geting the relevant files as shown in Figure 48:

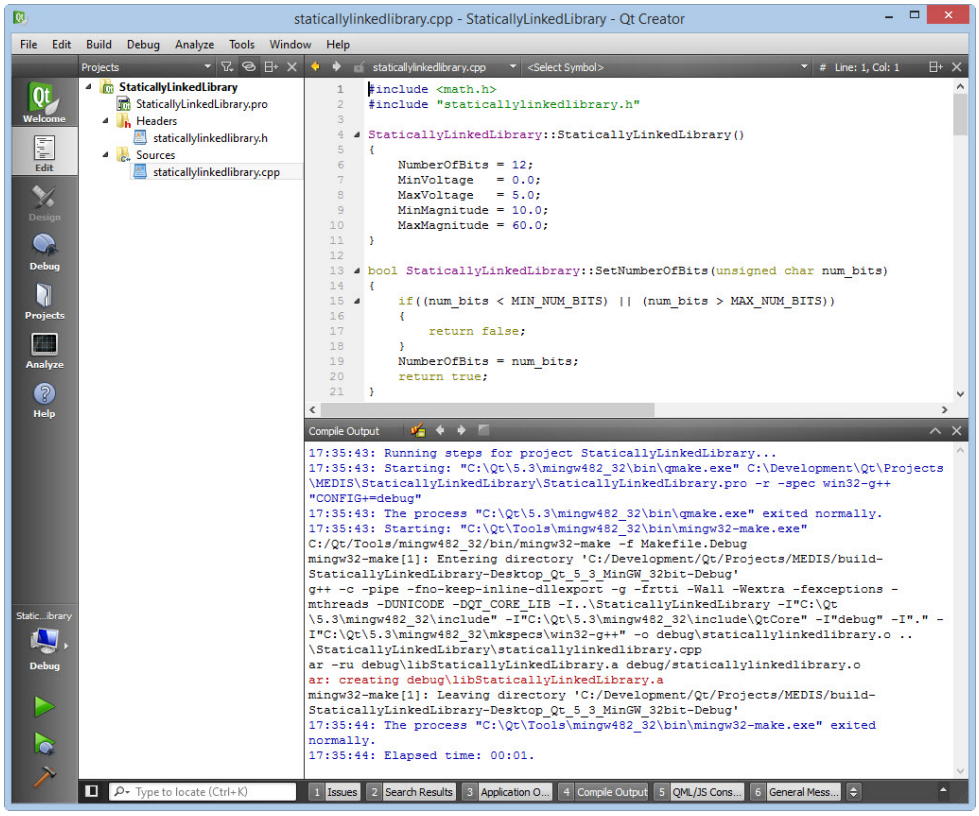

Figure 48: Compile Output

In Figure 49 is shown the directories generated:

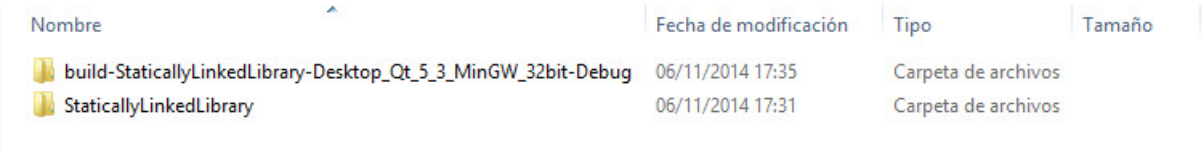

Figure 49: Directories generated

The source files are shown in Figure 50:

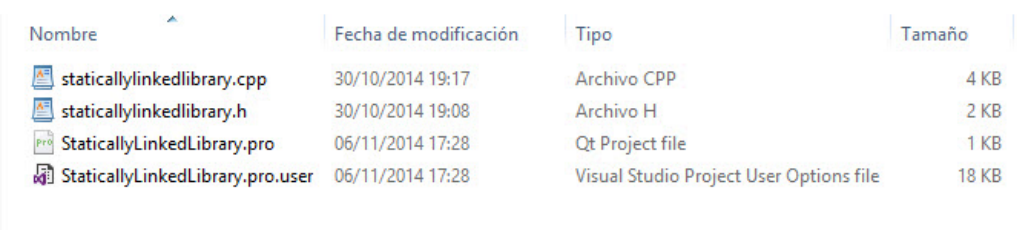

Figure 50: Source files

The building directories are shown in Figure 51:

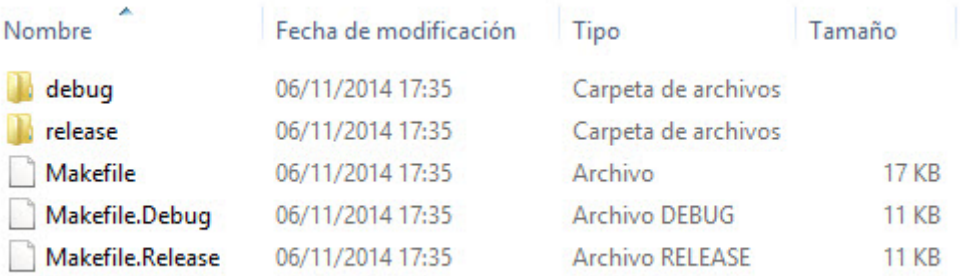

Figure 51: Building directories

The files with the generated library are shown in Figure 52:

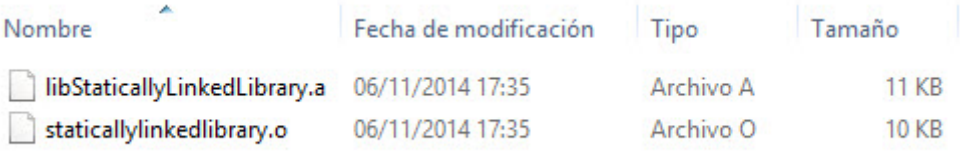

Figure 52: Generated Library

To finish the activity we propose the design of a console application that makes use of the static library created. In this case the program asks to the user to enter the value obtained by the sensor for example can be in volts and then using the transfer function calculation will provide the value of the physical quantity (degrees, liters, meters, pressure, humidity etc.). The c++ code can be:

```
#include <QCoreApplication>
#include <stdio.h>
#include "staticallylinkedlibrary.h"

int main(int argc, char *argv[])
{
    QCoreApplication a(argc, argv);
    StaticallyLinkedLibrary Converter;
```

```
    unsigned int quantification;


    printf("StaticallyLinkedLibrary\n\nquantification? ");
    scanf("%u",&quantification);
printf("magnitude=%lf",Converter.MagnitudeForQuantification(quantification)
);


    return a.exec();
}
```

To run this application in Qt, we need to select in the main menu:

File -> New File or Project...-> Aplications -> Qt Console Aplication

As is shown in Figure 53:

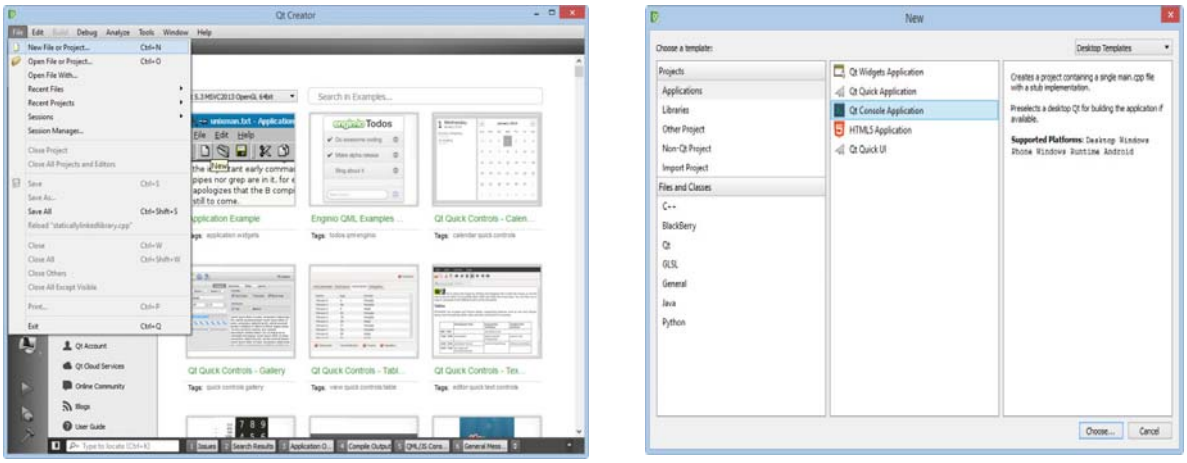

Figure 53: Using the Library with a Console Application

Next will be to select the project localization and the kit selection of *"DeskTop Qt 5.3MinGW 32bit"* as is shown in Figure 54:

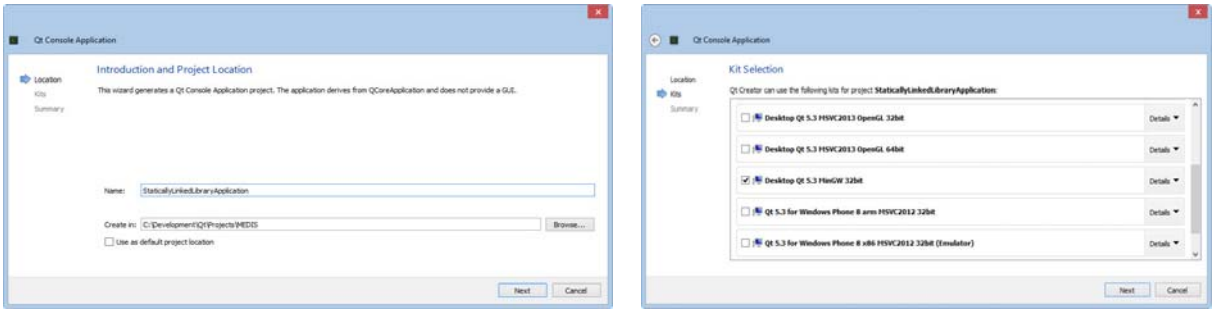

Figure 54: Project Location and Kit Selection

The next screen is for entering information concerning project management. In our case we select "*None*" as shown in Figure *55*:

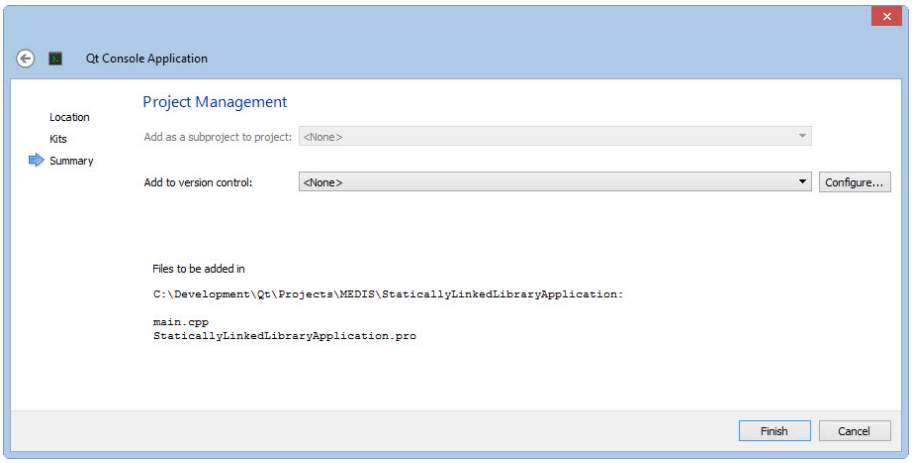

Figure 55: Project Management

After to press the "Finish" button we can see the generated template (Figure *56*):

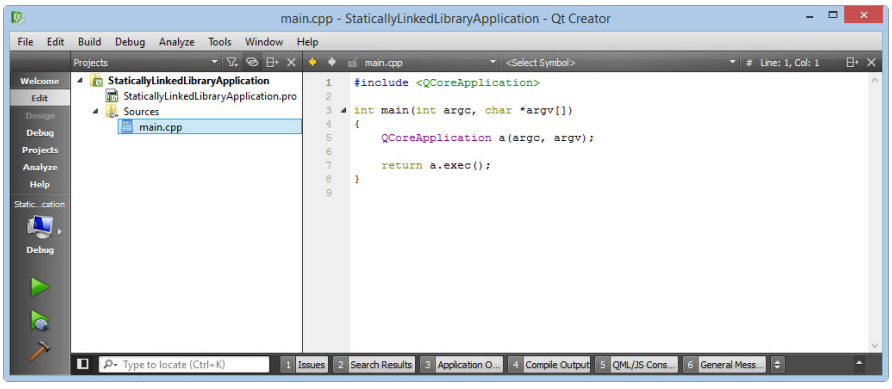

Figure 56: *.cpp* Template

You can check the contents included in the "*StaticallyLinkedLibraryApplication.pro*", as shown in Figure 57:

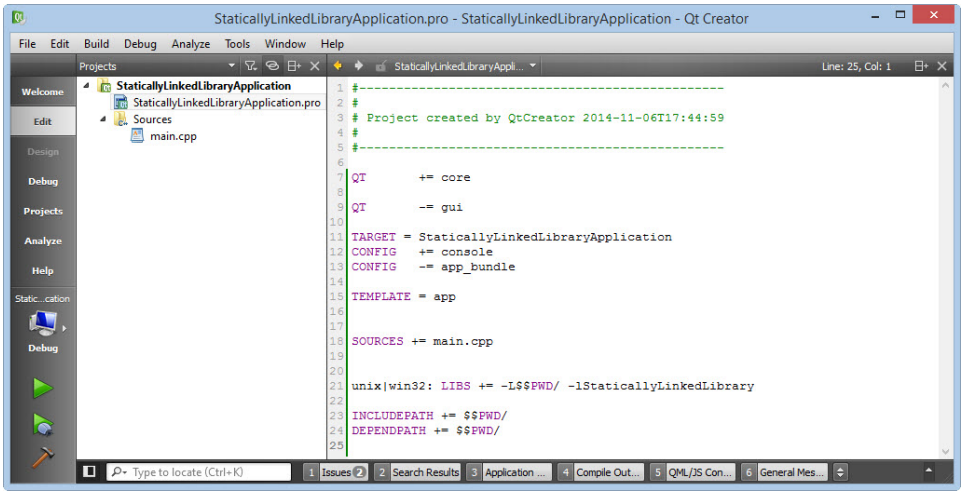

Figure 57: *.pro* Template

The directories and source files are (Figure 58):

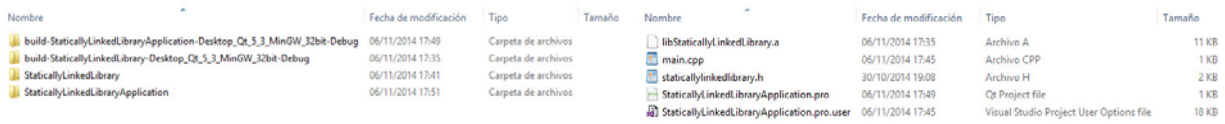

Figure 58: Directories and Source Files

Inside the template generated we can write the code of the console application described above as shown in Figure 59:

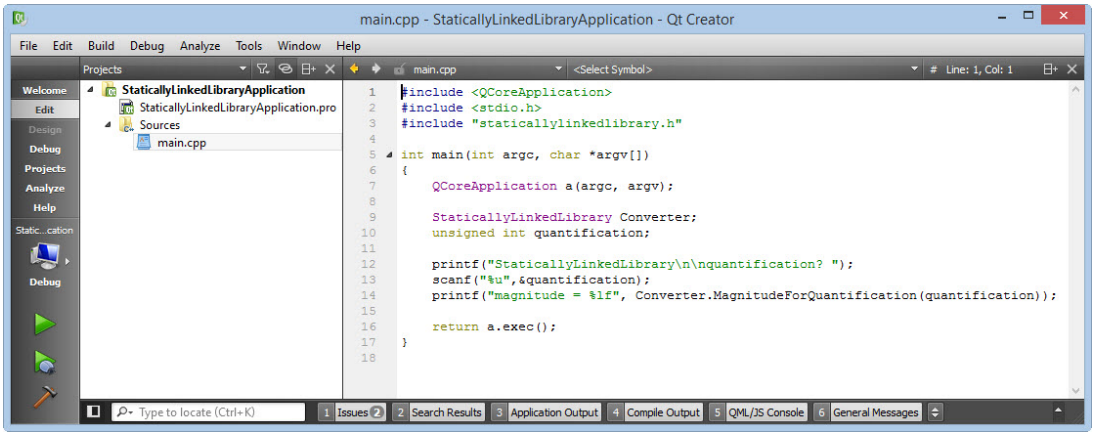

Figure 59: Code of the Console Application

Next step will be compiling and building the application (Figure 60):

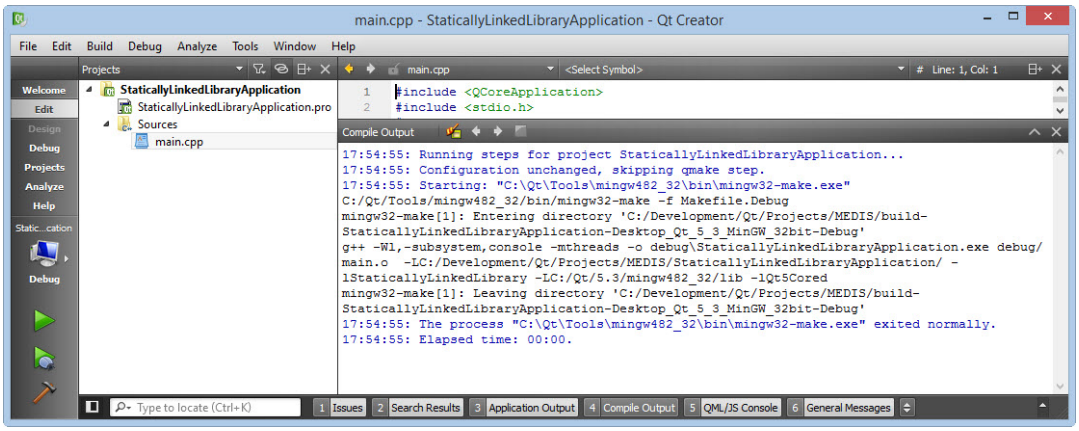

Figure 60: Compile Output

The Figure 61 shows an example of the execution screen generated by the program:

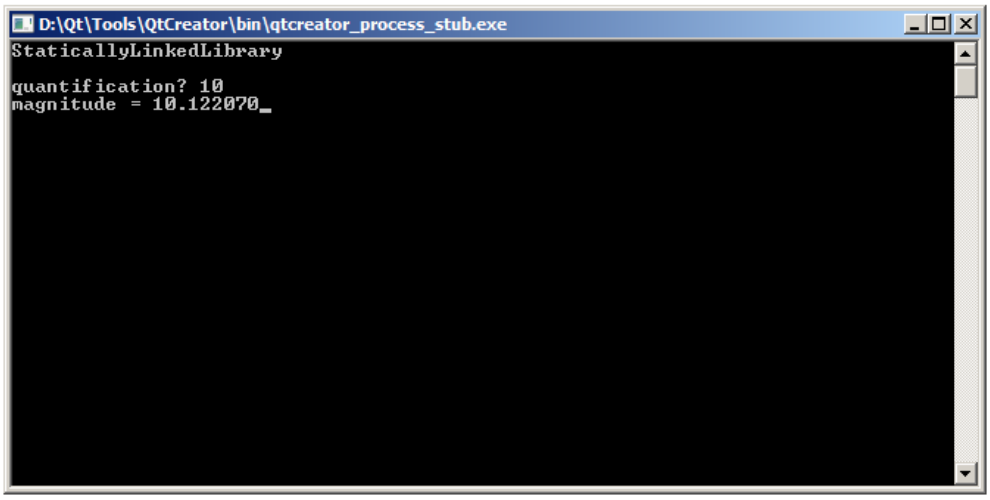

Figure 61: Screen after the execution of the application

6 Mini-project: Formal specification

6.1 Problem Definition

The proposed process is a level and temperature control system of a liquid tank. The Figure 62 shows a diagram of the process to be managed.

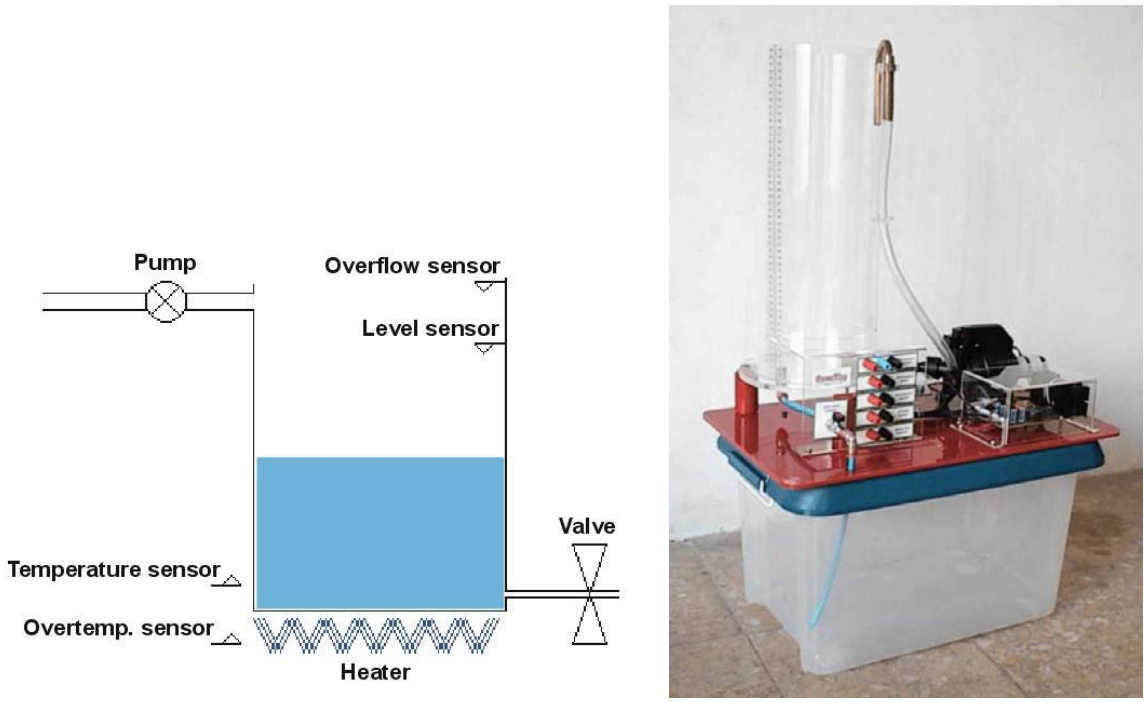

Figure 62: Diagram of the process

The computer system should control the process so that the level and temperature are maintained at desired values at all times. These desired values and ways of behaving the deposit will be decision of the working group of students.

The deposit is instrumented with a different sensors and actuators that allow controlling it.

An overflow TTL digital sensor lets us know if there is danger of overflow liquid. A "0" indicates a danger of overflow.

A TTL digital overheat sensor lets us know if there is danger of overheating. A "1" indicates that danger.

By a digital electro valve can dispense the liquid contained in the tank. A "1" opens the valve.

The temperature of the liquid is known by a linear sensor that provides a voltage of 10 volts (equivalent to 100 ° C) to 0 Volts (equivalent to 0 ° C).

The volume (level) of the liquid is obtained by a linear sensor for the height of liquid which gives a voltage of 6 volts to a height equivalent to 0 liters in the tank and 0 volts to a height equivalent to 3800 liters.

A pump allows introducing liquid into the tank. The pump power can be controlled with analog voltage from 5 V (100% power) and 0 volts (0% power).

The signals are connected to a National Instruments NI USB-6008 data acquisition board and the mandatory assignment of signals connections within the card are shown in Table 2:

Signal	NI USB-6008 Connection
Overflow	P0.0
Overtemperature	P0.1
Electrovalve	P1.0
Heater	P1.1
Temperature	ai0, GND reference
Level	ai1, GND reference
Pump	ao0

Table 2: Signals connections between process and data acquisition board

6.2 Some important aspects to be assessed

6.2.1 System design and development

It is the software developed in the mini-project. The following will be assessed:

- Effectiveness
  - Minimum Functional Requirements
  - Extra Functionality
- Sturdiness
  - Reliability
  - Testing
- Modularity
- Clarity and Maintainability of Source Code
  - Choosing Identifiers
  - Indenting Text
  - Comment
  - Etc.

6.2.2 Report

The final report will contain information obtained from the laboratory notebook, should be properly presented. The following content will be assessed:

- Problem Description and Objectives (Requirements document)
- Identification and Modeling of Process (Specifications document)
- Program Design
  - Modules that make up the program
  - List of functions at each module (prototype) and committed
- Program Operations Manual (User Manual).
- Annexes:
  - Full Source Code (in Courier New 10pt font).
  - Calculations (obtaining the equations of transformation of input and output)
- Etc.

It is important to properly present the report (front, indexes, page numbers, etc.).

6.2.3 Remarks

During the course the student must develop:

- Lab Notebook

- Design and development of the system
- The working report

The clarity, completeness and brevity in all aspects of the work will be taken into consideration.

At the end will be given:

- Report
- The developed software (source and executable).
- Exposure in computerized form.

Prior to the session final defense of the project will be compulsorily delivered the above materials in a compressed file in compressed format (zip, rar or tar.gz) should have the following structure:

Filename: **gNNN_alias_of_grup.zip**

For example: g223_cactusland.zip

Inside the zip file there will be 4 directories:

report: report with original format and in PDF format.

sources: the source code of the application.

program: the application executable.

presentation: the *Powerpoint* presentation in electronic format.

6.2.4 Laboratory book

During the semester project progress will be documented in a written book. The teacher will review the booklet and evaluate continuously during the semester.

7 Bibliography

B Govindarajalu, *Computer Architecture and Organization: Design Principles and Applications*, Ed: McGraw-Hill Education, 2004, ISBN 0070532362, 9780070532366

Jon Stokes, *Inside the Machine: An Illustrated Introduction to Microprocessors and Computer Architecture*, Ed: No Starch Press, 2013, ISBN: 978-1593271046

Page, Daniel, *A practical introduction to computer architecture*, Ed: Springer-Verlag Berlin H., 2009, ISBN: 9781848822559

M. Morris Mano, *Computer System Architecture*, Ed: Prentice Hall, 2007, ISBN 0131755633, 9780131755635

David A. Patterson, John L. Hennessy, *Computer Organization & Design: The Hardware/Software Interface*, Ed: Morgan Kaufmann Publishers, 1998, ISBN 1558604286

John P. Hayes, *Computer Architecture and Organization*, Ed: McGraw-Hill, 1998, ISBN 0070273553

Douglas E. Comer, *Essentials of Computer Architecture*, Ed: Addison-Wesley Professional, 2004, ISBN 0131491792

Andrew S. Tanenbaum, *Structured Computer Organization*, Ed: Prentice Hall, 1999, ISBN 0130959901

1/e Rajiv Chopra, *Computer Architecture and Organization (A Practical Approach)*, Ed: S. Chand Publishing, 2013, ISBN: 9788121942249

Qt Project, http://qt-project.org/

Data acquisition card NI USB 6008, http://www.ni.com/pdf/manuals/371303m.pdf

National Instruments Data acquisition card NI USB 6008, http://sine.ni.com/psp/app/doc/p/id/psp-117/lang/en