MASTER THESIS
AUTOMATION SYSTEMS AND INDUSTRIAL ELECTRONICS

STUDY OF AN INNOVATIVE SCADA SYSTEM

Oriol Filló Claret

April 2016
Supervisor: Jordi Zaragoza Bertomeu
ACKNOWLEDGEMENTS

This master thesis report has been written by Oriol Filló during the 3rd semester of the Master in Automation Systems and Industrial Electronics at Escola Superior d’Enginyeria Industrial Aeroespacial i Audiovisual de Terrassa. The work presented in the report has been carried out between September 2015 and March 2016, and to be presented in May 2016.

First of all I would like to express my gratitude to the supervisor of this project Jordi Zaragoza, who has been supportive, patient and open to discussion since the early beginning of the thesis. A special thanks is also extended to all the Industrial Shields team members, which have provided the PLC used for the development of this project, the contact with the company owning the reference plant and information on the actual plant and automation systems.
### CONTENTS

1. OBJECTIVES ........................................................................................................... 4
2. SCOPE OF THE STUDY ......................................................................................... 5
3. BACKGROUND AND JUSTIFICATION .................................................................. 6
4. SPECIFICATIONS OF THE STUDY ......................................................................... 7
   4.1 Reference plant .................................................................................................. 7
   4.2 Specifications of the SCADA system .................................................................. 8
      4.2.1 Layout of the SCADA system ................................................................... 11
5. CONSIDERATION OF ALTERNATIVES AND EVALUATION .............................. 12
   5.1 PLCs .................................................................................................................. 12
   5.2 HMI .................................................................................................................... 13
6. REVIEW ON THE IMPLEMENTED COMMUNICATIONS ..................................... 14
   6.1 RS-485 .............................................................................................................. 14
      6.1.1 RS-485 general description ....................................................................... 14
      6.1.2 RS-485 bus configuration ......................................................................... 17
      6.1.3 RS-485 transceiver operation ................................................................... 17
   6.2 TCP/IP Protocol Suite ...................................................................................... 18
      6.2.1 Layers characteristics .............................................................................. 20
      6.2.2 Addressing ................................................................................................. 27
      6.2.3 TCP/IP Protocol suite table summary ....................................................... 29
   6.3 Modbus .............................................................................................................. 29
      6.3.1 Origin and overview ................................................................................. 29
      6.3.2 General operation ...................................................................................... 30
      6.3.3 Modbus addressing ................................................................................... 31
      6.3.4 Modbus data storage: coils and registers ............................................... 31
      6.3.5 Modbus functions ..................................................................................... 32
      6.3.6 Modbus Erro Checking Methods .............................................................. 33
      6.3.7 Modbus modes ......................................................................................... 34
      6.3.8 Modbus RTU Message example .............................................................. 36
17  BIBLIOGRAFY & REGULATIONS.........................................................88
ANNEX A: Modbus RTU Test.............................................................91
ANNEX B: Real case examples.......................................................93
ANNEX C: Attached CD..................................................................94
ANNEX D: Datasheets....................................................................94
ABBREVIATIONS

**ADU** Application Data Unit

**ARP** Address Resolution Protocol

**CRC** Cyclical Redundancy Check

**DHCP** Dynamic Host Configuration Protocol

**DNS** Domain Name System

**DSC** Datalogging and Supervisory Control

**EIA** Electronic Industries Alliance

**FTP** File Transfer Protocol

**HMI** Human Machine Interface

**HTTP** Hypertext Transfer Protocol

**ICMP** Internet Control Message Protocol

**IGMP** Internet Group Management Protocol

**IoT** Internet of Things

**IP** Internet Protocol

**LAN** Local Area Network

**LRC** Longitudinal Redundancy Check

**MAC** Medium Access Control

**MAC** Media Access Control

**PDU** Protocol Data Unit

**SMTP** Simple Mail Transfer Protocol

**TCP** Transmission Control Protocol

**URL** Universal Resource Locator

**WAN** Wide Area Network

**WLAN** Wireless Local Area Network
1 OBJECTIVES

The main objective of this work is to adapt or develop industrial communication protocols to the open-source based Industrial Shields PLCs. A second goal of the study is to provide a prototype of a SCADA system using Industrial Shields PLCs. Finally, the work developed should serve as a proof of concept and reference showing that developing a SCADA system using low-cost alternative hardware, based on open-source, is nowadays a feasible alternative to traditional and closed-standard automation solutions.
2 SCOPE OF THE STUDY

The developed and implemented functionalities for Industrial Shields PLCs (Boot & Work Corp, S.L., 2016) in this work are highly demanded in industrial automation, supervision and control systems. The results and conclusions of the study will serve to provide some ready-to-use solutions to Industrial Shields customers and thus, upgrade the value-for-money of the products. The ultimate application of this study is to provide a prototype of a SCADA system based on Industrial Shields PLCs. Following are the main points that this study will cover.

- Implementation of Modbus RTU protocol over RS-485 (master and slave mode) on Industrial Shields PLCs.
- Development of a Human Machine Interface (HMI) for supervision and control purposes.
- Implementation of TCP protocol on Industrial Shields PLCs over Ethernet and Local Area Network (LAN) for interaction between the control unit (Industrial Shields PLC) and the HMI.
- Practical integration of sensors, data acquisition equipment and actuators with a PLC.

The SCADA system prototype is focused on a real-case project, which is a safe-water treatment plant where an upgrading of the current supervisory and control system is wanted.
3 BACKGROUND AND JUSTIFICATION

The development of this work is of particular interest to Industrial Shields firm and, more generally, to the industrial and automation sector. The paradigm in this field is beginning to evolve from the closed-systems concept to the open-systems. Studies on the ‘open’ concept related to Industry 4.0 and the Internet of Things (IoT) are on the rise in the scientific literature. The open-source and open-standard concepts are regarded as a key plank for the evolution of the industry. This new technologies will broaden the range of solutions for automation systems and boost the interaction capabilities between devices and systems of different nature or firms. The present project is encased within that framework and it will provide an example on how a SCADA system can be developed based on open technology, integrating devices from different firms, which communicate with each other via different open protocols.

The election of Modbus RTU and TCP-IP protocols as part of the development of this project lies on the following reasons.

- Modbus RTU, and Modbus more generally, are widely extended protocols in the industry nowadays. Therefore, it is a feature demanded in the industrial automation sector.
- TCP-IP is the standard protocol of the Internet and, for all the exposed reasons in this section, the devices supporting it bear a greater potential.
4 SPECIFICATIONS OF THE STUDY

This section presents the ‘status quo’ of the installation taken as reference for the development of this study and describes the set specifications that the prototype of the SCADA system to be developed should attain.

4.1 Reference plant

The reference installation is a water tank of the safe water supply system of a town in Catalonia. In this tank there is the need of monitoring and controlling several parameters (three are taken into account for this study) such as Chloride, PH and turbidity levels. All these variable values need to be contained within a specific range for the water to be conditioned for human consumption. For each of the variables, there is a dedicated dosing pump that will provide the necessary flow rate of the component that modulates the chloride, PH and turbidity levels. Figure 1 shows the layout of the water tank system.

Figure 1. Water tank system layout.

- P1: Chloride pump.
- P2: Acid pump.
- P3: Floculant pump.
- V1: Chloride valve.
- V2: Acid valve.
- V3: Floculant valve.
- V4: Water inlet.
- V5: Water outlet.
- V4: Water in valve.
- V5: Water out valve.

At the moment the installation runs with a simple standard PLC which regulates the pumps according to the control scheme, but there is no SCADA system. The only communication implemented is an SMS sent in case of an alarm. The only information included in the alarm is the stoppage of one of the dosing pumps. No information on the level of the control variables nor its evolution along the last hours is reported. Besides, no historical data of the system is stored. Figure 2 illustrates the control system of the plant nowadays.

![Figure 2. Current control system layout of the water tank system.](image)

An upgrading of this system is wanted. The new design should include the same automation logic, but improve the information available from the system and integrate up-to-date technologies.

### 4.2 Specifications of the SCADA system

Following are describe the specifications for the SCADA system to be develop.

**Capabilities and system configuration**

- The automated system has a centralized configuration
  - the controller receives all the variable values (chloride, PH, turbidity).
The controller evaluates the situation of each variable compared to the set limits and operates the dosing pumps accordingly.

- The limits for each variable are adjustable from a user interface which is not located at the same position as the controller.
- The variables values are updated on the user interface at a minimum rate of every 15 minutes.

**User interface**

- The user interfaces is an HMI on a remote PC with internet connection.
- All variables and actuators states are visible on the HMI
- The limits of the variables (max,min) are set manually on the HMI

**Data acquisition equipment and actuators**

- Inputs
  - The chloride, PH and turbidity values are acquired by a single Seko Kontrol 800 (ANNEX D: Datasheets).
  - Water intake and water outtake of the tank meters. Water tank level meter.
- Outputs
  - The dosing pumps reference is Seko Tekna EVO APG800 (ANNEX D: Datasheets).
  - Each dosing pump have a digital control valve.
  - There are 3 Seko Tekna EVO APG800 (one per each variable).
  - Water intake and outtake analog control valves.
  - The existing equipment (pumps, data acquisition ) and number of units is already dimensioned for the needs of the system.

**Control of the variables:**

The controlling algorithm for each variable (chloride, PH and turbidity) follows the same pattern. Each variable has a maximum and minimum limit and its value must be contained within this range.
Alarms

The existing alarm system must be upgraded in the following aspects:

- including more information on faults and
- using a more up-to-date communication such as Internet, instead of SMS.

Figure 3 shows the layout of the target SCADA system.
4.2.1 Layout of the SCADA system

Figure 3. Layout of the SCADA system defined in the specifications
5 CONSIDERATION OF ALTERNATIVES AND EVALUATION

Before beginning the project two decisions had to be made regarding the controlling devices (PLC) and the HMI technology.

5.1 PLCs

Concerning the PLC for automating the system, the first alternative was using Industrial Shields PLCs. The relevant advantages of this technology to this work, and compared against standard industrial PLC solutions are:

- Low-cost
- Flexibility
- No-licensing
- Accessibility
- Programmed as an Arduino (Arduino, 2005), a language similar to C, which offers an opportunity to students from IT, communications and electronics background that have no specific training in standard PLCs languages.

Advantages of standard industrial PLCs compared to Industrial Shields at the moment are.

- Widespread knowledge on the technology.
- Ready-to-use solutions.
- More mature technology.

Taking into account the project specifications, it was concluded that capabilities of Industrial Shields PLCs (ANNEX D: Datasheets) were more than enough to develop the application. Its flexibility was an important aspect, due to the need of integrating devices of different firms and specifications in the installation.

The Arduino IDE was also an opportunity for the developer because it had previous experience on the language (C,C++ and Java). Finally, the low-cost is also a point of relevance once the rest of requirements are satisfied.
5.2 HMI

The HMI was determined to be develop using LabView (National Instruments, 2012), because it offers some ready-to-use tools for data acquisition, monitoring and SCADA systems and supports an intuitive graphical programming language. In addition, all the points defined in the project specifications can be met with this software.

A considered alternative was to create a graphical application from scratch using C# and Visual Studio coding languages. This option was ruled out due to the amount of work that it required and because it was out of the scope of this project.
6 REVIEW ON THE IMPLEMENTED COMMUNICATIONS

This section will describe the basic knowledge acquired throughout the study on the implemented communication protocols and interfaces. The purpose of this section is to present and provide an overview of the used communication technologies in this work.

6.1 RS-485

This section presents a brief introduction into RS-485 serial line layer standard.

6.1.1 RS-485 general description

Serial line communications (SparkFun Electronics, 2013) include many different physical layers and standards. Among all of them, probably the most well-known is RS-232 (Saboya, 2012), because it is present in all computers and in the past was the most extended serial communication. However, it has some limitations that make the system not suitable for more demanding applications. This derived in the advent of new serial buses with upgraded capabilities, being RS485 the most implemented for industrial applications (Bies, 2002). The RS-485 standard was approved in 1983 by the Electronic Industries Alliance (EIA). Table 1 compiles the most representative features of RS232 and RS485 buses, and it remarks the higher potential of RS485 compared to other RS-XXXX serial lines (Bies, 2002).

One of the most relevant advantages of RS-485 in terms of operation are the drastic higher speed rates and distances of the bus. This difference is due to the increased immunity to disturbances and noise that its bus configuration provides. In the RS-232 bus, the receiver and the transmitter compare the voltage variations to a shared zero line. Any subtle voltage variation induced in the ground level disturbs the reference and, therefore, the voltage seen from the receiver perspective. Besides, this configuration is prone to the appearance of ground loops. All this reasons reduce the speed capacity and the length of the buses for RS-232 interfaces, because they are vulnerable to noise.
Table 1. RS-232 and RS-485 comparison chart. Ref Siemens Energy & Automation.

<table>
<thead>
<tr>
<th></th>
<th>RS232</th>
<th>RS485</th>
</tr>
</thead>
<tbody>
<tr>
<td>Differential</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Max nº of drivers</td>
<td>1</td>
<td>32 (it can be extended to 255) (Texas Instruments Incorporated, 2011)</td>
</tr>
<tr>
<td>Max nº of receivers</td>
<td>1</td>
<td>32 (it can be extended to 255) (Texas Instruments Incorporated, 2011)</td>
</tr>
<tr>
<td>Network topology</td>
<td>point-to-point</td>
<td>Multipoint</td>
</tr>
<tr>
<td>Max distance</td>
<td>15 m</td>
<td>1200 m</td>
</tr>
<tr>
<td>(acc.standard)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Max speed at 12m</td>
<td>20 kbs</td>
<td>35 Mbs</td>
</tr>
<tr>
<td>Max speed at 1200m</td>
<td>1kbs</td>
<td>100 kbs</td>
</tr>
<tr>
<td>Receiver Input sensitivity</td>
<td>-3/-3 V</td>
<td>-200/+200 mV</td>
</tr>
<tr>
<td>Minimal driver output voltage (54 Ohms)</td>
<td>-5V/+5V</td>
<td>-1.5V/+1.5V</td>
</tr>
<tr>
<td>Receiver input range</td>
<td>-15 / +15 V</td>
<td>-7 / +12 V</td>
</tr>
<tr>
<td>Max driver output voltage</td>
<td>-25 / +25 V</td>
<td>-7 / +12 V</td>
</tr>
</tbody>
</table>

On the other hand, RS-485 buses are based on a differential approach, which means that the signals are not transmitted as an absolute value but rather as the voltage difference between V+ and V- lines (the two lines are commonly named A and B). This implies that even if a voltage difference between the receiver and transmitter grounds is present, the communication will work effectively.

Another relevant feature that allows faster speed in RS-485 buses is the voltage sensitivity, which is remarkably lower for RS-485 (200mV, Figure 5) compared to RS-232 (3V). This narrow transition voltage gap allows faster voltage shifts on the lines and thus, faster bit rates.
RS-485 bus is usually half-duplex (Figure 4), yet depending on the devices characteristics it can be adapted to full-duplex. The connection of the devices to the bus is straight, A connector to A connector and B connector to B connector.

*Figure 4. Half-duplex RS-485 configuration. Ref. Texas Instruments.*

Another important feature for RS-485 lines is the multipoint configuration, which means that all the devices can initiate the communication if required. In RS-232 there is only the possibility of point-to-point configuration, limiting the maximum number of devices in a line to two (one receiver and one transmitter). Conversely, a signal broadcast from any device in the RS-485 bus is received by all ports in the line.
6.1.2 RS-485 bus configuration

The RS-485 network allows up to 255 devices connected in the same line depending on its internal characteristics (Texas Instruments Incorporated, 2011). Figure shows two common typologies of RS-485. The values of the resistors and configuration may vary depending on the specifications, immunity and the noise in the environment (Texas Instruments Incorporated, 2011). A major principle regarding the configurations is that the line terminations resistance equals the line characteristic impedance, in order to avoid reflections. Figure 6 depicts two types of possible terminations typologies. Standard RS-485 buses comprise two cables, A and B or V+ and V-. Yet, RS-485 buses can also exist for full-duplex communication (Texas Instruments Incorporated, 2011).

![Image of RS-485 termination typologies examples. Ref. Texas Instruments.]

6.1.3 RS-485 transceiver operation

In half-duplex configuration the driving and receiving mode must occur at different times. All the nodes include a direction control which is activated by software and sets the state of the device, either to transmitting or receiving mode. In a half-duplex line, it must be ensured that there is only a driver at any time; the rest of devices must be at receiving mode. Otherwise it may result in data collision. The standard pins of a RS-485 transceiver are summarized in Table 2.
Table 2. RS-485 transceiver basic pinout.

<table>
<thead>
<tr>
<th>RS-485 transceiver pin</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>DI (Data In)</td>
<td>Transmit pin (TX)</td>
</tr>
<tr>
<td>RO (Receive Out)</td>
<td>Receive pin (RX)</td>
</tr>
<tr>
<td>DE (Data Enable)</td>
<td>Data transmit enable</td>
</tr>
<tr>
<td>RE (Receive Enable - negated)</td>
<td>Data receive disable</td>
</tr>
</tbody>
</table>

RE and DE are usually bounded together so that whenever the transmit functionality is enabled, the receive functionality is disabled. Thus, a single pin from the controller is enough to control the direction of the device (transmit state or receive state). Figure 7 shows the schematics for a regular RS-485 half-duplex transceiver.

**Figure 7.** RS-485 transceiver basic schematic. Ref. Maximintegrated.

### 6.2 TCP/IP Protocol Suite

TCP/IP protocol suite is an open system layer-based collection of protocols that establishes the basis for communicating between different types of computers systems. It is widely well-known for being the protocol suite used for Internet communication. Among the supported protocols within the different layers, there
are the Transmission Control Protocol (TCP) and the Internet Protocol (IP), which serve as name for the entire protocol suite. At its origins in 1983, the protocol suite considered four layers. However, some years later it was adapted to a five layer structure so as to equate to the OSI Model (Forouzan, 2010). Figure 8 shows the layers for the TCP/IP protocol suite.

![Figure 8. TCP/IP Protocol suite layers at its origin and after rearrangement to become closer to the OSI Model. Ref. TCP/IP Protocol Suite.](image)

The TCP/IP protocol suite is based on a hierarchical model, which means that the protocols in one layer are supported by one or more protocols in the lower level layer. However, there is not a strong dependability among specific protocols between the layers. The protocols can be mixed and matched so as to satisfy the needs of the system, as long as they support and provide an interface to protocols in higher/lower layers, in that sense TCP/IP is relatively flexible. Figure 9 shows the layers and several protocols.

![Figure 9. TCP/IP suite layers. Ref. TCP/IP Protocol Suite.](image)
6.2.1 Layers characteristics

In the following paragraphs the characteristics of each of the five layers in the TCP-IP suite are described.

**Physical Layer**

TCP/IP does not provide a specific protocol for the physical layer, thus this can support all standard protocols. As it is induced in the name, this communication level is right above the hardware, such as a computer or a router in a network. The unit of communication at physical layer is a bit. When communication is established between two devices there is a stream of bits flowing from one to the other. In that level, each connection between nodes (R1, R3 or R4 in Figure 10) may use different protocols, because each of the links along the whole path may support different ones.

![Figure 10. Physical layer communication diagram. Ref. TCP/IP Protocol Suite.](image)

**Data Link Layer**

TCP/IP, as for the physical layer, does not provide any specific protocol for the data link layer (Figure 11). Therefore, it can support any standard or protocol. The data unit in this layer is a frame, which besides the data received from the network layer it includes a header containing the source and destination of the frame.
The destination address allows the frame to be transferred along several nodes to the recipient. Conversely, the sender address is needed for a possible required reply from the receiver, as many protocols do. The frame sent from node A to node B may be different and modified along the way by intermediate nodes, because as in the physical layer, there might be different protocols between datalinks.

![Data link layer communication diagram](image)

*Figure 11. Data link layer communication diagram. Ref. TCP/IP Protocol Suite.*

**Network Layer**

The transmission approach defined by the TCP/P protocol suite for the network layer (Figure 12) is the Internet Protocol (IP). The data units in this layer are the datagrams, which are transported separately, not all of them in a single cluster or pack. IP does not keep track of the datagrams and neither provides reordering capability for the arrived datagrams. The communication at this layer is end to end, in other words, the layer is only active for the initial sender and for final receiver, but not for the rest of intermediate nodes or devices. This is an important difference with the previously described layers.
IP is not a robust protocol and is dismissed as *best-effort* delivery service. This implies that IP packets are vulnerable to corruption, delays, losses and contribute to congestion of the network. Therefore, reliable systems require a higher protocol that takes care of all these issues and allows a consistent communication. Some higher-level protocols above IP are ICMP, IGMP, TCP, UDP, ARP... (Figure 13). These can be in the same layer (ICMP, IGMP) or in the above layer, the transport layer (TCP, UDP).

ICMP is a data error control that handles issues when datagrams are discarded for some of the reasons mentioned above (data corruption, missing header, missing receiver, delay...). It also provides host and management query service (for example requesting IP address of neighboring routers).
On the other hand, IGMP is responsible for the recognition and keeping track of the membership status of the hosts connected to a network. In other words, the IGMP collects information locally about the members in a subnetwork and broadcasts this to other elements or services for network management purposes.

Another protocol within the network layer and which is complementary to the IP is the ARP. It serves to match the logical address of a device within a network to its physical address Medium Access Control (MAC). For information about addresses see Section 6.2.2.

There exist several more protocols in the network layer which are out of scope for this study.

**Transport Layer**

The function of transport layer (Figure 14) is to hand in the whole message which travels from the initial node to the final node, this layer is only present in the two end nodes. The data units for the transport layer are named segments, user datagrams or packets and this are transferred from the source sender to the end receiver; an email from computer A to computer B for example. The segments are divided into datagrams in order to be forwarded to the network layer for transmission. The datagrams will follow different routes along the Internet and once all of them have arrived, these are compiled and reordered into the initial segment to be delivered to the receiver device. The transport layer performs all this actions.

Another important requirement of this layer is to provide control on the data flow and provide acknowledgement for received packets. The kind of communication at the transport level is known as process-to-process, because it is responsible to transmit information from one process in device A to the process in device B. The network layer (one level below) defines protocols for transmission from device A to device B, but not from specific processes going on in these devices.
The most extended protocols in the transport layer are TCP (Transmission Control Protocol) and UDP (User Datagram Protocol). The main features of these two protocols are the following:

- **UDP:**
  UDP is a connectionless and unreliable transport protocol. Apart from permitting the process-to-process communication and discarding packets if an error is detected (via checksum), it does not offer any control on the packets flow nor acknowledgment functions. UDP can send data without the need of initially establishing a connection to the recipient. This reduces the overhead and provides a simple protocol for communication. UDP, as connectionless protocol, is useful in applications where speed is more critical than reliability or in applications with simple messages (National Instruments Corporation, 2013).

- **TCP:**
  TCP protocol takes all the responsibilities defined for the transport layer. Not only does it provide process-to-process communication, but it also includes flow control and consistent error detecting (both byte-oriented). This two last features make TCP a robust transport protocol. In order to achieve these functionalities, TCP protocol needs to establish a connection between the two processes of the two devices before sending any segment. The connection establishes something seen as a dedicated tunnel across the network which allows data transmission as a flow of bytes and a persistent interaction (between the devices) to ensure data reliability. TCP is a connection-oriented protocol. Differently to UDP, it requires
more overhead and is not as fast, but this downsides are offset by far by the data reliability that it offers. Figure 15 illustrates the straight communication that TCP creates.

![Figure 15. Representation of TCP connection between two devices. Ref. TCP/IP Protocol Suite.](image)

**Application Layer**

The application layer (Figure 16) is the highest level layer in the TCP/IP protocol suite, and its responsibility is to allow a user to access the services of Internet, therefore, many of these applications concern human interaction (Thomas, 2012). The communication is end to end (as in the network and transport layer). As there is a great diversity of applications related to internet, there are many protocols defined for this layer and each one is related to a specific services such as electronic mail (STMP), file transfer (FTP), accessing World Wide Web (HTTP)...

![Figure 16. Application layer communication schematic. Ref. TCP/IP Protocol Suite.](image)
Some of the application layer protocols relevant for this project (seen in the upcoming sections) are the DHCP (Dynamic Host Configuration Protocol), the DNS (Domain Name System) and the SMTP (Simple Mail Transfer Protocol).

- **Dynamic Host Configuration Protocol (DHCP):**
  This application protocol automatically provides TCP/IP configuration parameters (IP address, subnet mask, gateway, DNS server...) to devices when these are connected to an internetwork (Thomas, 2012). In other words, it manages and assigns the required parameters to new devices to become operative in the network.

- **Domain Name System (DNS):**
  As mentioned for the network layer, any host/device connected to the Internet is identified with an IP address. This is the case of any websites (i.e.: www.etseiat.upc.edu), because all of them lie on a host connected to Internet. Yet, websites are always accessed by its name or domain (www.etseiat.upc.edu). And this is possible thanks to the DNS protocol, which maps the web domain to its IP address placing a query to a DNS server, which replies with the IP address for the entered domain. Then the IP address for the website is transferred to the transport layer. Figure 17 depicts the procedure of the DNS protocol.

![Figure 17. DNS service procedure.](image)

Oriol Filló Claret
April 2016
26
- Simple Mail Transfer Protocol (SMTP)

SMTP is the format protocol that defines the Mail Transfer Agents (MTA) in the internet, which are responsible for sending the message (MTA client) and to receive the message (MTA server). In most occasions when sending an e-mail, SMTP is used two times, between senders and sender’s email server and between sender’s e-mail server and receiver (Forouzan, 2010). Figure 18 depicts the described schematic.

![SMTP service procedure](image)

*Figure 18. SMTP service procedure. Ref. TCP/IP protocol suite.*

6.2.2 Addressing

In the previous sections the different layers of the TCP/IP protocol suite have been described. In this section the addressing methods or codes that each of the layers utilizes are presented. This offers a more practical perspective of the layers and binds them to more familiar ideas from an industrial engineering perspective such as the IP address or port number, which helps to better understand the whole concept of each layer.

The addressing codes in the TCP/IP protocol suite are: the physical address, the logical address, the port and the application-specific address.

**Physical or Link address:** It is the data link layer address (lower level) and it is strongly related with each hardware device connected to the network. In some occasions the devices have this addresses imprinted in their communication cards. This address have authority within a subnetwork (LAN for example). An example of physical address is the MAC. In the same subnetwork there can not be two devices with the same MAC.
Logical address: The logical address belongs to the network layer and it is related with the communication software build upon the hardware, and therefore it is independent of the physical network. The logical address, differently to the physical address, is universal for all the Internet (not just for subnetworks) and each host in the internet is uniquely identified. The logical address in the Internet is the 32-bits well-known IP (IPv4). However, it is currently being introduced the IPv6, which is 128-bits and will allow to embrace the drastically mounting number of devices connected to Internet worldwide. As mentioned in Section 6.2.1 the ARP protocol in the network layer is responsible for identifying the MAC corresponding to a particular IP.

Port address: The logical address and physical address serve the purpose of transmitting data from the source to the destination host. However, in a device connected to a network there are usually multiple communications and processes going on simultaneously. For example, at the same time there can be an open communication from computer A to B using FTP, and parallel another communication from computer A to a host using SMTP. For that reason, every opened communication process requires an address to differentiate it from the rest. The address assigned to each process in TCP/IP protocol is known as port address, and it is 16 bits length.

Application address: As mentioned before, application layer is on top of the TCP/IP protocol stack and some applications may require an identifier in that level, which is known as application address. These are probably the most familiar addresses to general users of Internet and examples of them are the e-mail address (i.e.: oriol.fillo@gmail.com) or the URLs (i.e.: http://www.etseiat.upc.edu). As it is derived from the examples and from the layer name, these addresses are specific for the applications and can be much more diverse than the rest of addresses.
6.2.3 TCP/IP Protocol suite table summary

This section includes Table 3, which compiles the main features of TCP/IP protocol suite described in Section 6.2.

Table 3. TCP/IP protocol suite summary chart.

<table>
<thead>
<tr>
<th>Data Unit</th>
<th>Layer</th>
<th>Protocols</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Message</td>
<td>Application</td>
<td>FTP, DHCP, TFTP, DNS, SMTP,</td>
<td>Application (i.e.: e-mail</td>
</tr>
<tr>
<td></td>
<td></td>
<td>HTTP, ...</td>
<td>address)</td>
</tr>
<tr>
<td>Segment</td>
<td>Transport</td>
<td>TCP, UDP, ...</td>
<td>Port (port number)</td>
</tr>
<tr>
<td>Datagram</td>
<td>Network</td>
<td>IP, ICMP, IGMP, ARP</td>
<td>Logical (IP)</td>
</tr>
<tr>
<td>Frame</td>
<td>Data</td>
<td>Ethernet, Wireless, ...</td>
<td>Physical or Link (MAC,...)</td>
</tr>
<tr>
<td>Bits</td>
<td>Physical</td>
<td>Ethernet, Wireless, ...</td>
<td></td>
</tr>
</tbody>
</table>

6.3 Modbus

This section will describe what the Modbus standard is, the most relevant features for its implementation and the different modalities of the standard.

6.3.1 Origin and overview

Modbus was the first accepted fieldbus standard and it was released by Modicon PLC manufacturer (now Schneider Electric) in 1979 with the purpose to have a robust and ordered method to exchange data between PLC in production plants (Siemens Energy & Automation, 2012). It is an open-standard communication protocol based on master/slave architecture that sets the structure for the data messages transmitted between the nodes. There was no restriction as for the physical layer over which Modbus was implemented; originally it was conceived
to be implemented over RS-232 lines, but later it moved over RS-485 for all the advantages that this includes (see Section 6.1). With the advent of the Internet, new extensions of the standard appeared and are widely utilized as Modbus TCP/IP (Modbus, I, 2006). Nowadays, many different devices apart from PLCs and controllers are eligible for Modbus, for example sensors, data acquisition equipment or actuators. The main features that drove Modbus to become the most extended and implemented automation protocol are its flexibility, simple structure, ease of implementation and reliability (Siemens Energy & Automation, 2012).

6.3.2 General operation

The master/slave transmitting approach implies that within a network it is the master device (usually one per network) which always initiates the communication. The slaves (one or multiple) react to the master message and respond. The action carried out by the slave consists in performing the requested operation or sending back the requested data by the master device. Regardless of the transmission mode (serial, TCP/IP) the communication cycle and structure is the same. Figure 19 illustrates the Modbus data transmission cycle, the most important elements in the exchanged messages can be seen in it.

![Modbus Communication Cycle Diagram](image)

**Query:** The master initial message includes the address of the slave device and the function code, which tells the targeted slave device what is the action to perform. The data bytes contain any data required for the slave action (for example the value of a parameter/variable, or the parameters or variables to be read). Finally, the error check field is the tool to validate the integrity of the message contents.

**Response:** The expected response from the slave device includes an echo of the address field and function code. If there is an error, the function code is modified to indicate an error response. The data bytes contain any data that was required by the master in the query. Finally, the error check field allows the master to confirm the validity of the response.

### 6.3.3 Modbus addressing

A paramount parameter for Modbus protocol is the identification of each device (and more precisely each slave) within the network. This parameter serves the master to point to each specific slave. In serial lines Modbus this value ranges from 1 to 247, thus in a conventional Modbus serial network there can be up to 247 slave devices (RS-485 bus). The 0 value is reserved for broadcasting to all the devices along a network.

### 6.3.4 Modbus data storage: coils and registers

In the Modbus protocol the data transmitted to/from the slaves is stored in the memory locations of the slaves as arrays ('tables' in some literature). The memory locations in that arrays that contain the data values are organized in four different groups, depending on its supported functionalities and data types. Generally, each group can store up to 9999 values (this varies among devices). Table 4 summarizes the characteristics for this groups of data.

Coils: single bit memory locations in the slaves to/from which the master can write or read the value. Its indexing within the Modbus protocol ranges from 1 to 9999. As an example, it can be used to define the state of a LED.

Inputs: single bit memory locations in the slaves from which the master can read value. The indexing within the Modbus protocol ranges from 10001 to 19999. As an example, it can be used to monitor the state of an on/off sensor.
Table 4. Modbus data tables.

<table>
<thead>
<tr>
<th></th>
<th>Size</th>
<th>Functions</th>
<th>Modbus address</th>
<th>Application used address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coil</td>
<td>1 bit</td>
<td>Read Write</td>
<td>1-9999</td>
<td>0-9998</td>
</tr>
<tr>
<td>Input</td>
<td>1 bit</td>
<td>Read</td>
<td>10001-19999</td>
<td>0-9998</td>
</tr>
<tr>
<td>Input Registers</td>
<td>16 bits</td>
<td>Read</td>
<td>30000-39999</td>
<td>0-9998</td>
</tr>
<tr>
<td>Holding Registers</td>
<td>16 bits</td>
<td>Read Write</td>
<td>40001-49999</td>
<td>0-9998</td>
</tr>
</tbody>
</table>

Input Registers: 16 bit memory locations from which the slaves can read its values. The indexing of the input registers within Modbus protocol ranges from 30000 to 39999. As an example, it can be used to monitor an analog sensor value.

Holding registers: 16 bit memory locations in the slaves to/from which the master can write or read the value. The indexing of the holding registers within Modbus protocol ranges from 40001 to 49999. As an example, it can be used to monitor an analog sensor value.

6.3.5 Modbus functions

The Modbus functions are one of the main parameters included in the query from the master and in the reply from the slave (it is the same function in both cases, an echo). These functions are identified with a code (which are in the range of 1-255) and by them the master defines the action that the slave must perform. Not all devices support all the functions. The most common are shown in Table 5.
Table 5. Most common Modbus functions.

<table>
<thead>
<tr>
<th>Function code</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>01</td>
<td>Read coil</td>
</tr>
<tr>
<td>02</td>
<td>Read input</td>
</tr>
<tr>
<td>03</td>
<td>Read holding registers</td>
</tr>
<tr>
<td>04</td>
<td>Read input registers</td>
</tr>
<tr>
<td>05</td>
<td>Force single coil</td>
</tr>
<tr>
<td>06</td>
<td>Preset single register</td>
</tr>
<tr>
<td>07</td>
<td>Read exception status</td>
</tr>
<tr>
<td>15</td>
<td>Force multiple coils</td>
</tr>
<tr>
<td>16</td>
<td>Preset multiple registers</td>
</tr>
<tr>
<td>17</td>
<td>Report slave ID</td>
</tr>
</tbody>
</table>

6.3.6 Modbus Error Checking Methods

All Modbus messages, regardless of the mode, always include an error detecting procedure in order to ensure reliability of the communication. The implemented methods are based on redundancy check and the specific type of redundancy check varies depending on the Modbus version. For serial lines the redundancy checks applied are the Cyclical Redundancy Check (CRC) or Longitudinal Redundancy Check (LRC) (Modbus Organization, Inc., 1996). This procedure consists in a function that takes all the fields in the message as inputs and then outputs a value according to the inputs. The master device calculates this value and appends it to the sent message, the device calculates the CRC upon receiving the messages and if the value is different from what the master calculated an error is reported. The same will happen for the response message from the slave to the master.
6.3.7 Modbus modes

As previously mentioned, Modbus protocol can be implemented over different layers being serial lines or ethernet the most common. The differences between these versions of the protocol lie on the peculiarities of the physical interfaces (configuration, speed, reliability...), but not on the communication cycle and message structure, which are both maintained. This remains as one of the most important features of the protocol and has allowed to adapt it to the more powerful interfaces since it was firstly developed.

Figure 20 illustrates a common Modbus structure and remarks the blocks that are present in all versions of the protocol and the blocks that can be modified or removed. The Protocol Data Unit (PDU) is independent of the utilized communication layers (serial, ethernet, wifi...), yet the Application Data Unit (ADU) is the part of the structure adaptable to every mode or version of Modbus. The PDU contain the information that needs to be transmitted, which is the data (normally the registers/coils that need to be read or written ) and the function code. The ADU is the part of the structure that takes care of the devices addressing and the reliability of the communication (error checking). This part of the structure and the way by which the addressing is achieved varies for each Modbus mode.

![Modbus data frame](image)

*Figure 20. Serial Modbus data frame. PDU block is universal for all Modbus modes. Ref. Modbus.org.*

- Modbus over serial lines:

  This was the first implementation of Modbus and the serial layer can be either RS-232 or RS-485, this last more extended (Siemens Energy & Automation, 2012). In both cases there are two versions of the protocol:

  - Modbus ASCII
  - Modbus RTU (Remote Terminal Unit)

  The main difference is that in ASCII mode, the messages are transmitted in ASCII format, while in RTU in binary format. Binary format condensed the message and, thus, it allows higher data rates. Another difference is that for RTU the message has to be send as a continuous stream and ASCII allows 1 second time gaps. Modbus RTU utilizes 16 bit CRC as error checking procedure, while ASCII 8 bits LRC.
Aside from the mode of the protocol (RTU or ASCII), for Modbus over serial lines there are other parameters, related to the interfaces characteristics, that need to be set:

- Baud rate: the data transmission speed
- Parity: if none, even or odd parity is used in the data transmission
- Stop bits: the number of stop bits used in the data transmission
- Time out: the maximum time span to receive a response to a query, after this time an error is reported

The communication lines are exclusively dedicated to the protocol and the messages are sent over the whole network. Another version of Modbus over serial lines is Modbus Plus, which allows peer-to-peer communication (HMS Industrial Networks, 2006), (AEG Schneider Automation, 1996).

- Modbus over ethernet/wifi (TCP/IP):
  The implementation of Modbus over ethernet and wifi interfaces provides an upgraded version of the standard, which is encased within TCP/IP protocol and results in the widely-extended Modbus TCP/IP. It includes the advantages of TCP/IP and it allows non-dedicated networks, there can be different connections or protocols over the same interfaces.

Concerning the data framework structure, there are some subtle differences in the ADU for Modbus TCP/IP, compared to the serial versions. These are mainly in the addressing and the Cyclical-Redundancy-Check, as it is induced in Figure 21. For Modbus TCP/IP the error checking procedure is 32 bits CRC (Modbus, I, 2006).

![Figure 21. Modbus TCP/IP data frame. Ref. Modbus.org.](image)

In terms of implementation or application development, the only parameter needed to be defined in the Modbus TCP/IP network devices is the IP of the device. The communication port is standardized to 502 (Modbus, I, 2006).
6.3.8 Modbus RTU Message example

In order to better illustrate the idea of Modbus messages and structure, in this section an example of a Modbus RTU communication cycle for a specific function and slave is described.

Query: The request from the master is to write two values (10 and 258) to the 40002 and 40003 slave’s (address 17) holding registers. The Modbus RTU code sent can be seen in Figure 22.

![Figure 22. Modbus RTU master code (query) example. Ref. SimplyModbus.ca.](image)

Slave Address: The address of the slave device.
Function code: The code of the function.
Address of the first register: Is the address of the first register to write to. In this case 1 (according to Figure 22).
 Nº of registers: The number of registers to write to. In this case 2
 Nº of Data bytes: Each register has two bytes, thus 2 bytes x 2 registers =4 bytes
Value to register 40002: The value to write to register 40002.
Value to register 40003: The value to write to register 40003.
CRC: The Cyclical Redundancy Check parameter (16 bits).

Response:

![Figure 23. Modbus RTU slave code (response) example. Ref. SimplyModbus.ca.](image)

Slave Address: The address of the slave device.
Function code: The code of the function to perform.
Address of the first register: Is the address of the first register to write to. In this case 1 (according to Figure 23).
 Nº of registers: The number of registers written. In this case two.
CRC: The Cyclical Redundancy Check parameter (16 bits).
7 MODBUS RTU IMPLEMENTATION ON INDUSTRIAL SHIELDS PLCs

In this section it is described how the Modbus RTU is implemented in the Industrial Shields PLCs. A master and slave will be configured with two PLCs and they will write data to each other. Firstly, both, the master and the slave have been tested with a master and slave simulators which are installed in the PC. The next step was to connect both PLCs together to one as a master and the other as a slave.

7.1 Libraries

The Arduino IDE libraries that have been adapted to Industrial Shields PLCs and finally used is the SimpleModbus (Bester, 2012), another library that was explored but did not provide the best results or were not available as master is (Armengol, 2014). The slave version of SimpleModbus only includes holding registers, and can perform Modbus functions 3 and 16 (read/write multiple holding registers).

7.1.1 Modification of the library

The SimpleModbus libraries are designed to be used in an Arduino board. The standard library assumes that the Arduino board used have all its UART ports (SparkFun Electronics, 2013) available for communication. However, in the case of Ardbox and MDuino Industrial Shields PLCs, there is no UART port available because these are being internally used for other functionalities and cannot be modified as it is a final product and it is built in a specific configuration (see Section ANNEX C: Attached CD).

SimpleModbus integrates HardwareSerial library, which is only compatible with UART ports. In order to initialize and operate the serial ports, Industrial Shields PLCs use the library SoftwareSerial (arduino.cc, 2012), instead of the HardwareSerial. So as to make SimpleModbus compatible with Industrial Shields PLCs, both the .cpp and .h files, which are the files that define the Arduino libraries, needed to be modified replacing the HardwareSerial for SoftwareSerial.
7.2 Implemented code

This section presents the code to configure Industrial Shields PLCs as Modbus RTU master or slave.

7.2.1 Slave Code

This code example serves to describe the main features of the SimpleModbus library for a slave device and how to use it on Industrial Shields PLCs. In this simple program, the slave responds to the masters requests. There are two holding registers, in one of them the master writes a value and in the other one the master reads its value.

1. #include <SimpleModbusSlaveSoftwareSerial.h>
   #include <SoftwareSerial.h>

2. int SSerialRX =11 ;// RO //RS485 Receive pin
   int SSerialTX=10; // DI //RS485 Transmit pin
   int SSerialTxControl=7; // DE //RS485 Direction pin
   int GroundRE=4 ;// RE
   SoftwareSerial RS4 85Serial(SSerialRX, SSerialTX);

3. int i;
   int Q5=5;

4. enum
   { ONE,
     TWO,
     HOLDING_REGS_SIZE // leave this one
   };

5. unsigned int holdingRegs[HOLDING_REGS_SIZE]; // function 3 and 16 register array

1. In this part the libraries used in the sketch are declared.
2. RS-485 pins and port (SoftwareSerial) are defined.
3. This are the two variables used in this sketch, Q5 is an output of the PLC.
4. This enum type defines all the holding registers that the slave will have, in this case only two.
5. The holding registers array is defined ‘holdingRegs’.
void setup()
{
6. pinMode(SSerialTxControl, OUTPUT);
   pinMode(GroundRE, OUTPUT);
   digitalWrite(GroundRE,LOW);
7. modbus_configure(&RS485Serial, 9600, SERIAL_8N1, 1, SSerialTxControl,
   HOLDING_REGS_SIZE, holdingRegs);
   modbus_update_comms(9600, SERIAL_8N1, 1);
8. pinMode(Q5, OUTPUT);
    i=10;
}

6. The RS-485 pins of the Arduino are set to either input or output. The RE pin is set to LOW because in the control is performed only by the DE MAX485 pin, which corresponds to SSerialTX Section 6.1.3.
7. The Modbus is configured with ‘modbus_configure’. The parameters needed in this function are:
a. RS-485 port.
b. Baud rate of the serial line.
c. Byte forma. Nº start/stop bits, Nº of data bits,type of parity. Parity and number of stop bits. 1 start bit, 1 stop bit, 8 data bit, no parity.
d. The slave ID.
e. The direction pin or control pin ‘SSerialTxControl’, which will drive RS-485 transceiver to transmit or receive mode.
f. The number of holding registers.
g. The array containing the holding registers.
8. Q5 is configured as an output and i variable is initialized.

void loop()
{
9. modbus_update();
10. holdingRegs[ONE]=i;
11. if (holdingRegs[TWO]==1){
      digitalWrite(Q5, HIGH)
   else{
      digitalWrite(Q5, LOW)
}

9. The modbus_update() functionis responsible for updating the holding registers values at each SCAN cycle of the PLC.
10. The first holding register is set to i value, in this example is kept at 10, but it could be replaced for an input value and the holding register would be updated with the variable value at each cycle. This value will be available for any master that reads the first holding register.

11. In this case the second holding register is aimed to drive Q5 value from the master. If the master writes a 1 to the holding register ‘TWO’, the slave will set Q5 to high.

### 7.2.2 Master code

The code for the master device is described here, the parts regarding the serial communication and Arduino IDE overhead will be equal to the slave code.

1. ```
#include <SimpleModbusMasterSoftwareSerial.h>
#include <SoftwareSerial.h>

2. ```
#define baud 9600
#define timeout 1000
#define polling 200 // the scan rate
#define retry_count 10

3. ```
int SSerialRX =11 ; // RO  //RS485 Receive pin
int SSerialTX=10; // DI  //RS485 Transmit pin
int  SSerialTxControl=7;  // DE  //RS485 Direction pin
int GroundRE=4 ; // RE
SoftwareSerial RS485Serial(SSerialRX, SSerialTX);
```

4. ```
int i;
enum
{
 PACKET1,
 PACKET2,
 TOTAL_NO_OF_PACKETS // leave this last entry
};
Packet packets[TOTAL_NO_OF_PACKETS];
```

1. Definition of the libraries used in the sketch.
2. Definition of parameters for the serial communication and Modbus.
3. RS-485 pins and port (SoftwareSerial) are defined.
4. Variable declaration (i) and declaration of other elements (Packets and Registers). More information on how packets work can be found in the library reference (Bester, 2012). The packets are not something specific to Modbus, but rather to this library. In the next part of code there is an explanation on what they are. The regs array is the variable where all the values to be written to and read from the slave through Modbus protocol are stored.

```c
void setup()
{
  5. pinMode(SSerialTxControl, OUTPUT);
  pinMode(GroundRE, OUTPUT);
  digitalWrite(GroundRE, LOW);

  6. modbus_construct(&packets[PACKET1], 1, READ_HOLDING_REGISTERS, 0, 1, 0);
      modbus_construct(&packets[PACKET2], 1, PRESET_MULTIPLE_REGISTERS, 1, 1, 1);
      modbus_configure(&RS485Serial, baud, SERIAL_8N1, timeout, polling, retry_count, SSerialTxControl, packets, TOTAL_NO_OF_PACKETS, regs);
}
```

5. This part is the same as part 4 for the slave code.
6. In this part, the packets defined in 4. are created. As it can be seen, each of the packets are constructed configured one by one. The elements in the modbus_construct function are as follows:
   a. The pointer to the declared packet that is being built.
   b. The slave ID.
   c. The function to be performed by the slave.
   d. The initial holding register address in the slave to write to/read from (0-9998).
   e. The number of holding registers to write to/read from.
   f. The index in the master registers array (regs) where the variable read or written is stored.

   The next function in the sketch is modbus_configure and it is very similar to the same function in the slave.
   a. The pointer to the serial port.
   b. The baud rate.
   c. Byte format. Nº start/stop bits, Nº of data bits, type of parity and number of stop bits (see Section 6.3.7) 1 start bit, 1 stop bit, 8 data bit, no parity.
   d. The timeout value.
   e. The polling rate.
   f. The time to retry communication.
   g. The direction pin for the RS-485 transceiver.
   h. The variable that contains the packets (packets).
i. The total amount of packets.

j. The array that stores the registers values (regs).

```c
void loop()
{
  7. modbus_update();
  8. Serial.print(regs[0]);
     Serial.println("/");
     delay(1000);
  9. regs[1]=i;
     i=i+1;
}
```

7. As in the slave code, the `modbus_update()` function is responsible for updating the registers values at each SCAN cycle (data is sent and received via Modbus to the slave).

8. In this case the value stored in the first position of the regs array is printed through the serial monitor of the Arduino IDE. It corresponds to the value read from the slave defined in ‘PACKET1’.

9. The value of i is stored in the second position of the ‘regs’ array. This value will be written to the slave as defined in ‘PACKET2’. After that the i value is updated.
8 MODBUS RTU FINAL IMPLEMENTED CODE

In this section the code for the PLC to interact with the data acquisition device Seko (see Section ANNEX C: Attached CD) is developed. The code will be very similar to the one described in the previous sections (7.2.1, 7.2.2). However, here it has been adapted to target the specific holding registers of the Seko device that contain the data of interest. The holding registers to be targeted are 1000 (Ph), 1001 (Chloride) and 1002 (Orp).

Table 6. Holding Registers. Ref. Seko Kontrol 800ML.

<table>
<thead>
<tr>
<th>Description</th>
<th>Property</th>
<th>Range</th>
<th>Operating state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>pH Measure</td>
<td>Read 0 to 1400</td>
<td>Full Operating</td>
</tr>
<tr>
<td>1001</td>
<td>Cl Measure</td>
<td>Read 0 to 500</td>
<td>Full Operating</td>
</tr>
<tr>
<td>1002</td>
<td>Orp Measure</td>
<td>Read -2000 to 2000</td>
<td>Full Operating</td>
</tr>
<tr>
<td>1003</td>
<td>Temperature Measure</td>
<td>Read 0 to 1050</td>
<td>Full Operating</td>
</tr>
</tbody>
</table>

The slave device ID it has been assumed to be “1”. The Seko device supports the “Report Slave ID” command that returns its ID. In the installation this would be done beforehand so as to acquire the slave ID.

```c
#include <SimpleModbusMasterSoftwareSerial.h>
#include <SoftwareSerial.h>
define baud 9600
#define timeout 1000
#define polling 200 // the scan rate
#define retry_count 10

int SSerialRX = 11; // RO //RS485 Receive pin
int SSerialTX = 10; // DI //RS485 Transmit pin
int SSerialTxControl = 7; // DE //RS485 Direction pin
int GroundRE = 4; // RE
SoftwareSerial RS485Serial(SSerialRX, SSerialTX);

int ph;
int chl;
int orp;

enum
{
  PACKET1,
  PACKET2,
  PACKET3,
  TOTAL_NO_OF_PACKETS // leave this last entry
};
Packet packets[TOTAL_NO_OF_PACKETS];
define TOTAL_NO_OF_REGISTERS 3
unsigned int regs[TOTAL_NO_OF_REGISTERS];
```
void setup()
{
    pinMode(SSerialTxControl, OUTPUT);
    pinMode(GroundRE, OUTPUT);
    digitalWrite(GroundRE, LOW);
    modbus_construct(&packets[PACKET1], 1, READ_HOLDING_REGISTERS, 1000, 1, 0);
    modbus_construct(&packets[PACKET2], 1, READ_HOLDING_REGISTERS, 1001, 1, 1);
    modbus_construct(&packets[PACKET3], 1, READ_HOLDING_REGISTERS, 1002, 1, 2);
    modbus_configure(&RS485Serial, baud, SERIAL_8N1, timeout, polling, retry_count, SSerialTxControl, packets, TOTAL_NO_OF_PACKETS, regs);
}

void loop()
{
    modbus_update();
    chl=regs[0];
    ph=regs[1];
    turb=regs[2];
}
9 TCP-IP PROTOCOL IMPLEMENTATION

In this section it is described the communication via TCP between the main controller (M-Duino) and the PC that will host the HMI and work as a server (see Section 4.2.1). The M-Duino will work as a client that connects to a TCP server (PC). The server has been developed using LabView, and it is integrated in the same application as the HMI (developed with LabView as well).

9.1 M-Duino TCP-IP client configuration

The library utilized in this application with the M-Duino to develop the TCP communication is the EthernetUIP (Truchsess, 2015). This is one of the libraries compatible with the Ethernet board integrated in the M-Duino (ENC28J60 see Section ANNEX C: Attached CD). This library has some examples to configure an Arduino as a TCP client or server.

9.1.1 M-Duino TCP-IP implemented code

In the following paragraphs it will be presented the code used to create a TCP client that connects to a TCP server.

1. `#include <UIPEthernet.h>`
2. `EthernetClient client;`
3. `uint8_t mac[6] = {0x00, 0x01, 0x02, 0x03, 0x04, 0x05};`
   `IPAddress myIP(192, 168, 1, 245);`
   `IPAddress dnsServer(192, 168, 1, 1);`
   `IPAddress gateway(192, 168, 1, 1);`
   `IPAddress subnetmask(255, 255, 255, 0);`
4. `String mes="Server data";`
   `String iter="";`
   `String message;`
   `int i;`

1. Declaration of the library EthernetUIP.
2. An ‘Ethernetclient’ type is created and given the ‘client’ name.
3. In this part the required parameters for a device connected to the Internet network (see Section 6.2.1) are defined.
   a. The value of the MAC must be an hexadecimal format value unique in the LAN network to which the PLC will be connected.
   b. The desired IP address of the device.
   c. The IP of the device that has a DNS server associated (in this case is the router of the LAN).
   d. The IP of the gateway device (it is the router of the LAN as well).
   e. The submask (it is provided by the router as well).

4. This part is not related with the communication itself but rather with the message. The String _message_ contains the data sent to the server. _mes_ and _iter_ contain the two parts of themessage that will be appended together in the next lines. _iter_ will be updated at each cycle with an increasing integer value.

```c
void setup() {
  Ethernet.begin(mac, myIP, dnsServer, gateway, subnetmask);
  client.connect(IPAddress(192, 168, 1, 200), 5020);
  Serial.println("Connecting...");
  client.println("Hello Server");
  while (!client.connected())
  {
  
  }
  if (client.connected()) {
    Serial.println("Connect SUCCESS");
  }
}
```

5. The _Ethernet.begin()_ function initializes and starts the Ethernet port on the ENC28J60 communication board in the M-Duino. The parameters in the function are the ones declared in 3.

6. The _connect()_ function tries to establish a connection between the M-Duino and the port address 5020 of the device with the IP 192.168.1.200.

7. If the connection is established the M-Duino will print ‘Hello Server’ to the server device.

8. This part of the code serves to check if the connection with the server is real. If the client is connected to the server (_client.connected()_ it will be seen on the serial Monitor ‘Connect SUCCESS’.
9. In this part the values from point 4 are integrated. \textit{iter} is updated to a new value and the \textit{message} is constructed. The message will have the following structure ‘Server data5’.

```c
void loop() {
    \texttt{9. iter=String(i);} \\
    \texttt{message=mes+iter;} \\
    \texttt{i=i+1;} \\
    \texttt{10. client.print(message.length());} \\
    \texttt{client.print(message);} \\
    \texttt{while (!client.available()) {}} \\
    \texttt{11. while (client.available() > 0) {}} \\
    \texttt{c = client.read();} \\
    \texttt{Serial.print(c);} \\
}
```

10. Once the communication has been established the client will send two more messages to the server. The first message will be an integer (\texttt{message.length()}) containing number of bytes that are going to be transmitted in the upcoming message (each character is a byte). The second message will be the data to be transmitted. It will have the number of bytes specified in the first message.

11. Once the server has received the data, the client waits for the master to reply with a message. The data that the server sends to the M-Duino is stored in the client buffer. The data must be retrieved or read from the buffer of the client to be used. \texttt{client.available()>0} will be true while there are characters in the client buffer to be read. The \texttt{client.read()} function reads the first character in the buffer and removes it from it. In the next iteration it will read the following character and this will go on until no more characters are available in the buffer.
9.2 Labview TCP-IP server configuration

LabView includes several blocks on its palette for TCP communication. A simple example on how its library works is described (National Instruments Corporation, 2015).

Figure 24 shows the basic TCP server configuration LabView will have. The first block will listen for a connection on a specific port. Once there is a client connected the server will read whatever the client is sending (it can be a request for example) and it will respond with a message back to the client. Finally, the connection is closed.

![Figure 24. TCP server LabView configuration.](image)

Each block may have several possible parameters as inputs and as outputs as well. Figure 25 shows all the possible parameters; information on those is fully detailed in (National Instruments Corp., 2011).

![Figure 25. TCP LabView blocks parameters.](image)

With the basic blocks and configuration in mind (Figure 25), it will be much easier to understand the next TCP server configuration. This configuration shows a further developed example of a TCP communication that will interact with the TCP client described in Section 9.1. The whole code of the server is shown in Figure 26.
1. The TCP Listen block (Figure 27) initiates the server and listens for communications on port 5020. The IP of the server will be the same as the host PC, in this case 192.168.1.200 (Figure 28).

2. The first Read TCP block (Figure 29) is aimed to read the first (and non-recursive) message sent by the client, which was in Section 9.1.1, ‘Hello Server’.
The message will be shown in the *Communication* string indicator blog. The number of incoming bytes is set to 15 (bytes that the block will expect at most) and the mode is ‘immediate’, by which the block will wait until any bites from the total amount expected arrive.

3. This part is the recursive structures of the code and it will be repeated until the client sends a stop token (‘Out’) to the server. In this structure there are two read blocks (Figure 30). The first one will wait for a two byte number indicating the number of bytes that the upcoming message will have. The second block (with the information from the previous block) will read the whole message sent by the client. As mentioned, this structure will go on until the ‘Out’ message is sent by the client.

4. Finally, the connection is closed with the TCP Connection Close block.
9.3 TCP-IP Implemented between M-Duino and LabView

Once the Arduino IDE code is generated and uploaded to the M-Duino and the LabView project is open the communication between both devices can begin. Ethernet cable must be connected from the M-Duino to the LAN router and the PC is also connected via Ethernet cable to the PC. Figure 31, Figure 32, Figure 33 and Figure 34 show how data is transmitted from the M-Duino to the PC. The code for the client and server of this application can be found in Section ANNEX C: Attached CD.

*Figure 31. Hello World from the M-Duino (client) to LabView-PC (server).*
Figure 32. M-Duino (client) sends iteration of data 6 ‘Server data6’. LabView response: ‘Hello Client’

Figure 33. M-Duino (client) sends iteration of data7 ‘Server data7’. LabView response: ‘Hello Client’
Figure 34. M-Duino (client) sends iteration of data 9 'Server data9'. LabView response: 'Hello Client'
10 DEVELOPMENT OF THE HMI

This section will describe the functionalities present in the HMI and the LabView code developed in order to implement them.

10.1 Variables

Figure 35 shows the overview of the HMI and the several menus, and blocks that it has. It can be seen that there is a representation of the water tank and the several ducts, pumps and valves. The water tank has an intake duct for incoming water and an outtake duct for water going into the save water town system.

Among all the variables in the graphical representation there are some which its value can modified and others that only serve monitoring purposes. The variable values that can be forced from the HMI are marked in a blue label, so as to differentiate them from monitoring values (yellow labeled).

**Forced variables**

In this graphical part of the system there are five variables that can be modified from the HMI.

1. The three pump-ducts valves, which can provide isolation between the water tank and the pump dosing system.
2. The intake and outtake water valves opening degree.

**Non-forced variables**

In the same graphical representation of the system there are some other elements that provide information about several variables so as the operator can have a rapid visual overview of the system. The following variables are the ones represented.

1. Dosing pumps operating point (0-100%).
2. Water level in the tank.
3. Intake and outtake water flow.

The values of these variables are forwarded from the PLC to the HMI and displayed. The PLC is responsible for either, calculating the values (pump operating point) or retrieving them from the acquisition equipment (water flow, tank level).
**Color and motion state**

The graphical elements present in the HMI have been programmed to modify its colors or its motion state according to their current value.

- **Valves.** Green color for open state and red color for close state.
- **Ducts.** Gray color for no-flow state and blue color for flow state.
- **Pumps.** Green color for on state (operation point different from 0%) and red color for off state. On the pumps icon there is also a fan (representing the inner turbine) that will modify its spinning speed according to the pump operating point.

**10.2 Control panels**

The second main area of the HMI includes a panel embracing three tabs (Figure 36), which include various functionalities such as alarm configuration and monitoring, emailing of alarm events and data storage. The three tabs are described in the following paragraphs.

- **Controls**
  - First of all it shows the TCP/IP communication parameters from the PLC (IP and remote port) communication.
  - Secondly there is a display of alarms that are triggered whenever ph, chloride and turbidity levels are surpassed. The system distinguishes if the alarm is due to a high level or a low level of the variable.
  - Another relevant block of this tab is the ‘Water parameter conditions’ which displays the chloride, ph and turbidity levels and allows setting the upper and lower limits for these three variables.
  - Finally, there is a led indicating if data is being saved or not (which is something that will be explained in the upcoming paragraphs).

- **Save**
  - This tab allows the configuration of a folder where .txt files will be created and stored daily with all the values of the system. It can also be configured the frequency of data saving. The .txt files name will be DD.MM.YYYY. A new file is created daily.
- **Alarms notifications**
  - This tab allows the configuration of a *Gmail* account from which messages will be sent whenever an alarm (see Section 0) is triggered. There is an activation button for the e-mailing functionality and a frequency setting (how often the email is resent when the alarm is still on). The email will include an indication of the low or high level of the variables and the .txt datafile (described in previous point) so as to have more information of each variable in the system and also the dynamics of the parameters derived from the cumulated values in the file.
Figure 35. LabView HMI layout.
Figure 36. HMI tab menus.
10.3 HMI implemented code

In this section it is described the developed LabView code lying below the HMI that sustains the functionalities presented in Sections 10.1 and 10.2.

The whole project code encompasses several while blocks, each one having its own purpose and being responsible for one of the multiple functionalities and tasks present in the HMI. The LabView file containing the HMI and code is found in Section ANNEX C: Attached CD.

10.3.1 Visual logic blocks

The first block to be described is responsible for the graphical part related to the dosing valves, ducts, pumps and fan. Its purpose is to modify the color or movement state of these elements depending on the state of the application. Therefore, this block does not hold any logic related to the application itself, but rather to the presentation and visual effects of the HMI. A better comprehension of this part of the code will be achieved after reading Section 10.4.

The inputs of this block is the operating point of the pump (PumFloc) and the state of the floculant valve (cV_Floc). The main logic for this while loop is encased inside the SubVI 5, which is shown in Figure 37.

![Visual logic blocks code](image)

Figure 37. Visual logic blocks code. Responsible for modifying the color and motion state of valves and pumps in the HMI.
The part of the code blocks conformed by DBL variables is responsible for the regulation of the switching rate between the two states of the Fan_Floc Boolean indicator, so as to achieve the sense that the fan is spinning and that the speed increases or decreases (according to the control needs of the system), the schematic is shown in Figure 38. As mentioned, this is better illustrated in Section 10.4.

![Figure 38. SubVI5. Contain the code that modifies the motion state of the pumps and colors of valves, ducts and pumps.](image)

**10.3.2 Data to PLC**

This *while* loop takes the values from the HMI that the user can force and adapts it into a single string which will be used in a different block to sent data to the PLC. Thus, the main responsibility of this code is to collect the data to be transmitted to the PLC and cluster it in a single string, which is the data type expected by the communication algorithm (see Section 9.3). The parameters collected in this block are:

- Chloride, Ph and turbidity limits.
- Dosing and water valves state.

This loop is shown in Figure 39.
Figure 39. HMI Parameters compiled in String format for transmission to the PLC.

The logic in this loop lies in SubVI 7. Figure 41 shows the code inside that block. There is a first step where all the input variables (booleans and doubles) are converted into a String type and the number of decimal values can be cut down to 1 or none depending on the needs. The block Round! (Figure 40) is responsible for this task. In the second stage of this SubVI all the input variables (already converted into Strings) are compiled in a single String, each value separated by a 'c' character, which is used as a delimiter in this project.

Figure 40. Round! block. Converts DBL into a String with the set number of decimal values.
Figure 41. SubVI7. SubVI contained in the Data to PLC block. It converts DBL and Boolean variables into a single string.
10.3.3 Data from PLC

This *while* loop is also a data treatment block, similar to the previous one. Here, the data coming from the PLC is unbundled and each of the values and data is converted into a suitable LabView variable or value. All these variables are used in different while loops to perform several functionalities.

Figure 42. Data from PLC block. Data coming from PLC is converted in suitable LabView format types for the application.

10.3.4 Water tank

This part of the code takes over the control of the water tank representation. In this part the opening degree of the valves is collected. Furthermore, the tank level, the intake and outtake flow is updated here. The color modification of the intake and outtake water valves is performed here as well. Figure 43 shows this loop.
10.3.5 Saving data

This *while* loop (Figure 44) is aimed to store data from the system or application into a .txt file. Every day a new .txt file is generated (`DD.MM.YYYY.txt`). First of all one of the main features of this block is that it allows adjustment of the savings frequency.

1. In terms of code, this derives in how often the while loop is repeated. This is achieved with the *Wait Until next ms* VI. This part of the code allows the user to adjust the frequency (in minutes) at which data is stored.

2. This blocks serve to create the string that will contain the data to be stored. First of all the current time is obtained by the *Get Day/Time in seconds* block. This is converted into date and time values. After that, the cluster *Data from PLC* (Section 10.3.3) is converted into an array and appended to the date and time. In order to format the array the *Array to Spreadsheet String* block is used.

3. In this part of the loop the folder path and file name where the files are to be stored is generated. It takes the current date (so as to name the file) and the path to the folder (which is set by the user, *Save* menu Section 10.2).

4. Finally, the code to create or open the files and store the data is conformed by the blocks, *Open/Create/Replace file*, *Set file Position*, *Write to Text File* and *Close File*.

In order to make the functionality optional, all the described code is contained in a case structure which is triggered only if the *Save data* button is active (*Save* menu Section 10.2).
The saved data contains the following parameters:
Timestamp / Date / Time / Incoming data from PLC (see Section 11.1)
*number of seconds that have elapsed since 12:00 a.m., Friday, January 1, 1904, Universal Time

**Figure 44.** Saving data block. This block saves data to a .txt file in the specified path.

### 10.3.6 Alarms and e-mailing

This is the part of the code (Figure 45) that takes care of visually reporting the alarms and sending e-mails if wanted.

1. All the needed information for the emailing is collected and clustered in this blocks. As it can be seen, the e-mailing requires a GMAIL account to be able to send messages. The cluster of data is used in the part 2 of the while loop.
2. This is the part of the code that triggers the alarms. Depending on the `Alarm_Chlor` value (which is sent from the PLC), the system will set off a high or low level alarm for the chloride (Alarms notification menu in Section 10.2). In order to do so, a Case structure is used. The `Alarm_Chlor` values are the following:
   - Case 0: Low level.
   - Case 1: High level.
   - Case 2: No alarm.
The part 2 of the code is reproduced in the same way for the pH and turbidity. Within the Case structure there is also the e-mailing code, contained in SubVI 1. Figure 46.

![Figure 45. Alarms and E-mailing block. Triggers an alarm and sends e-mails if wanted.](image)

**E-mail SubVI**

The emailing code (Figure 46) is structured in several parts which are described below.

1. These blocks set the frequency at which the e-mail is resent if the alarm persist.
2. This is the data treatment part of the SubVI, where the input cluster is unbundled and from it the required local variables are created, which will be used in other parts of the SubVI.
3. This is the main part of the SubVI, where the email message is created, the sender’s and receiver’s accounts are configured and the files attached.
4. This is a complementary part for the previous point. These blocks create the file path that contains the .txt with the data of the system. This crated path is used in part 3.
5. This part of the code accesses the Gmail account of the sender with its credentials, so that the e-mail is sent. Two important features in this part are the *Smtp client host* and *port*. According to (Google, 2012) and (Arduino Robotics, 2014) the Gmail host should be smtp.gmail.com and the port 587, this was proved by empirical methods to be the suitable configuration. Yet, in order to allow full access for LabView applications to Gmail accounts the security access needs to be set to “low security apps” described in (Google, 2011). This part of the code makes use of SMTP service (Section 6.2.1 and 6.2.3).
Figure 46. LabView code for sending emails with attached files from a GMAIL account.
10.3.7 TCP/IP communication to PLC

This part of the code takes care of the data exchange between the PLC and the HMI (Figure 47). The TCP communication blocks and code has been described in Section 9 and in the application it is applied in the same way (same data treatment). The difference lies in the set of data that is written to the PLC and that is read from it.

1. The data send to the PLC is the string ‘Data to PLC’ which was created in Section 10.3.2.
2. The data received from the PLC as a String is converted in the Cluster Data from PLC which was unbundled in Section 10.3.3. The SubVI 5 is responsible for adapting the input string from the PLC to a cluster.

SubVI 5

This sub VI (Figure 48) converts the string received from the PLC to a cluster of several strings and data types.

1. The block Spreadsheet String to Array is used to split the incoming string (which contains the character ‘c’ as a delimiter) and create an array of strings, each one being a specific value.
2. In the second main part of the code each of the strings are given tags and bundled into the output Cluster.
Figure 47. TCP/IP communication to PLC block
Figure 48. SubVI 5. Converts a string containing all parameters into a Cluster of parameters.

10.4 DSC Module

In this section it is described how the LabView controls such as valves, pumps or pipes have been customized, and also how to use some of the resources that the DSC Module provides.

10.4.1 DSC Module image navigator

The DSC Module image navigator is accessed from the LabView main panel, Tools/DSC Module/Image navigator. There are a great deal of categories and different kinds of images. Figure 49 and Figure 50 show the pump and the valve used in the HMI of this project.
Figure 49. DSC Module 3-D Pushbuttons category.

Figure 50. DSC Module Pumps category.
10.4.2 Creating a customized indicator

The steps to follow to create customized indicators, such as the ones in this project, are described below.

1. Create a new project with LabView and within it, create a new Control Figure 52.

2. In the Image navigator showed before, the color and some other characteristics of the icons can be modified. This is done by right-clicking and selecting Symbol options (Figure 53 and Figure 54).

Figure 51. Valves category.

Figure 52. Creating customized control.
Figure 53. Selecting control type.

Figure 54. Customizing control.
3. To the created New Control VI, a regular Boolean indicator from the Structures palette is added. Furthermore, the icon that we want to use in our control is copied/paste into the same panel in two different colors (one for FALSE state and the other one for TRUE state).

![Figure 55. New control creation.](image)

4. Modify the control VI mode to Customize (by clinking on the wrench icon shown in Figure 55). Once in that mode the red valve icon will be attached to the FALSE state of the Boolean indicator. This is done by copying the red icon to clipboard and, on the Boolean Import Picture from Clipboard (Figure 56). In order to set the TRUE state of the Boolean as a green valve, the same steps apply. It is just required to switch the Boolean from the FALSE state to the TRUE by toggle it using the Finger tool from Tools Palette. Figure 57 shows the customized Boolean.

![Figure 56. Attaching customized image to boolean control](image)
5. Once the new Control is created this must be saved in any directory. We will access this same directory from the LabView projects when the created control is required (Figure 58, Figure 59 and Figure 60).

Figure 57. Customized valve control.

Figure 58. Save the created customized control.
6. The fan animation (Figure 61) is created in the same way described above. An enlightening reference for the fan or any customizable indicator is (IndView, 2014).
11 INTEGRATION OF THE SOLUTION TO THE PLANT

This section provides an overview of the M-Duino code used to perform the automation services of the plant (updating the pumps and valves state and reading the sensor values). In previous sections, it has been described the Arduino code for programming Modbus RTU and TCP-IP on the M-Duino. The whole script is contained in Section ANNEX C: Attached CD. The wiring diagram for the plant is also presented in this section.

11.1 PLC Programming code

The code to be implanted in the final plant will contains the following elements.

Inputs and outputs

- **Q1.7**: Analog output that controls the valve for water intake (0-10V/8 bits).
- **Q1.6**: Analog output that controls the valve for water outtake (0-10V/8 bits).
- **Q0.7**: Analog output that controls the chloride regulation pump (0-10V/8 bits).
- **Q0.6**: Analog output that controls the pH regulation pump (0-10V/8 bits).
- **Q0.5**: Analog output that controls the turbidity regulation pump (0-10V/8 bits).
- **Q0.4**: Digital output that controls the chloride valve (0-24V).
- **Q0.3**: Digital output that controls the flocculant valve (0-24V).
- **Q0.2**: Digital output that controls the acid valve (0-24V).
- **I0.7**: Analog input from flow in meter (0-10V/10 bits).
- **I0.9**: Analog input from flow out meter (0-10V/10 bits).
- **I0.11**: Analog input from tank level (0-10V/10 bits).

Variables

**Float limits[]**: array of 6 float numbers that contains the lower and upper limits of the chloride, turbidity and pH.

0. Min Turbidity
1. Max Turbidity
2. Min Chloride
3. Max Chloride
4. Min pH
5. Max pH
**Int valves[]**: array of 3 integers that defines the state (closed=0/opened=1) of the valves:

- 0. Chloride
- 1. Flocculant (for turbidity)
- 2. Ph

**Int wat_valves[]**: array of 2 integers that define the state of the water flow in and out of the tank.

The string of data received from the TCP server has the structure in Table 7. Data from HMI to PLC string structure. The character ‘c’ is used as a delimiter between each value. This string is generated in the HMI (Section 10.3.2) and received by the PLC.

**Table 7. Data from HMI to PLC string structure**

<table>
<thead>
<tr>
<th>Pos</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>limits[0]</td>
<td>Turbidity low level limit</td>
</tr>
<tr>
<td>2</td>
<td>limits[1]</td>
<td>Turbidity high level limit</td>
</tr>
<tr>
<td>3</td>
<td>limits[2]</td>
<td>Chloride low level limit</td>
</tr>
<tr>
<td>4</td>
<td>limits[3]</td>
<td>Chloride high level limit</td>
</tr>
<tr>
<td>5</td>
<td>limits[4]</td>
<td>Ph low level limit</td>
</tr>
<tr>
<td>6</td>
<td>limits[5]</td>
<td>Ph high level limit</td>
</tr>
<tr>
<td>7</td>
<td>valves[0]</td>
<td>Chloride valve</td>
</tr>
<tr>
<td>8</td>
<td>valves[1]</td>
<td>Flocculant valve</td>
</tr>
<tr>
<td>9</td>
<td>valves[2]</td>
<td>Acid valve</td>
</tr>
<tr>
<td>10</td>
<td>wat_valves[0]</td>
<td>Water in valve</td>
</tr>
<tr>
<td>11</td>
<td>wat_valves[1]</td>
<td>Water out valve</td>
</tr>
</tbody>
</table>

**String s_Var[]**: Array of 3 strings which contain chloride, ph and turbidity values in a string type. This format will allow data to be send via TCP/IP as a string.

- 0. Chloride
- 1. Ph.
- 2. Turbidity

**String s_Pump[]**: Array of 3 strings containing pump values. This format will allow data to be send via TCP/IP as a string.

- 0. Chloride.
- 1. Ph.
- 2. Turbidity.
**Float float_Pump[]:** Array of 3 floats, the same as s_Pump but with a different type.

**String s_Tank[]:** Array of 3 strings that contain the flow in/flow out and tank level.

0. Flow in
1. Flow out
2. Tank level.

Array of 3 integers that contain the same values as s_Tank with a different type.

**String alarms[]:** Array of 3 strings that contain the alarm state for chloride, ph and turbidity levels.

**String reader:** String used to read the values coming from the HMI.

The values maybe 0, 1 or 2.
0→ level below lower limit
1→ level above upper limit
2→ level within range

0. Turbidity
1. Ph
2. Chloride.

**String message:** It is the string that contains all values sent to the HMI via TCP/IP. It is created in the PLC and received by the HMI (see Section 10.3.3.)


---

**Table 8. Data from PLC to HMI string.**

<table>
<thead>
<tr>
<th>Pos</th>
<th>Variable</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>s_Var[0]</td>
<td>Chloride level</td>
</tr>
<tr>
<td>2</td>
<td>s_Var[1]</td>
<td>Ph level</td>
</tr>
<tr>
<td>3</td>
<td>s_Var[2]</td>
<td>Turbidity level</td>
</tr>
<tr>
<td>4</td>
<td>s_Pump[0]</td>
<td>Chloride pump setpoint</td>
</tr>
<tr>
<td>5</td>
<td>s_Pump[1]</td>
<td>Ph pump setpoint</td>
</tr>
<tr>
<td>6</td>
<td>s_Pump[2]</td>
<td>Turbidity pump setpoint</td>
</tr>
<tr>
<td>7</td>
<td>s_Tank[0]</td>
<td>Water flow in</td>
</tr>
<tr>
<td>8</td>
<td>s_Tank[1]</td>
<td>Water flow out</td>
</tr>
<tr>
<td>9</td>
<td>s_Tank[2]</td>
<td>Water tank level</td>
</tr>
<tr>
<td>10</td>
<td>alarms[0]</td>
<td>Turbidity alarm</td>
</tr>
<tr>
<td>11</td>
<td>alarms[1]</td>
<td>Chloride alarm</td>
</tr>
<tr>
<td>12</td>
<td>alarms[2]</td>
<td>Ph alarm</td>
</tr>
</tbody>
</table>
11.2 Solution schematics
F1 - Water-in flow meter
F2 - Water-out flow meter
L - Water level
V1 - Chloride valve
V2 - Flocculant valve
V3 - Acid valve
V4 - Water-in valve
V5 - Water-out valve
P1 - Chloride pump
P2 - Acid pump
P3 - Flocculant pump

REFERENCE
12 BUDGET AND ECONOMICAL FEASIBILITY

The budget to implement the solution developed in the present work for a similar installation is exposed below. It must be clear that these are not the cost and timings of the development of this study, which would correspond to R+D division. The items included are the units involved in the control and automation of the system, in other words, the controller and the developed software.

Table 9. Budget for implementation of the developed solution.

<table>
<thead>
<tr>
<th>Components</th>
<th>Cost</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-DUINO PLC 42 I/Os Analog/Digital</td>
<td></td>
<td>275 €</td>
</tr>
</tbody>
</table>

Software implementation

<table>
<thead>
<tr>
<th></th>
<th>Cost</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Programming of the PLC</td>
<td>30 €/h</td>
<td>1.200 €</td>
</tr>
<tr>
<td>HMI development</td>
<td>30 €/h</td>
<td>2.400 €</td>
</tr>
</tbody>
</table>

*VAT not included

TOTAL 3.875 €

On regard of the budget above, it can be concluded that this project is totally feasible. First of all, the costs of the controller (taking into account all the specs that it includes) are significantly lower compared to standard industrial PLCs. Besides, the hourly fees for the software development are within range of current engineering market fares.
## STUDY TIMELINE

<table>
<thead>
<tr>
<th>Activity Description</th>
<th>September</th>
<th>October</th>
<th>November</th>
<th>December</th>
<th>January</th>
<th>February</th>
<th>March</th>
</tr>
</thead>
<tbody>
<tr>
<td>Information and training on SCADA tools for Arduino</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP/IP Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information and training on TCP/IP protocol/ Ethernet</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Arduino TCP/IP client development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP/IP Labview server development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modbus RTU Implementation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information and training on Modbus/Modbus RTU/RS-48S</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Information on Modbus RTU for Arduino</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modbus RTU Slave development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Modbus RTU Master development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HMI LabView development</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Report writing</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
14 ASSESSMENT OF ENVIRONMENTAL IMPLICATIONS

This study does not hold a relevant environmental implications, neither positive nor negative. As described in the objectives section, its ultimate purpose was not to improve the impact to the environment of the current technology. Yet, the developed solution for the reference plant implies an upgrading of the management and maintenance system, which in turn derives to savings of resources and time, and thus diminishes the effect on the environment. For instance, avoiding regular personnel check-ups to the plant as a consequence of remote availability of information of the system status through the developed SCADA system.

On a different note, another environmental advantage in this work ensues from using Industrial Shields open-source PLCs. The flexibility and adaptability that these devices offer result, in many cases, in a reduction of reinvestment in new equipment as a consequence of close-standard incompatibilities or outdated versions between existing installations when carrying out extensions of it or retrofitting tasks.
15 CONCLUSIONS

This section will examine how and to what extent the objectives set in Section 1 have been achieved.

In first place, the ultimate goal of the study was to develop and implement industrial communication protocols to Industrial Shields PLCs. It can be asserted that this objective has been wholly fulfilled. In Section 9 it has been shown how to establish a TCP-IP communication between an M-Duino and a server and define the functions and structures for exchanging data. Furthermore, Modbus RTU protocol over RS-485 serial bus has been adapted and implemented to an M-Duino in order to interface with a data acquisition device from another firm, Section 8.

These two developed protocols are widely extended in the industry nowadays, and TCP-IP has still a vast potential of expansion, as the standard protocol for Internet. The integration of these two communications with Industrial Shields products has already had a positive effects for customers, which have been implementing it from the beginning see ANNEX B: Real case examples. Moreover, the integration of this communication protocols has boost interest on the devices.

Another of the goals of this work was to develop a SCADA system integrating Industrial Shields PLCs. This point has been achieved indeed, as a SCADA system itself has been developed and tested. The communications between the PLC and the interactive HMI comprising the required variables and parameters to meet the specifications set in Section 4 has been shown to work. Moreover, this has already been a topic of interest among potential customers. Several of them contact us inquiring about communication between Industrial Shields and LabView, part of the work developed in this study serves as response, example and model for their applications.

In the paragraphs above it is has been proved, and supported with examples, that the developments of this study are useful realizations. Finally, taking into the fact that Industrial Shields PLCs are based on open-source, it can be concluded as well that this study serves as a proof of concept of a SCADA system developed with alternative open hardware and software solutions to traditional existing closed systems.
16 FUTURE DEVELOPMENT GUIDELINES

The objectives set at the beginning of the study have been fulfilled as argued in Section 15. It has been proved that developing a SCADA system with alternative technology is possible and its core functionalities have been implemented successfully. From that standpoint, there are some aspects that will upgrade the SCADA system and make possible to move from a prototype to a final product.

- Development of a thorough reconnection capability.

In this study it has been presented how a TCP communication can be established between an Industrial Shields PLC and a PC hosting a TCP server. Furthermore, it has been exposed how to exchange data between the PLC and the PC, and monitor and control the application parameters from an HMI in the PC. All this has been proved to work fine and serves as a prototype for a SCADA system. An important functionality for a final application would be a communication control algorithm on the PLC side. This would monitor the communication and ensure that whenever the transmission is disturbed or broken, the PLC follows the required procedure to reconnect to the server autonomously.

- Creation of a database and web server application

Another improvement to be considered for future developments is to create a web server HMI to interact with the PLC instead of a desktop based application. Furthermore, a database could also be created for data storage.

- Development of security protocol.

Security is critical nowadays in data systems. For the reference case considered in the present work this might not be an issue, but if the study is to be applied to different application where data might be confidential, encrypting the information and protecting the system against cyberattacks will be paramount.
17 BIBLIOGRAPHY & REGULATIONS


ANNEX A: Modbus RTU Test

Interfacing Industrial Shields PLC with Modbus RTU to a slave simulator

This test was performed during the development stage of the project to prove that the adapted library for Modbus RTU on Industrial Shields could work with different systems and not just between a Master and Slave of the same library (Section 7.2) and with Industrial Shields PLCs. Furthermore, in ANNEX B: Real case examples is also proved by Industrial Shields customers that Industrial Shields devices could communicate to other PLCs (from different firms) via Modbus RTU.

The test in this section was performed to an M-Duino configured as a Modbus RTU Master (see Section 7.2.2) communicating to a Modbus RTU slave simulator. The slave simulator (peakhmi.com) runs on a PC and therefore a RS-485 to USB converter was used to interface the RS-485 bus with the PC.

In the test the two first holding registers of the Slave are used for reading and writing a value in each case. The value in holding register 400001 was forced to a number (6 in the example) and this was read by the M-Duino (master) and printed in its Serial Monitor Figure 62.

Figure 62. Above: Modbus slave holding registers. Below: Modbus master code printing value in slave holding register 400001.
In parallel, holding register 400002 was used to write an integer value which was incremented in each cycle, in Figure 62 the screenshot shows iteration 91, is incremented in 1 unit every second.

```cpp
#include <SimpleModbusMasterSoftwareSerial.h>
#include <SoftwareSerial.h>

#define baud 9600
#define timeout 1000
#define polling 200 // the scan rate
#define retry_count 10

int SSerialRX =11 ; // RO //RS485 Receive pin
int SSerialTX=10; // DI //RS485 Transmit pin
int SSerialTxControl=7; // DE //RS485 Direction pin
int GroundRE=4 ; // RE
SoftwareSerial RS485Serial(SSerialRX, SSerialTX);
int i;
enum
{
    PACKET1,
    PACKET2,
    //PACKET3,
    TOTAL_NO_OF_PACKETS // leave this last entry
};

// Create an array of Packets to be configured
Packet packets[TOTAL_NO_OF_PACKETS];

// Masters register array
#define TOTAL_NO_OF_REGISTERS 2
unsigned int regs[TOTAL_NO_OF_REGISTERS];

void setup()
{
    Serial.begin(9600);
    pinMode(SSerialTxControl, OUTPUT);
    pinMode(GroundRE, OUTPUT);
    digitalWrite(GroundRE,LOW);

    modbus_construct(&packets[PACKET1], 1, READ_HOLDING_REGISTERS, 0, 1, 0);
    modbus_construct(&packets[PACKET2], 1, PRESET_MULTIPLE_REGISTERS, 1, 1, 1);
    modbus_configure(&RS485Serial, baud, SERIAL_8N1, timeout, polling, retry_count,
    SSerialTxControl, packets, TOTAL_NO_OF_PACKETS, regs);
}

void loop()
{
    modbus_update();

    Serial.print(regs[0]);
    Serial.println("/");
    regs[1]=i;

    i=i+1;
}
```
ANNEX B: Real case examples

Real case examples of implementation of Modbus RTU with Industrial Shields PLCs

After the work described in Sections 7 and 8 and implementing Modbus RTU on Industrial Shields PLCs, there have been several cases were Industrial Shields customers are using this functionality for their applications and systems. Following, two of the cases are described.

Montrol

Montrol Avda Jaume II, 97 1er Olot 17800, Girona

Montrol is a system integrator of automation and monitoring systems. In one of their projects they are using Ardbox Relay as a Modbus RTU slave interfaced with a master PLC DigiPoint. The Ardbox Relay acquires data from sensors and operates several pumps based on the DigiPoint commands.

Hitachi Air Conditioning Products Europe S A U

Ronda Shimizu, 1 Polígono Ind Can Torrella, 08233 Vacarisses, Barcelona

In this case the engineers in Hitachi are interested in replacing standard Siemens PLCs in a range of systems in their factory. The automation of the production lines is based on Siemens, Omron and Hitachi PLCs and the intention so far is maintaining the same devices. These same PLCs are currently being used as well in systems which are not as critical as the production line and are much diverse and prone to modifications and changes. The perspective from Hitachi in these applications is that traditional PLCs lack adaptability and flexibility capacity, costs are much higher and, furthermore, full potential of the controllers is not exploit. All these facts raised interest for Industrial Shields PLCs. A relevant requirement for the devices to fit in these systems is the possibility of Modbus RTU, which after this work is now available. At the moment Industrial Shields PLCs are being steadily introduced in several of their system, installations and projects.
ANNEX C: Attached CD

The attached CD contains essential files used in the project.

<table>
<thead>
<tr>
<th>File</th>
<th>Program</th>
<th>Report section</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCPsimple_client.ino</td>
<td>Arduino IDE</td>
<td>9.3</td>
</tr>
<tr>
<td>TCPsimple_server.vi</td>
<td>LabView</td>
<td>9.3</td>
</tr>
<tr>
<td>Final_application_PLC</td>
<td>Arduino IDE</td>
<td>11.1</td>
</tr>
<tr>
<td>Final_application_HMI</td>
<td>LabView</td>
<td>10.3</td>
</tr>
<tr>
<td>Arduino IDE libraries</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SimpleModbusSlaveSoftwareSerial</td>
<td>Arduino IDE</td>
<td></td>
</tr>
<tr>
<td>SimpleModbusMasterSoftwareSerial</td>
<td>Arduino IDE</td>
<td></td>
</tr>
<tr>
<td>arduino_uip-master</td>
<td>Arduino IDE</td>
<td></td>
</tr>
</tbody>
</table>

ANNEX D: Datasheets
Our Company

Industrial Shields was born in 2011 when a design engineer, Albert Prieto (Industrial Shields CEO), installing market PLCs in different machines was looking for lower cost and flexible PLCs. Then he decided to develop his own solutions, PLCs using Open Hardware.

Today Industrial Shields is a reality with a wide range of PLCs and Panel PCs that keeps growing.

Industrial Shields is present in more than 30 countries.

Main office (with Financial, R&D and logistics) and production are located close to Barcelona.

Our company develops and produce the 100% of the product range, including all aspects involved on the hardware and software.

Why?

Our company’s goal is to provide low cost solutions for automation, not only in the Universities and Hobbyist markets but also in Industrial environments.

The Open Source Hardware solutions are still not widely introduced in Industrial environments, it is a growing market and we are its pioneers.

The balance between quality and cost is very important for us and so for the market, using Open Source solutions we can provide more specifications at a better price.

Even more, the Open Source solutions are more flexible and accessible than the standard industrial solutions.

Our Customers

- Arduino Users, Universities and other training centers, hobbyist, robotics shops
- Industrial users looking for a low cost and flexible solution
- Integrators and Engineering companies providing turnkey solutions
Our Sales Channels

- Distributors and resellers.
- Industrial Shields online shop and web site (Google Adwords activity).

Applications

Manufacturing / Machining
Food / Beverage
Metals
Power
Mining
Petrochemical / Chemical

Products

ARDBOX Range - 2 Models - PLCs
- PLC Arduino ARDBOX 20 I/Os Analog
  10 In / 10 Out
- PLC Arduino ARDBOX PLC 20 I/Os RELAY
  10 In / 10 Out
Products

**M-DUINO Range - 6 Models - Ethernet PLCs**

- M-DUINO PLC Arduino 19R I/Os Rele / Analog / Digital
  - 7 In / 13 Out
- M-DUINO PLC Arduino 21 I/Os Analog/Digital
  - 13 In / 8 Out
- M-DUINO PLC Arduino 3B I/Os Rele / Analog / Digital
  - 14 In / 24 Out
- M-DUINO PLC Arduino 42 I/Os Analog/Digital
  - 26 In / 16 Out
- M-DUINO PLC Arduino 5R I/Os Rele / Analog / Digital
  - 21 In / 36 Out
- M-DUINO PLC Arduino 58 I/Os Analog/Digital
  - 36 In / 22 Out

**HummTOUCH 10.1” Android**
- 64Bit, dual core
- With digital inputs and outputs

**HummTOUCH 10.1” Linux**
- 64Bit, dual core
- With digital inputs and outputs

**TOUCHBERRY PI 10.1”**
- Linux OS
- With digital inputs and outputs

**Panel PCs – 3 Models**

- HummTOUCH 10.1” Android
  - 64Bit, dual core
  - With digital inputs and outputs
- HummTOUCH 10.1” Linux
  - 64Bit, dual core
  - With digital inputs and outputs
- TOUCHBERRY PI 10.1”
  - Linux OS
  - With digital inputs and outputs

**Programming environment**

- Arduino IDE

- Ladder/KOP (available soon)

**A Worldwide Reference for Open Source Industrial Equipment**
A compact PLC based in Open Source Hardware technology. With different Input/Outputs Units.

## 2. General Description M-DUINO FAMILY products

### CONNECTABLE PLC ARDUINO 24Vcc M-DUINO

<table>
<thead>
<tr>
<th>MODEL TYPE</th>
<th>21 I/Os</th>
<th>42 I/Os</th>
<th>58 I/Os</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input Voltage</td>
<td>12-24Vdc</td>
<td></td>
<td></td>
</tr>
<tr>
<td>I max.</td>
<td>0.5A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Size</td>
<td>101x119.5x70.1</td>
<td>101x119.5x94.7</td>
<td>101x119.5x119.3</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>16MHz</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Flash Memory</td>
<td>256KB of which 8KB used by bootloader</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SRAM</td>
<td>8KB</td>
<td></td>
<td></td>
</tr>
<tr>
<td>EEPROM</td>
<td>4KB</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

| TOTAL Input points | 13 | 26 | 36 |
| TOTAL Output points | 8 | 16 | 22 |

**Type of signals**

- An/Dig Input 10bit (0-10Vcc) | 6 | 12 | 16 |
- Digital Input (24Vcc) | 13 | 26 | 36 |
- * Interrupt Input HS (24Vcc) | 2 | 4 | 6 |
- Analog Output (0-10Vcc) | 3 | 6 | 8 |
- Digital Output (24Vcc) | 8 | 16 | 22 |
- PWM Output 8bit (24Vcc) | 3 | 6 | 8 |

**Expandability**

- I2C¹ – 127 elements – communication boards
- Ethernet

**Reference**

- IS.MDUINO.base.21
- IS.MDUINO.base.42
- IS.MDUINO.base.58

¹ By using this type of signal can no longer use Digital signal (24Vdc)

You must read product Datasheet.

1 IMPORTANT. Visit accessories/communication section.

---

¹ Pull-up resistance required [IS.ACI2C-4.7K]
4. Specifications

4.1. General Specifications:

<table>
<thead>
<tr>
<th>Item</th>
<th>M-DUINO 21 IOs</th>
<th>M-DUINO 42 IOs</th>
<th>M-DUINO 58 IOs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Power supply voltage</td>
<td>DC power supply 12 - 24Vdc</td>
<td>DC power supply 11.4 to 25.4Vdc</td>
<td>DC power supply 30VAC max.</td>
</tr>
<tr>
<td>Operating voltage range</td>
<td>DC power supply 11.4 to 25.4Vdc</td>
<td>DC power supply 30VAC max.</td>
<td>DC power supply 24Vdc</td>
</tr>
<tr>
<td>Power consumption</td>
<td>DC power supply 30VAC max.</td>
<td>DC power supply 700Ma</td>
<td>DC power supply 24Vdc</td>
</tr>
<tr>
<td>External power supply</td>
<td>Power supply voltage 24Vdc</td>
<td>Power supply output capacity 700Ma</td>
<td>Power supply output capacity 700Ma</td>
</tr>
<tr>
<td>Insulation resistance</td>
<td>20MΩ min. at 500Vdc between the AC terminals and the protective earth terminal.</td>
<td>20MΩ min. at 500Vdc between the AC terminals and the protective earth terminal.</td>
<td>20MΩ min. at 500Vdc between the AC terminals and the protective earth terminal.</td>
</tr>
<tr>
<td>Dielectric strength</td>
<td>2.300 VAC at 50/60 HZ for one minute with a leakage current of 10mA max. Between all the external AC terminals and the protective earth terminal.</td>
<td>2.300 VAC at 50/60 HZ for one minute with a leakage current of 10mA max. Between all the external AC terminals and the protective earth terminal.</td>
<td>2.300 VAC at 50/60 HZ for one minute with a leakage current of 10mA max. Between all the external AC terminals and the protective earth terminal.</td>
</tr>
<tr>
<td>Shock resistance</td>
<td>80m/s² in the X, Y and Z direction 2 times each.</td>
<td>80m/s² in the X, Y and Z direction 2 times each.</td>
<td>80m/s² in the X, Y and Z direction 2 times each.</td>
</tr>
<tr>
<td>Ambient temperature (operating)</td>
<td>0º to 45ºC</td>
<td>0º to 45ºC</td>
<td>0º to 45ºC</td>
</tr>
<tr>
<td>Ambient humidity (operating)</td>
<td>10% to 90% (no condensation)</td>
<td>10% to 90% (no condensation)</td>
<td>10% to 90% (no condensation)</td>
</tr>
<tr>
<td>Ambient environment (operating)</td>
<td>With no corrosive gas</td>
<td>With no corrosive gas</td>
<td>With no corrosive gas</td>
</tr>
<tr>
<td>Ambient temperature (storage)</td>
<td>-20º to 60ºC</td>
<td>-20º to 60ºC</td>
<td>-20º to 60ºC</td>
</tr>
<tr>
<td>Power supply holding time</td>
<td>2ms min.</td>
<td>2ms min.</td>
<td>2ms min.</td>
</tr>
<tr>
<td>Weight</td>
<td>445g max.</td>
<td>542g max.</td>
<td>850g max.</td>
</tr>
</tbody>
</table>

4.2. Performance Specification:

<table>
<thead>
<tr>
<th>Item</th>
<th>M-DUINO 21 IOs</th>
<th>M-DUINO 42 IOs</th>
<th>M-DUINO 58 IOs</th>
</tr>
</thead>
<tbody>
<tr>
<td>Arduino Board</td>
<td>ARDUINO MEGA 2560</td>
<td>ARDUINO MEGA 2560</td>
<td>ARDUINO MEGA 2560</td>
</tr>
<tr>
<td>Control method</td>
<td>Stored program method</td>
<td>Stored program method</td>
<td>Stored program method</td>
</tr>
<tr>
<td>I/O control method</td>
<td>Combination of the cyclic scan and immediate refresh processing methods.</td>
<td>Combination of the cyclic scan and immediate refresh processing methods.</td>
<td>Combination of the cyclic scan and immediate refresh processing methods.</td>
</tr>
<tr>
<td>Microcontroller</td>
<td>ATmega2560</td>
<td>ATmega2560</td>
<td>ATmega2560</td>
</tr>
<tr>
<td>Flash Memory</td>
<td>256kb of which 8 kb used by bootloader</td>
<td>256kb of which 8 kb used by bootloader</td>
<td>256kb of which 8 kb used by bootloader</td>
</tr>
<tr>
<td>Program capacity (SRAM)</td>
<td>8kb</td>
<td>8kb</td>
<td>8kb</td>
</tr>
<tr>
<td>EEPROM</td>
<td>4kb</td>
<td>4kb</td>
<td>4kb</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>16MHz</td>
<td>16MHz</td>
<td>16MHz</td>
</tr>
<tr>
<td>Clock Speed</td>
<td>16MHz</td>
<td>16MHz</td>
<td>16MHz</td>
</tr>
</tbody>
</table>
7. I/O Pinout (summary pinout/ Arduino PIN):

**M-DUINO 58 IOs**

**M-DUINO 42 IOs**

**M-DUINO 21 IOs**

**Base (common unit)**

<table>
<thead>
<tr>
<th>A Zone</th>
<th>B Zone</th>
<th>C Zone</th>
<th>D Zone</th>
</tr>
</thead>
<tbody>
<tr>
<td>M-Duino Connector</td>
<td>Arduino Pin</td>
<td>M-Duino Connector</td>
<td>Arduino Pin</td>
</tr>
<tr>
<td>SCL</td>
<td>21</td>
<td>SCL</td>
<td>IO.12</td>
</tr>
<tr>
<td>SDA</td>
<td>20</td>
<td>SDA</td>
<td>IO.11</td>
</tr>
<tr>
<td>RX0</td>
<td>1</td>
<td>RX0</td>
<td>IO.10</td>
</tr>
<tr>
<td>TX0</td>
<td>0</td>
<td>TX0</td>
<td>IO.9</td>
</tr>
<tr>
<td>RX1</td>
<td>19</td>
<td>RX1</td>
<td>IO.8</td>
</tr>
<tr>
<td>TX1</td>
<td>18</td>
<td>TX1</td>
<td>IO.7</td>
</tr>
<tr>
<td>TX</td>
<td>-</td>
<td>RS232</td>
<td></td>
</tr>
<tr>
<td>RX</td>
<td>-</td>
<td>RS232</td>
<td></td>
</tr>
<tr>
<td>RXS/RE</td>
<td>15</td>
<td>RS485</td>
<td></td>
</tr>
<tr>
<td>TXS/DE</td>
<td>14</td>
<td>RS485</td>
<td></td>
</tr>
<tr>
<td>A</td>
<td>3</td>
<td>Arduino Pin</td>
<td>COM-I0.8</td>
</tr>
<tr>
<td>B</td>
<td>50</td>
<td>SPI</td>
<td>I0.6</td>
</tr>
<tr>
<td>Gnd</td>
<td>-</td>
<td>SPI</td>
<td>I0.5</td>
</tr>
<tr>
<td>24Vdc</td>
<td>-</td>
<td>SPI</td>
<td>I0.4</td>
</tr>
<tr>
<td>PIN3</td>
<td>2</td>
<td>Arduino Pin</td>
<td>COM-I0.3</td>
</tr>
<tr>
<td>SO</td>
<td>52</td>
<td>SPI</td>
<td>I0.3</td>
</tr>
<tr>
<td>SI</td>
<td>51</td>
<td>SPI</td>
<td>I0.2</td>
</tr>
<tr>
<td>SCK</td>
<td>52</td>
<td>SPI</td>
<td>I0.1</td>
</tr>
<tr>
<td>Reset</td>
<td>-</td>
<td>SPI</td>
<td>I0.0</td>
</tr>
</tbody>
</table>

**42 IOs**

<table>
<thead>
<tr>
<th>M-Duino Connector</th>
<th>Arduino Pin</th>
<th>M-Duino Connector</th>
<th>Arduino Pin</th>
<th>M-Duino Connector</th>
<th>Arduino Pin</th>
<th>M-Duino Connector</th>
<th>Arduino Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREF 7Vdc (out)</td>
<td>AREF 5Vdc (out)</td>
<td>AREF 3.3V (out)</td>
<td>AREF 5Vdc (out)</td>
<td>AREF 3.3V (out)</td>
<td>AREF 5Vdc (out)</td>
<td>AREF 3.3V (out)</td>
<td>AREF 5Vdc (out)</td>
</tr>
<tr>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>Q0.7</td>
<td>6</td>
<td>Analog/ PWM/ digital Out</td>
<td>Q1.7</td>
<td>9</td>
<td>Analog/PWM/ digital Out</td>
<td>Q2.6</td>
<td>13</td>
</tr>
<tr>
<td>Q0.6</td>
<td>5</td>
<td>Analog/ PWM/ digital Out</td>
<td>Q1.6</td>
<td>7</td>
<td>Analog/PWM/ digital Out</td>
<td>Q2.5</td>
<td>12</td>
</tr>
<tr>
<td>Q0.5</td>
<td>4</td>
<td>Analog/ PWM/ digital Out</td>
<td>Q1.5</td>
<td>8</td>
<td>Analog/PWM/ digital Out</td>
<td>Q2.4</td>
<td>NC</td>
</tr>
<tr>
<td>Q0.4</td>
<td>40</td>
<td>Digital Out</td>
<td>Q1.4</td>
<td>45</td>
<td>Digital Out 2</td>
<td>Q2.3</td>
<td>49</td>
</tr>
<tr>
<td>Q0.3</td>
<td>39</td>
<td>Digital Out</td>
<td>Q1.3</td>
<td>44</td>
<td>Digital Out 2</td>
<td>Q2.2</td>
<td>48</td>
</tr>
<tr>
<td>Q0.2</td>
<td>38</td>
<td>Digital Out</td>
<td>Q1.2</td>
<td>43</td>
<td>Digital Out 2</td>
<td>Q2.1</td>
<td>47</td>
</tr>
<tr>
<td>Q0.1</td>
<td>37</td>
<td>Digital Out</td>
<td>Q1.1</td>
<td>42</td>
<td>Digital Out 2</td>
<td>Q2.0</td>
<td>46</td>
</tr>
<tr>
<td>Q0.0</td>
<td>36</td>
<td>Digital Out</td>
<td>Q1.0</td>
<td>41</td>
<td>Digital Out 2</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
</tr>
<tr>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
</tr>
</tbody>
</table>

**58 IOs**

<table>
<thead>
<tr>
<th>M-Duino Connector</th>
<th>Arduino Pin</th>
<th>M-Duino Connector</th>
<th>Arduino Pin</th>
<th>M-Duino Connector</th>
<th>Arduino Pin</th>
<th>M-Duino Connector</th>
<th>Arduino Pin</th>
</tr>
</thead>
<tbody>
<tr>
<td>AREF 7Vdc (out)</td>
<td>AREF 5Vdc (out)</td>
<td>AREF 3.3V (out)</td>
<td>AREF 5Vdc (out)</td>
<td>AREF 3.3V (out)</td>
<td>AREF 5Vdc (out)</td>
<td>AREF 3.3V (out)</td>
<td>AREF 5Vdc (out)</td>
</tr>
<tr>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>Q0.7</td>
<td>6</td>
<td>Analog/ PWM/ digital Out</td>
<td>Q1.7</td>
<td>9</td>
<td>Analog/PWM/ digital Out</td>
<td>Q2.6</td>
<td>13</td>
</tr>
<tr>
<td>Q0.6</td>
<td>5</td>
<td>Analog/ PWM/ digital Out</td>
<td>Q1.6</td>
<td>7</td>
<td>Analog/PWM/ digital Out</td>
<td>Q2.5</td>
<td>12</td>
</tr>
<tr>
<td>Q0.5</td>
<td>4</td>
<td>Analog/ PWM/ digital Out</td>
<td>Q1.5</td>
<td>8</td>
<td>Analog/PWM/ digital Out</td>
<td>Q2.4</td>
<td>NC</td>
</tr>
<tr>
<td>Q0.4</td>
<td>40</td>
<td>Digital Out</td>
<td>Q1.4</td>
<td>45</td>
<td>Digital Out 2</td>
<td>Q2.3</td>
<td>49</td>
</tr>
<tr>
<td>Q0.3</td>
<td>39</td>
<td>Digital Out</td>
<td>Q1.3</td>
<td>44</td>
<td>Digital Out 2</td>
<td>Q2.2</td>
<td>48</td>
</tr>
<tr>
<td>Q0.2</td>
<td>38</td>
<td>Digital Out</td>
<td>Q1.2</td>
<td>43</td>
<td>Digital Out 2</td>
<td>Q2.1</td>
<td>47</td>
</tr>
<tr>
<td>Q0.1</td>
<td>37</td>
<td>Digital Out</td>
<td>Q1.1</td>
<td>42</td>
<td>Digital Out 2</td>
<td>Q2.0</td>
<td>46</td>
</tr>
<tr>
<td>Q0.0</td>
<td>36</td>
<td>Digital Out</td>
<td>Q1.0</td>
<td>41</td>
<td>Digital Out 2</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
<td>NC</td>
</tr>
<tr>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
</tr>
<tr>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
<td>Gnd</td>
</tr>
</tbody>
</table>

*NOTE: Digital Inputs I0.5, I0.6, I1.5, I1.6, I2.5, I2.6 can be configured on Arduino IDE to be interrupts.*

**2** NOTE: Digital Outputs QX.0-QX.4 need to be provided a voltage supply and a reference to the COM-Q(+) and COM-Q(-).*
9. I/O technical details:

<table>
<thead>
<tr>
<th>M-duino family products</th>
<th>Signal</th>
<th>Vdc</th>
<th>Maxim current consumption</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Digital Input</td>
<td>24 Vdc</td>
<td>11mA</td>
</tr>
<tr>
<td></td>
<td>Analog/Digital Input</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Analog (impedance 78KΩ)</td>
<td>0-10Vdc</td>
<td>50mA</td>
</tr>
<tr>
<td></td>
<td>Digital</td>
<td>24Vdc</td>
<td>50mA</td>
</tr>
<tr>
<td></td>
<td>Digital Output</td>
<td>24Vdc</td>
<td>150mA</td>
</tr>
<tr>
<td></td>
<td>Analog/Digital/PWM</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Analog</td>
<td>0-10Vdc</td>
<td>50mA</td>
</tr>
<tr>
<td></td>
<td>Digital</td>
<td>24 Vdc</td>
<td>50mA</td>
</tr>
<tr>
<td></td>
<td>PWM</td>
<td>24Vdc</td>
<td>50mA</td>
</tr>
<tr>
<td></td>
<td>Cumulative outputs</td>
<td></td>
<td>All outputs can be working simultaneously at its max current value</td>
</tr>
</tbody>
</table>

6 You can select 24Vdc or 10Vdc (with correctly switch configuration). Digital output will be 12Vdc If you connect PLC to 12Vdc power supply.
ENC28J60

Stand-Alone Ethernet Controller with SPI™ Product Brief

Features

- IEEE 802.3 Compatible Ethernet Controller
- Integrated MAC and 10BASE-T PHY
- 8-Kbyte Transmit/Receive Packet Dual Port Buffer SRAM
- Receiver and Collision Squelch Circuit
- Supports one 10BASE-T Port with Automatic Polarity Detection and Correction
- Programmable Automatic Retransmit on Collision
- Programmable Padding and CRC Generation
- Programmable Automatic Rejection of Erroneous Packets
- 10 Mbit/s SPI™ Interface
- Buffer:
  - Configurable transmit/receive buffer size
  - Hardware managed circular receive FIFO
  - Byte-wide random and sequential access
  - Internal DMA for fast memory copying
  - Hardware assisted IP checksum calculation
- MAC:
  - Support for Unicast, Multicast and Broadcast packets
  - Programmable pattern matching
  - Programmable wake-up on multiple packet formats, including Magic Packet®, Unicast, Multicast, Broadcast, specific packet match or any packet
  - Loopback mode
- PHY:
  - Wave shaping output filter
  - Loopback mode
- Operational:
  - Outputs for 2 LED indicators
  - Transmit and receive interrupts
  - 25 MHz clock
  - Clock out pin with programmable prescaler
  - Operating voltage range of 3.14V to 3.45V
  - Temperature range: -40°C to +85°C Industrial, 0°C to +70°C Commercial (SSOP only)
- 28-pin SSOP, SOIC, SPDIP and QFN packages

Pin Diagrams

28-pin SSOP, SOIC and SPDIP

28-pin QFN
**Description**

The Microchip Technology Inc. ENC28J60 is a stand-alone Ethernet controller with an industry standard Serial Peripheral Interface (SPI™). It is designed to serve as an Ethernet network interface for any microcontroller equipped with SPI.

The ENC28J60 meets all of the specifications for IEEE 802.3. It incorporates a number of packet filtering schemes to limit incoming packets. It also provides internal DMA for fast data throughput and support for hardware assisted IP checksum calculation.

Communication with the microcontroller is implemented via SPI, with data rates up to 10 Mbit/s. Dedicated pins are used for LED link and activity indication and for transmit and receive interrupts.

**Block Diagram**

![Block Diagram of ENC28J60](image)
Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip’s Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as “unbreakable.”

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip’s code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip’s products as critical components in life support systems is not authorized except with express written approval by Microchip. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks
The Microchip name and logo, the Microchip logo, Accuron, dsPIC, KEELOQ, microID, MPLAB, PIC, PICmicro, PICSTART, PRO MATE, PowerSmart, rfPIC, and SmartShunt are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
AmpLab, FilterLab, MXDEV, MXLAB, PICMASTER, SEEVAL, SmartSensor and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.
Analog-for-the-Digital Age, Application Maestro, dsPICDEM, dsPICDEM.net, dsPICworks, ECAN, ECONOMONITOR, FanSense, FlexROM, fuzzyLAB, In-Circuit Serial Programming, ICSP, ICEPIC, Migratable Memory, MPASM, MPLIB, MPLINK, MPSIM, PICkit, PICDEM, PICDEM.net, PICLAB, PICtail, PowerCal, PowerInfo, PowerMate, PowerTool, rfLAB, rfPICDEM, Select Mode, Smart Serial, SmartTel and Total Endurance are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.
SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.
All other trademarks mentioned herein are property of their respective companies.
© 2004, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.
 Printed on recycled paper.

Microchip received ISO/TS-16949:2002 quality system certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona and Mountain View, California in October 2003. The Company’s quality system processes and procedures are for its PICmicro® 8-bit MCUs, KEELOQ® code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip’s quality system for the design and manufacture of development systems is ISO 9001:2000 certified.
# World Wide Sales and Service

<table>
<thead>
<tr>
<th>Region</th>
<th>Country/Location</th>
<th>Telephone Numbers</th>
<th>Fax Numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>AMERICAS</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Corporate Office</td>
<td>2355 West Chandler Blvd, Chandler, AZ 85224-6199</td>
<td>Tel: 480-792-7200</td>
<td>Fax: 480-792-7277</td>
</tr>
<tr>
<td>Technical Support:</td>
<td><a href="http://support.microchip.com">http://support.microchip.com</a></td>
<td>Web Address: <a href="http://www.microchip.com">www.microchip.com</a></td>
<td></td>
</tr>
<tr>
<td>Atlanta</td>
<td>Alpharetta, GA</td>
<td>Tel: 770-640-0034</td>
<td>Fax: 770-640-0307</td>
</tr>
<tr>
<td>Boston</td>
<td>Westford, MA</td>
<td>Tel: 978-692-3848</td>
<td>Fax: 978-692-3821</td>
</tr>
<tr>
<td>Chicago</td>
<td>Itasca, IL</td>
<td>Tel: 630-285-0071</td>
<td>Fax: 630-285-0075</td>
</tr>
<tr>
<td>Dallas</td>
<td>Addison, TX</td>
<td>Tel: 972-818-7423</td>
<td>Fax: 972-818-2924</td>
</tr>
<tr>
<td>Detroit</td>
<td>Farmington Hills, MI</td>
<td>Tel: 248-538-2250</td>
<td>Fax: 248-538-2260</td>
</tr>
<tr>
<td>Kokomo</td>
<td>Kokomo, IN</td>
<td>Tel: 765-864-8360</td>
<td>Fax: 765-864-8387</td>
</tr>
<tr>
<td>Los Angeles</td>
<td>Mission Viejo, CA</td>
<td>Tel: 949-462-9523</td>
<td>Fax: 949-462-9608</td>
</tr>
<tr>
<td>San Jose</td>
<td>Mountain View, CA</td>
<td>Tel: 650-215-1444</td>
<td>Fax: 650-961-0286</td>
</tr>
<tr>
<td>Toronto</td>
<td>Mississauga, Ontario, Canada</td>
<td>Tel: 905-673-0699</td>
<td>Fax: 905-673-6509</td>
</tr>
<tr>
<td><strong>ASIA/PACIFIC</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Australia - Sydney</td>
<td>Tel: 61-2-9868-6733</td>
<td>Fax: 61-2-9868-6755</td>
<td></td>
</tr>
<tr>
<td>China - Beijing</td>
<td>Tel: 86-10-8528-2100</td>
<td>Fax: 86-10-8528-2104</td>
<td></td>
</tr>
<tr>
<td>China - Chengdu</td>
<td>Tel: 86-28-8676-6200</td>
<td>Fax: 86-28-8676-6599</td>
<td></td>
</tr>
<tr>
<td>China - Fuzhou</td>
<td>Tel: 86-591-8750-3506</td>
<td>Fax: 86-591-8750-3521</td>
<td></td>
</tr>
<tr>
<td>China - Hong Kong SAR</td>
<td>Tel: 852-2401-1200</td>
<td>Fax: 852-2401-3431</td>
<td></td>
</tr>
<tr>
<td>China - Shanghai</td>
<td>Tel: 86-21-5407-5533</td>
<td>Fax: 86-21-5407-5066</td>
<td></td>
</tr>
<tr>
<td>China - Shenyang</td>
<td>Tel: 86-24-2334-2829</td>
<td>Fax: 86-24-2334-2393</td>
<td></td>
</tr>
<tr>
<td>China - Shenzhen</td>
<td>Tel: 86-755-8203-2660</td>
<td>Fax: 86-755-8203-1760</td>
<td></td>
</tr>
<tr>
<td>China - Shunde</td>
<td>Tel: 86-757-2839-5507</td>
<td>Fax: 86-757-2839-5571</td>
<td></td>
</tr>
<tr>
<td>China - Qingdao</td>
<td>Tel: 86-532-502-7355</td>
<td>Fax: 86-532-502-7205</td>
<td></td>
</tr>
<tr>
<td><strong>ASIA/PACIFIC</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>India - Bangalore</td>
<td>Tel: 91-80-2229-0061</td>
<td>Fax: 91-80-2229-0062</td>
<td></td>
</tr>
<tr>
<td>India - New Delhi</td>
<td>Tel: 91-11-5160-8631</td>
<td>Fax: 91-11-5160-8632</td>
<td></td>
</tr>
<tr>
<td>Japan - Kanagawa</td>
<td>Tel: 81-45-471-6166</td>
<td>Fax: 81-45-471-6122</td>
<td></td>
</tr>
<tr>
<td>Korea - Seoul</td>
<td>Tel: 82-2-554-7200</td>
<td>Fax: 82-2-558-5932 or 82-2-558-5934</td>
<td></td>
</tr>
<tr>
<td>Singapore</td>
<td>Tel: 65-6334-8870</td>
<td>Fax: 65-6334-8850</td>
<td></td>
</tr>
<tr>
<td>Taiwan - Kaohsiung</td>
<td>Tel: 886-7-536-4818</td>
<td>Fax: 886-7-536-4803</td>
<td></td>
</tr>
<tr>
<td>Taiwan - Taipei</td>
<td>Tel: 886-2-2500-6610</td>
<td>Fax: 886-2-2508-0102</td>
<td></td>
</tr>
<tr>
<td>Taiwan - Hsinchu</td>
<td>Tel: 886-3-572-9526</td>
<td>Fax: 886-3-572-6459</td>
<td></td>
</tr>
<tr>
<td><strong>ASIA/PACIFIC</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>EUROPE</strong></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Austria - Weis</td>
<td>Tel: 43-7242-2244-399</td>
<td>Fax: 43-7242-2244-393</td>
<td></td>
</tr>
<tr>
<td>Denmark - Ballerup</td>
<td>Tel: 45-4450-2828</td>
<td>Fax: 45-4485-2829</td>
<td></td>
</tr>
<tr>
<td>France - Massy</td>
<td>Tel: 33-1-69-53-63-20</td>
<td>Fax: 33-1-69-53-90-79</td>
<td></td>
</tr>
<tr>
<td>Germany - Ismaning</td>
<td>Tel: 49-89-627-144-0</td>
<td>Fax: 49-89-627-144-44</td>
<td></td>
</tr>
<tr>
<td>Italy - Milan</td>
<td>Tel: 39-0331-742611</td>
<td>Fax: 39-0331-466781</td>
<td></td>
</tr>
<tr>
<td>Netherlands - Drunen</td>
<td>Tel: 31-416-690399</td>
<td>Fax: 31-416-690340</td>
<td></td>
</tr>
<tr>
<td>England - Berkshire</td>
<td>Tel: 44-118-921-5869</td>
<td>Fax: 44-118-921-5820</td>
<td></td>
</tr>
</tbody>
</table>
Seko Tekna EVO APG Series is an analogic dosing pump with:

- constant flow rate manually adjustable
- proportional flow rate according to an external analogic (4-20 mA) or digital signal (water meter)
- Six positions adjustable switch:
  - 3 in division mode (1, 4, 10=n)
  - 1 in multiplication mode (n=1)
  - 1 for proportional mode 4-20 mA signal
  - 1 for constant functionality
- Pacing function adjustable by dip switch

Also with this pump, with only 5 sizes, is possible to cover a wide range of performances, having a flow rate range from 0.4 to 54 l/h and a back pressure from 0.1 to 20 bar. The power supply is 100÷240 Vac – 50/60 Hz therefore the same pump can operate with different supply voltage, moreover the model APG603 have the possibility for 24÷48 Vac/Vdc power supply.

The standard pump head is in PVDF, therefore high chemical compatibility with several liquids but is available in PVC as well, on request.

All Tekna series are equipped with a manually priming pump for the start up.

The pump is furnished with a complete standard installation kit, which includes: PVDF foot filter and injection valve, PVC suction tube, PE delivery tube. Moreover is available an installation kit in PVC, on request.
# PUMP KEY CODE

## 1° Model

**APG**

Analog dosing pump with constant flow rate manually adjustable, with proportional flow rate according to an external analog (4-20 mA) or digital signal (water meter). 6 positions adjustable switch: 3 in division mode (1, 4, 10=n), 1 in multiplication mode (n=1), 1 for proportional mode 4-20 mA signal, 1 for constant functionality. Pacing function adjustable by dip switch.

## 2° Hydraulic

<table>
<thead>
<tr>
<th>Pressure [bar]</th>
<th>Flow Rate [l/h]</th>
<th>Stroke/min</th>
<th>cc/stroke</th>
</tr>
</thead>
<tbody>
<tr>
<td>500</td>
<td>20 0.4</td>
<td>120</td>
<td>0.06</td>
</tr>
<tr>
<td></td>
<td>16 0.8</td>
<td></td>
<td>0.11</td>
</tr>
<tr>
<td></td>
<td>10 1.2</td>
<td></td>
<td>0.17</td>
</tr>
<tr>
<td></td>
<td>6 1.5</td>
<td></td>
<td>0.21</td>
</tr>
<tr>
<td>600</td>
<td>20 2.5</td>
<td>120</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>18 3</td>
<td></td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>14 4.2</td>
<td></td>
<td>0.58</td>
</tr>
<tr>
<td></td>
<td>8 7</td>
<td></td>
<td>0.97</td>
</tr>
<tr>
<td>603 N</td>
<td>12 4</td>
<td>160</td>
<td>0.42</td>
</tr>
<tr>
<td></td>
<td>10 5</td>
<td></td>
<td>0.52</td>
</tr>
<tr>
<td></td>
<td>8 6</td>
<td></td>
<td>0.53</td>
</tr>
<tr>
<td></td>
<td>2 8</td>
<td></td>
<td>0.63</td>
</tr>
<tr>
<td>603 O</td>
<td>7 2</td>
<td>160</td>
<td>0.21</td>
</tr>
<tr>
<td></td>
<td>5 6</td>
<td></td>
<td>0.62</td>
</tr>
<tr>
<td></td>
<td>0.2 15</td>
<td></td>
<td>1.56</td>
</tr>
<tr>
<td>800</td>
<td>16 7</td>
<td>300</td>
<td>0.39</td>
</tr>
<tr>
<td></td>
<td>10 10</td>
<td></td>
<td>0.55</td>
</tr>
<tr>
<td></td>
<td>5 15</td>
<td></td>
<td>0.83</td>
</tr>
<tr>
<td></td>
<td>1 18</td>
<td></td>
<td>1.00</td>
</tr>
<tr>
<td>803</td>
<td>5 20</td>
<td>300</td>
<td>1.11</td>
</tr>
<tr>
<td></td>
<td>4 25</td>
<td></td>
<td>1.39</td>
</tr>
<tr>
<td></td>
<td>2 38</td>
<td></td>
<td>2.11</td>
</tr>
<tr>
<td></td>
<td>0.1 54</td>
<td></td>
<td>3.00</td>
</tr>
</tbody>
</table>

## 3° Power Supply

- **N** 100-240 Vac – 50/60 Hz
- **O** 24-48 Vac/Vdc – 50/60 Hz (only for APG603)

## 4° Liquid End

<table>
<thead>
<tr>
<th>Pump head</th>
<th>Connections</th>
<th>Balls</th>
<th>Diaphragm</th>
</tr>
</thead>
<tbody>
<tr>
<td>H</td>
<td>PVDF</td>
<td>Ceramic</td>
<td>PTFE</td>
</tr>
<tr>
<td>P*</td>
<td>PVC</td>
<td>PVC</td>
<td>PTFE</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>*Automatic degassing valve only for APG 603 and 800</td>
</tr>
</tbody>
</table>

## 5° Installation Kit

| H         | PVDF |
| P         | PVC  |

## 6° Seals

- **0** FPM
- **1** EPDM
- **2** PTFE

## 7° Options

- **000** Standard
# DOSING PUMP

**Tekna EVO APG SERIES** - Solenoid dosing pump

## HYDRAULIC CHARACTERISTICS

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Min*</td>
</tr>
<tr>
<td>A P G 5 0 0 N</td>
<td>20 16 10 6</td>
<td>0,4 0,8 1,2 1,5</td>
<td>120</td>
<td>0,06 0,11 0,17 0,21</td>
<td>4/6 4/7</td>
<td>100÷240 Vac</td>
<td>8</td>
</tr>
<tr>
<td>A P G 6 0 0 N</td>
<td>20 18 14 8</td>
<td>2,5 3 4,2 7</td>
<td>120</td>
<td>0,35 0,42 0,58 0,97</td>
<td>4/6 4/7</td>
<td>100÷240 Vac</td>
<td>8</td>
</tr>
<tr>
<td>A P G 6 0 3 N</td>
<td>12 10 8 2</td>
<td>4 5 6 8</td>
<td>160</td>
<td>0,42 0,52 0,63 0,83</td>
<td>4/6 4/6</td>
<td>100÷240 Vac</td>
<td>15</td>
</tr>
<tr>
<td>A P G 6 0 3 O</td>
<td>7 5 0,2 0,4 2</td>
<td>2 6 10 15</td>
<td>160</td>
<td>0,21 0,62 1,56</td>
<td>4/6 4/6</td>
<td>24÷48 Vac</td>
<td>17</td>
</tr>
<tr>
<td>A P G 8 0 0 N</td>
<td>16 10 5 1</td>
<td>7 10 15 18</td>
<td>300</td>
<td>0,39 0,56 0,83 1,00</td>
<td>4/6 4/6</td>
<td>100÷240 Vac</td>
<td>15</td>
</tr>
<tr>
<td>A P G 8 0 3 N</td>
<td>5 4 2 0,1</td>
<td>20 25 38 54</td>
<td>300</td>
<td>1,11 1,39 2,11 3,00</td>
<td>8/12 8/12</td>
<td>100÷240 Vac</td>
<td>15</td>
</tr>
</tbody>
</table>

*Minimum consumption at 0 bar of back pressure (Patented)
DOSING PUMP
Tekna EVO APG SERIES- Solenoid dosing pump

DIMENSIONS

INSTALLATION KIT

In PVDF
- PVDF foot filter
- PVDF injection valve
- PVC suction tube (4 m)
- PE delivery tube (2 m)
- Wall and base fixing bracket

In PVC
- PVC foot filter
- PVC injection valve
- PVC suction tube (4 m)
- PE delivery tube (2 m)
- Wall and base fixing bracket
2.1 MAIN CHARACTERISTICS

- Power Supply: 100-240 Vac 50/60 Hz, 15Watt (Class 1 Electrical Insulation)
- System duration: 24 hours a day, 7 days a week for 5 years (43800 Hours)
- Operating temperature: 0 to 40°C, relative humidity 0 to 95% (without condensation)
- Data display: 4-line display with 20 large White and Blue characters.
- Keyboard: 7 Keys
- Cable connections: Dual row connectors
- Relays: Six (250 Vac 10 A); Four 100 to 240V Power relays and Two dry contact relays
- Measurements:
  - pH: 0.00 to 14.00 pH (precision ±0.01 pH)
  - Redox: ±2000 mV (precision ±1 mV)
  - Temperature: 0 to 105°C (precision 0.5 °C) (Predisposition for PT100 and PT1000 sensor)
  - Free chlorine: 0.01 to 5 ppm (precision ±0.01 ppm) (Amperometric Probe)
- Output Modules associated with the chemical measurements:
  - 2 channel current output, 0/4 to 20mA, 500 Ohm maximum load (precision ± 0.01 mA)
  - 2 channel Frequency Output (Open Collector NPN/PNP) 0 to 120 pulses per minute (precision 0.016 Hz)
- Input Modules:
  - Flow (pull up) (input for Reed sensor)
  - Hold
- Data transmission modules:
  - RS485 Serial Port (ModBus Standard Protocol)
- Modules integrated upon the mother board:
  - Clock module with backup battery.

2.2 MECHANICAL INSTALLATION

<table>
<thead>
<tr>
<th>Mechanical Dimensions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Dimensions (L x H x D)</td>
<td>300x290x143 mm</td>
</tr>
<tr>
<td>Installation depth</td>
<td>148 mm</td>
</tr>
<tr>
<td>Material</td>
<td>ABS</td>
</tr>
<tr>
<td>Installation typology</td>
<td>Wall-mounted</td>
</tr>
<tr>
<td>Weight</td>
<td>2.45 Kg</td>
</tr>
<tr>
<td>Front Panel</td>
<td>UV resistant polycarbonate</td>
</tr>
</tbody>
</table>
Drill the necessary holes and fasten the instrument to the wall using the support provided. The cable glands for the electrical connections are located on the lower portion of the control unit. In order to facilitate the connections, therefore, any other devices must be positioned at least 15 cm away. Protect the device against any drips and/or sprays of water from adjacent areas during the programming and calibration phases.

2.2 ELECTRICAL INSTALLATION

2.2.1 CONNECTION TO THE POWER SUPPLY

If possible, keep any high power cables away from the instrument and its connection cable, as these could cause inductive disturbances, especially for the analogical portion of the system. Use an alternating 100Vac to 240Vac-50/60Hz power supply. The power supply must be as stabilised as possible. Absolutely avoid connecting the device to rebuilt power supplies, using transformers for example, where the same power supply is also used to power other systems (perhaps of an inductive typology). This could lead to the generation of high voltage spikes which, once emitted, are difficult to block and/or eliminate.

ATTENTION: The electrical line must be equipped with an appropriate circuit breaker, in compliance with the proper installation standards

It is nevertheless always a good idea to check the quality of the grounding connector. In industrial facilities, it is not uncommon to find grounding connectors that cause electrical disturbances instead of preventing them; wherever doubts should arise regarding the quality of the facility's grounding connectors, it is best to connect the control unit's electrical system to a dedicated grounding rod.

2.2.2 CONNECTIONS TO DOSING SYSTEMS

ATTENTION: Before connecting the instrument to the external utilities (outputs and relays), make sure that the electrical panel is off and that the wires from the Utilities are not live.

WARNING: With a resistive load, each relay contact can sustain a maximum current of 1 amp, at max. 230V, and therefore a total power of 230 VA.
4.6 ModBus RTU protocol

The list of commands for data transmission using the RS485 RTU Serial Port is found below:

> Read Holding Register
> Write Multiple Register
> Report Slave ID

In response to the “Report Slave ID” command, the system returns the following string:

"Device"+ Firmware code + Edition

<table>
<thead>
<tr>
<th>ModBus profile table Address</th>
<th>Description</th>
<th>Property</th>
<th>Range</th>
<th>Operating state</th>
</tr>
</thead>
<tbody>
<tr>
<td>1000</td>
<td>pH Measure</td>
<td>Read 0 to 1400</td>
<td>Full Operating</td>
<td></td>
</tr>
<tr>
<td>1001</td>
<td>Cl Measure</td>
<td>Read 0 to 500</td>
<td>Full Operating</td>
<td></td>
</tr>
<tr>
<td>1002</td>
<td>Orp Measure</td>
<td>Read -2000 to +2000</td>
<td>Full Operating</td>
<td></td>
</tr>
<tr>
<td>1003</td>
<td>Temperature Measure</td>
<td>Read 0 to 1050</td>
<td>Full Operating</td>
<td></td>
</tr>
<tr>
<td>1004</td>
<td>Frequency Measure</td>
<td>Read 5 to 15000</td>
<td>Readable but not active</td>
<td></td>
</tr>
<tr>
<td>1005</td>
<td>Conductivity Measure</td>
<td>Read 0 to 10000</td>
<td>Readable but not active</td>
<td></td>
</tr>
<tr>
<td>1006</td>
<td>Pot1 Measure</td>
<td>Read 0 to 20000</td>
<td>Readable but not active</td>
<td></td>
</tr>
<tr>
<td>1007</td>
<td>Pot2 Measure</td>
<td>Read 0 to 20000</td>
<td>Readable but not active</td>
<td></td>
</tr>
<tr>
<td>1008</td>
<td>Status</td>
<td>Read See Note 1</td>
<td>Full Operating</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetPoint pH</td>
<td>Read/Write 0 to 1400</td>
<td>Full Operating</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetPoint Cl</td>
<td>Read/Write 0 to 500</td>
<td>Full Operating</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetPoint Orp</td>
<td>Read/Write -2000 to +2000</td>
<td>Full Operating</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SetPoint Temperature</td>
<td>Read/Write 0 to 1050</td>
<td>Full Operating</td>
<td></td>
</tr>
</tbody>
</table>

*Note1:* STATUS (bit field 16 bit register)

<table>
<thead>
<tr>
<th>Bit</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>LEVEL_0_ALARM</td>
</tr>
<tr>
<td>2</td>
<td>LEVEL_1_ALARM</td>
</tr>
<tr>
<td>3</td>
<td>HOLD_ALARM</td>
</tr>
<tr>
<td>4</td>
<td>REED_ALARM</td>
</tr>
<tr>
<td>5</td>
<td>Reserved</td>
</tr>
<tr>
<td>6</td>
<td>Reserved</td>
</tr>
<tr>
<td>7</td>
<td>Reserved</td>
</tr>
<tr>
<td>8</td>
<td>Reserved</td>
</tr>
<tr>
<td>9</td>
<td>Reserved</td>
</tr>
<tr>
<td>10</td>
<td>Reserved</td>
</tr>
<tr>
<td>11</td>
<td>Reserved</td>
</tr>
<tr>
<td>12</td>
<td>Reserved</td>
</tr>
<tr>
<td>13</td>
<td>STAT (Viewing statistics Mode)</td>
</tr>
<tr>
<td>14</td>
<td>Reserved</td>
</tr>
<tr>
<td>15</td>
<td>OFA (Over Feed Alarm)</td>
</tr>
<tr>
<td>16</td>
<td>PERMANENCY (Permanency Alarm)</td>
</tr>
</tbody>
</table>
Kontrol 800

Multi-parameter control instrument

The Kontrol 800 is a dedicated multi-parameter controller for complex applications that require a number of chemical parameters to be checked at the same time. The unit features independent proportional control output measures, two programmable frequency outputs, RS 485 serial port with MODBUS protocol, three relais outputs, probe quality checking and Data logging capability.

Parameters
- pH / ORP
- Conductivity
- Chlorine
- Chlorine Dioxide

Applications
- Waste Water
- Drinking Water
- Cooling Towers
- Boiler
- Legionella disinfection
- Reverse Osmosis
- Sludge
- Crate Wash
- Galvanic Process
- Dioxide Station
- CIP
- Irrigation
- Swimming Pool
- Fish Farming
- Sea water
- Dairy

Features

Graphic display and Keypad
Simultaneous value of the measure, Temperature and Relay status.
4-line, 20-character Alphanumeric Display.
Seven control keys for instrument calibration and configuration.

Enclosure Box and Power Supply
Wall mounting ABS plastic material IP65.
Universal Power Supply 100÷240 Vac 50/60 Hz

Manual controls
The user-friendly programming step menu makes starting up and checking the control and dosing system easy.

Data logging
Internal Flash memory to load record measures values.
Type: Circular (F.I.F.O.) or Filling.