Control module for a gas water heater

Conference Paper · May 2015

CITATIONS
0

READS
6,579

1 author:

Dajic Armin

10 PUBLICATIONS **0** CITATIONS

SEE PROFILE

Control module for a gas water heater

Armin Dajić

University of Sarajevo

Abstract

This paper gives a detailed description of a prototype of control module for a gas water heater. All manipulations over input variables, as well as control of output relays have been handled by the microcontroller Arduino Uno. The module can be controlled using a 4x4 keypad and a Nokia 3110 display, as well as over a network connection. The theoretical part describes the principle of operation of the gas water heater. There are also some topics related to the field of thermodynamics that have been discussed. A two-point controller applied to a thermodynamic system has been described as well. The paper deals with design of the module and with characteristics of hardware components that have been used. A compatible shield has been developed in order to adequately connect components to the microcontroller. The software has been created using the open-source Arduino environment. Besides standard libraries that are part of the environment, some additional libraries have been used: for graphics of the display, as well as for time interrupts. A special library for work with generic functions for reading and writing to EEPROM memory and a function which detects keystrokes have been created. Possibilities for remote control via TCP/IP (using the ethernet shield and Telnet protocol) have been clarified. The microcontroller acts as a server which can be accessed using any available Telnet client software. A Telnet php client has been developed. In combination with a web server available on a LAN network, it openes up numerous possibilites and at the same time it can provide the global Internet access to the control module. Furthermore, the module can be connected to the Internet directly through a router with Internet access.

Keywords: Control module, gas water heater, embedded system, Internet controlled device

Table of Contents

Introduction

Embedded systems can be found in a wide range of devices today. They are usually designed to perform a specific task. We find them with devices in consumer electronics (mp3 players, portable gaming consoles, home appliances), medical equipment and transportation systems (GPS devices for navigation, traffic lights control, automotive industry), industrial electronics and the like. Developement of an embedded system includes parallel design of hardware and software. A wide range of user interfaces are used, from simple push buttons to more complex menus displayed on an LCD. Ordinary microprocessors or microcontrollers with additional peripherals usually have the role of the central processor.

Considering that in the control module for a gas boiler one of the simplest types of control has been implemented (two-position control), in the process of developing and debugging it was enough to use a display for feedback, while the algorithm and module's components were tested piece by piece.

It is an embedded mounting system for temperature control for one room which provides interface for adjusting parameters such as: temperature setpoint, hysteresis, startup time, operation mode (auto/manual), the minimum temperature difference of incoming and outgoing water after the startup (in order to verify the functionality of the burner and circulation pump).

Numerical values have precision of one decimal place after decimal comma. While writing the program, the limited memory resources had to be taken into account, especially when it comes to RAM (only 2 KB of available memory). The goal was to create an easy to use interface, with a clear display of variables and a LAN access. The interface can be integrated into a web scheduling, or an alarm cron service.

1. The principle of operation of the gas boiler

1.1. Boiler

The term refers to closed containers where liquid is heated under pressure. The fluid does not necessarily boil. Heated fluid, or steam coming from the boiler is used for various purposes. It can be used for space heating, sanitation, cooking or as a generator. In this paper, the term boiler will imply water heaters (used to heat rooms) constructed to circulate heated water (rather than vapor) through pipes. In such boilers water must not warm up to the boiling point, because the presence of steam and excessive temperature in the pipes lead to increased pressure and risk of explosive bursting. Therefore, the temperature must be maintained below the boiling point. Pan is usually made of copper, iron alloy, carbon steel or low-alloy steel, because of low prices, low risk of rupture due to corrosion, good thermal conductivity and low coefficient of thermal expansion. As a source of energy to heat fluids different burning fuels are used, such as: wood, coal, oil and natural gas. Electric water heaters use heating elements that release energy by Joule's law. Boiler’s power supply system may include a generator. It could use wind power, hydropower or solar energy.

Since the topic of this paper is a control module for a gas water heater, this chapter describes basic features of this boiler, structural design, theoretical part related to the heat transfer between different media, a mathematical model of the process of heating and control.

1.2. Gas Water Heater

Classification according to the way of functioning:

- Classic wall-mounted gas boiler
- Condensing gas boiler

In a conventional boiler fuel burns and hot gases pass through the heat exchanger, where most of its heat transfers to water. One of the hot gases produced in the combustion process is water vapor. It's produced by combustion of hydrogen in the fuel. The condensing boiler extracts additional heat from waste gases by condensing the water vapor thus recovering its latent heat of evaporation. The condensate is then used to heat the incoming liquid resulting in increased efficiency of 10-12%.

Classification according to the amount of water:

- Flow heaters
- Accumulation heaters

Flow heaters have small volume (5, 10 and 15 liters). They must have strong heaters, because the water heats up quickly when flowing through them. The accumulation heaters are generally of larger volumes (30, 50, 80, 100 and 120 liters), and they gradually heat cold water from the tank. Due to its capacity these can have more outlets than flow water heaters. The heated water (tipically 60°C, maximum 70°C) is mixed with cold water according to user preferences. The required amount of hot water is one of the basic parameters when choosing a type of boiler. It depends on the number of household members, number of outlets, and a position at which the device can be placed.

Classification according to the purpose:

- Water heater
- Space heater
- Combi

Combi boilers with integral DHW cylinder (domestic hot water) are a hybrid combination of the first and second version. These are in demand due to the relatively small size and good characteristics.

The module developed as part of this work controls a flow water heater for room heating. The response is relatively fast due to the small capacity. Parameters are adapted, especially the startup time and the minimum temperature difference of outgoing and incoming fluid. The module can be used to control boilers with larger tanks, but the startup time in that case is likely to be much longer and temperature difference of outgoing and incoming fluid established during this time is smaller. User interface of the module allows setting these parameters. If the permitted range is not sufficient it's quite simple to extend it in the code.

The essential components of the gas water heater for heating a room (Figure 1.1.) are: Burner, Heat exchanger, Fan, Circulation pump, Controllable Valves, Water pipes, Chimney.

Heat exchanger allows transfer of heat from one medium to another, and can be implemented in a way to touch the media, or to be separated by partitions which prevent their direct contact. In that case it transfers the heat from hot gases to the water in hot water pipes. The use is very widespread, from household heaters and refrigerators, to automotive and industrial devices. A key element is a network of pipes or barrier that separates the two environments. Medias are usually exposed to different pressure. Heat transfer involves convective heat transfer from the first fluid to the barrier, conduction through the barrier and convection to the second fluid.

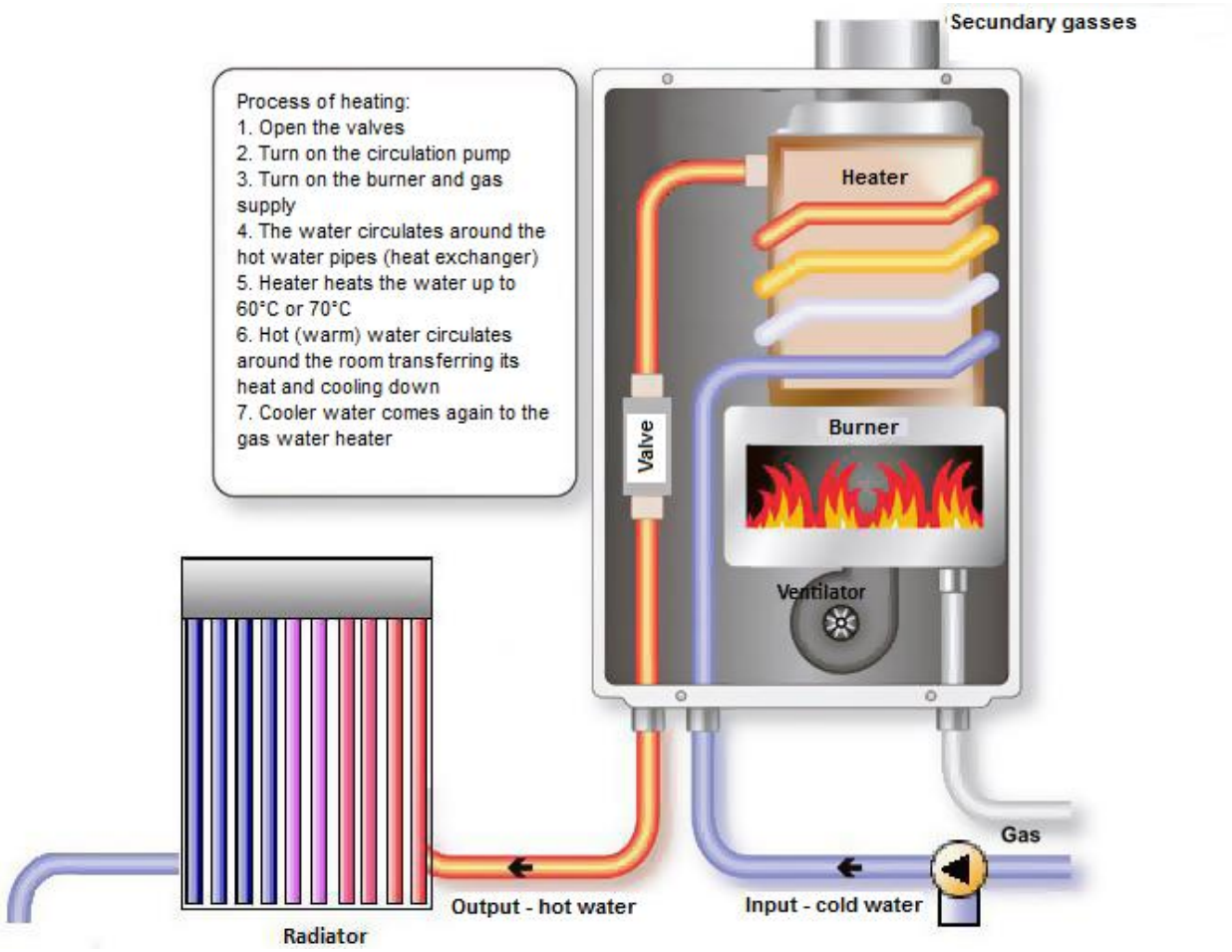


Figure 1.1. Configuration of the gas boiler and the heating process

1.3. Heat transfer

There are three ways of heat transfer: conduction, convection and radiation. The first two will be addressed, because their combination transmits energy from the hot gases to the liquid. Heat $Q[J]$ moves from areas of higher temperature to regions of lower temperature until thermal equilibrium is established.

1.3.1. Conducton

Let's introduce two physical variables:

The heat flux - heat that is transferred per unit time through a surface S.

$$\Phi[W] = \iint_S \vec{q} \cdot \overrightarrow{dS} = \frac{dQ}{dt} \quad (1.3.)$$

$\overrightarrow{dS}$ is a vector perpendicular to the elementary part of the surface, with an intensity equal to the area of the elementary part of the surface.

$$\overrightarrow{dS} = \overrightarrow{n_0} \cdot dS \quad (1.4.)$$

The density of heat flux - The specific heat flux $\vec{q}\left[\frac{W}{m^2}\right]$, ie. the average flux through a surface. According to the definition:

$$d\Phi = \vec{q} \cdot \overrightarrow{dS} \quad (1.5.)$$

According to the Fourier's law, it is known that:

$$\vec{q} = -\lambda \overrightarrow{grad}(T) \quad (1.6.)$$

The vector of density of heat flux is collinear with the temperature gradient. So, the gradient $\overrightarrow{grad}(T)$ defines the direction of the heat transfer. The coefficient of proportionality is a scalar λ. A negative sign indicates that heat diffuses in the direction of decreasing temperature. The overall heat flux is a scalar value, in contrast to the density of heat flux.

From (1.5.) and (1.6.) follows:

$$d\Phi = \vec{q} \cdot \overrightarrow{dS} = -\lambda \cdot \overrightarrow{grad}(T) \cdot \overrightarrow{dS} \quad (1.7.)$$

The elementary flux $d\Phi$ from (1.7.) is the amount of heat that is transferred through an elementary part dS of the surface S per unit time. From (1.3.):

$$dQ = \Phi \cdot dt = \left(\iint_S \vec{q} \cdot \overrightarrow{dS} \right) \cdot dt \tag{1.8.}$$

To find the total amount of heat transferred after time t through the whole surface S, it is necessary to integrate the equation:

$$Q = \int_0^t \Phi(t) \cdot dt = \int_0^t \left(\iint_S \vec{q} \cdot \overrightarrow{dS} \right) \cdot dt \tag{1.9.}$$

Applying (1.6.) gives:

$$Q = \int_0^t \Phi(t) \cdot dt = \int_0^t \left(\iint_S -\lambda \overrightarrow{grad}(T) \cdot \overrightarrow{dS} \right) \cdot dt \tag{1.10.}$$

In order to simplify (1.10) let's introduce presumptions:

Presumption 1: Overall flux through a surface S is constant over time t.

$$Q = \Phi \cdot t \tag{1.11.}$$

Presumption 2: The vector of density of heat flux is the same for every part of the surface S.

$$\Phi = \vec{q} \cdot \vec{S} \tag{1.12.}$$

These assumptions imply that the vector of density of heat flux is time invariant on each part of S during the time t. This conclusion can not be made taking into account only the first presumption, because it's about the time constancy of the total flux through the surface S, not about the time constancy of flux through parts of the surface.

In (1.12.) vector $\vec{S}$ is given as the sum of all N elementary vectors $\overrightarrow{\Delta S}_n$:

$$\vec{S} = \sum_{n=1}^{N} \overrightarrow{\Delta S_n} \tag{1.13.}$$

$\overrightarrow{\Delta S_n}$ vectors are perpendicular to the corresponding „flat“[1] parts of the surface S, with an intensity equal to the area of the corresponding "flat" part. As surfaces are usually not broken into "flat" segments, it is necessary to go with $N \to +\infty$ and choose infinitely small segments $\overrightarrow{\Delta S}$ which meet the requirement (1.14.):

$$\sum_{n=1}^{M} \Delta S_n < \varepsilon, \quad \forall \varepsilon \in \mathrm{R} \quad \wedge \quad \forall \mathrm{M} \in \mathrm{N} \tag{1.14.}$$

Sum of the areas of a final number of parts of the surface S should be less than no matter how small real number ε. From (1.13.) follows (1.15.):

$$\vec{S} = \lim_{N \to \infty} \int_{n=1}^{N} \overrightarrow{dS_n} \tag{1.15.}$$

Presumption 3: Surface through which heat is transferred is flat.

All points on the surface lie on the same plane. It is relatively easy to find the numerical value of the area and the direction of the vector perpendicular to the plane.

The expression for calculating the gradient is given by (1.16.):

$$\overrightarrow{grad}(T) = \left(\frac{\partial T(x,y,z)}{\partial x}, \frac{\partial T(x,y,z)}{\partial y}, \frac{\partial T(x,y,z)}{\partial z} \right) \tag{1.16.}$$

If the temperature T of the equation (1.6.) changes in all three coordinate directions, then the components of the vector of density of heat flux are given by (1.17.):

$$q_x = -\lambda \frac{\partial T(x,y,z)}{\partial x} \quad q_y = -\lambda \frac{\partial T(x,y,z)}{\partial y} \quad q_z = -\lambda \frac{\partial T(x,y,z)}{\partial z} \tag{1.17.}$$

Presumption 4: Heat is transmitted only in the direction of z axis.

[1] “flat” parts of surface S where all points lie on the same plane

Now the vector of density of heat flux is always in the direction of the z axis, ie:

$$\vec{q} = \left(0,0,-\lambda\frac{\partial T(x,y,z)}{\partial z}\right) = \left(0,0,-\lambda\frac{dT(x,y,z)}{dz}\right) = \left(0,0,q_z\right) \quad (1.18.)$$

Due to the Presumption 2 the vector of density of heat flux and the vector perpendicular to the surface S are collinear and from (1.12.) follows:

$$\Phi = \vec{q}\cdot\vec{S} = q\cdot S \quad (1.19.)$$

Here is also assumed that the direction of $\vec{S}$ coincides with the direction of $\vec{q}$.

Presumption 5: Let the surface S be homogenized so that the temperature changes in the direction of the z axis according to a law which is a constant on the surface S. Now the temperature does not depend on (x, y) coordinates so we can write:

$$\vec{q} = \left(0,0,-\lambda\frac{\partial T(z)}{\partial z}\right) = \left(0,0,-\lambda\frac{dT(z)}{dz}\right) = \left(0,0,q_z\right) \quad (1.20.)$$

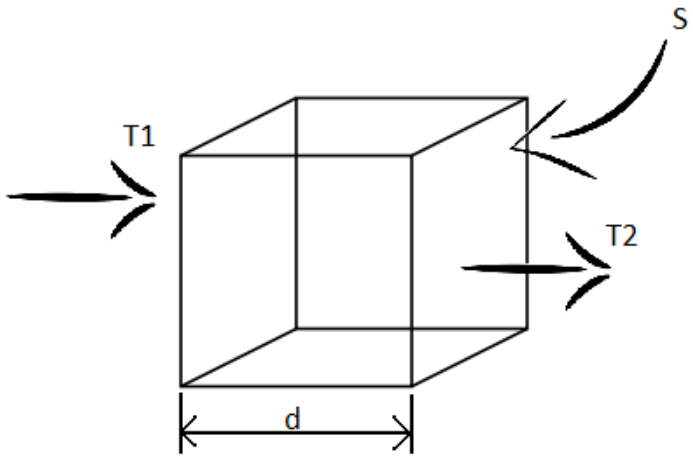


Figure 1.2. The body of a homogeneous material through which conduction is performed

The coefficient of proportionality λ is not completely constant for a given material. It's a coefficient of thermal conductivity and it depends on:

- bulk density of the material

- chemical composition
- moisture content
- temperature

Strictly, it depends on time and position (x,y,z). Conduction happens as the energy transmitts from particle to particle. Let the body be as in Figure 1.2.

Presumption 6: Let the coefficient of thermal conductivity λ be constant for the body.

Taking into account all of the presumptions listed above, and knowing that the temperature of the body changes only with the z axis (horizontal axis in Figure 1.2.):

$$q = q_z = -\lambda \frac{dT(z)}{dz} \tag{1.21.}$$

where λ is a constant.

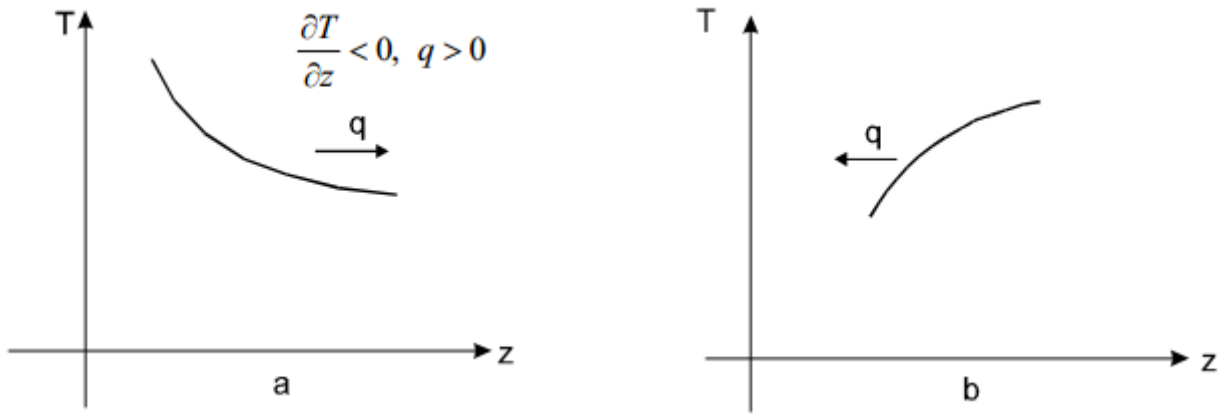


Figure 1.3. The direction of the vector of density of heat flux

Now these equations are valid:

$$Q = \Phi \cdot t = q \cdot S \cdot t = \lambda \cdot \frac{dT(z)}{dz} \cdot S \cdot t \tag{1.22.}$$

In the last expression minus sign is omitted. It only tells about the direction of heat transfer, and since only scalar values are observed now, the direction will be ignored. The heat is transferred only in the direction of the biggest temperature drop.

Presumption 7: Let the heat conduction through the plane of a wall of thickness d be stationary. Let the area of surface S be at temperature T_1, and another side of the same size at T_2.

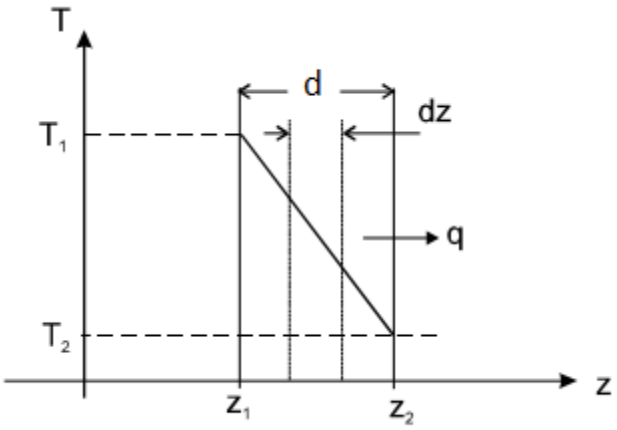


Figure 1.4. Stationary heat conduction through the wall

The difference in relation to the Figure 1.3. stems from the fact that in the case of stationary heat conduction derivative of density of heat flux remains constant along the axis z.

$$Q = \Phi \cdot t = q \cdot S \cdot t = \lambda \cdot S \cdot \frac{T_1 - T_2}{d} \cdot t \qquad (1.23.)$$

Thermal conductivity λ can be defined as the amount of heat [J] per time unit which passes through the cross-sectional area of 1 m^2 and of thickness of 1 m perpendicular to the surface at the temperature of 1 K . This assumes a linear temperature change along the material (1.24.).

$$\lambda = \frac{Q}{A \cdot (T_1 - T_2)} \cdot \frac{d}{t} \quad \left[\frac{W}{m \cdot K} \right] \tag{1.24.}$$

The coefficient of thermal transmittance Λ is equal to the ratio of the coefficient of thermal conductivity of material and its thickness d:

$$\Lambda = \frac{\lambda}{d} \quad \left[\frac{W}{m^2 K} \right] \tag{1.25.}$$

The coefficient of thermal transmittance is equal to the amount of heat per unit time which passes vertically through unit area of the construction element of thickness d at the unit temperature difference between the surface boundaries of the element once it reached steady state. Resistance to leakage of heat R is reciprocal to the coefficient of thermal transmittance:

$$R = \frac{1}{\Lambda} \quad \left[\frac{m^2 K}{W} \right] \tag{1.26.}$$

The greater the thermal resistance, the better insulator.

1.3.2. Convection

Convection is the transfer of heat from one fluid (gas or liquid) to a solid body, or vice versa. Convection can be classified according to the cause of movement. Therefore, there are:

- Natural convection (fluid particles move due to the difference in density caused due to uneven temperature).
- Forced convection (movement of fluid is supported by some mechanical device).

Heat transfer by convection is calculated using Newton's law, where the density of the heat flux is represented by the formula:

$$q = h_c \cdot (T_p - T_f) \quad \left[\frac{W}{m^2}\right] \tag{1.27.}$$

q - density of heat flow, $\left[W / m^2\right]$

h_c - convection coefficient, $\left[W / \left(m^2 \cdot K\right)\right]$

T_p - temperature of solid surface, [K]

T_f - fluid temperature, [K]

1.3.3. Heat exchanger

As noted above, the exchange of heat between two fluids (gas or liquid) happens via a heat exchanger. This includes convecttion from the first fluid to the barrier, conduction through the barrier and convection to the second fluid. The parameter that characterizes such heat transfer is called the heat transfer coefficient of the exchanger (K):

$$K = \frac{1}{\frac{1}{\alpha_i} + \frac{d}{\lambda} + \frac{1}{\alpha_e}} \quad \left[\frac{W}{m^2 \cdot K}\right] \tag{1.28.}$$

α_i - internal heat transfer coefficient. It is equal to the amount of heat per unit time which moves from hot gases in the combustion chamber to a heat exchanger's unit area at unit temperature difference of air and surface of the exchanger.

α_e - the coefficient of the external heat transfer. It is equal to the amount of heat per unit time which passes from the unit surface of the exchanger to the fluid at unit temperature difference between the exchanger's surface and fluid.

The coefficient K is equal to the amount of heat which passes vertically per unit time through unit area of the exchanger at unit temperature difference between fluids from both sides of the element. The heat transfer coefficient can be expressed using heat resistance:

$$K = \frac{1}{R_i + R + R_e} \tag{1.29.}$$

$R_i = \frac{1}{\alpha_i}$ - the resistance of the internal heat transfer

$R_e = \frac{1}{\alpha_e}$ - the resistance of the external heat transfer

$R = \frac{1}{\Lambda}$ - resistance to leakage of heat

Heat transfer resistance $\left(R_0\right)$ is the reciprocal value of the heat transfer coefficient of the exchanger:

$$R_0 = \frac{1}{K} = R_i + R + R_e \tag{1.30.}$$

The value of the coefficient of heat transfer can be used to assess the loss of heat from a room. For a barrier of thickness d made of many different materials in layers $d_1, \mathrm{d}_2, ..., \mathrm{d}_n$, and for different temperatures of end surfaces (T_1 i T_n), at a certain point of time a state of linear temperature change will be established at the intersections of individual layers of the barrier.

$$q = \frac{Q}{S \cdot t} = \frac{\lambda \cdot \left(T_1 - T_n\right)}{d} \tag{1.31.}$$

As the density of heat flow is equal for layers and the entire barrier (due to the assumption that the heat is transmitted only in the direction of the horizontal axis), follows (1.32.):

$$\frac{\lambda_1}{d_1}(T_1 - T_2) = \frac{\lambda_2}{d_2}(T_2 - T_3) = \ldots = \frac{\lambda_{n-1}}{d_{n-1}}(T_{n-1} - T_n) \tag{1.32.}$$

From this equation follows (1.33.):

$$\begin{aligned} T_1 - T_2 &= \frac{d_1}{\lambda_1} \cdot q \\ T_2 - T_3 &= \frac{d_2}{\lambda_2} \cdot q \\ &\vdots \\ T_{n-1} - T_n &= \frac{d_{n-1}}{\lambda_{n-1}} \cdot q \end{aligned} \tag{1.33.}$$

Summing up the above equations, the expression for the density of heat flux is given by:

$$q = \frac{T_1 - T_n}{\sum_{i=1}^{n-1} \frac{d_i}{\lambda_i}} \tag{1.34.}$$

1.4. Control of the gas water heater

This paper is about a control module that controls only the burner and circulation pump. Therefore, assumptions must be determined which the system must meet before installation of the specified module. The assumptions are listed below:

1) There is an already implemented system that openes all valves automatically once the pump is turned on to ensure the smooth circulation of water (if such valves exist).
2) When the circulation pump is activated (in the state ON), pre-implemented system must enable occasional discharge and replacement of water.
3) When the burner is turned on, the appropriate valves for the gas supply are automatically opened and the combustion process is initiated.

4) The removal of secondary gases is regulated in an appropriate manner.
5) The burner must have internal control which prevents overheating of the water exiting the boiler. The temperature of the outgoing fluid typically does not exceed 60°C, although there may be systems where it goes up to 70°C. No matter for how long the burner is ON, the outgoing fluid temperature must not be close to the boiling point.

1.4.1. Model of control

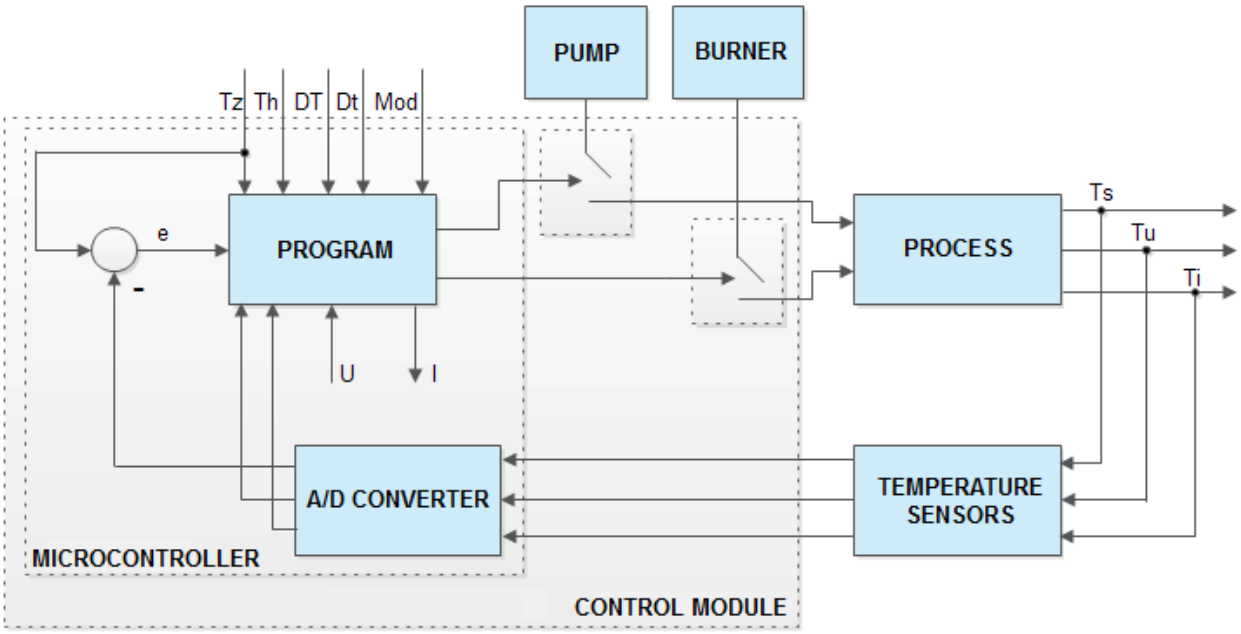


Figure 1.5. Structural block diagram of the control system

Once connected to the appropriate boiler system, the control module provides the following functionality:

- Turning ON the module and system
- Maintaining the room temperature Ts
- Displaying of temperatures Tu, Ti, Ts, Tz, hysteresis Th, status of pump and burner curcuit, mod of the module (auto/manual)

- Setting temperatures (and other parameters)
- (TCP/IP communication over a LAN, with the possibility of insight into the values that appear on the display and setting parameters the same way like when using the keyboard)

As seen in the previous picture, the composition of the module has a microcontroller with accompanying shield boards (ethernet shield, shield to connect with other components), relays for control of the pump and burner, display and a keyboard. "U" indicates that the signal represents a command sent by the user, while "I" denotes a signal sent from the module to the user (preview of all available options, preview of values of variables and parameters, messages and responses to commands sent via the Telnet protocol).

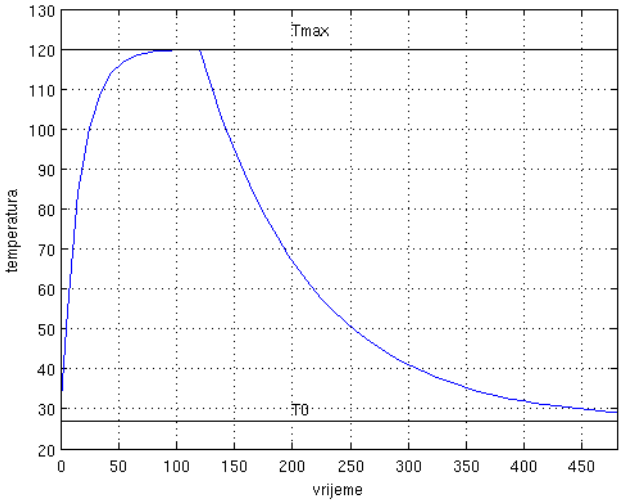
Temperature usually belongs to the class of processes of slow dynamics. It allows the sample period to be large enough so that the processor can perform all of its functions in a relatively short period of time (immediately after the sampling), within the period T. Clock speed of Arduino Uno microcontroller (16 MHz) allows the entire loop of the program to be carried out in a few milliseconds. Time constant of the process is several tens of minutes, so the condition of sampling theorem is certainly fulfilled. There was no need for precise determination of the sampling time. Moreover, in the program there was no need to perform synchronization, so the sampling period is a bit variable (it never exceeds 10 milliseconds). Use of delay in order to fixate the time of loop execution (or for any other purpose) would lower the response speed of the module.

1.4.2. Two-point regulation

The algorithm for temperature control is two-position regulation. The term „two-position controller“ is derived from the position of the output (either "on" or "off"). This system regularly

includes hysteresis. The dynamic behavior of the process of heating and cooling will be approximated with the differential equation of the first order:

$$G_{ob} = \frac{K_{ob}}{Ts+1} \cdot e^{-\tau}$$



a) transfer function *b) system response*

Figure 1.6. Thermodynamic system as a first order system

Time constants of heating and cooling are generally different. The parameters specified in the transfer function are:

K_{ob} - gain coefficient;

T - inertial time constant of the process;

τ - transport delay;

The transport delay is too small compared to the inertial time constant of the process. Therefore, the dynamic behavior of the temperature can be represented by the following differential equation:

$$T \cdot \frac{dx}{dt} + x = K_{kr} \cdot 1(t) \qquad (1.36.)$$

x - output variable (temperature) versus time, response to the excitation signal of step function

T - inertial time constant of the process

K_{kr} - the magnitude of the final value of the output variable (steady state)

$K_{poč}$ - the initial value of the output variable, tj. $x(0) = K_{poč}$

$1(t)$ - Heaviside unit step function

If we look at the system in the time interval: $t \in (0, \infty)$, then the equation is:

$$T \cdot \frac{dx}{dt} + x = K_{Kr} \tag{1.37.}$$

Particular solution is: $x_{par} = K_{kr}$. A homogeneous solution can be found by solving the following equation:

$$T \cdot \frac{dx}{dt} + x = 0 \Rightarrow \frac{dx}{dt} = -\frac{dt}{T} \Rightarrow \mathrm{x_{hom}} = C \cdot e^{-\frac{t}{T}} \tag{1.38.}$$

The general solution is:

$$x(t) = K_{kr} + C \cdot e^{-\frac{t}{T}} \tag{1.39.}$$

Let's take into account the initial condition:

$$x(0) = K_{poč} = K_{kr} + C \Rightarrow \mathrm{C = K_{poč} - K_{kr}} \tag{1.40.}$$

$$x(t) = K_{kr} + (K_{poč} - K_{kr}) \cdot e^{-\frac{t}{T}} = K_{kr} - \left(K_{kr} - K_{poč}\right) \cdot e^{-\frac{t}{T}} \tag{1.41.}$$

The derived equation describes the behavior of a thermodynamic system, and will be used to describe the behavior of a two-position controller. When this control system is applied to the first order system, output characteristics are given by:

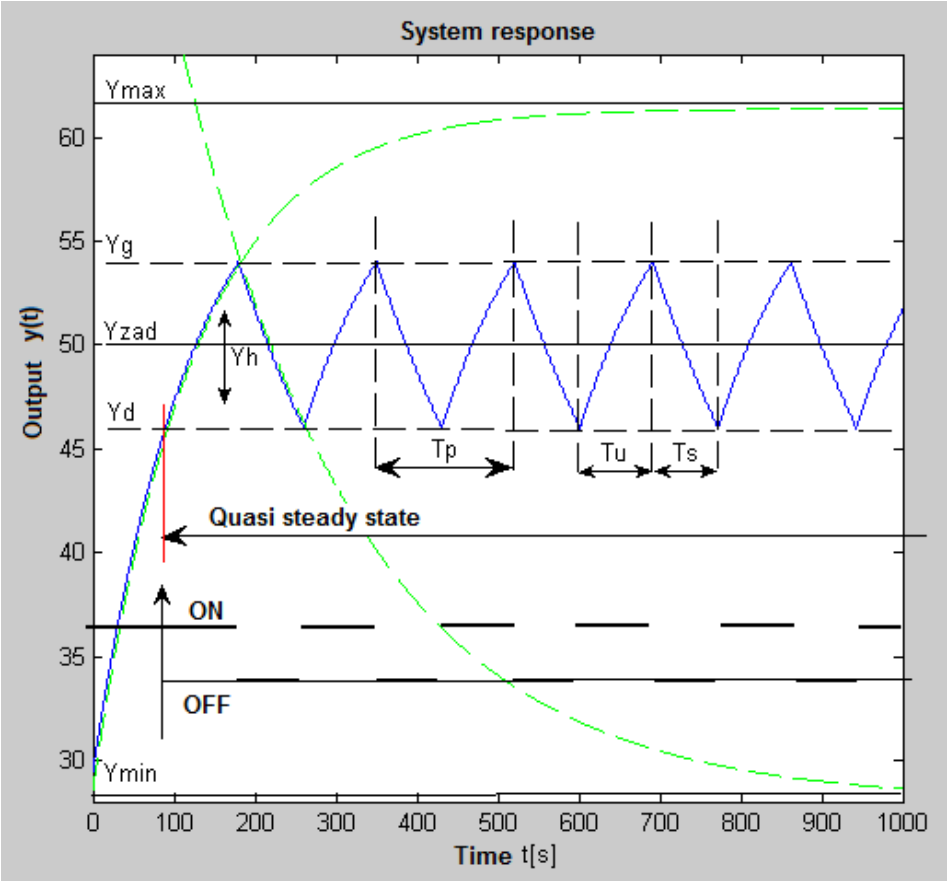


Figure 1.7. Response of the system regulated by two-position controller

$Y_{\max}$ - the maximum value that the output variable would achieve with heater always ON

$Y_{\min}$ - the minimum value that the output variable would achieve with heater always OFF

Y_{zad} - setpoint value of output temperature

Y_g - upper threshold

Y_d - lower threshold

Y_h - hysteresis

The value of output during the quasi-steady state has a value in the range:

$(Y_{zad} - Y_h,\ Y_{zad} + Y_h)$

T_p - quasi oscillation period

T_u - duration of pulse signal „ON“

T_s - duration of pulse signal „OFF“

1.4.2.1. Analytical expressions

$$Y_g = Y_{zad} + Y_h \tag{1.42.}$$

$$Y_d = Y_{zad} - Y_h \tag{1.43.}$$

$$T_p = T_u + T_s \tag{1.44.}$$

If the time constants of ascending and descending functions are equal, then: $T_u = T_s$. Let the quasi oscillatory state begin at the moment t=0. While the burner is in the state ON, the following equation describes the system:

$$Y(t) = Y_{\max} - (Y_{\max} - Y_d) \cdot e^{-\frac{t}{Tuzli}} \tag{1.45.}$$

The time required to reach Y_g is T_u:

$$\begin{aligned} Y(T_u) = Y_g &= Y_{\max} - (Y_{\max} - Y_d) \cdot e^{-\frac{T_u}{T_{uzli}}} \\ \Rightarrow T_u &= T_{uzli} \cdot \ln \frac{Y_{\max} - Y_d}{Y_{\max} - Y_g} \end{aligned} \tag{1.46.}$$

If the time constants are equal:

$$T_u = T_s = \frac{T}{2} \tag{1.47.}$$

The equation for T_s can be derived from (1.46.), if we:

T_{uzli} replace with T_{sili}

$Y_{\max}$ replace with $Y_{\min}$,

Y_d replace with new initial condition $Y(0) = Y_g$,

Y_g replace with Y_d, because that is the final value to which the output falls.

$$T_s = T_{sili} \cdot \ln \frac{Y_{\min} - Y_g}{Y_{\min} - Y_d} \tag{1.48.}$$

Taking into account the equations (1.42.) and (1.43.): $Y_g = Y_{zad} + Y_h$; $Y_d = Y_{zad} - Y_h$; we get final equations in a way where intervals of heating and cooling depend on Yh and Yzad:

$$T_u = T_{uzli} \cdot \ln \frac{Y_{\max} - Y_{zad} + Y_h}{Y_{\max} - Y_{zad} - Y_h} \tag{1.49.}$$

$$T_s = T_{sili} \cdot \ln \frac{Y_{\min} - Y_{zad} - Y_h}{Y_{\min} - Y_{zad} + Y_h} \tag{1.50.}$$

$$T = T_u + T_s \tag{1.51.}$$

Lowering hysteresis Y_h would reduce the amplitude of oscillations and a higher accuracy would be achieved. However, it would lead to an increase in the switching frequency which reduces the life of the relay (switch/actuator). It is necessary to find a compromise between the lifetime of actuator and accuracy of control. The user can select the value of hysteresis from the range $T_h \in [0.5°C,\ 7°\mathrm{C}]$.

1.4.2.2. Simulation of two-position control

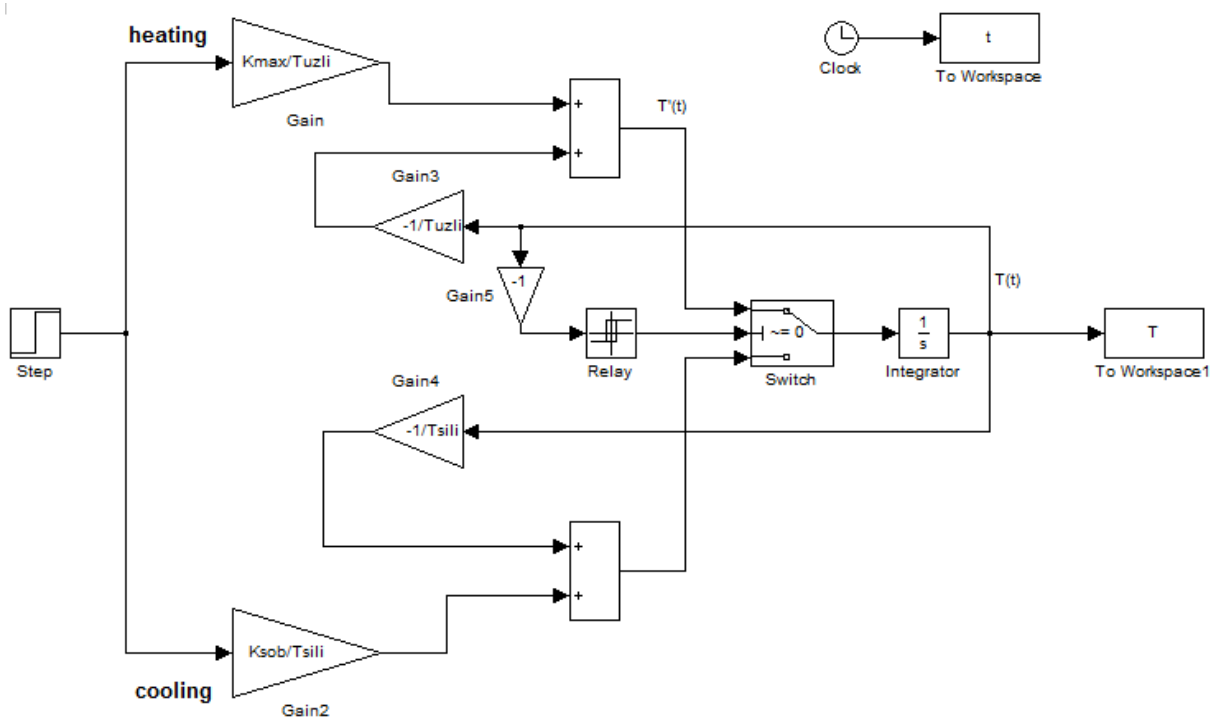


Figure 1.8. Simulink model of automatic regulation of a thermodynamic system

The resulting data can be demonstrated in a simulation of the control system in the Simulink software package. obtained responses are as in Figures 1.9. and 1.10.

The values of the parameters for the simulation are selected only to demonstrate the two-position control. The real module is configured so that the desired temperature can not be set outside of a predefined range $T_{zad} \in [8°C,\ 35°C]$.

Table 1.1. The parameters for the simulation of two-position control (Example 1)

T_h	T_{zad}	$T_{poč}$
4°C	50°C	28°C

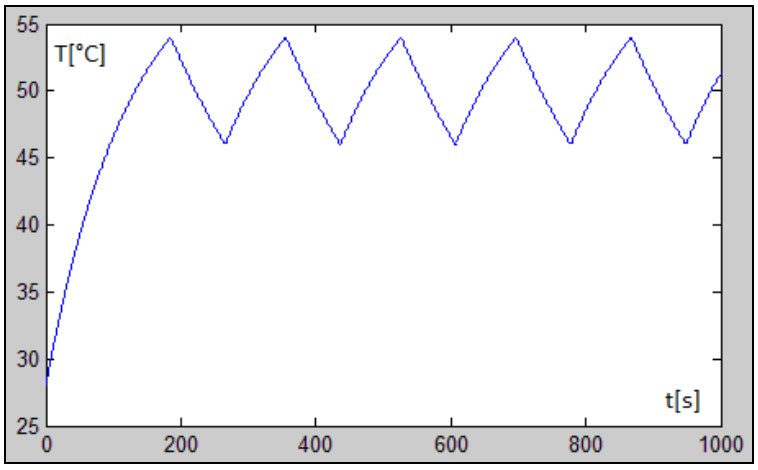


Figure 1.9. The response of the simulated system (Example 1)

Table 1.2. The parameters for the simulation of two-position control (Example 2)

T_h	T_{zad}	$T_{poč}$
4°C	50°C	28°C

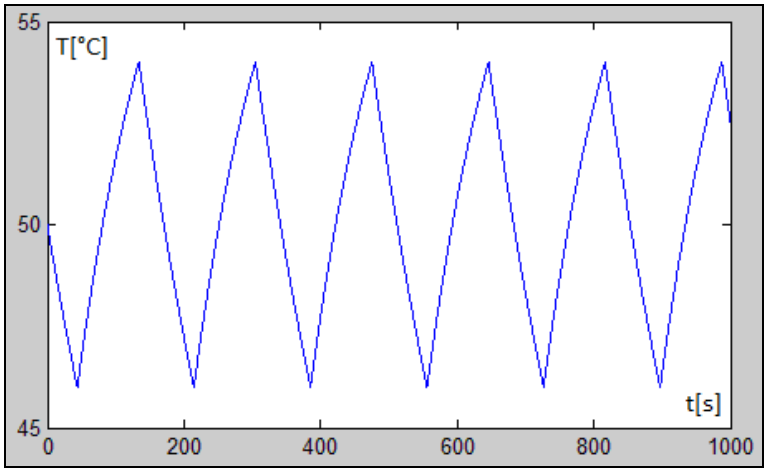


Figure 1.10. The response of the simulated system for (Example 2)

2. The hardware structure of the control module

The microcontroller is the central component of the module. It also provides interaction with other components which are necessary to complete the functionality of the system. Electrical scheme was created in a form in which the shield was actually realized (Appendix 2).

2.1. List of used components

Table 2.1. List of used components

Num	Element	Type	Label	Quantity
1.	Microcontroller	Arduino Uno	Based on ATmega328	1
2.	Adapter	AC/DC (9V)		1
3.	Two-point switch	Mechanical handle		1
4.	Membrane keypad	4x4		1
5.	LCD display	48x84 monochromatic	Nokia 3110	1
6.	Ttemperature sensor	Linear	LM35DZ	3
7.	Relay module	With 2 relays	Ywrobot 2 relay	1
8.	Three-state circuit	DIL	SN74LS244N	1
9.	Ethernet shield	Arduino	Based on Wiznet W5100	1
10.	LED diode			2
11.	Resistor		410 Ω	2
12.	Resistor		1.5 $k\Omega$	6
13.	Resistor		2.1 $k\Omega$	5
14.	Resistor		6.8 $k\Omega$	5
15.	Connectors			

2.2. Arduino Uno microcontroller

Figure 2.1. Arduino Uno microcontroller

Table 2.2. Overview of main features

Microcontroller	ATmega328
Operating voltage	5V
Vcc voltage (recommended)	7-12V
Vcc limits	6-20V
Digital I/O pins	14 (6 support PWM output)

Analog input pins	6
DC current per I/O pin	40 mA
DC current limit for 3.3V pin	50 mA
Flesh memory	32 KB (ATmega328) 0.5 KB for bootloader
SRAM	2 KB (ATmega328)
EEPROM	1 KB (ATmega328)
Clock speed	16 Mhz

2.3. Shield for connection with other components

In order to successfully connect display, keyboard, relays, sensors and LED diodes with the microcontroller, a shield that can easily connect with the Arduino has been developed (Appendix 2). Basic characteristics related to individual components will be presented here, with some notes about the wiring.

2.3.1. Nokia display 3110

It is a monochrome display as shown in Figure 2.2.

Figure 2.2. Nokia display 3110

Specifications:

- 48x84 pixels
- Interface with a serial bus with maximum speed of 4 Mbit/s
- Internal controller PCD8544

- LED illumination
- Working at voltages 2.7 – 5.0 V (recommended 3.3V)
- Low consumption, suitable for battery-powered devices
- The permitted temperature range: -25°C do 70°C

2.3.1.1. Addressing LCD display

The addresses in the memory (DDRAM) are arranged in a matrix of 6 rows (Y-address 0 to 5) and 84 columns (X address 0 to 83). Appropriate combinations address data bytes. The data sent to the display are 8-bit (1 byte), and are arranged to be vertical. In this case, the most significant bit (MSB) is at the bottom (Figure 2.3).

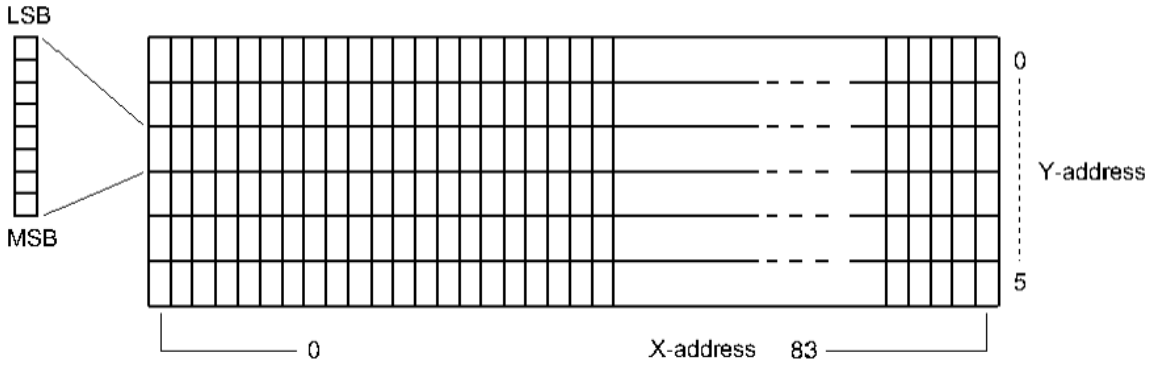


Figure 2.3. The structure of addresses and data of the LCD display

Data can be entered at the address in the memory (DDRAM) sequentially, and the value of X-address and Y-address will automatically increase. There are two methods:

- Vertical addressing mode (V=1). After each written byte, Y address increases by one.
- Horizontal addressing mode (V=0). After each written byte, X address increases by one.

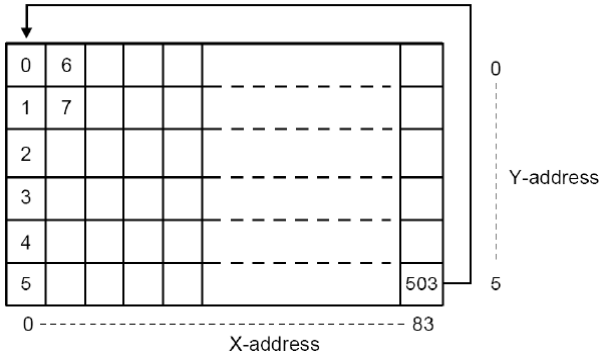


Figure 2.4. Vertical addressing mode

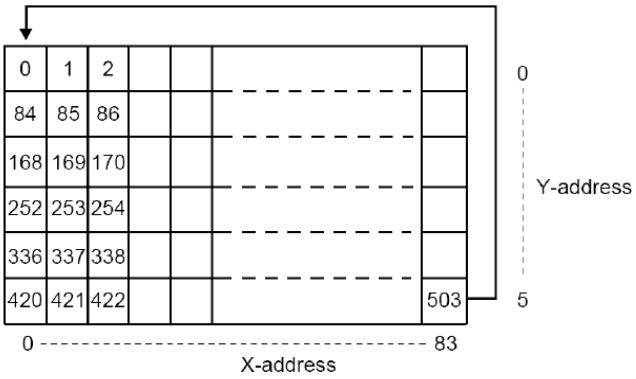


Figure 2.5. Horizontal addressing mode

2.3.1.2. The pins to control LCD display

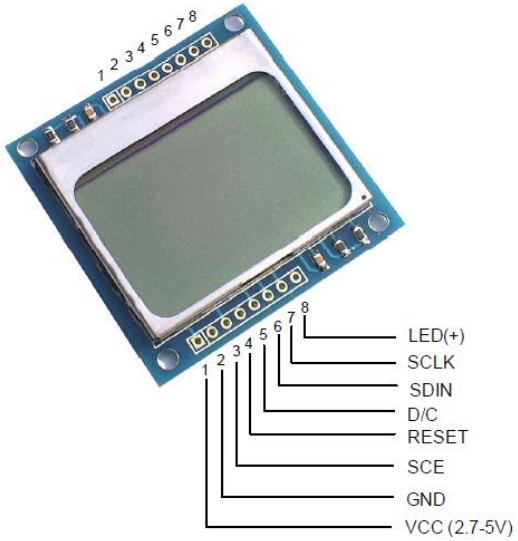


Figure 2.6. Pins of LCD display

Table 2.3. Pins of LCD display

Pin name	Function
1. VCC	Voltage supply (2.7 – 5V), 3.3V recommended
2. GND	Ground
3. SCE	Enable transfer
4. RESET	Reset signal for current operation of the LCD
5. D/C	Pin to set the type of signal (data or command)
6. SDIN	The line for serial data transmission
7. SCLK	Clock
8. LED	Pin to control the brightness of the display

2.3.1.3. Communication modes

Format of the command used to communicate with the LCD is divided into: the command mode and data mode. Pin D/C is used for the differentiation of these two types. If the D/C = 0 then the data that is sent to the LCD is a command, otherwise it is data that will be entered into DDRAM memory and displayed. After entering bytes in DDRAM memory address value will automatically increase. Format of data is serial and the first bit sent is the MSB:

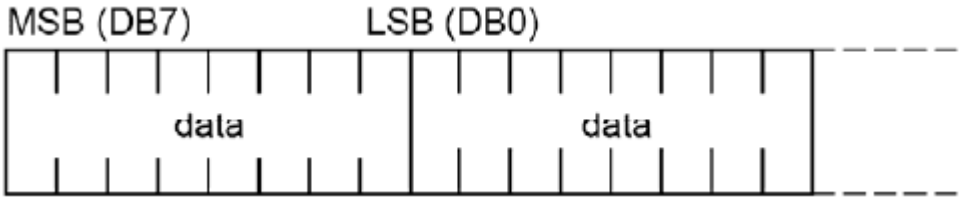


Figure 2.7. Serially distributed data sent to the display

There are two methods of sending data to LCD:

- Sending of one byte at a time *(Figure 2.8.) and*
- Sending of a serie of consecutive bytes *(Figure 2.9.)*

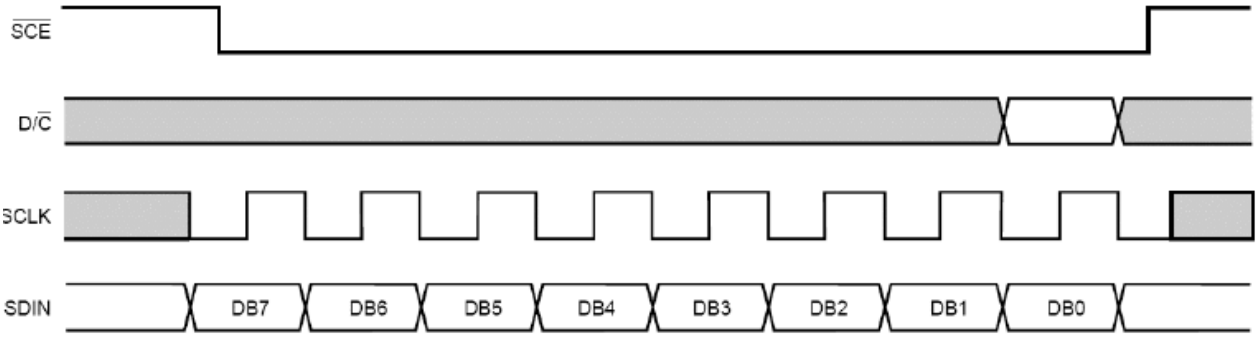


Figure 2.8. Sending of one byte at a time

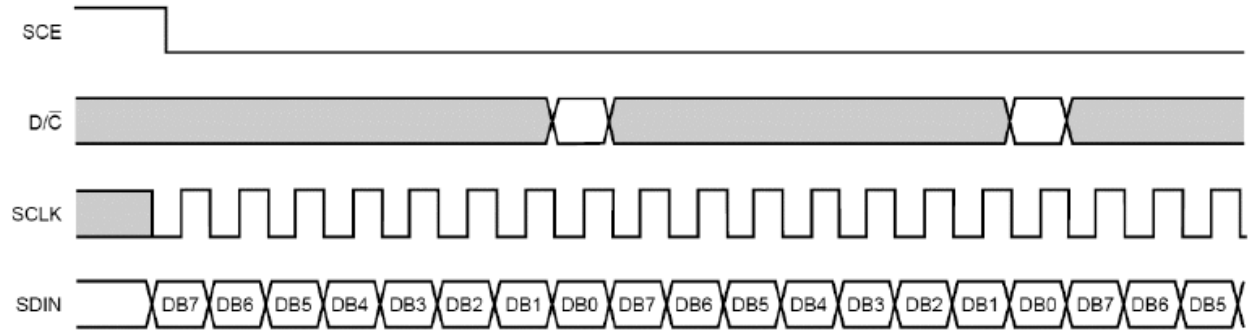


Figure 2.9. Sending of a serie of consecutive bytes

If the pin SCE has a value of logical one, any change of SCLK signal will have no effect on the display. The user can send data to the LCD display only when the value of the SCE is set to logical zero. Data is sent via SDIN input and follows time interval of the clock (rising edge). LCD concludes if it is a data for DDRAM or a command according to the logical value on pin D/C. If D/C = 0, then it's a command, otherwise it's data. Pin SCE should retain the value 0 until the data is successfully transferred. More information and a list of commands can be found in the datasheet of the LCD display. As a part of the program developed in the context of this work, finished libraries were used to send commands to the LCD display. The display is connected to the input/output pins of a microcontroller via voltage dividers which lower the output voltage of the microcontroller pins to about 3.3V. Voltage dividers consist of serially linked resistors (2.1k

and 6.8k). If the pin has logical one, the output gives 5 * (1-2.1 / 6.8) = 3.45V which is acceptable voltage for the LCD. Between 4 pins of the microcontroller and the voltage dividers there is a three-state circuit SN74LS244 which will be discussed below.

2.3.2. Membrane keypad 4x4

This keyboard provides the possibility of developing an appropriate interface for the user of the embedded system. These keyboards use a combination of 4 rows and 4 columns of conductors that overlap, but do not touch each other until the button is pressed. The microcontroller detects which button is pressed by detecting the position of the short circuit between a row and a column.

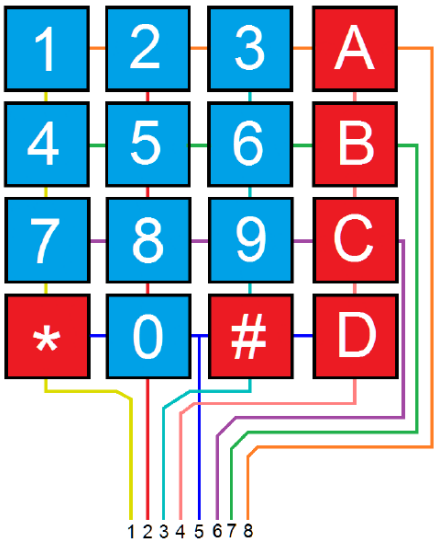


Figure 2.10. Scheme of a matrix membrane keyboard 4x4

It is possible to make a function that returns an array of all pressed buttons. In this work a function that returns a single pressed character has been created (section 3.2.).

2.3.3. SN74LS244 (three-state chip)

This is a DIL chip with 20 pins. There are two lines, each with 4 channels. Each line can be switched off (brought into the third state), so that the logical value of the input has no effect on the output. If the line is not brought into the third state, channels just forward unchanged input signals to the output.

It was necessary to share some pins between components. For the realization of all functionalities of the module there had to be 20 digital and 3 analog pins (analog pins are used for getting a temperature from each of the three sensors. That's a total of 23 pins, while the microcontroller has 14 digital and 6 analog pins, which in total gives 20. Analog pins can be used as digital, and lack of 3 pins will be compensated by using the three-state circuit to share 4 pins between display and keyboard. 1 pin is required for control of the chip, so saving is not 4 pins but 3. A component that can be occasionally turned off is the display. Wiring is shown in Appendix 2, which contains a comprehensive electrical schema of the shield.

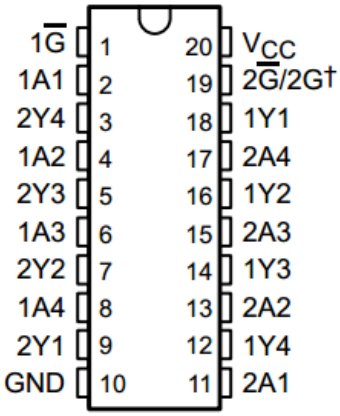


Figure 2.11. SN74LS244

This circuit is powered by a + 5V. Pin 1 is brought to + 5V so that the line 1G is always off. This is done to prevent rapid change of the output state in these channels due to the possible effects of noise. Pin 19 is a command pin which occasionally brings 4 pins connected to the

display into the state of high impedance. On the right side, inputs 2A4, 2A3, 2A2 and 2A1 are connected to the digital pins 5, 6, 7, 8 of Arduino respectively. Pin 9 of the microcontroller has the role of SCE pin for display (it's brought directly to the voltage divider, not through the circuit with three states). While the display is off, the pin SCE is set to the value of the logical one, so that the noise of the output of the three-state circuit can not result in unpredictable behavior of the LCD display.

In addition, it was noted that the display is still blocked as soon as the pins: SCLK, SDIN, D/C, and RESET disconnect from the microcontroller using the three-state circuit. RESET pin should not fall to logical zero while SCE is set to logical one. Therefore the RESET pin is linked through a 1.5k resistor to +5V. It will remain at +5V as long as the display is turned off (resistor has a role of a pull-up resistor). There were no blockades of the LCD display after adding the resistor in the circuit. All other resistors on the electrical scheme of the shield have a role to limit electricity.

Outputs 2Y4, 2Y3, 2Y2 and 2Y1 were brought to voltage dividers, which lead to the pins of the display: 4 (RESET), 5 (D/C), 6 (SDIN) and 7 (SCLK) respectively. These pins are in the state of high impedance during the time of checking the pressed key of the keyboard. After that, they are reactivated. The possible short circuits caused by pressing keys do not result in excessive currents due to 4 resistors of 1.5k which limit the current.

2.3.4. Temperature sensor LM35

The module measures: the temperature of the incoming and outgoing liquid, and the room temperature. LM35DZ provides output voltage linearly dependent on temperature. The

increment of temperature of 1°C causes increment of voltage of 10mV. Given that this is a 10 bit D/A conversion, the microcontroller can distinguish 1024 voltage values in the range of 0 to 5V.

$5 / 1024 \approx 0.0049V \approx 5mV$

This means that precission can not be greater than 0.5°C. However, through the software it is possible to use internal voltage of 1.1V as the upper limit for A/D conversion. This is done, and the scope is reduced. As the scope is reduced, the difference between quantum levels decreased:

$1.1 / 1024 = 0.00107V \approx 1mV$

The accuracy increased significantly and now stands at 1/10°C. That's why the module's UI provides the possibility to set parameters and measure variables with precision of one decimal place after decimal comma. This scope could be further reduced by using the AREF pin.

These sensors do not heat up much when turned on (0.08°C in still air). They have non-linearity of $\pm 1 / 4°C$. The module is not configured to measure temperatures below zero, so all of those values are considered zero. The sensors provide accuracy of $\pm 3 / 4°C$ for the entire range of 0 to 100°C.

2.4. Ethernet shield

Arduino Ethernet shield allows an Arduino microcontroller to connect to a network (LAN, Internet) via Ethernet cable. Its hardware, software and complete documentation are open-source and available on the Internet.

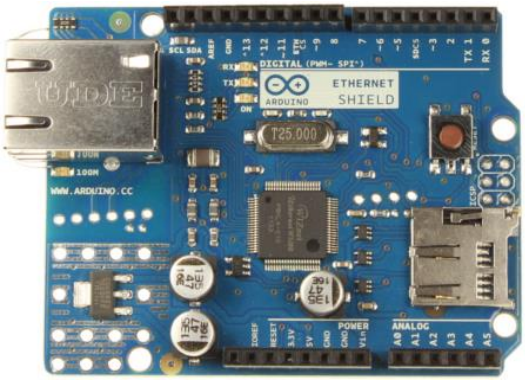


Figure 2.12. Ethernet shield

Specifications and requirements

- It is necessary to have the Arduino development board
- It has an ethernet controller: W5100 with internal 16 KB buffer
- Connection Speed: 10/100 Mb
- Communicates with Arduino using SPI

Description

The shield is based on Wiznet W5100 Ethernet chip. Wiznet W5100 allows networking with support for TCP and UDP. It allows up to 4 simultaneous connections. The shield leaves the pin interface of Arduino intact, so it can be connected to another shield, and it was done. The shield has a place for micro-SD card that can be used to store server files.

The shield has a reset controller that ensures that the W5100 Ethernet module is properly reset at startup. There is a PoE module developed by open-source Internet community which exploits the power of the conventional Ethernet cable, thus reducing the power consumption. PoE module does not come with an Ethernet shield, but can be purchased as a separate component.

Arduino communicates with W5100 and SD card using the SPI via digital pins 11, 12, 13. Pin 10 is used to select communication with the W5100 and pin 4 to select communication with SD. These pins can not be used as standard inputs or outputs. Since the W5100 and SD card share the same SPI channel, it's impossible to communicate simultaneously with both components. If we want to exclude communication with the SD card, we need to declare the output pin 4 and set it to logical one. This is why seemingly nothing is connected to pins 4, 10, 11, 12 and 13 (These pins are used by Ethernet shield).

3. The software for the Arduino UNO

The code has been developed in the Arduino software environment. Parameters can be set by the user in a number of ways. All parameters remain written in the EEPROM. This chapter explains important segments of the program, the way in which the connection is established with individual components and relevant commands.

3.1. Communication with display

For communication with the display there are two included libraries:

```
#include <Adafruit_GFX.h> // Graphics
#include <Adafruit_PCD8544.h> // Basic functions Nokia LCD
```

In order to do appropriate initialization of the display it is necessary to define the pins and certain variables that are necessary for the proper functioning of the library:

```
byte pinDisplayControl=14; // A0 as a digital pin
byte pinDisplaySCE=9;      // Digital pin
// -------- INITIALIZATION OF THE DISPLAY ----------
// Adafruit_PCD8544(SCLK,SDIN,D/C,SCE,RESET);
Adafruit_PCD8544 display =
Adafruit_PCD8544(pinKey[0],pinKey[1],pinKey[2],pinDisplaySCE,pinKey[3]);
```

```
#define XPOS 0
#define YPOS 1 // The starting position of DDRAM
#define DELTAY 2
#define LOGO16_GLCD_HEIGHT 16
#define LOGO16_GLCD_WIDTH  16 // Size of logo
// ---------------------------------------
```

To use commands from the library "Adafruit", the setup of the program must turn the display on through the three-state circuit, initialize the object display, set the desired contrast and display the logo of Adafruit (manufacturer of the library).

```
pinMode(pinDisplayControl,OUTPUT);
digitalWrite(pinDisplayControl,LOW); // Turn on
display.begin();                     // Initialization
display.setContrast(42);
display.display();                   // Logo
```

The display redraws itself every second and on a valid user interaction. Here are some commands that were used in the code:

```
display.clearDisplay();
```

// Clears screen

```
display.drawLine(x1, y1, x2, y2, BLACK);
```

// A line from the point (x1,y1) to the point (x2,y2)

```
display.setTextSize(size);
```

// The size of the text to be printed. If size=1 each character occupies 5x8

// pixels. If size=2, then characters occupy 10x16 pixels.

```
display.setCursor(x,y);
```

// Set the cursor to the default position

```
// Y should be increased by 8 (if TextSize=1) to move to a new line
display.print(F("Text"));
// Prints a string message, automatically moves the cursor in front of the text
display.println(F("Text"));
// After printing a string message goes to a new line and puts the cursor at the beginning of x axis
```

Arduino Uno has only 2 KB of SRAM. By putting F in front of strings, they are stored in the Flash memory (Arduino has considerably more Flash memory, 32 KB).

3.2. Detection of the pressed key on the keyboard

This is, as noted earlier, reduced to detection of the short circuit occurred between a row and a column of the matrix keyboard. One of the most common ways in which this can be done is the following:

- Include pull-up resistors to limit the current through pins that are connected to rows of the keyboard. They will be inputs with voltage 5V, unless a short circuit brings logic 0 (through resistors with much lower resistance than the resistance of the pull-up resistors).
- Set all the columns (1 to 4 on the keypad) to logic 0 (at 0V)
- Check if the voltage of one of the rows is brought down to 0V. If so, then it is known that at least one key is pressed from the line.

If we want to find out which button is pressed, it is necessary to carry out the same procedure for the column, and detect the column of the pressed button. The initialization related to the keyboard has the following commands:

```
// ----- Initialization-----
char matrix[4][4] = {
  {'1','2','3','A'},
```

```
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
char customKey=0;          // Pressed key, not released yet
char currKey=0;            // The auxiliary variable
boolean readyForKey=true; // While true, keystroke is to be accepted
// -------------------------------------
```

The role of these variables is best seen in the following code snippet:

```
  if (!readyForKey) { // Pressed, not released yet
    currKey=getKey();
    if (!currKey || currKey!=customKey) readyForKey=true;
    // Released, or a new one is pressed,
    // the program is ready to execute a new key command.
  }
```

readyForKey serves as an indicator that the program is ready to take a new command. Otherwise a command would be accepted a large number of times instead of just once. This is especially important when entering parameters via prompt. The function that performs the task of detecting which button is pressed is defined by the following code:

```
char getKey() { // Get pressed key
  byte red=10, kolona=10;
  Timer1.stop();
  for (byte i=0; i<=3; i++) {
    pinMode(pinKey[3-i],INPUT_PULLUP); // All rows to +5V (enable pull-up
                                       // resistors)
    pinMode(pinKey[7-i],OUTPUT);
    digitalWrite(pinKey[7-i],LOW); // Columns to LOW
```

```
  }
  for (byte i=0; i<=3; i++) {
    if (digitalRead(pinKey[3-i])==LOW) {
    // That's the one
      red=4-i; break;
    }
  }
  Timer1.resume(); // Continue the timer, as it will do all
                   // necessary things to avoid wrong key detection
                   // if an interrupt happenes below
  for (byte i=0; i<=3; i++) {
    pinMode(pinKey[7-i],INPUT_PULLUP); // Columns to +5V
    pinMode(pinKey[3-i],OUTPUT);
    digitalWrite(pinKey[3-i],LOW);     // Rows to LOW
  }
  for (byte i=0; i<=3; i++) {
    if (digitalRead(pinKey[7-i])==LOW) {
    // That's the one
      kolona=4-i; break;
    }
  }
  red--; kolona--;
  if (red>4 || kolona>4) return 0; // If still 10 something is wrong
  else return matrix[red][kolona];
}
```

It is important to notice that time interrupts must be temporarily stopped at one part of the function. During the time of the interrupt execution the screen is being renderred, but keyboard

and display share 4 pins. Use of the display may not only disturb the logic value of the pins, but also their mode of operation (which can be either an output or an input). It would lead to incorrect key reading and should not be allowed.

On the other hand, if the termination occurs after the command Timer1.resume(); ie. use of the display will not lead to errors in reading as a function that performs rendering calls a function which safely turns the display off through the three-state circuit and returns the value of shared pins to LOW, so that the function getKey() can continue its execution safely.

The time interrupt can occur at any part of the program unless it has previously been disabled. It can be seen that there are parts of the program where the execution of an interrupt is unacceptable.

3.3. EEPROM memorija

Arduino Uno has 1 KB of EEPROM memory (Electrically Erasable Programmable Read Only Memory). This memory allows the permanent storage of data as opposed to RAM. It would be possible to use the Flash memory, but in that case a single byte could not be addressed individually. Given these characteristics, EEPROM memory is suitable for storing 5 parameters that the user can specify either over a LAN connection or through the menu and keyboard of the module.

Used libraries:

```
#include <EEPROM.h> // The standard EEPROM library
#include <EEPROM_anything.h> // Generic functions, read/write any
                             // data type from/to EEPROM
```

The second of these libraries was created for the purpose of this paper. It consists of only three generic functions, and its content is provided in Appendix 1. The functions of the generic type can be used within the Arduino program only if they are included in a form of a library.

```
template <class T> int EEPROM_writeAnything(int ee, const T &value)
template <class T> int EEPROM_readAnything(int ee, T &value)
template <class T> void EEPROM_limitVar(const T &limit1, const T &limit2,
                                        const T &def, T &value, int &adress)
```

- **EEPROM_writeAnything:** Accepts integer address and a reference to an arbitrary variable. Executes writing of the variable to EEPROM memory starting from the given address, and returns the number of bytes that the variable occupies in memory.
- **EEPROM_readAnything:** Accepts integer address and a reference to an arbitrary variable. Reads the values from the EEPROM memory starting from the given address. It reads as many bytes as the received variable occupies, and places them in accepted variable passed by reference. Returns the number of loaded bytes.
- **EEPROM_limitVar:** Accepts 4 arbitrary variables of an equal type and a reference to an integer value representing the address in the EEPROM. The first two variables set the range. If the variable "value" is not from the given range, its value is set to the value of the variable "def" and as such is entered into the EEPROM memory over its old value. Address will be increased by the number of bytes of the variable "value", and the function returns nothing (void).

The following function demonstrates the use of the above functions:

```
void ucitajEEPROM() {
  adresa=0;
```

```
  // The order of variables must remain unchanged even though type can
  // be changed
  din_adr[0]=adresa; adresa+=EEPROM_readAnything(adresa,Tzadano);
  din_adr[1]=adresa; adresa+=EEPROM_readAnything(adresa,Histereza);
  din_adr[2]=adresa; adresa+=EEPROM_readAnything(adresa,Dtime);
  din_adr[3]=adresa; adresa+=EEPROM_readAnything(adresa,DTmin);
  din_adr[4]=adresa; adresa+=EEPROM_readAnything(adresa,mod);
  // Take the addresses, to access and change variables later
  // EEPROM: 0,1,2,3|4,5,6,7|8,9|10,11,12,13|14
    adresa=0;
  EEPROM_limitVar(8.0,35.0,20.0,Tzadano,adresa);
  // Executed only if EEPROM has invalid data
  EEPROM_limitVar(0.5,7.0,3.0,Histereza,adresa);
  // Address automatically increases within the function
  EEPROM_limitVar(1,60,15,Dtime,adresa);
  EEPROM_limitVar(1.0,30.0,4.0,DTmin,adresa);
  if (mod>0) mod=true; else mod=false;
  EEPROM_writeAnything(adresa,mod);
}
```

We see that the parameters occupy only 15 bytes of EEPROM memory. 3*4=12 bytes for double type, 2 bytes for an integer value (Dtime) and 1 byte for the logical value of the module's work mod (Manual/Auto).

If the entire code is loaded on an Arduino Uno microcontroller for the first time, then the EEPROM could have some unpredictable values. To ensure that the values are within the predefined limits, the generic function EEPROM_limitVar is used to bring the value back to the default scope.

3.4. Time interrupts

Interrupts are special events that can lead to the controlled interruption of the current piece of code to call the appropriate piece of code that "handles" the interrupt. Interrupt Driven Programming is a very powerful paradigm that significantly simplifies the implementation of complex systems on a microcontroller.

To use the time interrupts in the program, it is necessary to include the library:

```
#include <TimerOne.h> // For a time interrupt.
```

The setup of the program executes the following commands:

```
Timer1.initialize(1000000); // Initialization of interrupt in us
Timer1.attachInterrupt(provjeriCrtaj); // This function will run every second
```

When the time interrupt occurs, provjeriCrtaj() function will be called, and after that the microcontroller will proceed with the execution of code. Its main role is to count seconds. The countdown starts when the burner is turned on and starts from the value Dtime. When Dt falls to 0 (Dtime seconds after the burner has been turned on), then this function checks if the system works. It checks inequality: DT>DTmin. As soon as it is not true, the boiler and burner are switched off and an appropriate message about the defective functioning is displayed to the user. Testing is done only if the burner is turned on, after the startup time.

The secondary role of this function is rendering the screen every second. Which screen is drawn, depends on the value of a state variable:

```
byte state=1; // 1-main screen, 2-MENU, 3-prompt
```

4. TCP/IP communication and Telnet protocol

One of the main features of today's embedded systems is the ability to be accessed over the Internet. For example, it is possible to use cell phones, laptops, PCs, or other devices to

remotely turn on the water heater, light or air conditioning in the apartment. One possible way to achieve this is demonstrated in this chapter.

4.1. TCP/IP and Telnet protocol

TCP / IP is a group of protocols named after the two most important protocols: TCP (Transmission Control Protocol) and the IP protocol. TCP / IP can be used to communicate across a variety of interconnected networks, and is today's most widely used protocol on LAN and Internet.

Table 4.1. 4-layer model, TCP/IP group of protocols

Layer	Protocol
Application	DNS, DHCP, TLS/SSL, TFTP, FTP, HTTP, IMAP, IRC, NNTP, POP3, SIP, SMTP, SNMP, SSH, Telnet, BitTorrent, RTP, rlogin, …
Transport	TCP, UDP, DCCP, SCTP, IL, RUDP, ...
Internet	IP (IPv4, IPv6), ICMP, IGMP, ARP, RARP, ...
Data Access	Ethernet, Wi-Fi, Token ring, PPP, SLIP, FDDI, ATM, DTM, Frame Relay, SMDS, …

Telnet is a network protocol within the IP group of protocols. The purpose of this protocol is to establish a two-way 8-bit communication channel between two network devices. It is often used to provide a user session for a command line interface to another device. The name comes from the words TELephone NETwork, which shows that the protocol was designed with the intention of connecting a terminal to a remote device. It is common that the term telnet is used as a name of the interactive client program that lets you connect using the Telnet protocol.

4.2. Arduino code for communication establishing

The header includes a library containing the functions for the proper use of the Ethernet Shield.

```
#include <SPI.h>      // for Ethernet
#include <Ethernet.h> // for Ethernet
```

Declared variables necessary for the further course of the program:

```
// Variables for Ethernet
```

```
byte mac[]={0xDE,0xAD,0xBE,0xEF,0xFE,0xEE};
byte ip[]={192,168,1,177};
#define textBuffSize 7       // Longest command plus 2 places for CR + LF
char textBuff[textBuffSize]; // For received text
byte charsReceived=0;
boolean connectFlag=false; // To know if a new connection has been created
unsigned long timeOfLastActivity; // Time of last activity [ms]
unsigned long allowedConnectTime = 300000; // After 300000 ms (5 min)
                                           // of inactivity, Telnet session
                                           // is closed
EthernetServer server(23); // Telnet, tipically goes through the port 23
EthernetClient client=0;
```

In order to make the initialization of object *server*, it is necessary to call the following two commands in the Setup part:

```
  Ethernet.begin(mac,ip);
  server.begin();
```

The gets a MAC (Media Access Control address) and IP address. Here is used the IP address 192.168.1.177. Range of addresses 192.168.1.0 - 192.168.1.255 is usually used for private LAN. Therefore, the computer that is connected to the module must configure the LAN IP address to be within that range.

Initialization is possible with any of the following function calls:

Ethernet.begin(mac);

Ethernet.begin(mac, ip);

Ethernet.begin(mac, ip, dns);

Ethernet.begin(mac, ip, dns, gateway);

Ethernet.begin(mac, ip, dns, gateway, subnet);

The module acts as a server, listening awaiting commands. The principle of operation is clearly seen in the following code running in each iteration of the infinite loop:

```
  if (server.connected() && !connectFlag) {
  // If someone accessed the Telnet server, and the session is not open
    connectFlag=true;
    client=server.connected();
    client.println(F("Arduino Telnet Server"));
    client.println(F("----------------------"));
    printHelpMessage();
    printPrompt();
  }
  if (connectFlag && client.available()) getReceivedText();
  // If there is a new character waiting to be read, and the flag is HIGH
  if (connectFlag) checkConnectionTimeout(); // If session open,
  // check whether to do disconnection for inactivity
```

The function getReceivedText(); is used to receive a command sent by the user, and then the function parseReceivedText(); interprets the command and calls an appropriate function:

```
void parseReceivedText() {
  switch (textBuff[0]) {
    case 'i' : ispisiVarijable(); break;
    case 'A' : promjTzad(); break;
    case 'B' : promjHist(); break;
    case 'C' : promjDTmin(); break;
    case 'D' : promjDtime(); break;
    case 'E' : modus(); break;
    case 'F' : bojlerch(); break;
    case 'R' : restart(); client.println(F("  Modul restartovan")); break;
```

```
    case 'c' : diskonektuj(); break;
    case '?' : printHelpMessage(); break;
    case 0x0d : break; // Enter command is ignored
    default: printErrorMessage(); break;
  }
}
```

4.3. Overview of available commands

Commands which module recognizes are listed in the following table:

Table 4.2. The list of available commands that module recognizes

Syntax	Function
i	print all significant variables
Axx.x	change setpoint of the room temperature (8,35). For example: A27.5
Bxx.x	change hysteresis (0.5,7). For example: B4.1
Cxx.x	change DTmin (1,30). For example: C5.2
Dxx	change Dtime (1,60). For example: D20
Ex	change mod, x=0 manual, x=1 automatic. For example: E0
Eu	turn burner on (only in manual mod)
Ei	turn burner off (only in manual mod)
Fu	turn boiler on
Fi	turn boiler off
R	restart of the module
c	disconnection
?	help: prints all commands

4.4. Demonstration using the Telnet client php website

After correctly connecting the module to a boiler system and to a computer (or a router) with an Ethernet cable, we run a web server on a LAN with a website implemented specifically for demonstration of the module's interface. Opening of the web page displays the following screen:

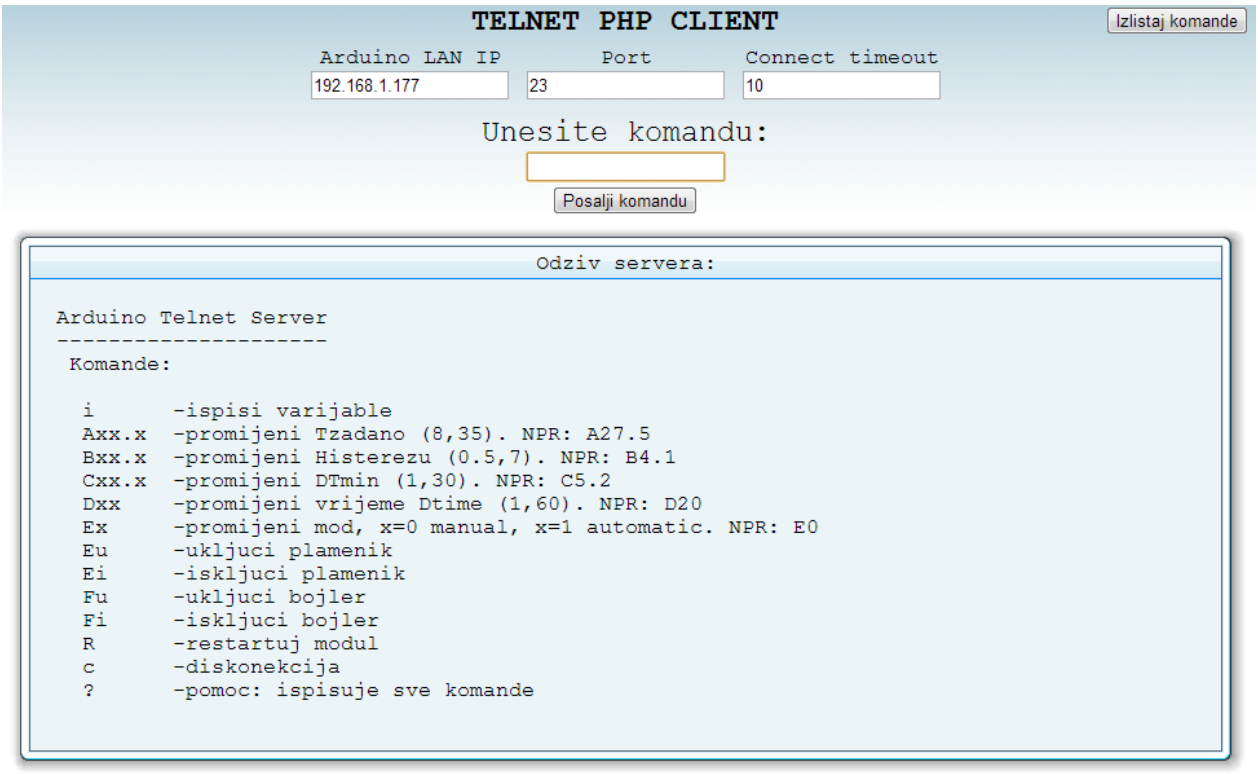


Figure 4.1. Telnet php client, home page

The predefined variables within the PHP script filled fields: IP address, Port and Connect timeout. If the module is not properly connected, that is, if communication is not established after 10 seconds, a message will appear as shown in Figure 4.2.

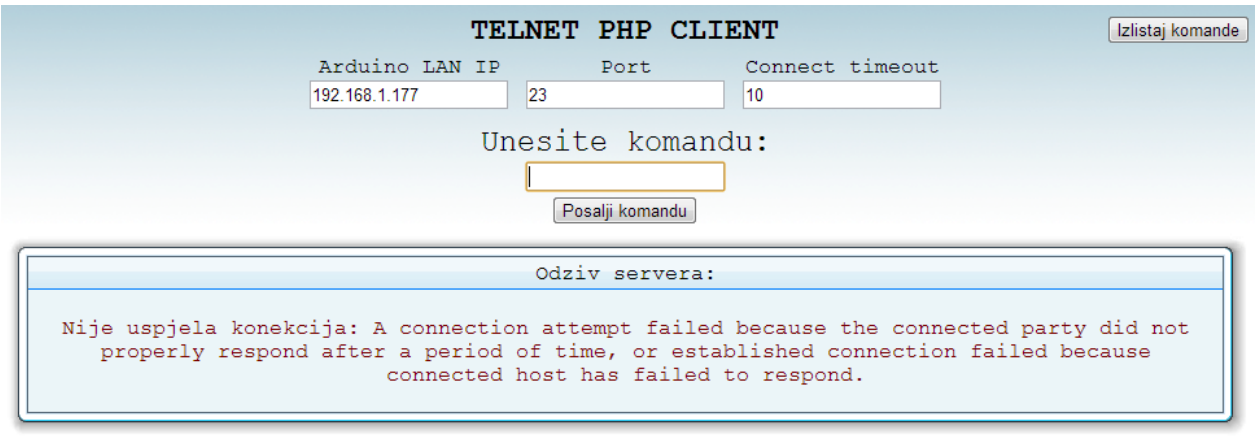


Figure 4.2. Telnet php client, communication error

If everything is fine, then by entering commands a user can control the boiler system. After the execution of each of the commands, the user will get an answer that either confirms that the command is executed successfully, or explains why it has not been executed. The following figures show a few examples:

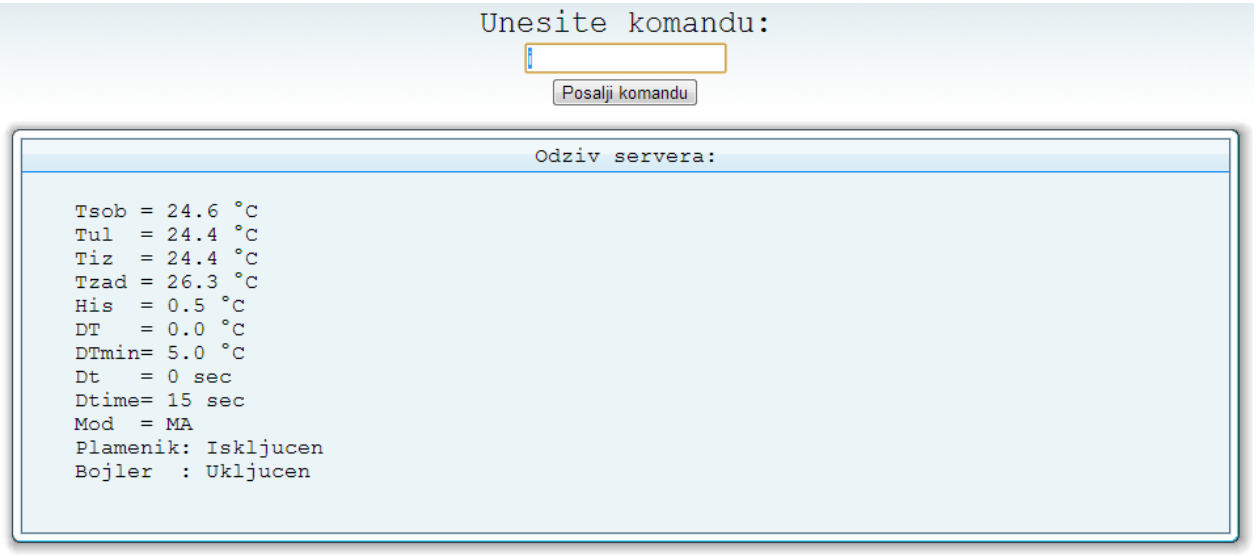


Figure 4.3. Print variables

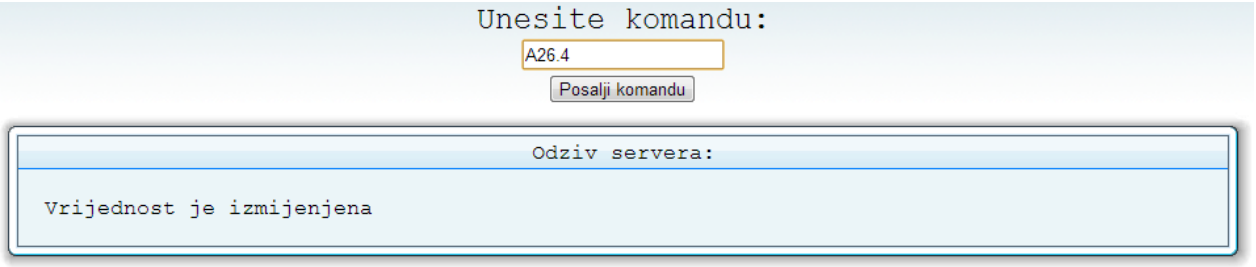


Figure 4.4. Sets the desired temperature to 26.4°C

CONCLUSION

This work shows the entire process of development of the control module for a gas water heater. Starting from the theoretical description, the physicality and the principle of operation of the gas water heater, all necessary requirements for a boiler system are recognised so that the module can be implemented successfully. Development of modern engineering disciplines leads to better performance of hardware structures. The processors are faster, all hardware components are cheaper, smaller, and more reliable. Usually a hardware failure is attributed to the carelessness of users, or to the age of the device. On the other hand, the software part of the solution is not subject to aging. However, it is noticeable that the commercial applications as well as embedded systems have errors from time to time. These errors can sometimes lead to a disaster if it comes to a system with an important function. The reason is that, unlike the process of developing the hardware, one can not introduce standardization in the development of software solutions, nor in testing of the software. When one wants to develop a module system based on a microprocessor technology, it is necessary to achieve the overall functionality using a synergy of hardware and software. Implementation of the hardware system began collecting all relevant data and specifications of components. In particular, one should take into account the limits of voltage and current (and possibly power) to which a component can be exposed. Attention was paid to the complexity and price of components. The system is made of very cheap components, but it fulfills all specifications. Shield for connecting components with Arduino consists of ordinary resistors, two LEDs and one three-state chip, while the keyboard is nothing more than a 4x4 matrix of conductors. LED display is worth a few dollars. Arduino and Ethernet shield together cost about 18$. The system could be created using even cheaper components. Within the described control module, a microcontroller allows us to control output

components (LCD, relays, Ethernet response) and it does a correct interpretation of data and commands coming from input components (sensors, keyboard, Ethernet). Secondary applications developed on a PC (or other device) can utilize this basic interface as part of useful extensions. The module can be improved in several directions. The support for micro-SD card could be used (there are cards with 1 TB capacity). Within LAN, it would be useful to develop an application that stores data, displays variables in real time and possibly allows animated and interactive burner, boiler and parameter editing. The module could be extended in terms of supporting independent temperature control for multiple rooms. Accuracy largely depends on used temperature sensors. This module used sensors LM35DZ, which are sufficiently accurate. Greater precision and accuracy can be achieved by using LM35CZ sensors, which are slightly more expensive. One could also prepare a look-up table for thermistors, but simple thermistors have undesired hysteresis and are not suitable for control of a more dynamic rates of change of temperature. It would be nice to make a scheduling CRON web service and an alerting system. Users would be able to schedule automatic water heating at specific time, for example time of cheap tariff. Smart home is a project which is noteworthy and based on pure innovation.

References

1. P. Godse, A. O. Mulani (2009) „*Embedded Systems“*, Technical Publications Pune

2. Hephaestus Books (2011) *„Articles on Motherboard Form Factors, Including: Atx, Mini-Itx, Nlx, Btx (Form Factor), at (Form Factor), Lpx (Form Factor), Wtx (Form Factor), PC/104“*, Hephaestus Books

3. G. F. Gilman (2010) *„Boiler Control Systems Engineering - Second edition“*, International Society of Automation

4. Tankless Water Heaters (2013) *„http://www.plumbing-solutions-inc.com/faq.html“*

5. Latif M. Jiji (2009) *„Heat Conduction”*, Springer

6. Xtreme I/O ADC-DAC (2011) datasheet

7. PXA255 PC/104 Single Board Computer VIPER (2007) datasheet

8. Arduino Uno (2014) *„http://arduino.cc/en/Main/arduinoBoardUno“*

9. ATmega328 (2009) datasheet

10. Nokia display (2008) *„User_Manual_ET_LCD5110“*

11. PCD8544 48x84 pixels matrix LCD controller/driver (1999) datasheet

12. 4x4 Matrix Membrane Keypad (2011) *„http://www.galeon.com/oswagar2/tec_mem.pdf“*

13. SN74LS244 (2008) datasheet

14. LM35 Precision Centigrade Temperature Sensors (2000) datasheet

15. Ethernet shield (2013) *„http://arduino.cc/en/Main/ArduinoEthernetShield“*

16. EEPROM library (2013) *„http://arduino.cc/en/Reference/EEPROM“*

17. Timer1 time interrupt library (2008) *„http://playground.arduino.cc/code/timer1“*

18. Arduino Ethernet library (2013) *„http://arduino.cc/en/reference/ethernet“*

19. On-Demand Water Heater System Design Manual (2009)

„http://www.equipmentcontrols.com/pdf/186965-001.pdf“

20. Adafruit PCD8544 Nokia 5110 LCD library (2013) *"https://github.com/adafruit/Adafruit-PCD8544-Nokia-5110-LCD-library"*

21. Adafruit GFX Library (2013) *"https://github.com/adafruit/Adafruit-GFX-Library"*

APPENDIX 1 - A library with generic functions for EEPROM

The use of this library requires the inclusion of the standard Ethernet library for work with EEPROM and the Arduino environment (`#include <EEPROM.h>`).

Content of the library „EEPROM_anything.h“

```
template <class T> int EEPROM_writeAnything(int ee, const T &value) {
    const byte* p = (const byte*)(const void*)&value;
    unsigned int i;
    for (i = 0; i < sizeof(value); i++)
          EEPROM.write(ee++, *p++);
    return i;
}
template <class T> int EEPROM_readAnything(int ee, T &value) {
    byte* p = (byte*)(void*)&value;
    unsigned int i;
    for (i = 0; i < sizeof(value); i++)
          *p++ = EEPROM.read(ee++);
    return i;
}
template <class T> void EEPROM_limitVar(const T &limit1, const T &limit2,
const T &def, T &value, int &adress) {
    // This function returns a variable into the given range,
    // automatically increases the address
    if (value<limit1 || value>limit2) {
        value=def; // "Default" value
        EEPROM_writeAnything(adress,value);
    } adress+=sizeof(value);
}
```

APPENDIX 2 - Electrical scheme of the module

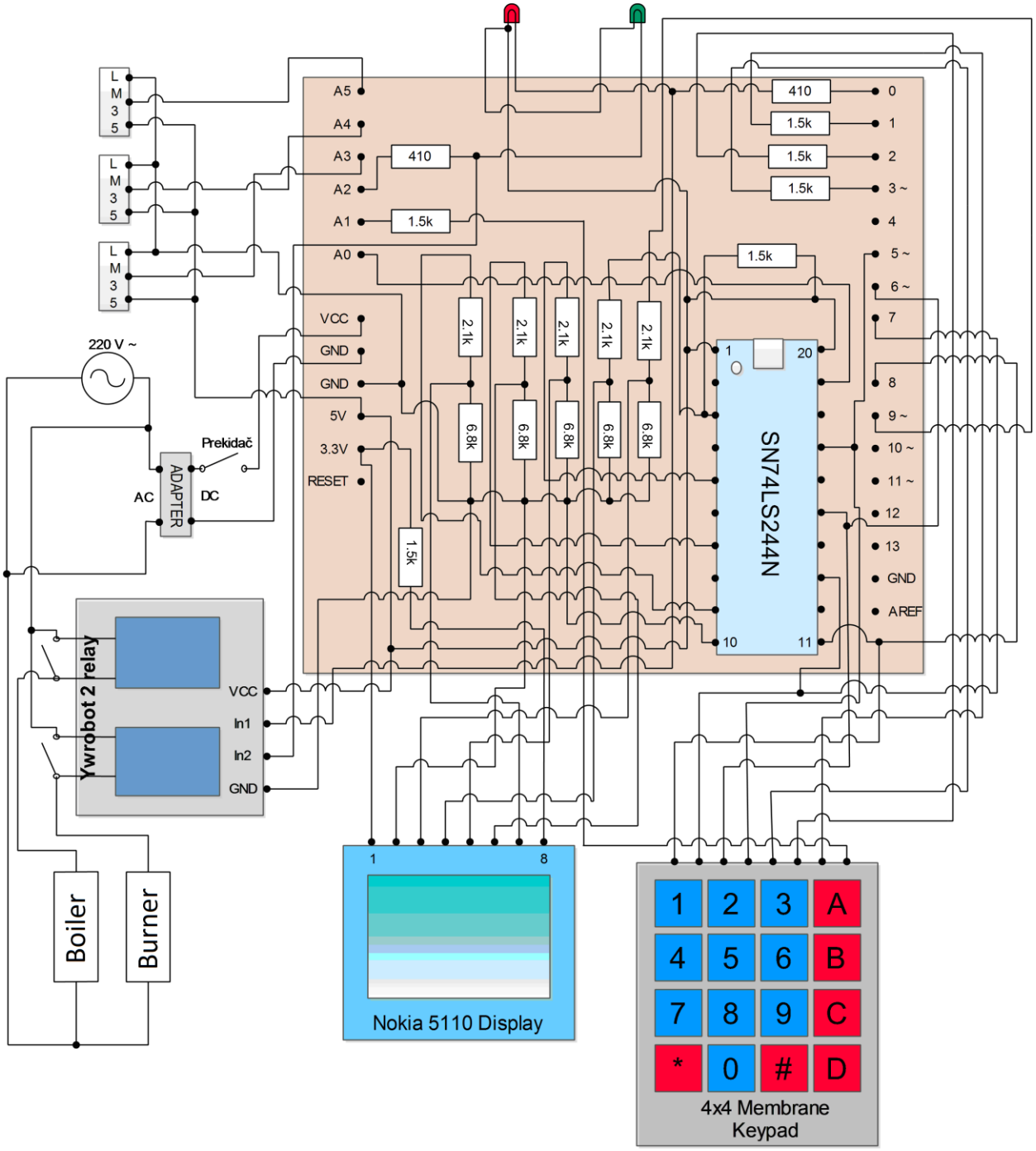


Figure P2.1. Electrical scheme of the module

APPENDIX 3 - Appearance of the implemented control module

*Figure P3.1. A*ppearance of the control module

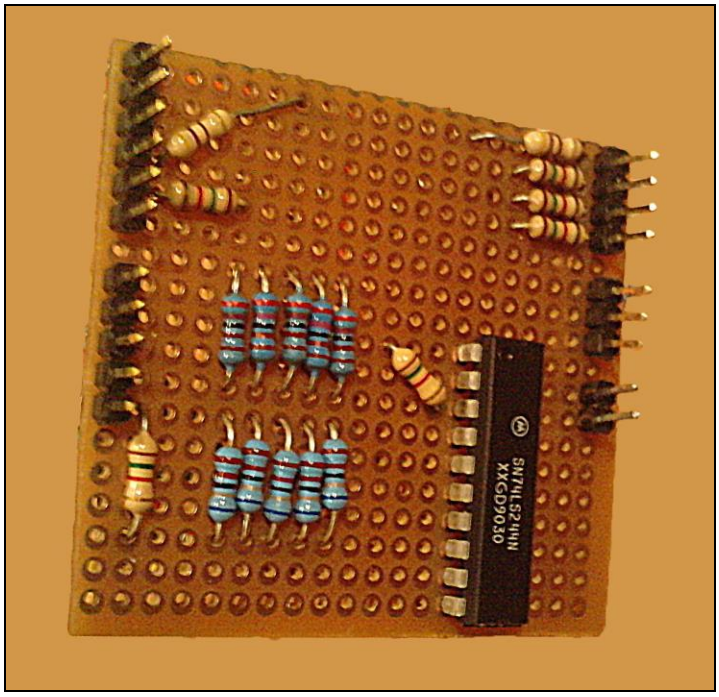


Figure P3.2. Shield board for connection of the microcontroller with other components