MONITORING A FLUID CIRCUIT: A COMPARISON BETWEEN ARDUINO UNO AND PIC

C. PERJARU¹, G. PREDUȘCĂ², MIRUNA AVRÁM², M. RADU², C. STĂNESCU²
¹Otelinox, ²Valahia University of Targoviste, Faculty of Electrical Engineering, Electronics, and Information Technology
E-mail: cony82coni@gmail.com, gabriel.predusca@valahia.ro, adina.avram.ie@valahia.ro, marius.radu.ie@valahia.ro, cristian.stanescu.ie@valahia.ro,

Abstract. The purpose of this project is to design an electrical circuit for measuring specific parameters, such as liquid level and engine RPM, to monitor a fluid circuit. The project involves a comparison between the Arduino Uno development platform and the PIC16 development platform. The devices utilized include a level sensor, motor potentiometer, ultrasonic sensor (HC-SR04), submersible pump, a 4x20 LCD (Liquid Crystal Display) with an I2C (Inter-Integrated Circuits) interface, and various connection cables.

Keywords: Arduino Uno, Microchip, PIC16F15376, Arduino IDE, MPLab.

1. INTRODUCTION

The Arduino platform offers a diverse range of well-developed systems, catering to various informational needs and device connections. Whether you seek data from the environment or wish to establish connections with other devices, there is always an Arduino system to suit your requirements. To gather information from the environment, consider sensors such as those measuring alcohol levels in the breathed air, fire sensors, LPG gas sensors, carbon monoxide sensors, accelerometers for moving devices, current sensors for household appliances, power pressure sensors, tilt sensors, and more. For communication with other systems, options include Ethernet-type network cards for Arduino, facilitating internet connectivity; development platforms capable of transmitting or receiving data via radio or WiFi connections; GSM systems for sending or receiving SMS and initiating voice or data calls through 2G, 3G, or 4G networks; and Bluetooth devices for connecting the SMS and initiating voice or data calls through 2G, 3G, or WiFi connections; GSM systems for sending or receiving capable of transmitting or receiving data via radio or facilitating internet connectivity; development platforms.

The Xpress Board platform offers a diverse range of well- connected to the Arduino, ranging from simpler options like a character LCD with 16x2 lines to more advanced graphic LCD screens.

In addition to the Arduino platform, this project also utilizes the Xpress Board, based on the PIC16F15376 microcontroller. The Xpress Board supports various peripherals that can be connected to the platform, facilitating the development of new applications. Most of the sensors mentioned earlier are compatible with the Xpress Board platform [3].

Data acquisition involves determining electrical or physical states, such as voltage, current, temperature, and pressure. A Data Acquisition System (DAS) comprises sensors, DAS measurement hardware, and a computer with programmable software. In comparison to traditional measurement systems, PC-based DAS systems leverage the processing, productivity, display, and connectivity capabilities of industry-standard computers. This provides a more powerful, flexible, and cost-effective solution for measurements.

Data acquisition is prevalent in various fields, including industry, scientific research, communications, and automotive applications, such as on-board computers for monitoring water levels in evaporation pans used in evaporation and irrigation studies [4], fuel level monitoring systems in tanks [5], water level indicators in pools [6], volume monitoring of liquids in tanks [7], and control and data acquisition in liquid pipelines [8].

The essential components of a data acquisition system are illustrated in Figure 1. These components include:

a. Sensors: These devices transform physical phenomena into measurable electrical signals.

b. Input filter: This component eliminates alloying and parasitic effects.

c. Signal conditioning circuits: These include amplifiers, analog conversion circuits, isolation circuits, etc.

d. Sample and hold circuits: These circuits maintain a constant analog signal for a short period.

e. Multiplexers: Devices that select one of several analog input signals and route it to a single output.


g. RAM/ROM: Used for temporary or permanent data storage.

h. Digital-to-analog converter (DAC): Converts digital data back into analog signals.

i. Interface: This part facilitates communication between compatible units, devices, or components in a unidirectional or bidirectional manner.

j. Software (drivers): Specialized acquisition software usually provided by the manufacturer of the acquisition board. It is compatible with current operating systems (Windows, Mac, etc.).

k. Application software: Examples include LabVIEW, Matlab, etc., which provide virtual tools for data acquisition and analysis.

DOi: 10.2478/sbeef-2023-0019
1. Computing system: This comprises both hardware (the physical part) and software (the logical part). Its role is to retrieve, store, process, and transmit information [9].

2. DESCRIPTION OF THE EQUIPMENT USED IN THE PRACTICAL IMPLEMENTATION

The Arduino Uno and the Microchip Technology DM164143 are compact computer systems designed for programmable interaction with various forms of input and output, as depicted in Figure 2.

Figure 1. Structure of a data acquisition system.

Figure 2. Development platforms: Arduino Uno and Microchip Technology DM164143.

The microcontroller serves as the 'brain' for both the Arduino platform and the Xpress Board platform, equipped with the necessary processor to execute instructions. It encompasses various memory models to store data and instructions in our projects, offering versatile methods for sending and receiving data.

Arduino employs an Atmega328 AVR 8-bit microcontroller, a high-performance IC based on a RISC-type microcontroller. This microcontroller features a 32 KB ISP flash memory with read-while-writing capability, 1 KB EEPROM, 2 KB SRAM, 23 pins, three flexible counters, internal and external interrupts, a USART-type programmer, an SPI serial port, a 10-bit ADC, and five selectable internal power-saving modes. Operating at voltages ranging from 1.8 to 5.5 V, this microcontroller provides a robust foundation for project development [10].

The microcontroller utilized by Microchip Technology is the PIC16F15376, an advanced IC based on a RISC-type microcontroller. It boasts a 28 KB ISP flash memory (with read-while-writing capability), 2 KB of SRAM, 36 I/O, three flexible counters, internal and external interrupts, a USART-type programmer, an SPI serial port, a 10-bit ADC, and a programmable 'watchdog timer' using an internal oscillator [11].

For the Arduino platform, the program acts as an inseparable companion, providing a set of instructions to guide the platform on what actions to take and how to execute them. In this project, the integrated development environment (IDE) serves as the software, which can be installed on a personal computer. This IDE allows for the creation and submission of projects to the Arduino development platform, as illustrated in Figure 3 [12].

Figure 3. Arduino IDE.

To program the microcontroller, the MPLAB Xpress Cloud-Based IDE is employed. This online development environment incorporates the key features of MPLAB X IDE, a desktop application (as depicted in Figure 4) [13].

Figure 4. MPLab X IDE.

The LCD HD44780 20x4 allows the display of 20 columns and 4 lines of characters, totalling 80 characters. To use it with Arduino, connecting wires and a 10kΩ potentiometer are required to adjust the contrast [14].

The ultrasonic sensor HC-SR04 emits ultrasound at 40 kHz, which travels through the air. If there is an object or
obstacle in its path, the signal is reflected to the sensor, as illustrated in Figure 5. By measuring the travel time of the ultrasound and knowing the speed of sound, we can calculate the distance to the object [15].

Figure 5. The ultrasonic sensor HC-SR04.

The HC-SR04 ultrasonic sensor exhibits several characteristics, including a supply voltage of 5V, a current of 15mA, a minimum detection distance of 2cm, a maximum range of 4m, and a measuring angle of 15 degrees.

The HC-SR04 ultrasonic module is equipped with four pins: GND, VCC, trigger, and echo. For proper connectivity, the sensor's GND and VCC pins should be linked to the GND and 5V pins on the Arduino or Microchip Technology DM164143 platform, respectively. Additionally, the trigger and echo pins should be connected to any available digital I/O pins on the Arduino or Microchip Technology DM164143, as depicted in Figure 6.

Figure 6. Connecting HC-SR04 to Arduino and Microchip Technology DM164143.

The water level sensor is a user-friendly device employed to measure the volume of water drops and determine the water level. In our work, we utilized the sensor to detect water levels, and the values read directly by the Arduino can be utilized to trigger an alarm when the water reaches a specified level. The sensor operates within a voltage range of 3 to 5 V, with a current consumption below 20 mA, making it an analog sensor. The connection scheme for the sensor is illustrated in Figure 7 [16].

Figure 7. Level sensor connection: Arduino and Microchip.

The mini submersible pump, depicted in Figure 8, operates silently and is suitable for applications such as water circulation for cooling or other water recycling purposes. Key characteristics of the submersible pump are outlined in Table 1 [17].

Figure 8. The mini submersible pump.

<table>
<thead>
<tr>
<th>Model</th>
<th>JT-DC3W</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nominal voltage</td>
<td>DC3V … 5V</td>
</tr>
<tr>
<td>Nominal current without load</td>
<td>0,05 A</td>
</tr>
<tr>
<td>Rated charging current max.</td>
<td>0,18A</td>
</tr>
<tr>
<td>Noise</td>
<td>40dB MAX</td>
</tr>
<tr>
<td>The water</td>
<td>tap water, ground water, and sea water</td>
</tr>
<tr>
<td>Water temperature</td>
<td>-20 °C ~ 50 °C</td>
</tr>
</tbody>
</table>

The 10kΩ potentiometer, illustrated in Figure 9, serves the purpose of adjusting the water flow of the mini submersible pump, allowing users to increase or decrease
the flow rate. This rotary potentiometer is linear, single axial, and single-turn, with a resistance of 10kΩ. Its features include a 6mm shaft diameter, serrated spindle surface, total height of 29mm, spindle length of 20mm, and a mounting hole diameter of 7mm.

![Figure 9. The 10kΩ potentiometer.](image)

The 2N3904 transistor, depicted in Figure 10, is a widely used NPN transistor patented by Motorola Semiconductor in the 1960s. Alongside the 2N3906 PNP transistor, it is designed for operation at low currents, medium voltages, and relatively high frequencies. This cost-effective and robust transistor is commonly employed in electronic experiments.

![Figure 10. 2N3904 transistor.](image)

The FTDI FT232RL USB Serial Adapter, based on the FTDI FT232RL chipset, provides a convenient way to connect TTL (time to live) serial devices to a computer through a USB port, as shown in Figure 11. This adapter supports both 5V and 3.3V operations, allowing users to choose between voltages by adjusting the jumper on the board. It was essential to use this adapter with the DM164143 development platform for monitoring information on the serial monitor [18].

![Figure 11. The FTDI FT232RL USB Serial Adapter.](image)

3. EXPERIMENTAL DETERMINATIONS

In creating the fluid circuit monitoring system, we utilized various components: a plexiglass structure, two development boards (Microchip and Arduino), the FT232RL serial adapter, a 20x4 LCD screen, a 10k potentiometer, an ultrasonic sensor, a fluid level sensor, a submersible mini pump, a 2N3904 transistor, a fluid channel and level valve, a fluid pool, a fluid recirculation tube, and connection cables, as illustrated in Figure 12.

![Figure 12. Front and side views of the mock-up.](image)

The connections between components and sensors (Figure 12) were established based on the diagrams presented in Figures 13 and 14.

![Figure 13. Arduino development platform component connection diagram.](image)

![Figure 14. Microchip development platform component connection diagram.](image)
To program the Arduino Uno development board, I utilized the Arduino IDE software, which offers easy implementation thanks to existing libraries. The LCD with the I2C module initialization utilized the 'LCD.h' and 'LiquidCrystal' libraries.

In contrast, programming the Microchip development platform involved a larger number of lines of code due to the absence of specific libraries for the LCD with an I2C module, as it is a newer generation platform.

The experimental determinations aimed to compare the stability and precision performances of the two developed platforms: the Arduino Uno and Microchip Technology DM164143. For this purpose, various liquids with different viscosities and densities were employed, including plain water, saline water, carbonated water, Schweppes mandarin, Solevita cherry, Solevita multifruit, fruit nectar, and sea buckthorn syrup.

3.1 Case 1: Flat water

After conducting the initial experimental determination with still water, we observed that the monitoring system using Arduino on the serial monitor experienced freezing issues during data acquisition within a relatively short timeframe (4–10 seconds). Conversely, when utilizing the Microchip development platform, the system functioned normally, and data acquisition proved to be highly stable, as illustrated in Figure 15.

For Arduino, a reliable and deadlock-free data acquisition was achieved exclusively through the LCD viewing mode. As evident in the serial port COM5, data acquisition on Arduino halted, whereas on the LCD, the process continued without any issues.

The results obtained revealed that the parameters read with the Microchip development platform, provided on the COM3 serial input, matched those displayed by the LCD. Achieving a flow line level of approximately 100% required the submersible pump to operate at an RPM of 1026, causing the water collection basin's level to drop to 80% (Figure 15).

3.2 Case 2: Saline water

In the second experimental determination using saline water, the monitoring system employing Arduino on the serial monitor once again experienced freezing during data acquisition within a short time frame (4–10 seconds), as shown in Figure 16. In contrast, the LCD monitoring proceeded without any issues, leading to differing values between the COM5 serial reading and the LCD.

With the Microchip development platform, the system smoothly acquired data, and the displayed values were consistent on both the LCD and the COM3 serial readout.

Notably, due to the higher weight of saline solution compared to plain water, maintaining the same level on the flow line required an increase in the pump's RPM to approximately 1110 RPM (Figure 16). However, both systems encountered a common challenge related to the higher conductivity of the saline solution, resulting in the pool level sensor indicating an erroneous value compared to the actual one.

3.3 Case 3: Carbonated Water

In Figure 17, a new issue arises with the Arduino, where the system encounters a blockage during data acquisition. In contrast, the development platform from Microchip continues to function seamlessly in acquiring data without any problems. Notably, the error encountered by both systems with the saline solution was resolved at the time of the fluid change.

As we transition to dealing with a carbonated liquid, the submersible pump operates effortlessly at an RPM of approximately 970, as depicted in Figure 17. This RPM ensures a 100% level in the drain line, and the pool level experiences only a minimal drop.
3.4 Case 4: Schweppes Mandarin

Figure 18. Experimental determinations using Schweppes Mandarin: serial values.

Once again, the block in data acquisition using the Arduino development platform occurs swiftly and is depicted in Figure 18. Conversely, the development platform from Microchip continues to operate flawlessly during data acquisition.

Even though this experiment involves a carbonated liquid, the Arduino development platform encounters a rapid blockage in data acquisition. This time, with the presence of multiple components contributing to a higher density in the liquid composition, achieving a flow line level of 100% necessitates an increase in the pump’s RPM to 1098. Remarkably, the pool level remains consistent with the carbonated mineral water experiment, reaching 96% (Figure 18).

3.5 Case 5: Solevita Cherry

Figure 19. Experimental determinations using Solevita Cherry: serial values.

Although it took longer compared to the previous determination, a blockage in data acquisition using the Arduino development platform occurred once again. Conversely, the development platform from Microchip continued to operate smoothly without any issues. In this scenario, achieving a flow line level of 100% with the viscous liquid necessitates an increase in the pump RPM to 1062 RPM for the Microchip PIC platform. The pool level, in this case, ranged between 95 and 97% (Figure 19).

3.6 Case 6: Solevita Multifruit

Figure 20. Experimental determinations using Solevita Multifruit: serial values.

Once again, a deadlock occurred in data acquisition using the Arduino development platform, while the Microchip development platform seamlessly acquired data without any issues. The fluid used in this experiment being more viscous, achieving a flow line level of 100% required an increase in the pump RPM to approximately 1200 RPM for both systems. Notably, the water level in the pool remained at a consistent percentage of 92%, although the RPM values differed slightly: 1192 for Arduino and 1200 for Microchip (Figure 20).

3.7 Case 7: Nectar Fruits

Figure 21. Experimental determinations using Nectar Fruits: serial values.

Once again, the Arduino development platform experiences challenges in data acquisition, while the Microchip development platform operates smoothly without any issues. In this scenario, achieving a flow line level of 100% with the viscous liquid necessitates an increase in the pump RPM to approximately 1400 RPM, as illustrated in Figure 21.

3.8 Case 8: Sea Buckthorn Syrup

Figure 22. Experimental determinations using Sea Buckthorn Syrup: serial values.

Once again, the Arduino development platform challenges in data acquisition, while the Microchip development platform operates smoothly without any issues. In this scenario, achieving a flow line level of 100% with the viscous liquid necessitates an increase in the pump RPM to approximately 1400 RPM, as illustrated in Figure 21.
In the final experimental determination, a similar challenge arose in data acquisition using the Arduino development platform, while the Microchip development platform executed data acquisition seamlessly. Notably, at maximum RPM—1524 for the Arduino development platform and 1746 for the Microchip development platform—we observed a very low percentage of 44% on the drain line, with the pool level reaching approximately 95%, as depicted in Figure 22. Both systems utilized the same Arduino LCD readout.

Examining the received and transmitted signals, Figure 23 illustrates two signals on two channels: channel one displays the input voltage as a signal, and channel two presents the analog signal of the pool level sensor.

In Figure 23, signals were captured from the supply voltage depicted on CH1 (5 V) and the voltage from the level sensor depicted on CH2. Three distinct states are observed:
1. The first state occurs with the potentiometer at zero and without a diode, resulting in constant signals (4.75 V for the supply voltage and 2.25 V for the voltage on the level sensor).
2. The second state occurs between 0 and 700 RPM (before the pump starts), and it reveals an amplification of parasitic voltages on the level sensor (approximately 1.5 V peak-to-peak). These variations are proportional to the parasitic voltages introduced by the input voltage.
3. In the third state, as the pump circulates fluid between 700 and 800 RPM, the noise level on the level sensor further increases, reaching approximately 3.5 V peak-to-peak. Simultaneously, the input experiences an elevated noise signal of around 2 V peak-to-peak.

For a more detailed examination of these three states, we have narrowed the time interval from 0s - 10s to 0ms - 10ms, as illustrated in the subsequent figures (Figures 24–26).

In Figure 24, the input voltage signal exhibits a linearity of approximately 4.75 V, while the level sensor signal demonstrates a voltage linearity of approximately 2.5 V.

Moving to Figure 25, when the pump begins operation, the signals lose their linearity. Specifically, the supply voltage signal shows a voltage ripple ranging from approximately 4.5V to 4.75V. Simultaneously, the level signal displays a ripple voltage of approximately 1.95V to 2.25V, with an additional peak (noise) occurring every 2 ms, reaching up to 3.1V.

In the third state, a slight increase in ripple is observed for the input voltage signal, ranging from 4.4V to 4.75V. Similarly, the level sensor signal exhibits an increased voltage ripple of 0.4V, with peaks reaching up to 3.25V and 3.6V noise (Figure 26).
Figure 27 illustrates the reduction in supply voltage signal noise due to the presence of the diode, ensuring it doesn't interfere with data acquisition on the Arduino development platform.

To examine the diode's effect in detail, we have further divided the results into three distinct states (Figures 28 - 30).

In the third state with the diode connected to the pump terminals (Figure 30), the input voltage ripple remains consistent with the system without a diode. The level sensor signal also exhibits a ripple of 0.4V, without any voltage peaks, as observed in the state without a diode (Figure 26). However, there is an increase in the period of maintaining the HIGH level to 1 ms compared to 0.5 ms.

4. CONCLUSIONS

Following the experiments, centralized results are presented in Table 2.

<table>
<thead>
<tr>
<th>Case</th>
<th>Flow line [%]</th>
<th>Fluid level [%]</th>
<th>Pump flow [RPM]</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Arduino</td>
<td>Microchip</td>
<td>Arduino</td>
</tr>
<tr>
<td>3.1</td>
<td>100</td>
<td>100</td>
<td>74</td>
</tr>
<tr>
<td>3.2</td>
<td>95</td>
<td>98</td>
<td>158</td>
</tr>
<tr>
<td>3.3</td>
<td>95</td>
<td>98</td>
<td>93</td>
</tr>
<tr>
<td>3.4</td>
<td>100</td>
<td>100</td>
<td>89</td>
</tr>
</tbody>
</table>
To derive the data presented in Table 2, efforts were made to fine-tune the RPM of the submersible pump for both systems to maintain the flow line at a consistent 100%. Notably, the submersible pump demonstrates its least demand when dealing with 'carbonated mineral water' as the fluid, requiring a mere 918 RPM for the Arduino platform and 972 RPM for the Microchip platform.

As the viscosity of the fluid increases, the demand on the submersible pump intensifies proportionally. This is evident in the case of 'sea buckthorn syrup,' where the RPM escalates to accommodate the higher viscosity of the fluid.

However, when working with the 'buckthorn syrup' fluid, it becomes apparent that, even at maximum RPM - 1524 for Arduino and 1746 for Microchip - a flow line of only 44% can be ensured for both systems.

A noteworthy observation is the programming aspect, where using the Arduino IDE proves to be considerably simpler. The code for Arduino involves fewer lines (73) compared to MPLab (354).

5. REFERENCES

[10] https://docs.arduino.cc/hardware/uno-rev3