Water Tank Control and Monitoring System

Control system code documentation

Zahi Abou Chakra – 5468740

Last update 08/07/2025

Index

1. Program Description

The control system program was written in C++ based on the Arduino platform.

The project's layout includes two water tanks. The first accumulates water from a natural water source and the second is directly connected to the structure's plumbing system and is provided with water from the first one. This means that the system that will control the mentioned layouts must support multi-tasking.

Taking into consideration the limits of classic Arduinos' performance (single core) and the necessity to build a program that can control different parts of the system without getting blocked on a single task, the program should be created in a non-sequential manner. This means that the microcontroller (probably with a single core) running the program must perform different actions while the system is running and should not wait until the current task is done.

2. Code Description

2.1 Code Structure

To make this possible, the code was built with modularity. Starting with single actions, functions were created to do small and precise tasks (check section 3. Code Functions) such as turning on the pump, checking if the tank is being filled, checking the current water level in one of the tanks…

After defining single functions, a global system logic was created to control the system efficiently using those functions. The global system logic describes how and when single functions should be used and handles the presence of errors of different types (check section 5. Error Handling).

To make the previously mentioned logic work, the system should keep track of its own status. For this reason, a global variable called "status" was created and belongs to an enumeration list containing the different possible statuses of the system. The "status" variable allows the system to decide what actions to do and when.

But how can the status of the system change?

Only single functions discussed in paragraph 3 can alter the status of the system (change the global variable "status"). While these short functions perform small actions, they also change the "status" variable signaling to the logic a change in the system.

In case of errors, the system logic can take actions on "status", changing the system's status according to the error's severity.

2.2 Code Files

To respect modularity and make it easier for the reader and debugger to understand, the code's components were inserted into different files:

2.2.1 control.h

control.h declares the functions that perform input or output actions such as stopping the pump or reading a sensor value. The corresponding control.cpp file defines the functions and their tasks.

The functions declared and defined in these files are discussed in section 3. Code Functions.

2.2.2 config.h

This file contains all the parameters of the system. It allows the developer to set the parameter from one file instead of changing every value in all the remaining files. It also lets the developer tune the program according to the real-world setup.

The parameters are described in section 4 (System parameters).

2.2.3 globalVariables.h

It simply contains the declaration of all the global variables that will be used by different functions and parts of the program, it also contains the declaration of the enumeration list for the system status.

Global variables are 8:

1. status

Defines the global system status with the enumeration format SystemStatus. (system status is described in section 5. System Status and Logic)

2. previous_status

This variable is used for the system logic and for the system log via Serial Monitor. It simply memorizes the status before the current one and belongs to the enumeration SystemStatus.

3. source_time

   source_time is used by functions to calculate the elapsed time of filling the accumulator tank. It is useful to check if the tank is being filled after a certain time has passed.

4. pump_time

   Like source_time, pump_time is useful to check the elapsed time of the pump.

5. sup_level_at_start

This global integer variable memorizes the water level of the second tank when the pump has just started. It is used with pump_time to check if the pump is working correctly.

6. inf_level_at_start

   Same as the previous one. Memorizes the water level of the accumulator tank when the natural source was released.

7. sup_current_level

   sup_current_level is used to memorize the water level in the second tank for the current logic loop.

   Measuring different values of the water level in a tank caused by sensor noise or uncertainty can cause errors in the logic loop. This variable solves the problem.

8. inf_current_level

   Same as sup_current_level, used for the accumulator tank. Stores the water level of the accumulator tank for the current loop.

2.2.4 mainLogic.ino

This file contains the main system logic. It uses the primary functions to perform actions based on the system status. It features a switch statement that enters the case according to the “status” variable.

It also handles errors. Depending on the error type, it tries if possible, to keep the system going without blocking the main features. However, some errors makes the system not usable or dangerous, in thist case the system is completely blocked and error message is sent to the Serial Monitor every 5 seconds.

On system boot, mainLogic.ino checks for errors before starting the program (this is done inside setup()). The only errors to check in this case are sensor errors or program errors.

3. Code Functions

There are 7 functions in the program. These functions are defined and described in the “control.h” and "control.cpp” files:

3.1 int get_level (int sensor_pin)

The function “get_level” takes in as a parameter the microcontroller’s pin number connected to the water level sensor (e.g. ultrasonic sensor) and returns the water level as an integer.

In case of errors, it changes the system status and returns -1.

3.2 void start_pump()

This function simply switches ON the pin connected to the pump and changes system status correspondingly.

It also memorizes the water level and time when the pump starts, which are then used in the function check_pumping() described later.

3.3 void stop_pump()

This function stops the pump by switching the connected pin OFF, then changes the status of the system.

3.4 int check_pumping()

check_pumping is used to check if the pump is working properly. After a certain time has passed it checks if the water level in the second tank has changed (increased). It compares the current water level and the one memorized when the task has started.

It returns "0" if not enough time has passed to check, "1" if the water level has increased (pump working correctly) and "-1" if the water level has not increased (pump not pumping).

The time to check is calculated using the difference between the time set by start_pump() and the current time, and it is configurable in the config files discussed earlier.

3.5 void start_source()

This function starts filling the accumulator tank releasing the water coming from the natural source.

3.6 void stop_source()

This function stops filling the accumulator tank by switching the connected pin OFF, then changes the status of the system.

3.7 int check_source()

Just like check_pumping, check_source() checks if the accumulator tank is being filled correctly. If not, it throws a SOURCE_ERROR. It compares the current water level and the one memorized when the task has started.

It returns "0" if not enough time has passed to check, "1" if the water level has increased (source available and tank being filled) and "-1" if the water level has not increased (tank not being filled).

The time to check is calculated using the difference between the time set by start_source() and the current time, and it is configurable in the config files discussed earlier.

4. System parameters

System parameters are declared and defined in the config.h and config.cpp files. There are 3 categories of parameters:

4.1 Connection parameters

- **INF_SENSOR_PIN**: pin number of the microcontroller connected to the water level sensor of the accumulator tank. (input)
- **SUP_SENSOR_PIN**: pin number of the microcontroller connected to the water level sensor of the second tank. (input)
- **SOURCE_PIN**: pin number of the microcontroller connected to the valve that controls the water flow from the natural source. (output)
- **PUMP_PIN**: pin number of the microcontroller connected to the pump (probably via relay. (output)

4.2 Maximum and minimum levels

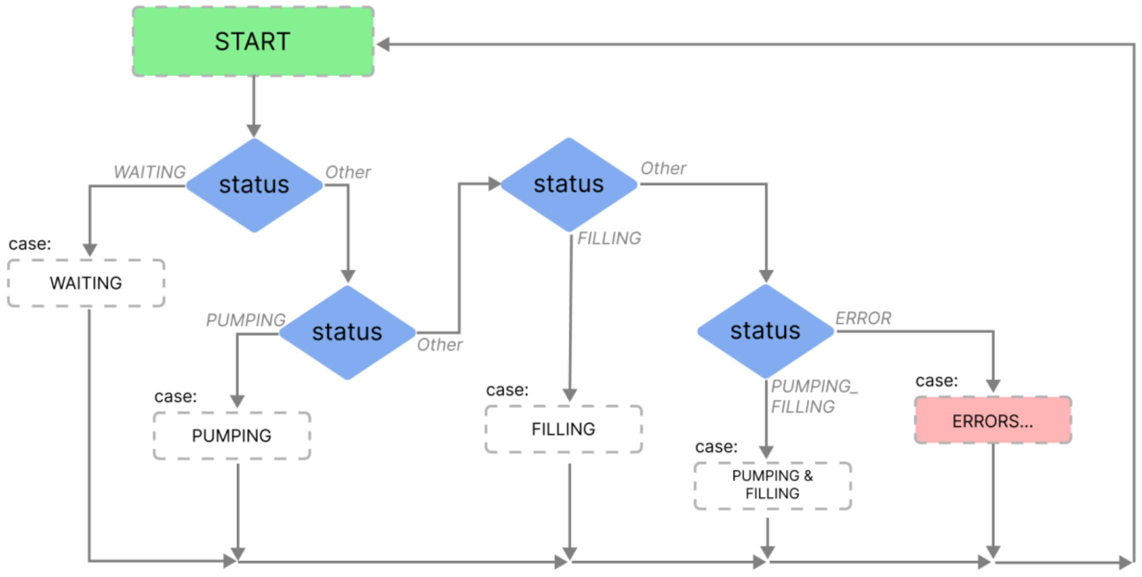
- **INF_TANK_LO**: Sensor value or minimum value when accumulator tank is Empty (Low)
- **INF_TANK_HI**: Sensor value or maximum value when accumulator tank is Full (High)
- **SUP_TANK_LO**: Sensor value or minimum value when second tank is Empty (Low)
- **SUP_TANK_HI**: Sensor value or maximum value when second tank is Full (High)

4.3 Thresholds

- **MIN_INF_TO_PUMP**: minimum acceptable water level in the accumulator tank to start pumping.
- **SUP_ACC_LEVEL**: second tank threshold for pumping (lower than threshold --> pump)
- **TIME_TO_CHECK**: time in milliseconds to check if pumping or source are working. It defines the time after which the system should check for water level difference.
- **SENSOR_RANGE**: used to avoid sensor errors due to sensor precision. It is used as the divider of the maximum water level of the tank and allows the system to create a threshold for max/min.

5. System Status and Logic

As discussed in section 2.1, the system keeps track of its status, and the logic proceeds according to it. Here's a flow chart showing the main operation of the program:



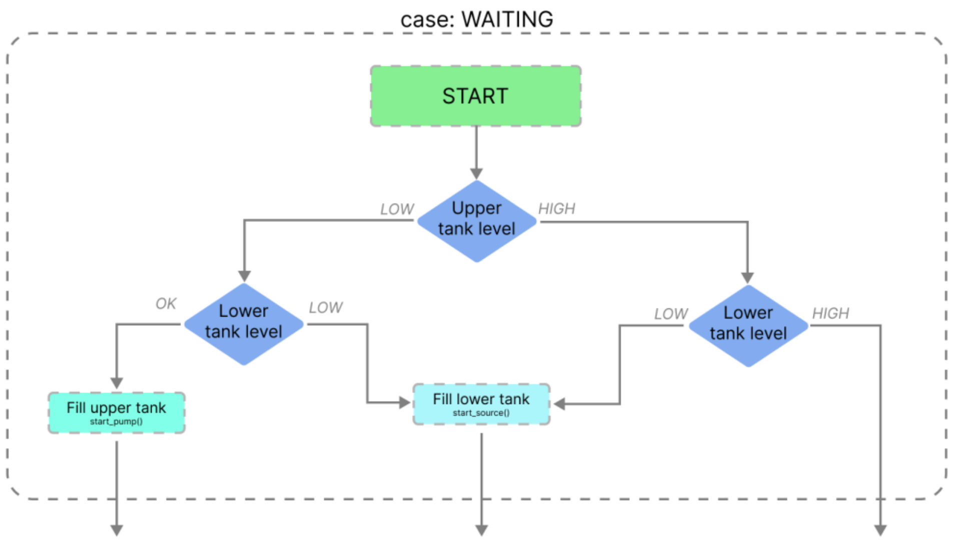
For each status, a logic program (flow chart) was designed. Every logic prioritizes the second tank, which means that the system always wants to keep the second tank filled (above the threshold), as it is the water indispensable provider of the structure.

Four statuses were declared as enum in globalVariables.h and define how the system logic should work.

Here are the flow charts that describe the behavior of the system according to its status:

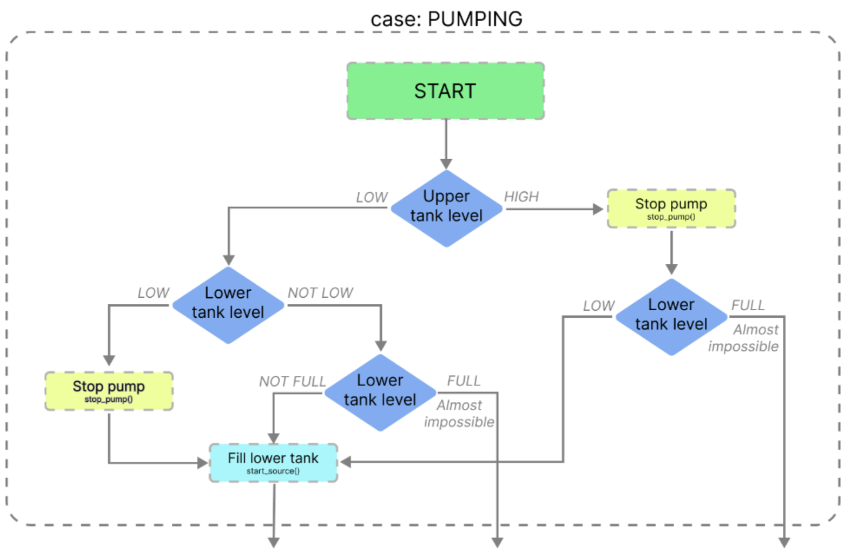
5.1 WAITING

The 'Waiting' status indicates that the system is neither filling the accumulator nor pumping to the second tank. And it is always monitoring both tanks and ready to take action:



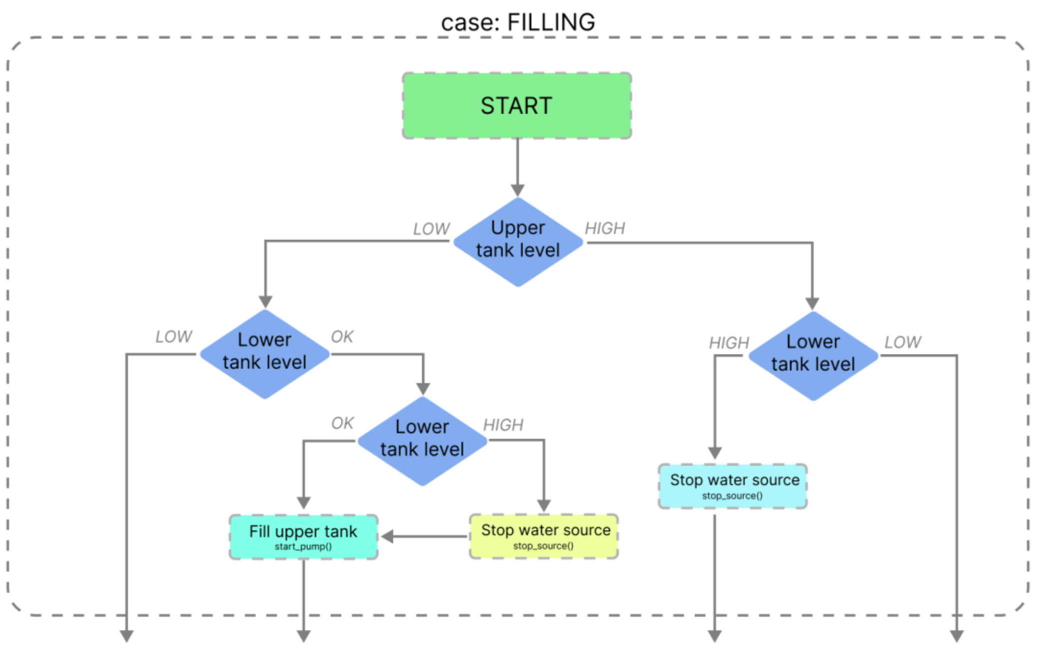
5.2 PUMPING

The system status is “PUMPING” when the pump is ON and the accumulator tank is not being filled:



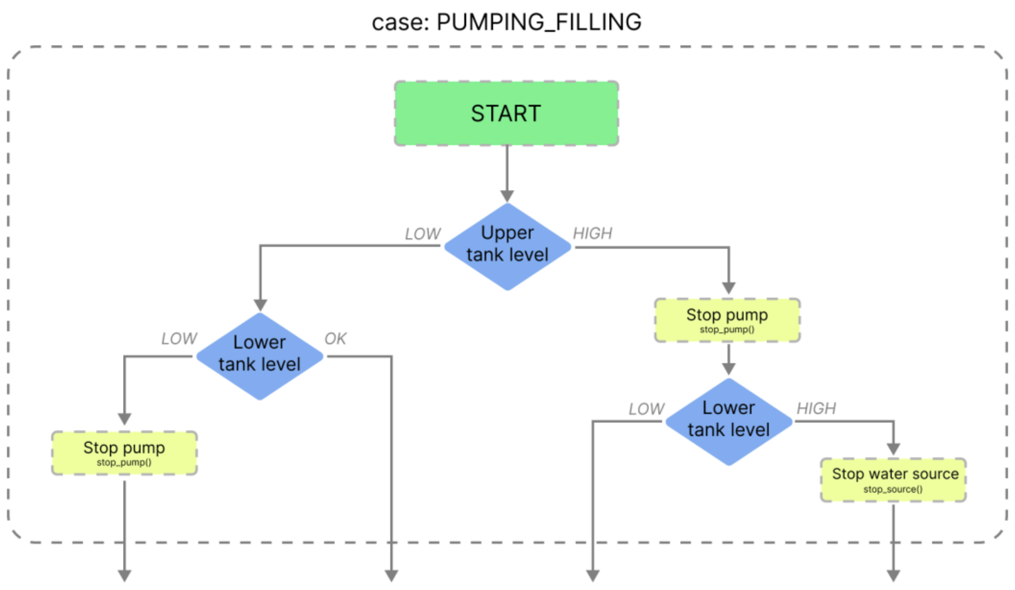
5.3 FILLING

The status is “FILLING” when the accumulator is being filled by the natural water source and the pump is inactive:



5.4 PUMPING_FILLING

“PUMPING_FILLING” means that the pump is active and the water source is released to the accumulator tank:



6. Error Handling

6.1 Program Error

In case the system notices errors in the code, it throws a program error. Such errors can be typos, errors while writing the code, not supported function parameters…

This error is irreversible, which means that once the system enters PROGRAM_ERROR it stays there for as long as the microcontroller is powered. It sends to the Serial Monitor the error every 5 seconds.

6.2 Sensor Error

When the values received from sensors are out of bound (lower than INF_TANK_LO or SUP_TANK_LO or higher than INF_TANK_HI or SUP_TANK_HI) the system is blocked. However, it will keep checking the values of the sensors. When everything gets back to normal the system will resume its operations.

It logs the outputs into the Serial Monitor every 5 seconds.

6.3 Pump Error

This error indicates that the system switched ON the pump but the second tank is not being filled.

In case of a PUMP_ERROR the system will check the water level one last time and blocks if the water level of the second tank did not change, turning OFF the corresponding pin. In case the water level increases it will resume the operation.

6.4 Source Error

Indicates that the source was released to the accumulator tank but the tank is not being filled. In this case the system will not be blocked. If there is remaining water in the accumulator tank and the second tank needs to be filled, the remaining water will be pumped. After that, the system will keep the source released and will check continuously if the water level changes.

The pumping operation will not be interrupted, so if an alternative source supplied the accumulator with water, the system would pump it to the second tank.

7. Useful Links

Github: https://github.com/ZacOS123/Water-Tank-Control-and-Monitoring-System.git