TAMPEREEN TEKNILLINEN YLIOPISTO
TAMPERE UNIVERSITY OF TECHNOLOGY

EDUARD KOBAK

TESTING FACILITY FOR A WATER RECYCLING SHOWER

Master’s thesis

MSc Thesis topic and examiner approved in the Faculty of Computing and Electrical Engineering faculty council meeting on December 4th, 2013.

Examiner: Jari Nurmi

ABSTRACT

TAMPERE UNIVERSITY OF TECHNOLOGY
Master's Degree Programme in Information Technology
Kobak, Eduard: Testing Facility for a Water Recycling Shower
Master of Science Thesis, 51 pages
2014
Major: Embedded Systems
Examiner: Jari Nurmi
Keywords: Embedded system, shower, recycling, water, testing facility, testing, carbon footprint, filtering, sustainable development

Reducing carbon footprint is one of the main global objectives today in order to stop or at least slow down the global warming. Carbon footprint reduction can be done by trying to use our resources more efficiently without wasting them on a path to a sustainable development. One such resource is water and an activity in which the water could be used more efficiently is showering. Utilizing water efficiently could be done with a help of a water recycling shower, the idea of which is to filter and recycle the showering water or graywater, as it is also called, back into the process of showering. With filters already tested, more stress testing is needed to actually create a safe and reliable product. This thesis concentrates on building a testing facility around the filters and other cleaning mechanisms. The testing facility is built out of necessary tools and an interface to control the water flow, to collect and store the data about the system for further analysis. Thus the testing facility is there to help automate the testing of the shower filters. Different tools have been tested and selected for further stages of prototype development. The basic algorithm for hot and cold water mixing has also been tested successfully for future modifications. Well selected tools and great results from the functioning testing facility have created a foundation for future product development.

ACKNOWLEDGEMENTS

I would like to thank my thesis supervisor Jari Nurmi for valuable advice concerning this thesis and his help with all the necessary paper work.

Thanks to my wife Dana for support.

Also thanks to my team Jason Selvarajan and Keiran Holland for help, hard work and a great team spirit. We can do this!

CONTENT

TERMINOLOGY

AC	Alternating Current
ASCII	American Standard Code for Information Interchange
DC	Direct Current
DUV	Design Under Verification
EEPROM	Electrically Erasable Programmable Read Only Memory
GND	Ground in Electrical Circuits
IDE	Integrated Development Environment
JTAG	Joint Test Action Group
LED	Light-Emitting Diode
LSB	Least Significant Bit
MC	Microcontoller
MOSFET	Metal-Oxide-Semiconductor Field-Effect Transistor
MSB	Most Significant Bit
MVC	Model-View-Controller Design Pattern
PC	Personal Computer
PWM	Pulse Width Modulation
RAM	Random Access Memory
SPI	Serial Peripheral Interface Bus
SRAM	Static Random Access Memory
TIP	Texas Instruments Power Transistor
UML	Unified Modeling Language
UNICEF	The United Nations Children’s Fund
USART	Universal Asynchronous Receiver/Transmitter
USB	Universal Serial Bus
UV	Ultraviolet Light
VCC	Integrated Circuit Power Supply Voltage
WHO	World Health Organization

1 INTRODUCTION

Our planet Earth is a home of 7 billion people constantly producing products out of raw materials and consuming resources while shaping our surroundings in our image. The impact on the environment because of all this activity is way too large and serious to be ignored. Majority of scientists are certain that human's activity and especially burning of fossil fuels and deforestation are the main cause for the average global temperature rise or global warming as it is called. Additionally much of the used resources are not endless but scarce and the consumption rate is unsustainable and even unhealthy for our environment in the long run.

Fortunately there are many ways engineers can help to reduce wasteful usage of important resources as well as reducing human's carbon footprint and global warming on a path to a sustainable development. More powerful, cheaper and energy saving electronic components and devices can help engineers to achieve this sustainability and cleaner future while making the use of important resources more efficient.

One of such resources is water that is used on a daily basis by households. Finding better and more efficient water use techniques is more than encouraged with rising population growth, because cleaning used or dirty water consumes tremendous amounts of energy. The best solution is the one that with certain adjustments and slight changes in how we use water at home would reduce significantly not only the water used, but also energy. This is the main motivator of this thesis work and showering is the household activity that this work will concentrate on.

Showering is one of many water consuming activities practiced by households, an activity where a lot of clean water is wasted too. There are also those who prefer rather long showering sessions where water consumption rises enormously [1]. Reducing water consumption can be done by anyone by reducing the lengths of showering sessions, which can already drop water usage dramatically per person. This offers an area for research to find other low cost solutions to reducing water consumption during showering. One such solution is to try to clean water locally and use it again or in other words recycle it with the help of a water recycling shower.

There is also an enormous need and a great benefit of creating a low-cost shower that can consume small amount of water with cleaning and recycling capabilities. According to the WHO/UNICEF latest report released in 2013, about one third of the global population lacks improved sanitation facilities. Mostly these are the areas of poor infrastructure and hygiene with the lack of clean water or where clean water is a scarce resource [2]. This is another reason why a water recycling shower is an ideal target for research.

The task of a water recycling shower is to do exactly what its name implies, to filter used showering water and return it back into the process of showering thus reducing the amount of used water per person per one showering session. The key component of the shower is the filter, yet testing of the filter is outside the scope of this project and is done as a part of another work. The goal of this thesis work is to create a testing facility for such water recycling shower and give way for a possible future product. The idea is to simulate a water recycling shower, with a possibility to observe the behavior of a system as a whole, measure water temperature changes within the system and power consumption of the components used as well as offer us a tool for revealing weak spots for later modification and design changes. The testing facility for a water recycling shower will be a necessary step before creating a prototype. The testing facility will include various sensors, a mechanism for water circulation and data collection and a control interface. Thus it will combine mechanical and electronic components, software and hardware forming a complete system. Therefore there are many things that need to be considered in the design process.

A concept of a water recycling shower is discussed in chapter 2 where various water recycling approaches are described as well. Design process of a testing facility for a water recycling shower is discussed in chapter 3 along with the theory behind it and a complete design plan. Component selection and implementation are discussed in chapter 4. Graphical interface of the testing facility is discussed in chapter 5. Chapter 6 deals with system functionality, verification, results and discussions about possible problems. Chapter 7 includes conclusion and further discussions. Sources and any additional material can be found in the end of this work.

2 STATE-OF-THE-ART DESIGN

What makes a traditional shower that can be found in every westernized household and that is used on a daily basis? A piping system for delivering hot and cold water, a constant flow of pressurized water, a drainage system for waste water and a shower head. Then of course there is a cosmetic part which can raise substantially the total cost of the whole shower facility. Yet the core idea is the same in every shower. You use clean water only once as it continues its flow down the drain all the way to a wastewater processing station.

Recycling the water locally is another way to re-imagine a shower and showering. It is a new approach to save water and to make showering into a much “greener” activity. Instead of allowing the used water to exit into a drain, the water is kept within the system of the shower facility for the whole showering experience and is reused. Approaches of achieving this might differ, yet there are some essential components that need to be present and are addressed next.

To keep water within the system of the shower facility, used water needs to be collected into a basin or tank. During showering the tank’s exit to drainage is closed and exit to circulation is open. Differences in design can already be done here by choosing tank’s material and size. The tendency is to keep the height of the tank or basin as low as possible. The water recycling shower facility should not require too much space for installation and should use only the smallest amount of water necessary to create a proper showering experience. This is the requirement for a truly revolutionary design.

To generate water flow the shower facility should be equipped with a pump. There are also special requirements for such a pump. It must be silent, small, energy efficient, yet powerful enough to deliver the water from the basin through the filters all the way to the shower head. The water should come out of the shower head with enough velocity and pressure to replicate traditional showering. Thus a good pump can do much, yet a well optimized short piping system of the showering facility can help to reduce water path as well as a smartly designed shower head can sprinkle water at higher pressure.

Before water is delivered from the basin into reuse, it must be filtered and cleaned. This is the core and necessity for water recycling shower facility. There are several of filters and cleaning methods that can be used, and many of them can be applied together like activated carbon and UV light. The filtering techniques, activated carbon types and cleaning methods as well as testing results and theory of the filter are outside the scope of this thesis and are discussed in another thesis [1]. No matter what techniques are used, they must be well tested, efficient and safe and produce water that is drinkable over and over again. It is way too easy to frighten possible users with technology that is uncertain, especially with technology that deals with cleaning and recycling dirty water. There is no room for mistakes.

It is not enough for a household product to be safe and efficient; it must also be well designed and attractive to consumers. All of the previously mentioned parts must be integrated into specially designed casing. A minimalistic design with analogue knobs, a display and sensors with feedback can give a water recycling shower facility a futuristic look and take of what was before a traditional shower into the 21st century.

For a truly “green” product for outdoor usage in remote areas where clean water resource is scarce, solar panels can really revolutionize an essential hygiene product with a carbon footprint equaling zero.

There have been those who have tried to create something close to a water recycling shower. Probably the most successful in this is an Australian company called Cintep. They have created commercially available, very futuristic looking water recycling shower that uses pasteurization to clean the water. The shower uses a mixing mechanism to mix hot and cold water, a pump to pump the water up into the mesh filter. The water is then directed into a hydrocyclone that removes particles that are heavier than water with 30% of the dirtier water going into the drain, leaving 70% of the clean water for further filtration in carbon filters. After the carbon filters, water enters a heater that performs pasteurization, then comes the water mixer and showering can begin. The design is truly fascinating compared to a normal non-circulating conventional shower, yet there is a 30% water loss and 70% is re-used [3]. Then there is another Australian company called Quench Recycling Showers. Their commercially available shower works in three phases, washing yourself with soap and letting the water exit through the drain following by a water circulation phase where water is filtered, pressurized, heated and re-used. Third phase is shower cleaning phase that is done automatically. Easy installation is guaranteed by the company. There seems to be water loss only in the first and last phases, while the second phase continuously cleans and circulates the water. The usage of soap is not recommended during the second phase, which leaves space for research of cleaning soap water in real time energy efficiently. This way all three phases could be combined into one without pauses, to copy the working of a conventional shower [4].

If close to 100% percent water re-use and circulation can be achieved in a shower safely and energy efficiently, something we believe is possible, then a shower could be built as an appliance without any need of an additional piping system to deliver the water, giving a great added value to a product that could be used in remote areas and areas that lack proper water delivery infrastructure.

To achieve a state-of-the-art design of a water recycling shower with a purpose of reducing our global carbon footprint requires a lot of hours of testing, component selection, design and redesign. This thesis describes the beginning of such an attempt.

3 EMBEDDED SYSTEM DESIGN PROCESS

Designing embedded systems, or systems that are designed for certain purposes and are a part of a bigger system, is a process that requires broad and diverse knowledge from the designer. Often big projects have different people with different skills participating at different stages of the project. Electrical engineer is there to design and test circuits while mechanics designer selects the right mechanical components to meet the requirements of a specified system. Then there is a software engineer to program a possible graphical interface that could allow a user to use the components and the integrated circuit to achieve a certain goal. Each of these individuals should have a wider scope of knowledge and skill than is required just by their narrow specialization to create a more or less flawless product that is safe, efficient in achieving its goal and is user friendly. Software and electrical engineers must understand how components work and mechanical designer must have knowledge in electronics. Thus wide technical understanding is very important and so are the design patterns.

In embedded system development process, design patterns matter and designers must be aware of the design process techniques and approaches to sustain effective teamwork, to stay within strict deadlines that are dictated by the market, to avoid mistakes that can come at a high cost and to create robust systems that can be supported throughout products' life time and upgraded if necessary. It is well known that the more time is used for specifying an embedded system accurately, the smaller is the risk of mistakes and misunderstandings. Design patterns help exactly in this case by laying a way not only how to specify, but how to produce and test systems in a time saving way effectively.

There are two approaches to the design process and both of them are normally used, a top-down approach and a bottom-up approach. Top-down approach is used to make the design itself with various sketches figuring out the connections between different parts of the system, its components and their responsibilities, starting from the highest level of design abstraction and approaching towards lower levels of design abstraction until the specification is reduced to simplest building units. Bottom-up approach begins with specifying the simplest elements that form a bigger subsystem, and subsystems that form the entire system, thus moving from lowest abstraction levels towards highest abstraction levels [5].

Bottom-up and top-down approaches are divided into various phases and each of these phases includes a specification. In each specification there is an implementation description that explains how certain things of this specification are implemented, and works as a basis for the next specification [5]. The V-cycle model in figure 3.1 encapsulates the usefulness of both approaches fairly well, where the top-down approach is used in the design process while the bottom-up approach in the implementation process.

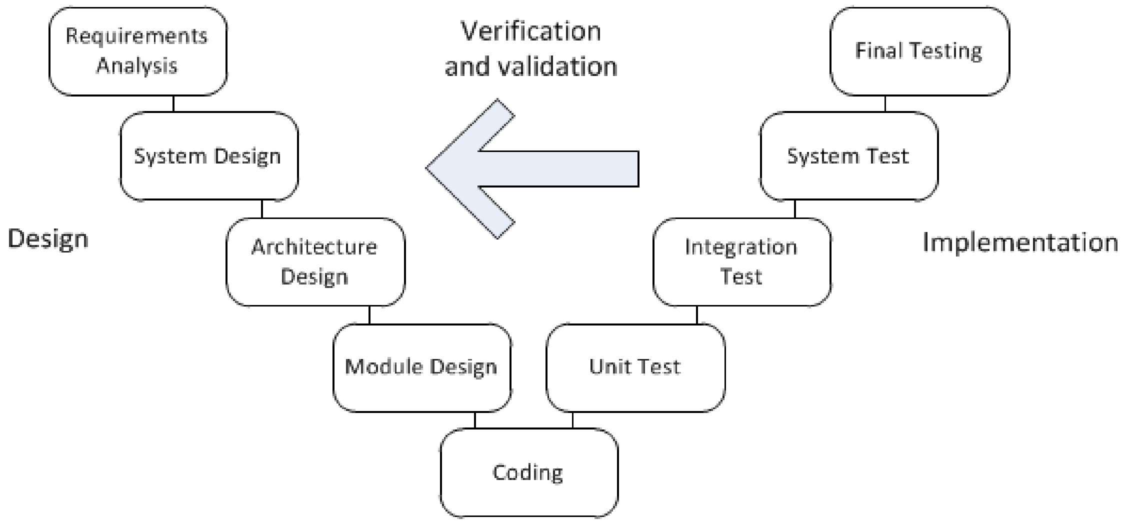


Figure 3.1 V cycle model

In the V-cycle model, each phase is not only tested for correct behavior but also verified against corresponding specifications made in the design phase. In the final phase a complete system is tested and compared to original system requirements. This model emphasizes an early and frequent design testing and verification [5][6].

Diagrams are used in specification to help to visualize the design at different abstraction levels and show hierarchy and dependencies between components. One such helpful diagram is a block diagram where the main system parts are described as blocks that have connections with other blocks [7]. With block diagram essential connections and possible bottlenecks can be recognized and emphasized during the implementation process, where written and specified ideas are transformed into code and physical systems. Block diagrams or similar can be used to display connections between functional units on any abstraction levels. Other useful diagrams like UML diagrams can help to describe the relationships between objects and can be used at different stages of design process. Use case diagram, one of many UML diagrams, can describe an outside system view with actors or system users and various use cases representing ways of using the system. Another UML diagram is a class diagram which is used to show an interface of system or its modules constructed out of classes and having properties and dependencies [5][6].

After the blocks and their tasks are recognized, they are encapsulated into modules for example with similar tasks. The idea of encapsulation, or hiding functionality within modules and offering only interfaces to that functionality can make the whole system and some of its subsystems maintainable and easily testable. These interfaces become the only way for different modules to communicate with each other. The idea of modularity and encapsulation is especially important when writing code for better maintenance, testing and reusability. One clear benefit is that two communicating parts of the system don't need to know anything about their inner implementation as long as they have a common language to communicate, in other words if the interfaces of both units are familiar to both parts.

There are various software architecture patterns that can help to divide certain functionality into modules. Architecture patterns are normally high abstraction level representations of the system that guide design process. One such pattern is a Model-View-Controller pattern or MVC shown in figure 3.2 [7].

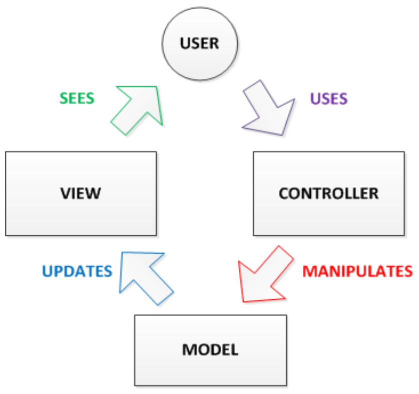


Figure 3.2 MVC pattern

In MVC the separation is made between a controller that user uses, a model that stores the system state or any other logic and a view that displays whatever is needed to display. Thus, the user uses a controller to manipulate the model which state or data is then updated to the view and which is seen by the user for possible further actions. This separation makes each part of the system independent, reusable and replaceable [7]. Manipulation of the model and view update happen through interfaces as mentioned earlier.

If design is supposed to be constructed out of ready and available components, the difficult part is then to find the right tools and components to achieve a specified goal. To be able to find the right components, behavioral requirements of the system to be implemented and its constraints must be well specified and known. The search is done within those limitations.

Thus the core ideas of the design process is to understand the environment where the future system will function and specify this future system with enough details in such a way, so that there would not be any question in the development process about system functionality. This means that a highly abstract idea of a system should be broken down into details about it, which are then broken further into more specific details and so on until the smallest easily implementable units of the system are described and can be implemented without any problems and with total understanding of their functionality. Design process is the foundation for design implementation, testing and integration and the success of these post-design stages heavily depend on how well the foundation has been constructed.

3.1 Design Process of the Testing Facility and Goals

The same design rules apply here as with designing any embedded system. Testing facility should consume as little power as possible and have a low cost. From the point of view of performance, design should be good enough to do the tasks it is supposed to do. Design should be safe to use and be able to provide reliable results. Environmental effects should also be taken into account.

The lowest element of this work, not counting transistors used in wiring and needed to control components, is a component already designed to perform a certain task, like a sensor, that is then connected to other devices to form a bigger subsystem. The components themselves are more or less considered as black boxes without the need to know how they are made inside, as long as their functionality and characteristics are understood. Because of system's size and complexity having hardware, software and communication wiring, both of design approaches are used, each to a certain extent. Bottom-up approach is used with testing each component separately and making sure that the smaller parts work correctly before combining them to form bigger subsystems.

A top-down approach begins with a design specification or functional specification. The functional specification states all the requirements the system must fulfill and gives a starting point for the whole design process. There are various types of specifications depending on how abstract or technical they are. Before more technical or architectural specification can be done, the goal, or what is it we want to achieve, must be clear and specified. The goals of this thesis are stated below:

1. Build a testing system or a test shower with water recycling and cleaning capabilities with a manual control as well as an automatic execution options
2. Collect valuable data and information from a working prototype system
   a) Allow to measure water temperature changes at full flow
   b) Power consumption calculations
3. Find components that can be used in product development process
4. Create a method for mixing water from cold and hot sources to achieve a certain temperature
5. Offer a design for finding bottlenecks
6. Create a system that can help further product development

Thus the goal of this thesis is not to create an end product, but a system or a testing facility that will simulate a working shower and help to draw future conclusions and make design changes that will help to create an end product.

3.2 Overview of the Testing Facility

Part of the functional specification is to set guidelines for the design process. In this subchapter general characteristics of the testing facility are described before moving into more technical details.

Testing facility will have a main basin where water will be collected, stored, flown into the recycling process and eventually out of the system. Testing facility should be able to circulate water out of the main basin into the process of showering through the filters. The showering water is then collected back into the main basin for reuse. Thus, the first key component is a pump that will do the circulation. The pump should withstand high water temperatures, as well as have a waterproof wiring. The pump should also be powerful enough to create a real showering experience. The pump should be controlled, switched on and off digitally though a graphical interface. The main basin where the pump is located will have another exit point from which water will exit without reentering the testing facility any more. There will be two entrance points through which new water will enter the main basin. One entrance point is for hot water and the second entrance point if for cold water. The mixing of water will happen in the main basin. Initially there will be no additional devices to fasten the mixing process, it will happen naturally.

To control new water flow into the main basin and used water flow out of the whole system, valves will be used. One valve will be stopping hot water and another valve will be stopping cold water from entering the main basin. Whenever new water is needed, valves will be opened for short period of time to let new water enter the main basin. One valve will be stopping the water from exiting the main basin and will be opened whenever used water is not needed any more. Valves will be controlled or opened and closed through the graphical interface. Valves will need to withstand high water temperatures.

Graphical interface will offer user a possibility to select water temperature. This will be the temperature of water in the main basin. To correctly mix hot and cold water to achieve the selected temperature, the initial hot and cold water temperatures must be known. For this purpose waterproof digital thermometers will be used, with characteristics to work at high temperatures. One temperature meter will be measuring cold water source and another temperature meter will be measuring hot water source. Additionally one temperature meter will be in the main basin to follow the temperature of the mixed water. To help mix water to temperature wanted in the main basin, a digital fluid level meter will be used to track the water level in the main basin.

Connecting and controlling all previously mentioned components will require two microcontrollers. The two microcontrollers will be used to form a network of connections, sensors and controllable devices and allow access to these components. One microcontroller will be responsible for data collection. It will have all digital water temperature sensors and a fluid level meter connected to it. The second microcontroller will be responsible for controlling the devices attached. It will have all the valves and the pump connected to it. This way data collecting part and controlling part of the testing facility will be kept separated for practical reasons and efficiency.

Both microcontrollers will be connected to a data collecting and controlling center, which in this case will be a PC. All collected data will be gathered and displayed in a PC's graphical interface. Controlling part will be done on a PC though a graphical interface. This means that a user will be able to open or close a valve from an interface. The user will also have an option to select automatic execution and will be able to select a few other options, like water temperature wanted. In automatic execution the system will do everything else to create a mix of water in the main basin at a selected temperature by opening and closing valves automatically. The sketch of the testing facility at the highest abstraction level is shown in figure 3.3.

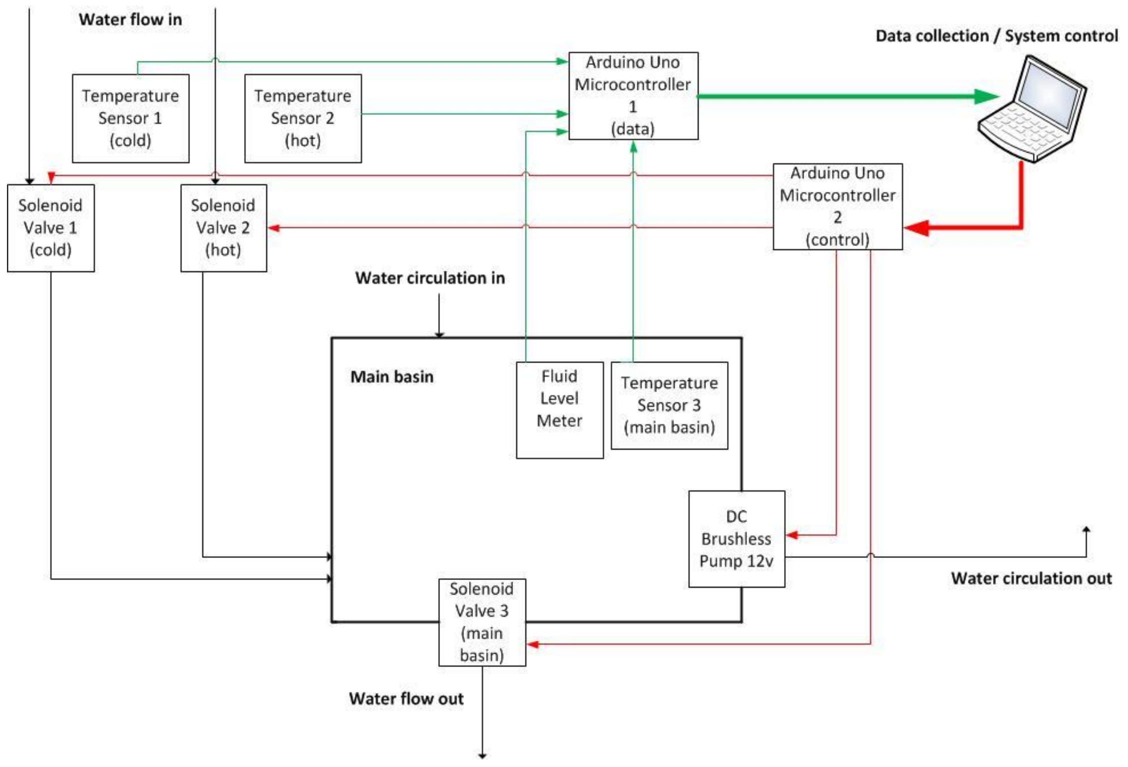

Water flow in
Data collection / System control
Arduino Uno Microcontroller 1 (data)
Temperature Sensor 1 (cold)
Temperature Sensor 2 (hot)
Arduino Uno Microcontroller 2 (control)
Solenoid Valve 1 (cold)
Solenoid Valve 2 (hot)
Water circulation in
Main basin
Fluid Level Meter
Temperature Sensor 3 (main basin)
DC Brushless Pump 12v
Water circulation out
Solenoid Valve 3 (main basin)
Water flow out

Figure 3.3 Sketch of the testing facility

3.3 Valve States of the Testing Facility

As mentioned in the system overview, thee valves will be used to control water flow into and out of the main basin. In this subchapter the states of the valves and the states of the testing facility will be described in more details as a part of a functional design process. In this discussion valve 1 will mean a cold water valve, valve 2 will mean hot water valve and valve 3 will mean a main basin valve. Each valve can be either open or closed.

In the starting state of the testing facility as the system is activated, all three valves are closed. This is also the case in the ending state as the system is deactivated. The main basin is empty and there is no water flow. Filling the main basin happens with valves 1 and/or 2 open and valve 3 closed. Valve 1 is opened to let the cold water into the main basin. Valve 2 is opened to let the hot water enter the main basin. Valve 3 is never open at the same time with valve 2 and/or valve 1. Main basin is emptied with valve 1 and valve 2 closed and valve 3 open. The main basin emptying phase is always performed before the ending state.

The valve states are shown in the table 3.4 along with valve state signs: “0” means a closed valve and “1” means an open valve.

Table 3.4 Valve states of the testing facility

Testing facility states	**Valve 1 (cold water)**	**Valve 2 (hot water)**	**Valve 3 (main basin out)**
Initial state, starting/ending state/working state	0 - closed	0 - closed	0 - closed
Main basin emptying state	0 - closed	0 - closed	1 - open
Main basin filling state – hot water	0 - closed	1 - open	0 - closed
Not used	0 - closed	1 - open	1 - open
Main basin filling state – cold water	1 - open	0 - closed	0 - closed
Not used	1 - open	0 - closed	1 - open
Main basin filling state – hot and cold water	1 - open	1 - open	0 - closed
Not used	1 - open	1 - open	1 - open

3.4 Selecting Specific Main Basin Temperature

Selecting main basin water temperature happens through a graphical interface. After temperature selection, the rest of the system takes care of mixing hot and cold water so that the selected temperature of water could be achieved in the main basin. Three digital temperature meters as well as a fluid level meter are there to help achieve selected temperature of water. Because this is a prototype building phase to see how the whole idea of a water recycling shower can be materialized into a working design, slight variations of main basin water temperature from the selected temperature are allowed.
Differences of ±2°C from the selected temperature will be satisfying. This will form a perfect basis for further modification of the water mixing process.

The heater to warm the water in the main basin or to just keep main basin water at certain temperature will not be used. No additional devices or methods will be used to cool main basin water either. Thus there will be a natural process of water either getting cooler or getting warmer depending on the initial basin water temperature and the temperature of the surrounding place where the testing facility is used. Graphical interface will offer a user to select a minimum and maximum temperature of water in the main basin. If main basin water temperature will hit minimum temperature or below, hot and cold water mixing process will begin to achieve initially selected water temperature in the main basin. Mixing of hot and cold water will also happen if the maximum temperature point is exceeded to achieve a selected cooler water temperature. Therefore the user will select a range of allowed water temperature for water in the main basin. Thus the minimum range will be 4°C, ±2°C from the selected temperature. The lowest value for the minimum point will be +6°C. Therefore the lowest selectable value will be +8°C. The highest value of the maximum temperature will be +50°C. This approach will help to test and find out if it is possible to make a working product without a water heater/cooler. Either way this will create a starting point for any other possible alternatives.

There are two options where to perform calculations based on which hot or cold water valves will be opened or closed. One option is to carry them out in the application running on a PC that will offer a graphical interface. Another option is to do it in the microcontroller responsible for controlling valves directly. The first option is chosen in this project for several reasons. By performing all the necessary calculations on the PC, only the commands of which valve to open and which valve to close are sent to the valve-controlling microcontroller. This way a clear hierarchy of one master (PC) and two slaves (data gathering and controlling microcontrollers) is formed. Another reason is that the PC offers better computation capabilities and concurrency for performing different processes. This division of responsibilities will also simplify the code for the valve and pump controlling microcontroller and allow it to do its only main tasks: activate and deactivate certain components based on the commands from the PC.

3.4.1 Calculating Water Ratio for Attaining Final Temperature

When mixing two liquids at different temperatures, heat is gained by one liquid and lost by another until equilibrium is attained. Thus there is a temperature change in both liquids. The amount of heat required for temperature change of certain mass in any material or liquid can be calculated by using the following formula:

(3.1) $$Q = mc\Delta T$$

In this (3.1) formula ***Q*** represents heat required, ***m*** stand for mass, ***c*** stands for specific heat of the material and Δ***T*** represents the change in temperature [8]. Assuming that there is no heat exchange with the surroundings, mixing the two liquids of different temperature and finding out solution’s final temperature can be done by using the following derived formula:

(3.2) $$Q_{liquid\_1} + Q_{liquid\_2} = 0$$

From formula (3.2) it can be seen that no heat is lost based on the assumption made earlier, only transferred. In the case of this work, liquid 1 and liquid 2 are water. Let liquid 1 be water from the hot source (whot) and liquid 2 be water from the cold source (wcold). Thus rewriting formula (3.2):

(3.3) $$(m_{whot} * c_{whot} * \Delta T_{whot}) + (m_{wcold} * c_{wcold} * \Delta T_{wcold}) = 0$$

Because we are dealing with the same substance, water, specific heat value (***c***) will be more or less the same throughout the equation (3.3) equaling 4190 J/kg*K at constant pressure. In fact, the value of specific heat (***c***) of water varies by less than 1% between 0°C and 100°C and can be excluded from formula (3.3) in this case [8]. Thus the formula is simplified even further:

(3.4) $$(m_{whot} * \Delta T_{whot}) + (m_{wcold} * \Delta T_{wcold}) = 0$$

Formula (3.4) ends up with values of mass (***m***), and temperature change (Δ***T***). By setting approximate real values for cold and hot water that reach households, 8°C and 50°C respectively, Δ***T*** can be opened for further calculations:

(3.5) $$(m_{whot} * (T-50°C)) + (m_{wcold} * (T-8°C)) = 0$$

In formula (3.5), ***T*** stands for final temperature after attaining thermal equilibrium. If selected final temperature to attain is set to be 20°C, by replacing ***T*** with this value we get the following result:

(3.6) $$m_{wcold} = 2{,}5 m_{whot}$$

The ratio in previous result (3.6) tells that with every water portion of m_{whot} there needs to be 2,5 times the water portion of m_{wcold} to reach a specified outcome, assuming that the mass of water portions of cold and hot water are the same.

The next step is to find out what is the water portion size in liters after the kilogram ratio is known. Density of the liquid defines how many liters of liquid equals one kilogram. In the case of water, one liter of water equal one kilogram. The final temperature to attain is also known and is 20°C in this example, but the amount of water with this final temperature is not yet stated. Stating water amount to be attained at certain temperature is important, because water portion size will change according to this amount. From the assumption of final water amount being 10 at liters and final temperature of it being at 20°C, we can calculate water portions of hot and cold water by dividing the target amount by the total amount of portions in the following way:

(3.7) $$10\ l / (1 + 2{,}5) \approx 2{,}86\ l$$

From this (3.7) follows that adding approximately 2,86 liters of hot water at 50°C and total mass of approximately 2,8 kg with 7,15 (= 2,5*2,86) liters of cold water at 8°C and total mass of approximately 7,15 kg into the same container, the final outcome is a water amount of approximately 10 liters at 20°C as the thermal equilibrium is attained. This excludes the fact than in the real example, the transfer of heat would not be so perfect because of other environmental factors and in this case a container, which would either capture or release heat depending on its initial temperature, mass and its material.

After finding out the ratio in liters of cold and hot water of certain temperature to attain a certain outcome, a way to implement this must be found. To achieve this, the area of a container's base must be known in advance. The liquid level sensor can help to tell the level of the liquid in centimeters. Thus from these values the volume occupied by water can be calculated with familiar formula (3.8).

(3.8) $$\textit{Volume} = \textit{Length} * \textit{Width} * \textit{Height}$$

Because 1 liter equals to 1000 cubic centimeters (cm^3), the volume in cubic centimeters is easily convertible into liters. Therefor the container can be filled with cold water until certain point and filled with hot water until the highest point in the container that would equal to 10 l or 10000 cm^3. For example by using the previous ratio values of hot and cold water and converting these ratios into cm^3, the result would be 2860 cm^3 and 7150 cm^3 respectively. Assuming the length and width of the container to be 50 cm and 20 cm respectively, the volume of 10000 cm^3 will be at 10 cm height (3.9).

(3.9) $$Volume / (Length * Width) = Height$$

With this container size and a goal of having 10 liters at 20°C, by using formula (3.9) cold water at 8°C would need to occupy the container until 7,15 cm height value and the rest approximately 2,85 cm would be occupied by hot water at 50°C.

3.4.2 The Algorithms: Water Ratio and Mixing

The code that will perform water ratio calculations should have as little hard coded constant values as possible for it to be reusable and applicable to various settings. Before the water ratio calculating algorithm can be allowed to start, the system must be aware of the maximum and minimum boundaries for temperature selection, the width and length of the container base must be defined, the final water amount to be attained must also be defined. These are the necessary values that must be given by the user to make the testing process more convenient for these values can easily change as the product develops. Thus the algorithm should be scalable for container size increases or decreases and changes in final water amount that should occupy the container. Pseudocode of the algorithm for water ratio calculation is shown below:

```
1: Start this algorithm as final temperature is selected.
2: Calculate cold water and hot water coefficients.
3: Get absolute values of the coefficients.
4: If cold coefficient is bigger,
        divide it by hot water coefficient and after this, replace hot water coefficient
        by 1.
5: Else if hot coefficient is bigger,
        divide hot coefficient by cold water coefficient and after this, replace cold water
        coefficient by 1.
6: Else,
        replace both coefficients by 1.
7: Divide final water amount that should be attained by the sum of hot and cold water coefficients to find out the portion size in liters.
8: Calculate hot water and cold water total amounts to be mixed by multiplying each coefficient by standard portion size.
9: Convert previous values into cubic centimeters to get volume size of each water type.
10: Divide both previous volume sizes by the base area of the container to find out the corresponding height values.
11: Calculate the maximum height value by dividing the final water amount (in cubic centimeters) that should be attained by the base area of the container.
12: Start water mixing process.
```

After the calculations are done, the water mixing process can begin. The decision of what valve to open first and sensor data gathering for state selection are done in this part. Water ratio results are used by this algorithm. The system state must be “0” and the main basin must be empty before this water mixing algorithm can be executed. The following algorithm begins with comparing height values calculated by the water ratio algorithm. Pseudocode of the algorithm for mixing and pre-showering state selection is shown below:

```
1: If the height value of cold water is bigger,
        select a state to open cold water source valve.
2: Else if the height value of hot water is bigger,
        select a state to open hot water source valve.
3: Else,
        select a state to open a cold water source valve.
4: Collect data information from data controlling MC.
5: If the water basin level has not reached the height value calculated for the water which valve is now open, go to 4.
6: Select a state to open the other water source valve and close the one that was just open.
7: Collect data information from data controlling MC.
8: If the maximum height level has not been reached, go to 7.
9: Main basin full, close all the valves and start water circulation/showering by selecting appropriate state.
```

The water mixing algorithm fills the main basin with water which is supposed to occupy at least half of the container. As the appropriate level is attained corresponding to the height value, the other valve is opened to fill the rest of the container. As soon as the maximum height value is attained, the valves are closed and the pump is turned on by selecting corresponding state.

The water mixing algorithm described in this chapter is in its basic form. It ignores the temperature variation in the water first arriving to the tank. The more precise way of mixing different water temperatures would be to do it in various steps with recalculation of ratios after each step. The main goal is to make the algorithm work in its basic form first, all in one step without recalculation. Modifications of the algorithm will be done later if needed, yet this is outside the scope of this thesis.

4 SYSTEM IMPLEMENTATION

Implementation of the testing facility for a water recycling shower covers many subjects that are discussed in this chapter. To implement a system the right components are needed, thus component selection is discussed in section 4.1. Subchapter 4.2 discusses about an important communication method that is used within the system. Coupling of the selected components is described in subchapter 4.3 in greater detail. Programming microcontrollers that are used to form the core of the system is explained in subchapter 4.4. Then of course there is the graphical interface that is also a part of the system, yet designing and implementing graphical interface is a much wider subject and is discussed separately in chapter 5.

4.1 Component Selection and Characteristics

Finding the right components that can solve a particular problem takes time. First of all the problem that has to be solved must be well specified to set boundaries for the search. Being familiar with various technologies available, what could possible work and what is unnecessary to achieve the goal, can help system designer tremendously and save time. Thus staying up to date with technological development is extremely important in the IT field, especially in the case of working with hardware and software.

Component datasheets are the place to start. They offer all the necessary information about the component: its usage, coupling information, schematics and its characteristics. Already at this stage some components can be preferred over some other components. For example a component can be discarded for not being able to withstand high temperatures in the case where the final product must have this capability. Other most important characteristics for embedded systems that area often considered are memory sizes (RAM, flash, ROM), processor speed, power consumption and amount of supportable peripheral devices by the processor. Therefore studying datasheets is an essential task any embedded system designer has to do.

One obvious criterion that heavily influences any component selection process for the design is the price. This is especially true when designing consumer products of the future. Any expensive component will be reflected in the higher price of the final product. This might raise a question of the product being any more profitable to create it as it is, or if the alternatives for the component should be searched for. Components with less unnecessary characteristics can help to drive the total costs down. This is especially true with embedded systems that are designed to be good for achieving a certain purpose, and not be good in everything. Again the importance of the well-made specification cannot be overstated.

There are other factors that can influence component selection. For this project, time schedules and component availability on the market led to selection of some components, because of the lack of the other components on the market. Additionally, wide availability of support in the form of guides, forums and communities was highly valued.

The main components of the project and their crucial characteristics are introduced next. The reasons why these components were selected are also discussed. More of this is discussed in the subsection 4.1.7, where I go through other considerable components.

4.1.1 Solenoid Valve

Solenoid devices are used to convert electrical energy into mechanical energy. Such electromechanical solenoids have an inductive coil around a mechanism that can be moved with magnetic force when enough electric current applied to the coil. Solenoid valves belong to this category and are used to control the flow of fluid.

There are various solenoid valves designed for commercial and industrial purposes that can withstand steam, high pressure and extreme temperatures and transport hot water, acids and oils. Materials for the valves are selected accordingly for each purpose. Solenoid valves can be either unidirectional, allowing water to flow only in one direction, or they can be bi-directional allowing the flow in both directions.

Preference is given to DC solenoids over AC solenoids. DC solenoids have a longer lifetime, are less noisy, have better magnetic force. AC solenoid valves have a smaller response time and thus work faster than DC solenoid valves. DC solenoids have no current alteration and thus cannot be damaged by it.

Working temperature of the plastic solenoid valves chosen for this project is up to 75°C. Each solenoid valve is unidirectional and requires between 6-12 V of DC to work. Initially the valve is normally closed meaning that there is no flow of fluid and can be opened by applying the necessary voltage. This kind of solenoid valve was chosen for its low cost, size and lower power requirements compared to some other solenoid valves on the market. Additional technical specifications of the plastic solenoid valve can be found on the vendor’s website [9].

4.1.2 DS18B20 Digital Temperature Sensor

DS18B20 is a digital temperature sensor from Maxim with plenty of features that make it applicable to a variety of systems. The version of the sensor used in the project is waterproof with capabilities to withstand up to 125°C and precision of ±0.5°C over the range of -10°C to 85°C. The SRAM scratchpad on board of the sensor offers nonvolatile EEPROM registers for additional configurations, like storing alarm triggering temperature values. It is also possible to configure resolution for analog-to-digital conversion up to 12 bits for better precision.

The SRAM scratchpad is 9 bytes long. First two bytes (byte 0 and byte 1) store temperature value as LSB and MSB respectively. Bytes 2 and 3 are used for storing high and low alarming values respectively. Byte 4 is a configuration register for setting the resolution for A-D conversion. Bytes 5, 6 and 7 are for the device’s internal use and cannot be overwritten. Byte 8 contains a cyclic redundancy check byte for data verification.

The DS18B20 sensor can be connected in parallel on the same 1-Wire bus with other DS18B20 sensors, because each sensor has a unique 64-bit serial code. This serial code is used for sensor identification purposes on the bus and in communication over 1-Wire with the microcontroller.

Sensor can function in “parasite power” mode. This is one way of powering the DS18B20 sensor, in which power is stolen from another bus when its signal is high. The stolen power not only helps to power DS18B20, a portion of its charge is stored in the capacitor for later use when the bus is low. This mode also requires fewer wires for coupling it with the microcontroller. Additional technical specifications can be found in the sensor’s datasheet [10].

4.1.3 Liquid Level Sensor

The task of a fluid level meter is to measure the fluid or powder level within a container. One fluid level meter is used for testing purposes to follow the water level in the main tank. Knowing the water level and the size of the container allows mixing of water for achieving certain water temperatures in the main tank. It is also used to inform how much free space there is in the main tank when filling it with water.

Fluid level meter operates by sending a resistive output. This value is formed by fluid or powder in which the level meter is in, which causes hydrostatic pressure on the meter’s surface. This causes a change in resistance corresponding to the height from the fluid level to the top of the meter. The resistive output is inversely proportional to the level of fluid, thus the actual water level can be calculated.

Fluid level meter can withstand water temperature up to 65°C. The level meter used for this project is around 25cm long and requires 0.5W to function. Various other lengths of the level meter can be ordered from the producer for specific applications. The sensor's datasheet can be found on a distributor's website [11].

4.1.4 Arduino Uno R3

Arduino is a commercial low cost open source microcontroller platform for hobbyists and artists, who want to build electronic projects for different purposes. Arduino and its utilities offer an easy and affordable way to learn more about electronics, sensors and interactions between different electronic devices. Universal serial bus (USB) can be used for communication between a computer and Arduino microcontroller. Therefore it is possible to control Arduino and electronics attached to it from a computer through a graphical interface, gather different sensor data and display or manipulate it.

Arduino microcontrollers are armed with a processor, memories and many input/output pins for digital and analog signals. Arduino Uno R3 board with ATmega328 low powered microcontroller created by Atmel is used in this project. ATmega328 has a 16 MHz clock oscillator, offers 32 kB of flash memory for the program sketch, 2 kB of random access memory (RAM) for working and 1 kB of electrically erasable programmable read only memory (EEPROM) for storing data that won't be lost after reset or power off. Arduino's operating voltage is 5 V. Additionally SPI and I2C communication methods are supported.

Arduino Uno R3 has 14 digital I/O pins and 6 analog input pins. It can also be configured for external interrupts on two pins. The use of pulse-width modulation (PWM), or controlling the amount of power on the output, is possible on 6 digital pins. More information about Arduino Uno R3 can be found on Arduino website [12].

4.1.5 TIP120 Darlington Power Transistor

TIP120 Darlington transistor is used to control high-power electronic devices. This is done with three pins that a transistor has: a base pin, a collector pin and an emitter pin. The base pin that uses a smaller current controls collector current flow, which is a larger current, through the transistor and exiting through the emitter pin. This means that in the circuit, a transistor can operate as a switch or as a flow controller. A transistor operating as a switch is either completely on, allowing current to flow, or completely off with no current flowing. A transistor operating as a flow controller can allow more or less current to flow. Thus it can be used along with a microcontroller to form a circuit and allow controlling the behavior of other devices in the system. This particular transistor can switch up to 60V at continuous 5A current.

In this circuit, three solenoid valves and a pump are connected to Arduino and controlled by Arduino with the help of TIP120 transistor, which operates as a switch. Additional heat sink is used with TIP120 transistor to dissipate the heat. More information about TIP120 including a datasheet can be found on the distributor's website [13].

4.1.6 DC Brushless Pump

There is a large selection of direct current (DC) pumps on the market with various characteristics: voltage range from 12V to 24V, providing different water pressure with different water flow capabilities. Being able to withstand hot water temperatures above 60C, being able to provide 5-8 bar water pressure, having 12V power supply and being easily connectable to the circuit were the main criteria for selecting a pump for this project. The pump selected for testing purposes was not the cheapest on the market and cost approximately 55€. The pump utilizes a brushless electric motor powered by a DC source with an integrated switching power supply to produce alternating current (AC) signal to control the motor. When compared with much older and cheaper brushed DC motor, brushless motor is more efficient in converting electrical energy to mechanical energy and provides more torque per weight, is less noisy and more reliable. Brushless DC motor's speed can be adjustable yet this characteristic is not used in this project, thus the pump either works at full speed or is completely off.

The pump selected for the project is 45W with a water flow rate at 40l/h. It is possible that this pump will not be powerful enough to move the water through the filters to provide a proper showering experience. Though the pump is powerful on its own, filters and the length of the pipes from the pump to the shower head will provide plenty of resistance and slow down the water flow. Thus, various other pumps with different power characteristics will be considered if necessary.

4.1.7 Other Considerable Components

One difficulty with probably any similar project is to estimate minimum requirements for the components used. There are various other components that are considered during this project. The idea is to look for components that can help to make the most efficient, low cost and durable system with little power consumption. Sometimes it means that to achieve better durability or performance, the cost can go up. Sometimes it is possible to do exactly the same thing in a more cost-efficient way by excluding all the unnecessary component characteristics. Thus not everything needs to be a tradeoff. The only way to achieve the most balanced solution is that one must search and study components and their characteristics.

Brass solenoid valve can be used to replace a plastic solenoid valve. Brass valve is more durable yet costs more and is bigger in size. It also doesn't need any initial pressure to open unlike a plastic solenoid valve that is chosen for this project. One advantage of the plastic solenoid valve is that it has lower power requirements. Brass solenoid valve requires 12V with 3A current to function. Both solenoid valves will be tested.

Waterproof DS18B20 temperature meter can be replaced by another cheaper version of the same sensor with lesser capabilities to withstand high temperature to do exactly the same task. With the cheaper version it is recommended to keep the temperature below 100°C which would be enough for this project. Finding cost efficient components is one of the goals for the shower testing facility and future development, yet a more expensive sensor was selected for the lack of the other in the store. Availability of products can set its own limitations.

In case of the 45W pump used to circulate the main tank water, twice as powerful pump is considered with power output around 90W. TIP-120 transistor can regulate the working of this kind of pump also. This pump needs 24V power supply and constant flow of 3,8A current to function at full potential. More powerful pumps than 90W are not considered yet with tendency to keep power consumption as low as possible.

From electronic component point of view, TIP120 transistor can be replaced by N-channel MOSFET transistor, which has less voltage and heat losses. Though TIP120 transistors are used in this project, this is more like a reminder that for the final product even various small electronic components can help to achieve a more efficient product and should be studied and considered.

Fluid level sensor is a convenient way to measure the level of liquid in the container. It will be used to build a testing facility but because of its high price, other ways to achieve the same results for a lesser cost will be welcome.

Arduino microcontrollers are excellent for this kind of prototyping or building of the testing systems. There are various microcontrollers on the market with different characteristics and it is all up to finding the exact one. For now Arduino Uno R3 seems to be exactly the kind of controller that can handle this kind of task. It has a low cost and is well documented with a large internet community for support. Just in case if the pins run out or if there is a need for more on board memory, Uno’s bigger brother Arduino Mega 2560 R3 can help with Atmel’s ATmega 2560 microcontroller having 54 input/output pins and 256KB of flash memory. These better than Uno’s characteristics are reflected in Mega’s doubled price.

4.2 DS18B20 Sensor Communication Over 1-Wire Bus

DS18B20 temperature sensor uses Dallas 1-wire protocol to receive commands from the microcontroller and send temperature data back. In this communication method data is transferred over a single data wire, thus the name. 1-Wire communication uses half-duplex mode meaning that the data is sent in one direction at a time. Both master and slave can be transmitters and receivers, yet the master side initiates and controls operations over the wire. Data is transferred in time slots using the least significant bit first order.

Before the microcontroller, which functions as a master, begins its communication with the DS18B20 sensor, which functions as a slave, it sends a search signal to get a unique 64-bit ROM code that identifies DS18B20 sensor slave. After gaining the unique ROM code or ROM codes (if there are many slaves on the bus), the main transaction with DS18B20 sensor can begin and happens in three necessary steps. Master sends a reset pulse in the initialization phase to find out if there are any slaves ready for communication. Master's reset pulse equals to keeping a signal low on a bus for at least 480 microseconds. After this the bus is released as the master goes into a receiving mode. The release of the bus by master allows the pull-up resistor (check chapter 4.3.1 for coupling) to pull up the bus signal and cause a transition from a low bus signal to a high bus signal. As soon as this rising edge of this transition is detected, the DS18B20 sensor enters a short waiting mode that lasts between 15 to 60 microseconds. After this, the sensor transmits its own presence signal by setting the bus low for 60 to 240 microseconds. The response is a must for communication to continue and all existing slaves on a bus have to do it. After getting a response signal, master can send one of the various ROM commands, this is the second step. One such ROM command (0x55) can address a specific DS18B20 slave on the bus by sending a unique 64-bit ROM code after the command. After this ROM command only the slave with the exact ROM code will respond to the master's function command, which is the third step and comes next. Function commands allow master to access DS18B20 sensor's scratchpad memory with read or write rights, initiate temperature conversion and more. To be able to get a temperature reading, function command (0x44) must first initiate temperature conversion. DS18B20 sensor will store raw temperature reading in two first bytes of its scratchpad memory. Before reading the temperature value, master must wait for at least 750ms and repeat the same steps one and two again and only then send a different function command (0xBE) to read the scratchpad memory. Instead of reading all 9 bytes with only 2 first bytes expressing the temperature value, it is possible to read only two first bytes following by master's reset signal afterwards to terminate the transaction. The raw reading is combined into a single integer by uniting first byte (low byte) and second byte (high byte) [14].

4.3 System Coupling

Data collecting microcontroller and system controlling microcontrollers are each connected to a PC via USB. The fluid level meter and three temperature sensors are coupled to a data collecting microcontroller, and three valves and a pump are coupled to a system controlling microcontroller.

All of the components are connected to a power supply or power supply pin (VCC) in one way or another. It is obvious why any component needs to be connected to a power supply or power supply pin. Connecting component to the ground (GND) on the other hand has many reasons. First of all it helps to discharge static electricity that can damage any electrostatic-sensitive components. There is of course a safety aspect to grounding devices, mainly protecting users from high voltage levels in case of failure. In physics, there needs to be voltage difference between two points for current to flow. The greater the voltage, the more current flow there can be. Thus ground works as a zero-volt reference point helping to create voltage difference between itself and power supply, creating a return path for the current.

Pull-up resistors, or resistors connected to VCC, are most common electronic components used in almost all of the coupling settings discussed further on. These are used to prevent a so called floating state, an unknown or changing state on the input pin that can either be high or low, that brings uncertainty to a system functioning. Floating state can occur when for example a switch is open, meaning contacts are separated with no current flow, and the input pin is not connected to anything causing a fluctuation in reading on the input pin. On the other hand, open switch with a pull-up resistor will cause a high state on the input pin. Closed switch, or pressed button connecting contacts, with a pull-up resistor will make the current flow from VCC to GND through the pull-up resistor preventing a shortage (directly connecting VCC to GND) at the same time, causing a low state on the input pin. Selecting a pull-up resistor value needs to be high enough not to cause too big current to flow from VCC to GND through the resistor when the button is pressed, while at the same time it must have a low enough resistance value to count as a high state on the input pin the switch is open. Thus pull-up resistor makes the input pin state predictable, either high or low and not randomly changing, even with lower amount of current. Instead pull-down resistors can also be used for the same purpose, except these are connected to the GND and not to VCC as in the case of pull-up resistors.

Coupling the components can be done either by soldering connectors or by using screw terminals for connecting two connectors together. For the testing period, the latter approach is preferable for possible changes in design. Following chapters explain how the valves, sensors and a pump are coupled to microcontrollers.

4.3.1 Coupling DS18S20 Temperature Sensor

Each DS18S20 temperature sensor has three wires: one for ground, one for power and one for data. There are two different ways of coupling this temperature sensor. The method shown in figure 4.1 has a ground wire connected to Arduino's ground pin and each power wire is connected to Arduino's 5V power supply pin. The sensor's data wire is connected to a digital pin on Arduino (data pin in the picture for reading). In this project the data wire of sensor of the warm water source is connected to Arduino's pin 9, the data wire of sensor of the cold water source is connected to Arduino's pin 10 and Arduino's digital pin 11 is reserved for main basin sensor's data wire. There is a pull-up resistor of 4.7K ohms used between data and 5V power wire.

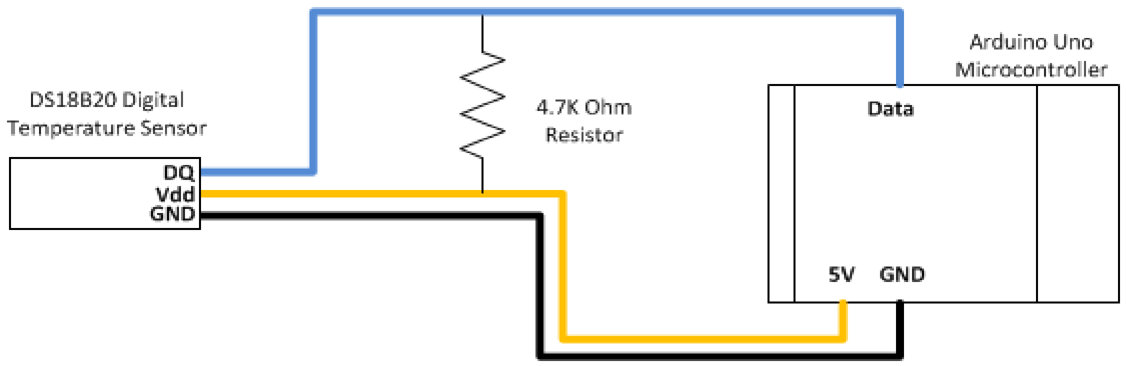


Figure 4.1 Three wires used in coupling

Another way of using only two wires is shown in figure 4.2 where Vdd and GND wires from the sensor are coupled together to be further coupled with Arduino's GRD pin. This way only two wires can be used to connect the temperature sensor to Arduino.

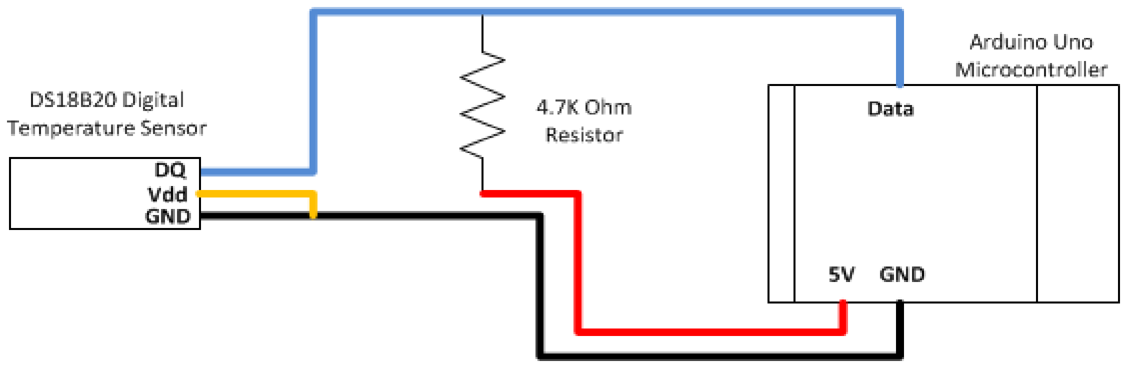


Figure 4.2 Two wires used in coupling

4.3.2 Coupling Liquid Level Sensor

The way liquid level sensor is coupled to data collecting microcontroller is shown in figure 4.3. Two middle pins of liquid level sensor (pins 2 and 3) are used with one connected to the ground, and the other to a 560 ohm resistor. Connecting the other side of the resistor is connected to VCC creating a voltage divider. Another wire from the resistor is connected to Arduino's analog input pin (data pin for reading in the picture), in this case to analog A0 pin. Liquid level sensor's pins 1 and 4 can be used for better accuracy and are not used during this project for now.

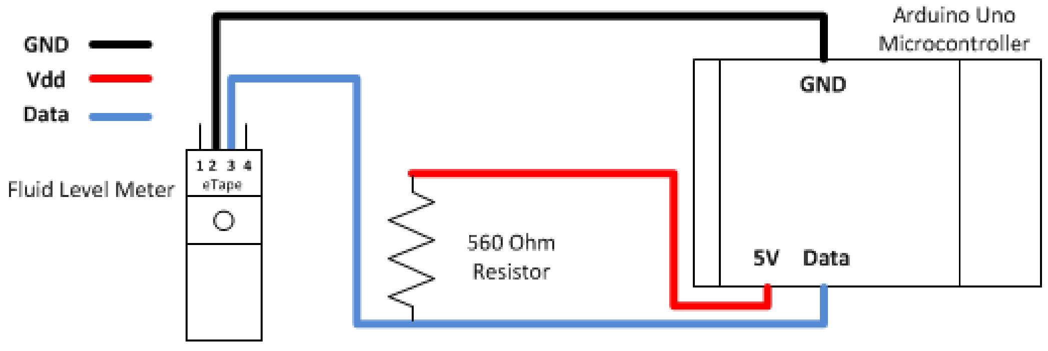


Figure 4.3 Coupling liquid level sensor

4.3.3 Coupling Solenoid Valve

In coupling a solenoid valve to a microcontroller, a TIP-120 transistor is used. The other two solenoid valves are coupled with the system controlling microcontroller in the same way. Arduino's Vin power pin is connected to valve's one end. The other end of the valve is coupled to transistor's collector pin. Transistor's emitter pin is connected to Arduino's ground pin. Arduino's digital output pins 6, 7, 8 are coupled to the base pin of each of the tree transistors used. These pins control the current though valves 1, 2 and 3 respectively. Coupling of the solenoid valve with the TIP-120 transistor and Arduino is shown in figure 4.4.

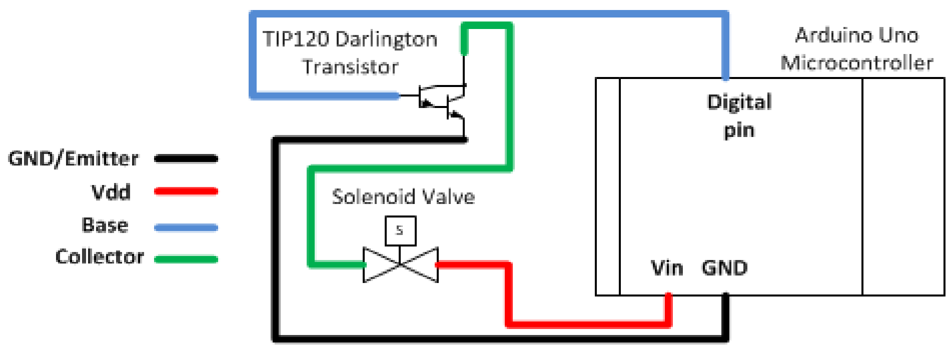


Figure 4.4 Coupling of solenoid valve

4.3.4 Coupling DC Brushless Pump

The pump is coupled with the TIP-120 transistor also, except in this case, the power is supplied by a separate power supply. Separate DC source is used because Arduino can't provide big enough current for the pump to work at its full potential. Coupling of a DC brushless pump is shown in the figure 4.5. Plus-wire is connected to a plus wire of the pump; the minus-wire the pump is connected to transistor's collector pin, the emitter pin is connected to the minus-wire of the power supply and ground pin on Arduino (GND pin in the picture). Arduino's digital output pin 9 controls (digital pin in the picture) the current flow through the pump.

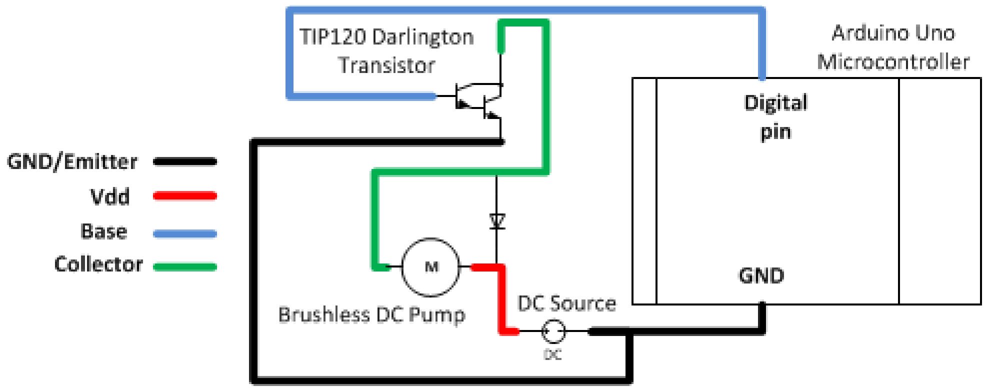


Figure 4.5 Coupling of the DC pump

Each transistor is equipped with a heat sink for heat dissipation. The transistor that is used with a pump has a larger heat sink because of allowing a bigger constant current to pass through. There is a small 1N4001 power blocking diode or TVS diode in-between the plus and minus polarities of the brushless DC pump. It is there to protect any sensitive electronic device like a transistor from short but fast reverse voltage spikes that can occur when the device that has a coil (a DC motor or a solenoid) is powered off. Thus these can be used with solenoid valves also. The diode allows a current to pass only in one direction and is coupled towards the plus polarity away from the minus polarity of the pump as shown in the previous figure. It does nothing when the pump is activated (diode blocks the current to pass through itself) in a normal situation but when the reverse voltage spike comes, the diode allows current to flow back to the coil and not the transistor.

4.4 Programming Arduino Uno R3

For convenient and fast program development, Arduino has its own open-source environment including compiler available for download for major operating systems. With this environment written programs can be uploaded to Arduino board through USB.

Arduino environment comes with a serial monitor, which is possible to use when testing/debugging written program code. Programs are written in C programming language. Programs for Arduino microcontrollers have two compulsory functions: *void setup()* and *void loop()*. These functions do not return anything because of the *void*–keyword. Instead, *setup()* function is used for initial pin settings either as outputs or inputs, serial port settings and variables initialization. Second compulsory function *loop()* includes the main functionality, commands that are performed over and over again in a loop, one after another. From *loop()* function it is possible to call any other defined function using normal C language syntax.

Both microcontrollers used in this project need to communicate over serial port with PC. Thus serial port initialization is required and is done by defining the baud rate in parentheses after the function name: *Serial.begin(9600)*. Baud rate tells receiving and sending parties at which speed data will be transmitted in bits per second. Both parties must use the same baud rate for transmission to be successful.

Data collecting microcontroller has more functionality compared to system controlling microcontroller, as it gathers temperature readings and liquid level readings. Its working is optimized to gather sensor readings only when the receiving part, or in this case PC, needs the data. When sensor readings are not needed, sensor data is not gathered by the microcontroller as it enters an idle state. When sensor data is needed and the request from the PC is received by Arduino with a *Serial.available()* function, everything is done in one loop iteration: temperature information is collected over 1-wire bus and converted into Celsius and raw data from the liquid level meter is read from an analog pin with a *analogRead(int pin)* function. The function *analogRead(int pin)* provides a value between 0 and 1023 and represents a conversion of input voltage range between 0 and 5 volts respectively. Arduino’s analog-to-digital converter performs this conversion. From this digital value liquid level in centimeters is calculated. As soon as the data is available, it is sent to the receiving party (PC) over serial port with a *Serial.print()* function. Because of the necessary *delay()* function needed to wait before reading the scratchpad memory, it takes at least 750ms plus communication overhead to see the visual representation of the data.

System controlling Arduino Uno runs a much simpler program. It also uses polling technique to look for available serial data from the sender (PC). As soon as the command is received, it is translated from ASCII scheme into a single integer which is then used in a *case* statement to choose a corresponding functionality by setting a signal high or low on a certain digital pin with two function: *digitalWrite(int pin, HIGH)* and *digitalWrite(int pin, LOW)*. In this example high and low signals are the base signals for TIP120 transistors as explained earlier. The response to a command is immediate. With 3 valves and a pump, there are 16 different combinations in a *case* statement.

5 INTERFACE OF THE TESTING FACILITY

The two microcontrollers, one for data collection and another for system control, are connected to a PC via USB cables. To make sense of data and allow user some control over the system, a graphical interface is used on PC using Windows 7 operating system. The graphical interface to microcontrollers is activated by launching an executable file developed in QT framework. Graphical interface of the testing facility is used to show the current status of the system, collected sensor data and offer controls to change system state.

There are three main system states that user can select through interface options. First state is disconnected state which is a starting state of the system. At this point computer with the interface is not connected to data collecting and control enabling microcontrollers. Thus there is no water circulation and no sensor data. After a successful connection to microcontrollers through the interface options, system enters a second state with abilities to collect and display sensor data. There is no water circulation at this state either. By setting temperature wanted and activating the pump, system enters the third state where sensor data is also collected and displayed and water is circulated: exiting the main basin through the pump, flowing through filters into the showering process and back into the main basin. Disconnecting from microcontrollers at any state will enable the starting state. Stopping water flow will enable first connected state with sensor data and no water circulation.

The flowchart of the main states of the system with options offered by the interface is shown in figure 5.1.

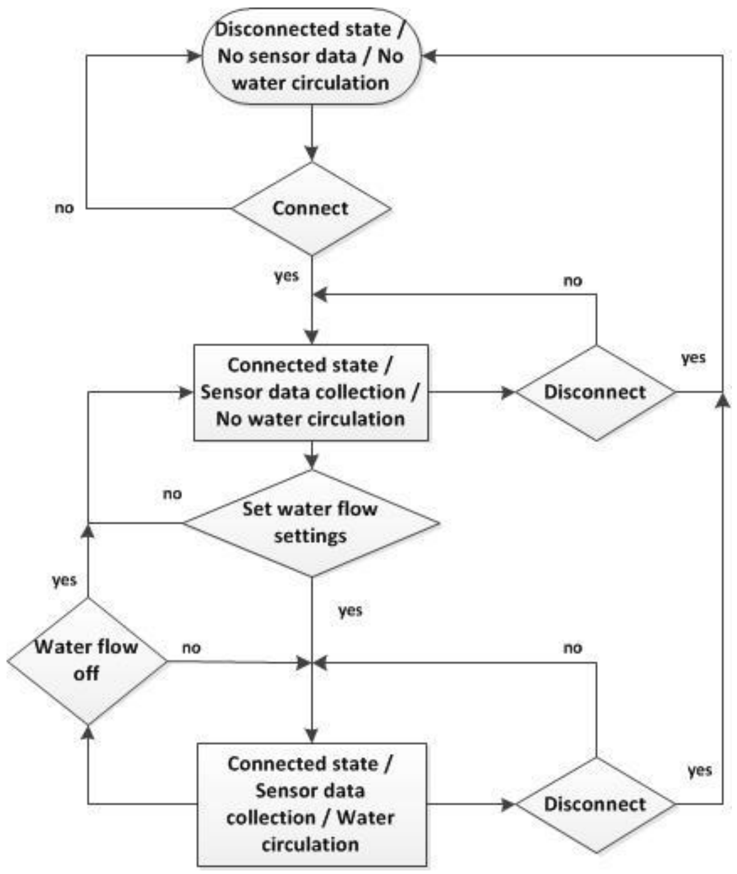

Disconnected state / No sensor data / No water circulation
Connect
no
yes
no
Connected state / Sensor data collection / No water circulation
Disconnect
yes
no
Set water flow settings
yes
yes
Water flow off
no
no
Connected state / Sensor data collection / Water circulation
Disconnect
yes

Figure 5.1 Flowchart of the interface

5.1 QT Framework

The testing facility interface is developed in QT, a cross-platform framework for graphical applications. There were several reasons for this choice. The framework is available and is free of charge with a large community and enough support material online. QT is quite powerful when it comes to graphical programming. It is based on object oriented C++, a programming language the author is familiar with.

In QT, basic object oriented C++ features of classes and inheritance are expanded with automatic garbage collection, signals and slots. Automatic garbage collection makes graphical application design process much easier by leaving memory management to the framework to take care of. In other words designer can allocate memory dynamically for various data structures or pointers with a use of a memory construct *new* and release this dynamically allocated memory automatically at the end without any additional code. Otherwise without automatic garbage collection, C++ would always require the designer to manually deallocate all dynamically allocated memory with a *delete* construct at the end to avoid memory leak or incorrect memory management.

Signals and slots are used to interconnect various objects. A signal is a method that is emitted when called. A slot is a member function that is executed after the signal emission. Signals and slots offer a way to connect graphical components with certain functions that will be executed as soon as something happens to the component itself. For example a graphical component can be a button, that when clicked, will emit a signal and cause an execution of a certain function mentioned in a slot connected to that button. This method can be used to create pretty complex systems, because any number of signals can be connected to any number of slots and as soon as a signal or more are emitted, the slots connected to emitted signals are called in an unpredictable order. A cautious approach is advised, especially when graphical application design in QT is heavily relied on signals and slots. The version of the QT framework used in this project is 5.0.2.

5.2 Designing the User Interface

The interface of the testing facility must provide controls for options described in the flowchart earlier (figure 5.1). Designing any user friendly interface is an art in its own name and deserves mentioning. Programmer or designer must approach interface design process from a point of view of a user which makes it that much harder. Software interface designing skills come with practice and take time to master, yet there are some basic things that can help and will be discussed next.

Three golden interface designing rules by Theo Mandel state that a good user friendly interface places the user in control, reduces user’s memory load and is consistent [15]. The designer should not try to limit the user too much even though it might simplify the design of the interface and various interface customization options should be provided [15]. Because of the nature of this project, mainly being a testing system for a small user group, limited resources and time, the interface of the testing facility implements the golden rules to a certain extent. There are some restrictions in what the user can do at a certain time. This limiting guidance, in a form of displaying only certain selectable options at certain moment, is present throughout the whole application’s running cycle. The purpose of this is not only to simplify the application, but to have some kind of control of what system states are available and are a logical transition from the current state without causing any damage to the system while providing needed results. For example before the user can connect to both microcontrollers, he can choose what showering modes and options will be used during the testing procedure that will begin after connecting to microcontrollers. Additionally to the water mixing mode, another showering mode is added where the system uses whatever water source it has regardless of water temperature. Selecting new settings like a different showering mode will not be possible during the testing procedure. Therefore each testing procedure is like a session with last selected settings. The main controlling interface does not change by hiding unnecessary components. For example main tank water temperature selection interface is still present even in the mode where the main tank water temperature is ignored. The same limiting guidance is used here also by simply disabling the controls for temperature selection. Thus there are many options that are locked and many unlocked depending of what the user will do next which might be confusing at first. At the same time this approach reduces user's memory load since there are only few options to choose from at each moment. Again, it is important to remember that this is an application for controlling and observing the testing environment and such limiting approach to user interfaces might not be applicable when considering applications designed for the bigger audience.

The interface provides an interface message box for guiding the user, where information is given about what is happening at a significant moment. The interface is designed so that only the information needed for testing and observation are shown and technical internals are hidden. Consistency, an important concept of designing user interfaces, is maintained by having same kind of menu styles (for example texture, text and button sizes) in each application window.

Because this is a system with many components attached, an additional selectable option provides a separate window with manual controls for all possible states of the testing facility. If activated, the testing facility will only switch states manually. This is designed to provide the user with more options for various testing situations, as well as for testing the components and circuit connections. Thus this option has nothing to do with showering modes, but is an additional feature for testing and troubleshooting.

5.3 Programming the User Interface

In this subchapter the most important parts of the testing facility interface and its C++ implementation are discussed. The program flow and QT techniques that are used are explained in greater detail.

As the program is executed, all the necessary tasks are done beforehand in the background before the *main* function is called. Compiler takes care of integrating these tasks as a part of the program. The actual program code begins with creating and showing a testing facility interface window (figure 5.2), which in the code is a *class* structure that inherits *QMainWindow* class that provides a main application window. This class also provides a ready layout with tool and menu bars and allows further main application window management through its public functions. Before showing the actual window, at its creation phase, a constructor is called that initializes the main application class, its variables, performs possible dynamic memory allocations and connects signals and slots.

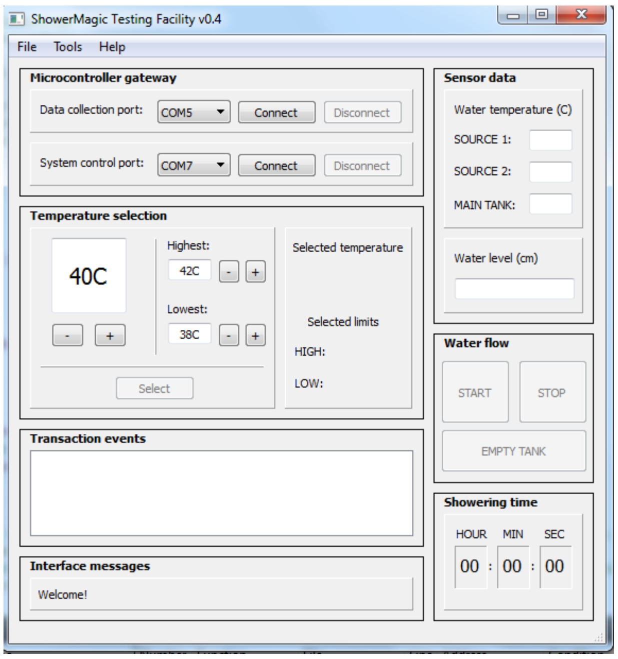


Figure 5.2 Interface of the testing facility

Signals and slots form the main functionality of the QT programs and are tied together with a special static public member. The following example of the public member uses only the necessary parameters:

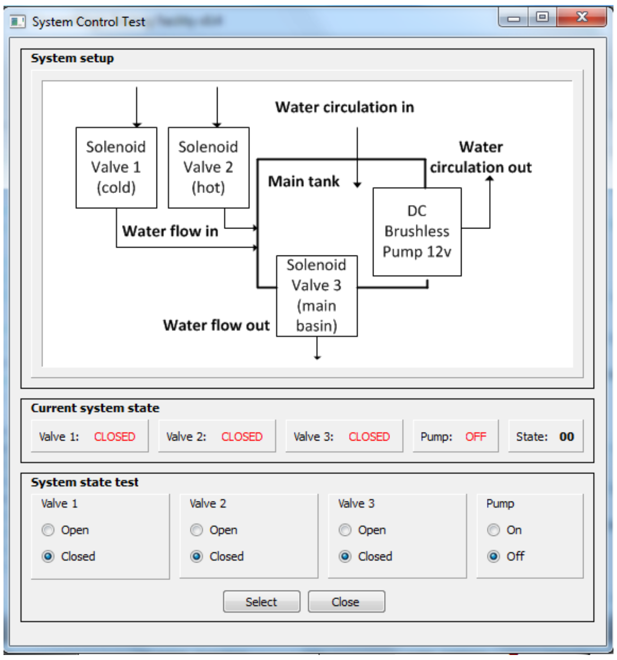
*connect(const QObject * sender, const char * signal, const QObject * receiver, const char * method)*

The first parameter in the *connect* member is a pointer to the sender, for example a button component. Second parameter defines a signal and its type, for example *SIGNAL(clicked())*, a signal macro with a *clicked()* type, meaning that a signal is emitted when the button is pressed. The third parameter is a pointer to the receiving object. The fourth parameter is a method of the receiving object that will be executed after the signal is emitted. The method is introduced within a *SLOT()* macro and can be a function like *SLOT(doSomethingAfterClick())*. Thus when the defined button is clicked, the function *doSomethingAfterClick()* is executed. The slot function must be defined in the header file within the class under the *private slots*, which is an extension of the traditional C++ in QT.

After it is created and shown, the main application window is ready for user interaction. The user can either select “Settings” menu under the “Tools” option in the menubar, or select an USB-port from the pull-down menu in the “Microcontroller gateway” section and connect to microcontrollers by pressing the “Connect” button. Connection with the microcontroller is established by opening a COM-port given by the user. Baud rate is set automatically and is the same as with microcontrollers, in this case 9600. Other necessary COM-port settings are set with default values within the code also. This includes flow control that is not used and set as none, parity for detecting errors in transmission is not used either and is set as none, 8 bits for each character in data bits settings and 2 stop bits for detecting the end of a character. Additional QextSerialPort-library is used that provides an interface for opening, closing, reading, writing and controlling communication over serial ports with QT-based applications [16]. After the “Connect” button is pressed and if the device is found on a given COM-port, the “Connect” button is disabled and “Disconnect” button is enabled after a short delay of five seconds. Forming a connection from the program with a hardware and disconnecting (and connecting again) within a very short time causes an error in the program. The short delay was necessary to allow connecting to the hardware without any user interruptions and disallowing users to use the interface in erroneous way. For this purpose a timer of *QTimer()* class of five seconds is launched with classes’ *start(int msec)* function with time in milliseconds as a parameter after each connecting/disconnecting button press. As the timeout of the timer is reached, a function is executed to enable the other button and start/stop communication. Below is an example of how signals and slots are used in this case with a *timeout()* signal type:

connect(timer_ , SIGNAL(timeout()), this, doSomethingAfterTimeout());

Signals and slots are also used in a more complex situation where one class, that has a graphical representation and where the change happens, needs to inform the other graphically represented class about this change. This is especially needed in one important case in this program. After connecting to both microcontrollers, it is possible to open a “System Control Test” window where all possible states can be tested (figure 5.3). At the same time testing facility interface window is also visible to the user.



Current system state

Valve 1: CLOSED | Valve 2: CLOSED | Valve 3: CLOSED | Pump: OFF | State: 00

System state test

Valve 1: Open [✓ Closed]
Valve 2: Open [✓ Closed]
Valve 3: Open [✓ Closed]
Pump: On [✓ Off]

Figure 5.3 State testing screen

“System Control Test” window inherits *QWidget* class which is the most basic user interface class in QT and is a *private* (object accessible only by the owner) member of the testing facility interface in the program. Communication with the system controlling microcontroller has to happen through the testing facility interface, because all the necessary functions for communication over USB are located there for encapsulation reasons and because of class responsibilities. This inevitably means that when a state is selected in the state testing screen, the parameters have to be forwarded to the testing facility interface and to microcontroller from there on. Like in the previous examples, the “Select” button that emits a signal must be connected to a slot function, let it be called *changeState()*. After the *changeState()* function is called with a press of a button, a next state is defined based on user selections and a new signal is emitted from within the function *changeState()* in the following way:

```
emit this-> setNewState(change_to_this_state);
```

With *emit* command it is possible to manually emit a signal. This of course requires that the function that is being emitted is introduced under *signals* in the header file of the class and here is an example of how it is done:

```
signals:
        void setNewState(QString change_to_this_state);
```

This signal function passes a parameter of a *QString* type that represents the next state of the system. To respond to this signal, a receiving side must connect it to a slot. In this example the receiving side is *QMainWindow* class (testing facility interface window shown in figure 5.3.1) and the following connection is written in its code file:

```
connect(sys_control_test_,SIGNAL(setNewState(const
QString)),this,SLOT(performStateTransition(const QString)));
```

The purpose of this connecting function is to connect a signal function of one class that a pointer named *sys_control_test_* points at in the example above, to a slot function named *performStateTransition(const QString*) of another class object (*this* is a pointer to the object itself, it represents another class in the example above). In other words a signal of class X is connected to a slot of class Y. The parameter *const QString* is passed from a signal to a slot function. As the slot function *performStateTransition(const QString)* is reached in testing facility interface, the parameter is then communicated to the system controlling microcontroller and the state is changed accordingly.

During the system control test any state is selectable for testing reasons, main control buttons in water flow section in the testing facility interface are disabled. These buttons are meant to be used during the normal showering process with a lot of automation, like appropriate system state selection, done in the background. Flow control buttons are enabled again as the user exits the “System Control Test” window. Enabling of these buttons and informing the main window about exiting the state testing phase is done by using signals and slots in the same way as in the previous example with two classes.

There is another user friendly function in the program. When the user exits the program either with an “Exit” option or with pressing “X” in the upper right corner of the program window, user’s settings are saved into a file for restoring them next time the user will execute the program. To be able to do this as the program is closing, the following *protected* type function is used:

```
void closeEvent(QCloseEvent *event);
```

Function with a *protected* access modifier is accessible only by the member functions of the class and by functions of classes that are derived from it. The execution of this function happens automatically as the window is being closed.

Testing and verification of code sections during runtime can be done in a debugging mode of QT environment. Execution can be paused at any given code line by using breakpoints. One way to test and verify correct functionality of a code section is to print results into a console by calling *qDebug()* function in the following way:

```
qDebug() <<  “Result of an algorithm: ” << algorithm_result_;
```

Previous example would print a local variable *algorithm_result_* into a console. This can be helpful for example to verify a correct functionality of an algorithm or generally, it can help a programmer to verify that a certain variable is what it should be at a certain program execution point. Using *qDebug()* function requires including it with an *include* preprocessor directive in the following way:

```
#include <QDebug>
```

6 SYSTEM VERIFICATION AND TESTING

System verification and testing for errors is a time consuming task. It is not guaranteed that all the bugs will be found during testing, some false system behavior can manifest itself after tests when the system is already available for consumers. Such errors can come at a high cost. The best approach to avoid or at least minimize such outcomes is to verify the system for correct behavior and test for errors on each stage of its development. Next subchapters discuss how this can be done with software and hardware effectively. In embedded systems software and hardware are often tightly related, thus some of the testing methods discussed are interconnected. Testing of the shower facility system and its parts is discussed in the following subchapters as well. The final subchapter discusses conducted tests to measure power consumption in a tendency to reduce it.

6.1 Hardware and Software Testing and Verification

Implementing a graphical interface or any software with a lot of details and functionality can hide in itself unwanted behavior. In software testing as with hardware, starting with unit and module testing before integrating them can substantially reduce unwanted behavior and errors. Behavior of modules can be verified by creating testbenches like the one shown in figure 6.1.

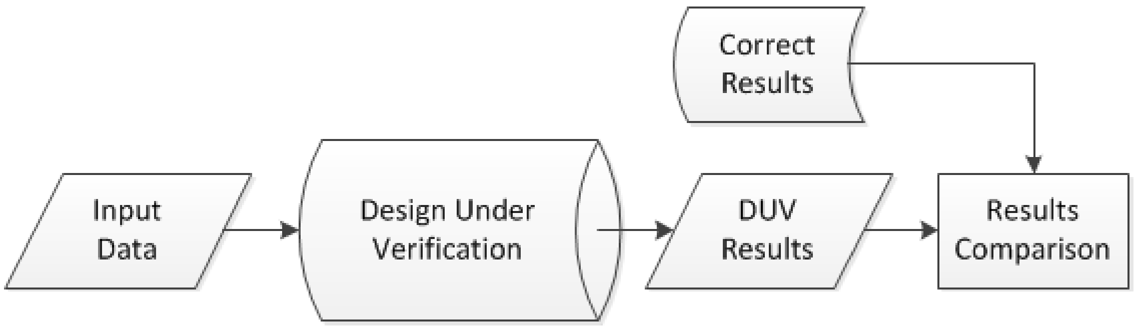


Figure 6.1 Design Under Verification

The idea of a testbench is to make verification process easier and more automatic to verify a correct behavior of a module or a design (design under verification or simply DUV in the picture above) [5]. Testbenches have input data that goes inside the DUV and produces results that are then compared with the correct results. If DUV results produced are different from the correct results, then the design should be inspected closer for false behavior. If any errors should occur during runtime at a crucial point, programs and its modules should be designed to be able to recover from them. Thus a proper error handling should be considered. This can also help debugging and finding errors by using informative error messages.

Usability testing adds another level to testing any design, especially its interfaces. In usability testing the goal is to figure out how well the user manages to achieve his goal for which the design is being made. It is held by having small group of future users to use the application [5]. This process might reveal many blind spots in the design that were left unnoticed by the designer. Thus it is helpful to think about the end user and what might happen if he or she does action X instead of action Y while using the future product and its interface.

Testing hardware for correct functionality can be done in many ways. Microcontrollers often come with an integrated development environment (IDE) that includes many useful tools (like debugger), as well as a console. The console window can be used to print the text out of the program that is running on the device to specify the current location of execution. This is a good way to test what program parts can be reached during runtime and see how the hardware is responding, thus it is also a good on-board software testing method. If there is a LED on a device, it can also be a good indicator to show if a wanted part of the code has been reached. If the LED light is not turning on when it should, then another revision of the code is needed. The weakness of these approaches is that they might not test everything and some functionality might be left untested, especially if the code is complex.

Knowing the essential electrical characteristics is important when dealing with hardware and its components. Components can be damaged not only by high current, but also by electrostatic discharge and this should be acknowledged by proper grounding. In coupling components, an essential tool is a digital multimeter that can measure voltage, current and thus the existence of electric connection between two circuit points. By knowing the required current of a component, multimeter can show if a component is getting enough current to work at its full potential. Generally digital multimeter is a good tool to confirm that the connections are functional.

For more detailed observation and debugging of system signals, an oscilloscope is the tool. Oscilloscope can display a waveform of a digital signal. It allows to observe how signal changes within a system and its behavior over time. Oscilloscope functions within a certain frequency range and thus signals with frequencies outside the range will not be displayed.

There are more advanced ways for testing embedded systems by using a logic analyzer or an in-circuit emulator if such is available. Logic analyzers are costly but can help to monitor and diagnose many digital signals simultaneously, which is especially good for systems that use buses. In-circuit emulator is a module that replicates the processor of the system. It is placed into the system being developed and is controlled by the software executed on a PC, emulating the processor and providing great debugging opportunities. The downside of relying too much on an in-circuit emulator is that it could be slightly different than the actual thing [17].

Another way to test the working of hardware is through JTAG, if such is available. JTAG is a port that offers an access to the processor and the rest of the embedded system. JTAG port provides real-time, in-circuit hardware and software debug capabilities allowing to single-step or multi-step through running code. This means that setting breakpoints, at which the execution pauses, is possible to allow for examination of register and memory location values. This is an excellent way to observe how the code behaves in hardware, yet the JTAG port is not available in every embedded device but is becoming more and more popular [17].

6.2 Testing and Verification of the Graphical Interface

QT IDE and its debugger were used intensively in testing and verifying functionality of graphical objects. Every added functional part and its graphical objects were tested as they were added before going any further. Debugging was done by printing text into the console of the IDE and by using breakpoints. For example this was used to check if the program read the user file with options correctly by printing the option information into the console after reading the file. Closing the program would activate a function that would write the same user options information into the same file overwriting old information. The file was then read to see if the options information corresponded to actual user’s selection during the program runtime. This latter part of the test was done manually by comparing results.

Slowly more functionality was integrated and tested with already working parts of the code. The technique proved to be effective as cases of faulty behavior were few and some of them insignificant. Most of faulty behavior was due to the lack of usability testing, because there was no time and resources to conduct such testing. The only tests were conducted during the programming process by the designer.

Lack of testing outside of the debugger and no usability testing did have its issues that manifested after intensive use of the program of the shower testing system. One problem occurred when rapidly opening and closing serial port connection within software through the main window interface. Problem manifested itself by freezing the graphical interface. This problem was corrected by forcing the user to wait 5 seconds before the other button would become active, thus allowing to either connect or disconnect depending on the previous choice. The problem that occurred outside the debugger in field use manifested itself by crashing the program as it was closing, after the “X”-option of the window had been pressed. This was caused by a clear mistake in the code, where a class object was being accessed after its memory had been released.

There was another problem that occurred after a long use of the program. The problem manifested itself in a slow response by the interface to user actions. It became almost impossible to use the program. This problem was caused by a component in the main window screen that was getting filled with sensor data information almost every second by adding more and more data. In the end there was no point to display all this information because the program stored everything in the log anyway which could be written into a file later. Additional option was added for the user that if selected, the same component would only show the latest message and not all of them. This option fixed the problem. Clearly this demonstrates that usability testing is extremely important as well as a more extensive testing of the final product and a longer program execution outside the debugger.

The main window was also tested for consistency. This was important, because the program and its interface was designed in such a way that it would guide the user by enabling and disabling options or buttons that can be pressed. So when one button was pressed (as a selection of temperature for example), it would activate another button (like start button to activate shower water flow) meaning that action “X” had to be done before action “Y” would be available. In some cases the button that was pressed would need to be disabled (like in the example of opening the serial port connection). Forgetting to disable certain buttons in certain situations would allow the user to press it or choose the same option again which would lead to faulty behavior or even crashes.

System state screen was used to see if every state could be set. This screen also helped in integration testing where more components were added and their responses to the commands from the interface observed. System state screen proved to be a very useful tool in testing. First of all the screen worked without errors or usability inconsistencies and second of all, the correct behavior of components within the circuit was verified quite fast. This screen was added much later after the main window was already designed and became the most essential tool during testing and would even serve as a manual state control tool during normal use.

Later the main interface of the shower testing facility was expanded by adding new options and allowing to turn other options off in the “Settings” window. This did add its own difficulties and make the code more complex with some changes made to the execution of the program. It also reflected on the upkeep of the program, which became slightly harder to do. Designing an application for possible future expansion as well as relatively easy upkeep are two program features that can save a lot of time and resources in the future and should be implemented, especially if there is a great chance that the application might grow.

6.3 Testing and Verification of Microcontrollers and Components

The components were tested with Arduino Uno R3 microcontroller separately to understand how each of them works. Valves were tested separately with MC and a button coupled in the same circuit. Every time the button would be pressed, the high signal would be sent to controlling TIP-120 transistor and the valve would open. The same was done with the pump to turn it on and off. The sensors were also tested separately and their readings printed into the console of Arduino IDE. Each component was then integrated into a circuit with its microcontroller for further testing. Verified code was later reused in the final code version to program each microcontroller.

Communication over USB between Arduino and a PC was tested by using the LED on the Arduino side to indicate a successful communication and printing values received from Arduino into grapchical components on the PC side. This method helped to understand, test and verify the essential link between microcontrollers and a PC. Verifying correct functionality of the communication path gave way for testing state changes and control of components from a PC as well as full data gathering for further analysis.

Originally a pump with a power supply of 12V and 3,5A was selected to pump water into the filters. As the test run was made, it was obvious that the pump was not powerful enough to push the water through to provide a proper showering experience. This pump was replaced with a twice as powerful pump, working at 24V and 3,7A, which provided a strong water flow even after the filters. Another thing that was revealed in a test run was that the wire that used to connect the pump with the transistor was thin enough for it to start melting. The constant current flow was simply too high for the original thin wire. After this all similar thin wires were replaced with a proper much thicker wire. Multimeter was used to measure that the pump and the valves get the right amount of current that they need to function at full power.

Constant current flow of a more powerful pump at 3,7A and 24V also caused TIP-120 transistor to heat up fast. Even though TIP-120 could withstand such electric voltage and current values, the breadboard into which the transistor was connected could not and started to melt. In the case of the pump, the breadboard was replaced by other type of connectors. The breadboard was still used with the valves, because of a smaller current flow.

For debugging purposes both the LED and IDE’s console were used. Printing text into console was probably the most efficient way to find out if the program execution on Arduino entered certain code area. The LED was used to indicate a certain important state when a complete system was tested.

Temperature sensors were tested by building exactly the same setup around each of them and comparing their results. Sensors were connected to microcontroller that would measure and print water temperature results into console window. In one test all of them were put into the same cup with water to see if the result would be the same and how fast each of the sensors would reach its final result. This was done to test for flawed sensors. Each of the sensors did provide similar results with minor differences, small enough to be discarded.

Liquid level meter testing was conducted with it being connected to a microcontroller that would print readings into the console. The liquid level meter was placed into a bucket. Then water was slowly poured into the bucket and the reading was compared to the actual mark on the liquid level meter. It did take quite some time to properly calibrate this sensor. Later it was realized to use an average of the sum of four values to get a more precise water level reading.

As integration test progressed and the final part of microcontroller testing was reached, system controlling MC was tested with its components, and data collecting MC with its components to verify functionality and look for possible false behavior. The most important was to observe if any component would interrupt the correct behavior of another component in the same circuit. This did not happen as all components worked together perfectly.

6.4 Power Consumption Measurements

Power consumption measurements were conducted to find out how much power is consumed by the microcontrollers and their components, an attempt to reduce power consumption was also made. The PC was excluded from power consumption measurements altogether because it can vary a lot depending on the computer used and its components. Also the prototype or even an end product are not going to have a PC connected to them but will use a more elegant embedded solution, thus it was more relevant to measure the power consumption of just the microcontrollers and their components at work.

For measuring power consumption, a digital multimeter was used to measure current flow. Digital multimeter was placed between the positive polarity of the DC source and the system. The system is connected to the negative polarity of the dc source. The DC power supply used in the measurements was a power adapter of 12V and 5A. Setup for power consumption measurements is displayed in the figure 6.2.

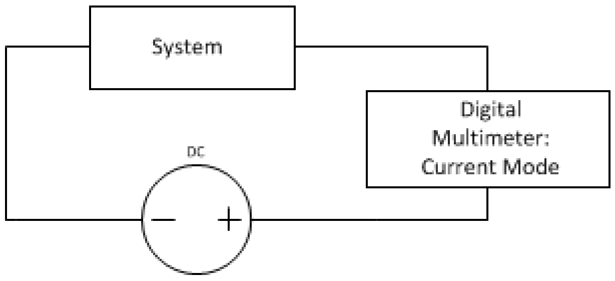


Figure 6.2 Setup for measurements of power consumption

The system in the figure 6.2 is a microcontroller with its components, thus there are two systems and two setups: data collecting microcontroller and system controlling micro-controller. Current flow results from both setups are displayed in the table 6.3.

Table 6.3 Current Flow Results

Microcontroller (with its components)	Minimum Current Flow	Maximum Current Flow
Data collecting MC	56mA	60mA
System controlling MC	54mA	1,3A + (3,75A)

The first setup for power consumption measurements was conducted with data collecting microcontroller and its components. The current flow then rose to 60mA (which is the maximum current flow of this setup) as the microcontroller started to collect data constantly from its sensors. Later the code was edited so that microcontroller would only start (and stop) collecting sensor data on a command, thus staying in an idle mode until asked to begin sensor data collection or entering idle mode if no more sensor data required by the PC. In the idle mode current flow dropped to 56mA, which is also the minimum current flow of this setup. Second setup's measurements were conducted with system controlling microcontroller. The minimum current flow required by the system was around 54mA in microcontroller's idle stage, where controller waits for a command from the host (PC). The maximum current flow reached 1,3A when all the valves (plastic) were opened and a pump's transistor was switched on. Turning on the pump, by supplying current from Arduino to transistor's base pin, consumed additional 3,75A at 24V voltage from another power adapter used only for the pump. Thus the actual maximum current required to open all the valves and make the pump work at its highest potential was much higher than what was required only by the system controlling MC. From these results it was possible to calculate power consumption of both microcontroller systems with the following formula (6.1).

(6.1) $$\mathbf{P = V * I}$$

P = power in Watts (W)
V = voltage in Volts (V)
I = current in Amps (A)

Calculated power consumption results from measured current flow results and used voltage levels (12V for microcontrollers and 24V for the pump) are shown in the table 6.4.

Table 6.4 Power Consumption Results

Microcontroller (with its components)	Minimum Power Consumption	Maximum Power Consumption
Data collecting MC	0,672W	0,72W
System controlling MC	0,648W	15,6W + (90W)

From the power consumption results it can be seen that by forcing the data collecting MC into an idle mode when sensor data is not needed, the power usage can be reduced. Also around 110W are needed to fully power system controlling MC and its components (including a pump). It is true that in this case, the reduction of power consumption by using an idle mode is very tiny, especially if compared to the total power consumption of the complete and active system. Forcing a system into an idle mode is still a valid method in an attempt to reduce system's power usage, especially when designing low-power systems.

Slight energy-efficiency was achieved with additional code modification for data collecting microcontroller by using a delay and collecting the sensor data less often. This means that the MC went into the idle mode for 2 seconds. This was possible without degrading overall performance. Using the following formula (6.2), the energy saved during this moment can be calculated.

$$\mathbf{W = P * T} \tag{6.2}$$

W = energy in Joules (J)
P = power in Watts (W)
T = time in seconds (s)

The initial code of collecting sensor data consumed 0,72W while the idle mode consumed 0,672W thus having the difference of 0,048W. By moving the data collecting MC into idle or waiting mode before collecting sensor data again, the 2 seconds saved 0.096J of energy.

Arduino Uno R3 does offer various sleep modes to reduce power consumption. In our case the events on serial port (USART) have to wake it up, thus USART should be left enabled. There is only one sleep mode available POWER_MODE_IDLE that does not disable USART. This is the least power saving mode. Unfortunately this is the same as allowing microcontroller to run in an idle mode which is already implemented and provides no additional power consumption savings.

7 CONCLUSIONS AND FUTURE WORK

The idea of having an eco-shower that can clean and recycle water and that could have a great impact on our everyday water consumption influenced me to join the team and help. The initial tests were slow and used manual valves. This project brought automation to controls and measurements as well as helped to build a foundation for such automated shower with a futuristic design. This project has also helped our team to engage in a relatively fast pace product development process, facing new problems and solving them.

In the design process some design methods described in previous chapters were applied. Block diagram helped to visualize and identify the problem. After this the appropriate components were selected and tested to better understand how they work, followed by the implementation process.

The implementation process of the testing facility could be divided into two significant and time consuming parts that were conducted at the same time. Creating the graphical interface and understanding how to communicate with microcontrollers was the first part. The second part dealt with studying electronics, programming microcontrollers and coupling them with sensors and controllable parts. Testing was done often especially in the second part, where each component was tested separately with its microcontroller before adding more components into the same system, an implementation of a bottom-up approach. The rule of making sure that the simplest things work before moving to create more complex things proved once again to be the most efficient and a fast way of building the whole system with each microcontroller.

Design and implementation processes followed the V-cycle model to certain extent. The design did not have any strict system requirements, yet there was a tendency towards a solution that would have a low cost and use as little power as possible. For example there were no restrictions on the kind of processor to use or memory size. The system was designed following certain guidelines and not strict limitations, something that would be normally present in a corporate project. The graphical interface also had loose requirements to offer controls to control the testing facility system manually and/or with some automation in the background and be able to gather and display data. There were no strict and predefined requirements of how this would look like or how this would be done, other than what was mentioned in this thesis. Thus this left a lot of space for possible related design changes at any time during project lifecycle, something that is not common in a V-cycle model. This too would be defined in great detail in corporate project by using possible UML diagrams, which is generally a better approach than the one used in this project. It is the nature of this project and lack of time and resources that did not allow to apply certain design patterns, yet it is also an indication that this project did not really need them for the target group is our team and not a consumer. Though, the benefits and importance of having well specified design and following design patterns mentioned earlier are fully acknowledged.

Tests and system verification were conducted during system integration phases.
After some test runs and minor changes and fixes, a complete system was up and ready for extensive filter testing. It was during this point when the 12V pump had to be changed for a 24V pump, because the former was not powerful enough. Water mixing tests were also conducted after the system was built. Water mixing test was the most difficult part of this project, for it took a lot of time to calibrate a water level meter to be accurate. Having the end product in mind, the water level meter cannot be used as such but it is enough for the testing facility to test the working of algorithms. Nevertheless we managed to get positive results that indicated the correct functionality of the water mixing algorithm. With water ratio recalculation it would have been possible to achieve more precise results, thus future development of the water mixing algorithm is needed. Based on these results we could not make any conclusions about using heater/cooler within the system. More tests are needed to see how well the mixing algorithm (with possible modifications) will work during showering process and not only in the beginning. For filling the main basin with certain amount of water, a flow sensor could be used to replace the water level meter, yet it must also be tested for accuracy.

Arduino microcontroller offered a fast and easy path to prototyping, testing and product development. It has helped to create a successful testing platform for recycling shower filters. The testing facility gathers temperature data and allows controlling shower states manually as well as provides automation to shower state selection and execution (goals n.1 and n.2). Power measurements were conducted and will play an even more important role in a development of an end product (goal n.2). We have also selected some components for further testing while some are under consideration for replacement (goal n.3). After a long and hard process of building a water mixing setting, successful cases of achieving main basin water at a selected temperature were conducted (goal n.4). The testing facility has helped to find bottlenecks and improve some of them (goal n.5).

The testing facility has also set a foundation for continuing to develop a product. The project of creating a prototype of a water recycling shower has been advancing at a fast pace. While creating a testing environment, new ideas have led on an exciting adventure of testing new devices. A physical user interface with waterproof buttons, potentiometer (for mode selection) and a led display is now under development for a stand-alone version of the prototype (goal n.6). Similar techniques (similar coupling, code partially) are used in a stand-alone prototype as in the testing facility. A couple of more valves (brass) are added with a more complex water circulation path allowing backwash, water circulation in a different direction through the filters and into the drain. The idea of a backwash is to try to clean the filters by allowing the water to flow in the opposite direction and take possible dirt into the drain, thus cleaning the filters.

The backwash method and filters' capabilities of cleaning water in the long run need extensive testing to find out how effective they are, even though filters' effectiveness has been proved in a short run. This would also provide us with estimation of how often the filters would need to be changed. The testing facility and a stand-alone version of the showering system that is under development will help us to test the filters not only on their own, but also as a part of a showering system. Further testing will also stress test the components that are used. Thus testing and development will continue side by side in our attempt to create a prototype and hopefully a final product.

REFERENCES

[1] Selvarajan, J. & Holland, K. ShowerMagic: A Hygienic and Eco-Efficient Real Time Greywater Reuse System for Showers. 2013, Helsinki Metropolia University of Applied Sciences.

[2] Water, Sanitation and Hygiene. [Online document - Referenced 10.09.2013]. Available:
http://www.unicef.org/wash/

[3] This Futuristic Shower Will Save You Thousands of Gallons of Water. [Online document - Referenced 10.10.2013]. Available: http://www.businessinsider.com/water-recycling-shower-uses-70-percent-less-energy-and-water-than-a-conventional-shower-2012-8

[4] Quench Recycling Showers. [Online document - Referenced 10.10.2013]. Available: http://quenchshowers.com/

[5] Haikala, I. & Märijärvi, J. Ohjelmistotuotanto. 2006, Talentum.

[6] Peckol, K.J. Embedded Systems: A Contemporary Design Tool. 2008, John Wiley & Sons Inc. p. 191-201 and p.352-353

[7] White, E. Making Embedded Systems. 2012, O’Reilly Media. p.10-12 and p.27-31

[8] Young, D.H. & Freedman, A.R. University Physics with Modern Physics, 11th Edition. 2004, Pearson Education Inc. / Addison Wesley. p. 652-663

[9] Plastic Solenoid Valve. [Online document - Referenced 18.08.2013]. Available: http://www.adafruit.com/products/997#Technical_Details

[10] High Temperature Waterproof DS18B20 Digital Sensor.
[Online document - Referenced 18.08.2013].
Available: http://www.adafruit.com/products/642#Technical_Details

[11] Liquid Level Sensor. [Online document - Referenced 18.08.2013].
Available: http://www.adafruit.com/products/463#Downloads

[12] Arduino Uno R3. [Online document - Referenced 18.08.2013]. Available: http://arduino.cc/en/Main/ArduinoBoardUno

[13] TIP120 Darlington Transistor. [Online document - Referenced 18.08.2013]. Available: http://www.adafruit.com/products/976#Description

[14] High Temperature Waterproof DS18B20 Digital Sensor datasheet. [Online document - Referenced 18.08.2013]. Available: http://www.adafruit.com/products/642#Downloads

[15] Pressman, S.R. Software Engineering: A Practitioner's Approach, 7th Edition. 2010, The McGraw-Hill Companies. p.312-335

[16] QExtSerialPort library. [Online document - Referenced 25.07.2013]. Available: http://code.google.com/p/qextserialport/

[17] Catsoulis, J. Designing Embedded Hardware, 2nd Edition. 2005, O'Reilly Media. p.126-129 and p.157-159