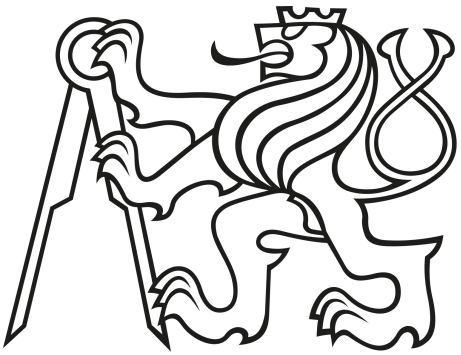
# OPTIMIZATION OF POWER GENERATION OF A PHOTOVOLTAIC SYSTEM ON A RESIDENTIAL HOUSE

EVŽEN KONSTANTINOV



Diploma thesis

Dept. of Electrical Machines, Apparatus and Drives

Electrical Engineering, Power Engineering and Management

Faculty of Electrical Engineering

Czech Technical University in Prague

Supervisor: Ing. Tomáš Haubert

České vysoké učení technické v Praze
Fakulta elektrotechnická

katedra elektrických pohonů a trakce

# ZADÁNÍ DIPLOMOVÉ PRÁCE

Student: **Bc. Evžen Konstantinov**

Studijní program: Elektrotechnika, energetika a management
Obor: Elektrické stroje, přístroje a pohony

Název tématu: **Optimalizace výroby FVE v rodinném domě**

Pokyny pro vypracování:

1) Navrhněte řídicí hardware s modulem G120-E obsahující připojení skrze Ethernet, vstupy pro teplotní senzory, výstupy s relé a výstup na LEM senzory.
2) Do tohoto modulu naprogramujte software, který bude řídit vytápění, komunikovat s fotovoltaickou elektrárnou, logovat data na SD kartu a odesílat na webový server.
3) Řídicí software bude navržen tak, aby pro topení byla co nejvíce využívaná fotovolatická elektrárna, tedy minimalizující spotřebu pelet v automatickém kotli.
4) Pro logovaná data vytvořte software umožňující jejich zobrazení a statistické vyhodnocení.

Seznam odborné literatury:

[1] ZÁHLAVA V.: Návrh a konstrukce desek plošných spojů: principy a pravidla praktického návrhu
[2] GANSSLE J.: The Art of Designing Embedded Systems
[3] HRADÍLEK Z., LÁZNIČKOVÁ I., KRÁL V.: Elektrotepelná technika

Vedoucí: Ing. Tomáš Haubert

Platnost zadání: do konce letního semestru 2016/2017

L.S.

Ing. Jan Bauer, Ph.D.
vedoucí katedry

prof. Ing. Pavel Ripka, CSc.
děkan

V Praze dne 1. 4. 2016

# ČESTNÉ PROHÁŠENÍ

Prohlašuji, že jsem zadanou diplomovou práci vypracoval samostatně a použil jsem k tomu pouze literaturu, kterou uvádím v seznamu přiloženém k této diplomové práci.

Nemám závažný důvod proti užití tohoto školního díla ve smyslu § 60 Zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon).

*Praha, Květen 2016*

Evžen Konstantinov

# ACKNOWLEDGEMENTS

I would like to thank my supervisor Ing. Tomáš Haubert for guidance and valuable professional advice, as well as passed knowledge. I would also like to thank my family for their patience and support during my studies and the creation of this thesis.

# ANOTACE

Předmětem této diplomové práce je optimalizace výroby fotovoltaické elektrárny na rodinném domě. Jejím cílem je navrhnout a vyrobit řídící a měřící systém pro fotovoltaiku a topení v domě, implementovat algoritmus pro řízení, měření elektrického proudu a ukládání dat v programovacím jazyce C# a navrhnout grafické rozhraní pro vizualizaci naměřených dat.

# ABSTRACT

The subject of this Diploma thesis is Optimization of power generation of a photovoltaic system on a residential house. Its aim is to design and manufacture a control and measurement system for the solar and heating system of the house, to implement algorithms for control, current measurement and data-logging in C# programming language and to design an software interface for data visualization.

KEYWORDS

G120E module, .NET Micro Framework, C#, Photovoltaic system, Heating, Data logging, Residential house, Embedded system, Current measurement, LEM, Visual Studio

# CONTENTS

# LIST OF FIGURES

# LISTINGS

# ACRONYMS

AD Analog-to-Digital

ADC Analog-to-Digital Converter

CAD Computer Aided Design

CAN Controller Area Network

COM Communication port

DC Direct current

DIO Digital Input/Output

EMC Electromagnetic compatibility

GC Garbage collector

GPIO General Purpose Input/Output

GUI Graphical User Interface

HTML Hypertext Markup Language

HTTP Hypertext Transfer Protocol

HTTPS Hypertext Transfer Protocol Secured

HVAC Heating, Ventilating, Air-Conditioning

IC Integrated circuit

IDE Integrated Development Environment

IO Input/Output

LAN Local Area Network

LCD Liquid Crystal Display

MIMO Multiple-In, Multiple-Out

MMC Multi-media card

MPC Model-Predictive Control

MPP Maximum Power Point

MW Mega-Watt

NETMF .NET Micro Framework

OOP Object-Oriented Programming

PCB Printed Board Circuit

PID Proportional-Integrated-Derivative

PV Photo-voltaic

PWM Pulse-width modulation

RMS Root-Mean-Square

SD Secure Digital

SMD Surface-Mounted Device

SPI Serial Peripheral Interface

TUV Czech abbreviation of Warm Utility Water

UART Universal Asynchronous Receiver/Transmitter

USB Universal Serial Bus

UV Ultra-violet

# 1

# INTRODUCTION & THEORY

In the past decades power engineering has undergone great trasformation. Thanks to knowledge of impacts of conventional power plants on planet Earth many new means of power generation have been discovered. The society became aware of negative effects of green-house gas emmission and began pressing power industry into developing clean power plants. At the same time, news of the forthcoming depletion of fossil fuels further encouraged development of utilization of renewable energy sources. Since then, people started with large scale power generation using:

1. Direct sunlight
2. Wind
3. River flow
4. Tidal waves
5. Geothermal energy and more

Renewable energy sources have become part of today's power industry and are gaining on significance by the day. One of the most dynamically developing means of power generation are photovoltaic power plants. This branch of power engineering has undergone great evolution thanks to the expansion of semiconductor technology, which is a major part of the electrical and electronical industry as a whole.

The benefit of this way of power generation is its versatility. Photovoltaic power plants can be projected as giant photovoltaic fields generating MWs of energy, as well as small systems no larger than a few $cm^2$ and generating fractions of Ws. Since it is so modular, many people living in a house decide to install a photovoltaic plant of top

of their roof. The reasons are obvious: less power is drawn from the electrical grid (lower energy bills) and it is a decent power backup for blackouts or unexpected events, if combined with an energy accumulator (electrochemical, thermal, etc.).

However, to be able to utilize the energy generated in the photovoltaic modules, it is necessary to adjust the electrical parameters of the output of the modules. That is the duty of the control system, without which the photovoltaic system cannot function properly. The system consists of many devices and sensors operating under control of a microprocessor unit. This controller unit is the main subject of this thesis.[3]

### SYSTEM IDENTIFICATION

The aim of this thesis is to design a control unit for control and optimization of power generation on an existing house with a rooftop photovoltaic power plant. This section will be dedicated to the description and identification of the system.

The house is located in the Czech Republic, in the Northern region in a village named Horní Slivno.[8] It is a two-storey new residential house with a photovoltaic plant on the roof. Its performance is 15,75 kWp. It is connected as three strings of 63 solar panels in total, 250 Wp each panel (manufacturer Canadian Solar - CS6P). The orientation of the panels is 190 ° (nearly South) and they are tilted by 35 ° from the vertical axis, which gives an optimal irradiance throughout the day.

The output DC current of the photovoltaic plant is connected to an inverter, which converts it to the standard 400 V and 50 Hz AC. The inverter is manufactured by Danfoss, it is the model TLX 15k. It has a built-in MPP tracker for every phase of the output current (this optimizes the power drainage from the photovoltaic modules). The maximum output current of the inverter is 3x22.4 A AC and the input DC voltage is 700 V.

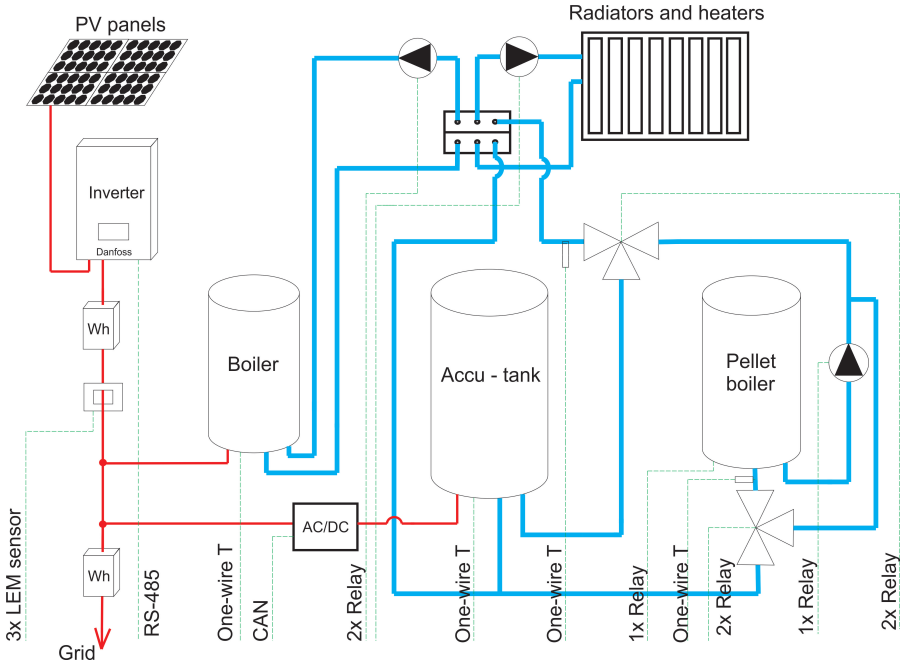



Figure 1: Schematic of the system.

The photovoltaic modules with the inverter create the photovoltaic system. This system shall communicate with the control system and according to its commands, it shall distribute the generated power into the house grid. The problem of energy accumulation is solved with an accumulation tank of water with electrical heating. The tank is connected to the heating system of the house and heats water flowing into the radiators via a heat exchanger.

In the same water circuit, there is a boiler, which generates heat by burning wood pellets. The pellets are automatically fed into the boiler with an electric motor. The water flow through this boiler is created by a water pump, the flow is switched on/off with a valve.

There is also a boiler for heating of utility water (washing, cooking, etc.) which is heated electrically, it is in a seperate water circuit and controlled by another set of water pump and armature.

The whole system is visualized in Figure 1. There is an array of sensors which function as input for the mathematical model in the control system. The measurement system receives measurement readings from the following sensors:

- Temperature in the boiler (One-Wire)
- Temperature of the heating water (One-Wire)
- Temperature in the accumulation tank (One-Wire)
- Temperature in the pellet boiler (One-Wire)
- Logical value of the thermostat (Opto-couple)
- 3x LEM electrical current sensors - actual current/power measurement

There are also some communication buses, which communicate with and control active parts of the system:

- RS − 485 bus - communication with inverter
- CAN bus - communication with rectifier for the accumulation tank
- 6x electromagnetic relay - controls the switches and directs power from the photovoltaic plant as needed
- Ethernet - connection to the internet (data logging, time synchronization)

All the means of control and communication are described and explained in Chapter 2, as well as the control system. So, the system's duty is to measure and log data and according to the readings control the power-flow around the house. The data logging hasn't been mentioned yet. Data is saved to two types of media: SD Card and to a server via internet connection. This shall be described in Chapter 3 with the software.

*Inverter*

The inverter is probably the most important part of the PV system. Without it, the system wouldn't be able to utilize the energy in the

form it comes. Modern inverters aren't just the power electronics, they are computer controlled, intelligent systems with data-logging and various means of communication.

The power electronics of the Danfoss TLX 15k are equipped with a 3-level bridge inverter. This means that each phase switches between 3 states: $\frac{+V_{DC}}{2}$, 0, $\frac{-V_{DC}}{2}$. An illustrative schematic is in Figure 2.

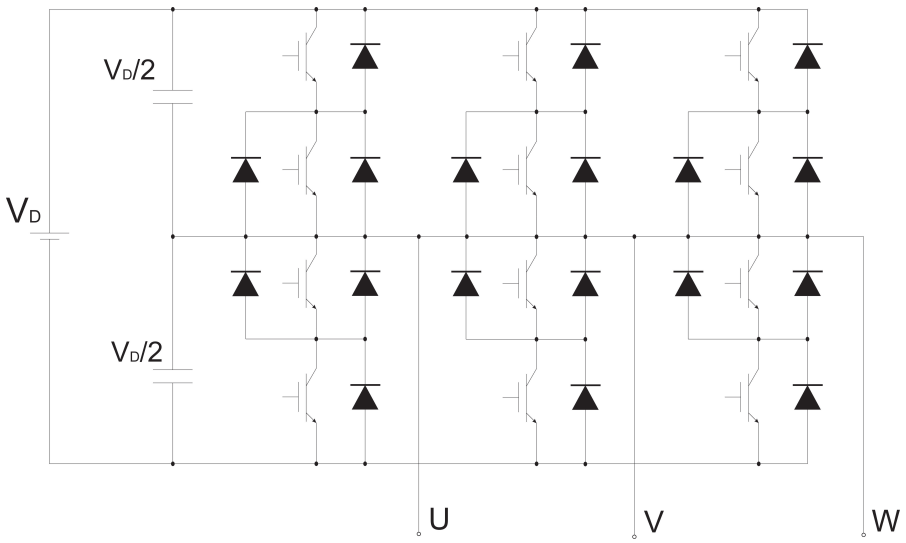



Figure 2: Schematic of a 3-level 3-phase inverter.

The electrical parameters of the inverter are:

- $V_{RMS}$ = 230,47 V
- $I_{RMS}$ = 21.921 A
- P = 5046.5 W per phase (15 kW in total)
- $\cos(\phi)$ = −0.999

The inverter is able to provide the user with a great amount of data including actual power, daily energy generation and even PV panel irradiation. The data is accessible via various communication buses, this project has chosen the RS − 485 bus, which is used with the Danfoss *ComLynx Protocol*.

*ComLynx Protocol*

The *ComLynx* Protocol was developed by the company Danfoss for secure and stable communication with their products, in our case the inverter TLX 15k. It is a protocol primarily designed for use with the $RS-485$ bus. The protocol is specifically designed for the products of *Danfoss* - the inverters DLX, TLX, ULX, FLX, each one having different parameters and IDs.

| ComLynx message | | | | | | |
|---|---|---|---|---|---|---|
| Start of Frame | | | Message Content | | End of Frame | |
| Start | Address | Control | Header | Data | FCS | Stop |
| 1 byte | 1 byte | 1 byte | 0 bytes | 0-253 bytes | 2 bytes | 1 byte |
| 0x7E | 0xFF | 0x03 | | | | 0x7E |

Figure 3: Table of *ComLynx* message.

In the table of Figure 3 the common message is depicted. The message consists of 7 parts: the Start, Address, Control (Start of Frame), 2 Data parts (Message), FCS and Stop (End of Frame). The Start and Stop bytes are always set to 0x7E, which has a reason in further processing. The FCS (Frame Check Sequence) is a 16−bit checksum based on the polynomial $X^{16} + X^{12} + X^5 + 1$ and is calculated over the whole message (except the Start and Stop bytes). The FCS calculation algorithm is designed in such a way that the result of the FCS calculation becomes a certain unique value at the moment when the calculation algorithm passes over the 2 FCS bytes of an incoming message. This can be useful if the FCS is calculated "on the fly" as bytes are received, as it can be used to identify the end of a message.[2]

Before the message is received, the checksum must be calculated to ensure correct transfer and detect the end of the message.

Listing 1: Code: FCS checksum

```
int i = 0;
int length = 18;
int fcs = 0xFFFF;
while (length != 0)
{
    length--;
    i++;
    fcs = (fcs >> 8) ^ FCSTable[(fcs ^ output_buffer[i]) & 0xFF];
}
fcs ^= 0xFFFF;
```

The algorithm performs the checksum over a byte[] array which defines the message. The code in Listing 1 performs the checksum over the output byte[] array containing the message for the inverter. When this encoded message is received by the inverter, an appropriate reply is sent back. The datasheet [2] to the inverter and to the *ComLynx* protocol defines at which positions of the returned message the desired information is. In the case of this project, the desired values (Actual power of the inverter) are placed in **bytes** 18-15. The decoding of the message in this thesis is described in Listing 4.

## PRINTED CIRCUIT BOARD

The first problem was to design the Printed Circuit Board for control purposes of the system. This section will be devoted to description of the manufacturing process of PCBs.

### *PCB Manufacture*

Todays electronic industry is ruled by Printed Circuit Boards. Practice has shown that integrating circuits into a small area flooded with electrical components, conductors and isolants enables miniaturization and optimization of circuit manufacture, robustness and quality. Miniaturization is possible thanks to great discoveries in material sciences, especially isolation materials, which enable us to place many electrical components very close to each other without hav-

ing concerns of EMC. Advances in semiconductor technology make way for effective utilization of microcontrollers and integrated circuits. Thanks to many years of practice in industry, technologists and scientists have designed and realized a very viable procedure for creation of PCBs. Todays technologies enable engineers to design and create complex and miniature circuits on multi-layered boards with power-planes, grounding planes etc. with a very low cost.

So, what is a PCB? PCBs consist of 2 basic parts: the substrate, which acts as a carrier for the components and conductors (the substrate is usually made of FR4, which is a fiberglass-epoxy laminate), and the printed wires, which are layered on top of the substrate. The substrate is there to carry the components and to isolate them from each other, while the copper layer functions as a conductor connecting the components. The basic layout is shown in Figure 4.

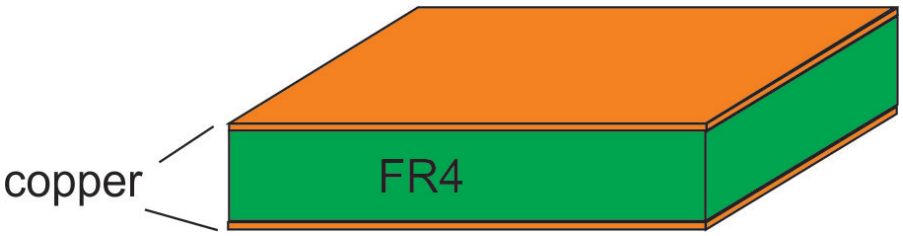



Figure 4: Typical PCB structure.

The copper cladding can be mounted onto the substrate by 2 methods: copper plating onto the surface or copper foil gluing to the substrate. The basic double-sided PCB consists of one FR4 substrate and two copper layers (one on each side). It is possible to make multilayered PCBs by layering these basic boards onto each other, resulting in boards with $4, 6, 8$ copper layers, which enables higher level of integration and complexity.

The traces and pads which are visible on the board are usually fabricated using the photolitographical method and acid etching. The desired shapes are obtained by selectively removing the redundant copper. The usual fabrication follows this order:

- Deposition of the polymer coating ("Photoresist") on top of the board
- Attachment of the mask (positive or negative)
- Exposure of the board to ultra-violet light
- Removing of the mask
- Removing of the photoresist which wasn't exposed to the UV light
- Etching in a corrosive solution (e.g. alkaline ammonia) which washes away the redundant copper
- Removing the remaining photoresist

As the photoresist is exposed to the UV light, it creates a shield on the desired parts of copper, which protects it in the process of chemical etching of the redundant copper. As a result, only the protected copper stays attached on top of the board and the rest is washed away. This way the conductive traces are fabricated on all copper layers of the board, resulting in an interconnected net of circuits.

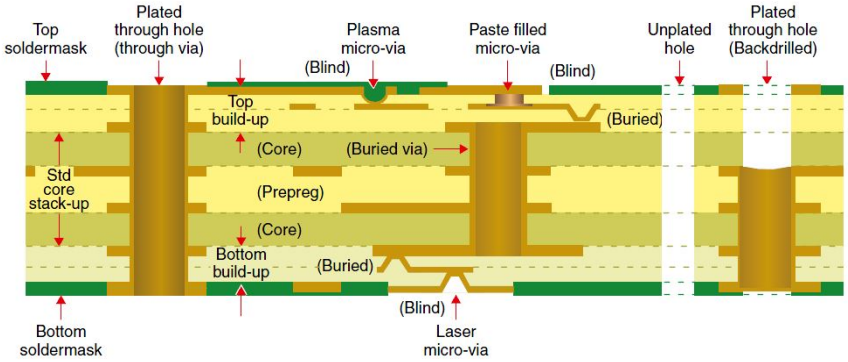



Figure 5: Types of vias. Picture taken from [7]

The copper layers are connected to each other with drilled holes, called vias, so usually a computer-controlled drill is incorporated into the process. Once the holes have been drilled and desmeared, it is necessary to make an electrical connection between the copper layers,

so the board is placed into a plating bath and the insides of the holes are coated with copper.

Some copper layers function as planes for low-impedance connection to ground or power supply, but some layers must be isolated from certain vias, so it is surrounded by a protective clearance area. To make a grounding plane without the use of another plane, copper pour is used. Copper pour is a technology where copper is "poured" all around the copper traces, pads and vias with a defined clearance. All the grounding pads of components are connected with the poured copper.

Last, the solder mask and the silk screen are applied. The solder mask covers all the traces except for the pads. Its purpose is to simplify the soldering process, to further isolate the traces from the copper pour and to protect the board from oxidation. The silk screen is a layer for creating marks and text on the board. Last comes the soldering of electrical components onto the board.

Electrical components come in many shapes, sizes and packages. The components are mainly divided into 2 groups: surface mounted (SMD) and through-hole. SMD components are preferred because they enable further miniaturization and a higher degree of automation of mounting and manufacture.

Many problems need to be taken into account during the design phase of the PCB. The engineer needs to determine the number of copper planes and layers on the board. The less layers, the smaller the cost, however there are more constraints due to the reduced space for traces. It is recommended to use grounding and power planes in multi-layered designs. The electrical components must be taken into account as well, their mounting technology (SMD x through-hole), size, EMC, availability etc. The key part of the design process is to choose the suitable software instrument.

Today, the whole process may be realized with a single CAD instrument. There are many software products for computer-aided PCB design available which enable the engineer to design the circuits from

scratch and finalize the project by exporting the results into files compatible with the PCB fabrication machines. Next to these design programs there are circuit-simulation developing environments which enable the user to simulate and test the circuits without the need to spend extra money on test hardware. The PCB for this project was designed in the software Cadsoft Eagle.

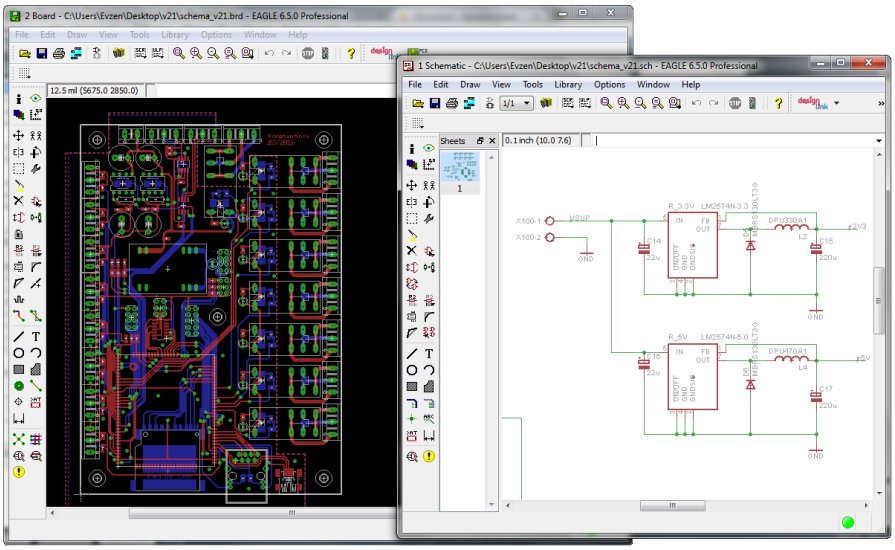



Figure 6: Example screenshot of PCB design software Eagle.

The software lets the engineer create the schematic and simultaneously design the layout of the board. Very large component libraries make it simple to create the schematic and layout. The software is equipped with extensive settings of physical parameters of the traces, pads, vias, clearances etc. with standardized sizes and values. The difficult part of the process is the design of the wiring and connections of the components in accordance with the limited space, EMC and heat dissipation. Eagle has many instruments that aid the designer during routing. Most CAD products even have an "auto-route" function. It is generally not recommended to use it on the whole layout though, however if applied to certain modularised parts of the circuit, the auto-router may have great results. Useful are also functions that check the electrical connections and discover malicious board layout (e.g. traces crossing each other).

I will not list the design rules for creating PCBs since there is a great amount of literature describing proper PCB design. I have gained many insights upon this topic in the book [11].

## CURRENT MEASUREMENT

This section describes measurement of electric current in all 3 phases of the house grid. There are many possible ways of how to measure alternating current, some are more suitable, some are less suitable for utilization in this project. When measuring alternating current, we usually measure the RMS value, which expresses the effects of power dissipation. Another significant value is Peak value, which gives us information about the maximum possible value of the current. For the purpose of microprocessor control and data logging, it is necessary to sample the signal and quantize it to suitable values. The sampling is done by the analog-to-digital converters which are embedded on the microprocessor module. The AD conversion and RMS value calculation will be described in Chapter 3. The following paragraphs of this section have been inspired by [9].

### *Measurement methods*

To better understand the system for further optimization, it was necessary to measure the alternating current in the electrical grid of the house. There are 3 main methods for current measurement:

- Ohm's method
- Current transformer
- Contactless sensor

The simplest one of them is the Ohm's method. The current flows through a resistor of a defined resistance and the voltage across the

resistor is measured. The current is calculated using Ohm's law. The problem with this method is that there is great dissipation of energy in the resistor which rises with rising current, which also limits the measurement range. Therefore this method hasn't been chosen.

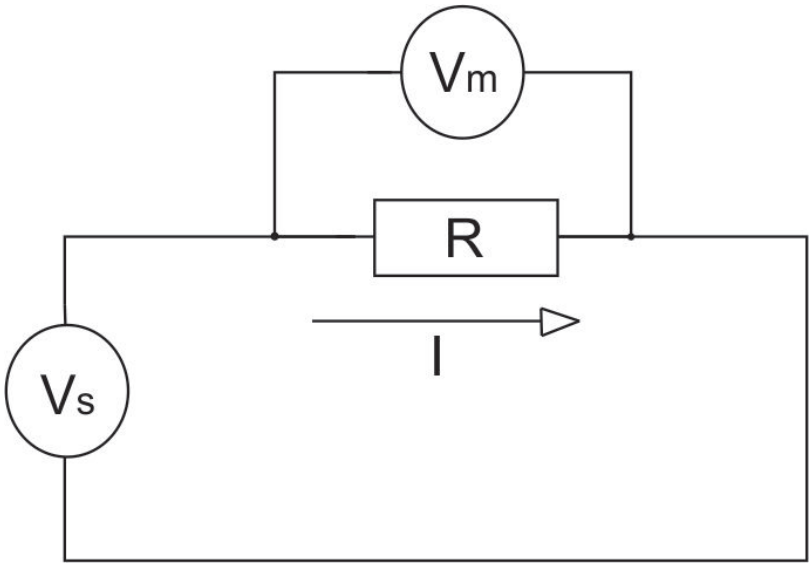



Figure 7: The Ohm's method circuit.

The Current transformer is suitable for measurement of higher currents. It is basically a transformer with many turns on the secondary winding and a few turns on the primary winding. Its pros are that it galvanically separates the circuits and that it doesn't deform the waveform much. However it is most suitable for measurement of high currents and its price is high as well as its size.

The last method incorporates contactless sensors. They are mostly magnetic sensors with Hall effect components. Their pros are a significant measurement range, galvanical separation of the circuits and over-current tolerance. Thanks to their favourable size and price, this alternative has been chosen for this project.

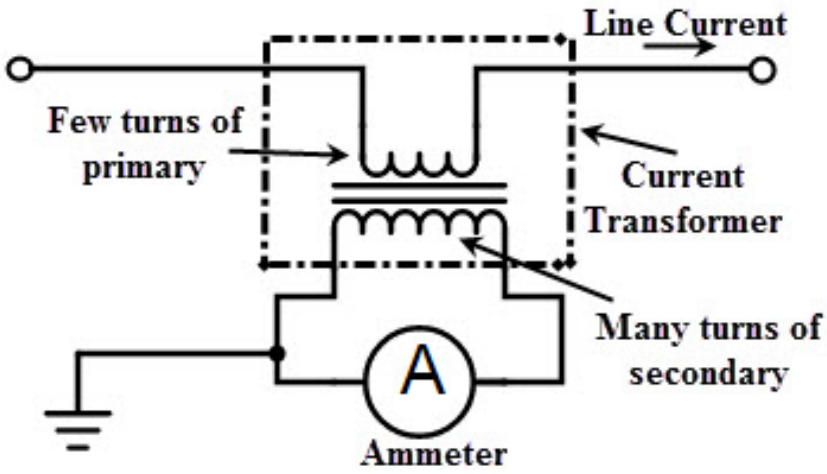



Figure 8: The current transformer measurement circuit.[12]

*Hall effect sensor*

The Hall effect sensor is a contactless magnetic field sensor. It consists of nothing more than a material with high electron mobility (e.g. semiconductors GaAs, InAs or graphene) and four leads. The measured current is in close vicinty of the sensor. A constant current flows through the semiconductor probe from one end to the other. The measured current induces a magnetic field which affects the electrons of the constant flowing current of the probe and deviates their path thanks to magnetic force. This results in a higher concentration of electrons on one side of the semiconductor and therefore creates a measureable difference in electric potential across the semiconductor (according to Figure 9). B(T) represents the magnetic field of the measured current, I (A) is the current of the measurement circuit and Fm (N) is the magnetic force which deviates the trajectory of electrons.

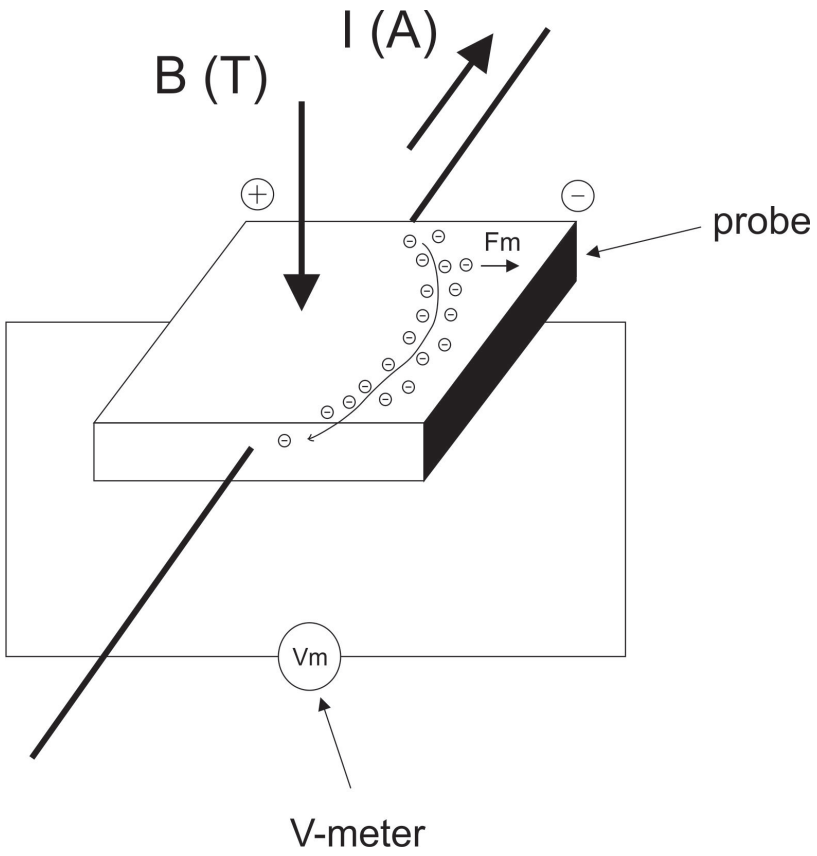



Figure 9: The Hall effect probe measurement circuit.

The output signal is lead to a transducer which converts it to a signal of parameters more suitable for measurement. The whole circuit is integrated into one device with just 3 wires: $+V_{cc}$, $-V_{cc}$ and M. It generally acts as a current-to-voltage converter, but with galvanic separation and a wider range of utilization. The output signal is a voltage signal and when measured across a defined resistor, we can easily calculate the electric current value. The measurement circuit is shown in Figure 15

The Hall effect sensor is superior thanks to its universality, it can be used to measure any type of current signal with very little signal distortion. Its only drawback is that it has a significant drift, so the circuitry requires compensation.

## CONTROL METHOD

In this section the control method for energy distribution will be covered. The actual diagram and algorithm will be covered in Chapter 3, this section will provide information about the control theory used.

Upon inspecting the heating system of a residential house, it is clear that the system consists mainly of components with long time constants. Heat exchange time constants range from units of seconds upto tens of minutes, so there are no great demands in responsiveness and precise timing. Incorporating fast PID regulators would therefore be unnecessary and even very difficult. Systems with single input and single output are easily tuned, however it is difficult, sometimes even impossible, to tune a PID regulator with multiple inputs and multiple outputs. Rather than fast response, we would be looking for a control method which would include predictive behaviour, as well as reaction to predicted stochastic events and accounting for as many inputs as possible.

There is need of feedback which would report the current temperatures and electrical parameters, so a closed-loop regulator would be suitable. It would be possible to use an open-loop control schematic based on the mathematical model of the house, however it is a difficult task to cover all possible disruptions that could cause faulty behaviour and instability of the system, so this is not a viable option. However, years of practice in industry [4] have proven that model-based control of systems with feedback and well covered disruptions can give very good results. Model-predictive control is an ideal method for this project, thanks to its capability to handle MIMO systems and thanks to its predictive behaviour (which could take advantage of weather forecasts). The next subsection will present the Model-predictive control method.

*Model-Predictive Control*

Model-Predictive Control is a control method which was invented in the late 1970s in chemical and oil industrial facilities. It was incorporated into the refining process with great success. Automated refining plants were well known and identified systems, so the dynamic model of the process was obtained which enabled the expansion of this method.

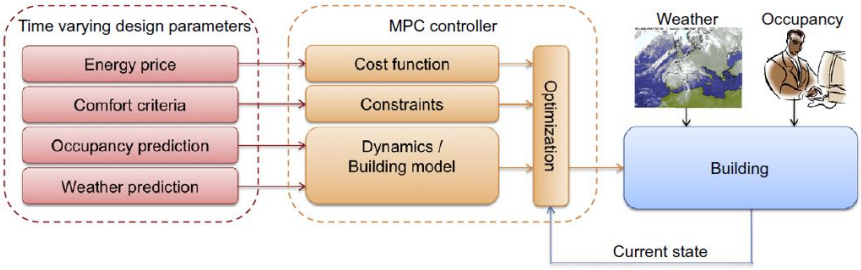



Figure 10: Basic diagram of MPC controlled system.[4]

MPC is a method for constrained control of the system. It is based on the dynamic model of the system. MPC uses the dynamic state of the system, MPC models, target values, limits, process variables and current measurements to calculate the future changes in dependent variables. It uses a minimalized cost-function to predict the future changes in a finite-time horizon.

The MPC controller calculates the future values according to the model and defines the next steps in the time horizon. The output change is optimized for every step. Theoretically the controller could calculate the control steps for a finite-time horizon, however this would not take into account the stochastic distruptions that could destabilize the control system. For example, if the controller would calculate the output for the next 2 hours and set the heating on and a disruption would occur (e.g. a person opens the window), the controller would't be able to react to this disruption (and a lot of energy would be wasted). Therefore a precaution is made: there is feedback measurement of many parameters and the MPC controller calculates

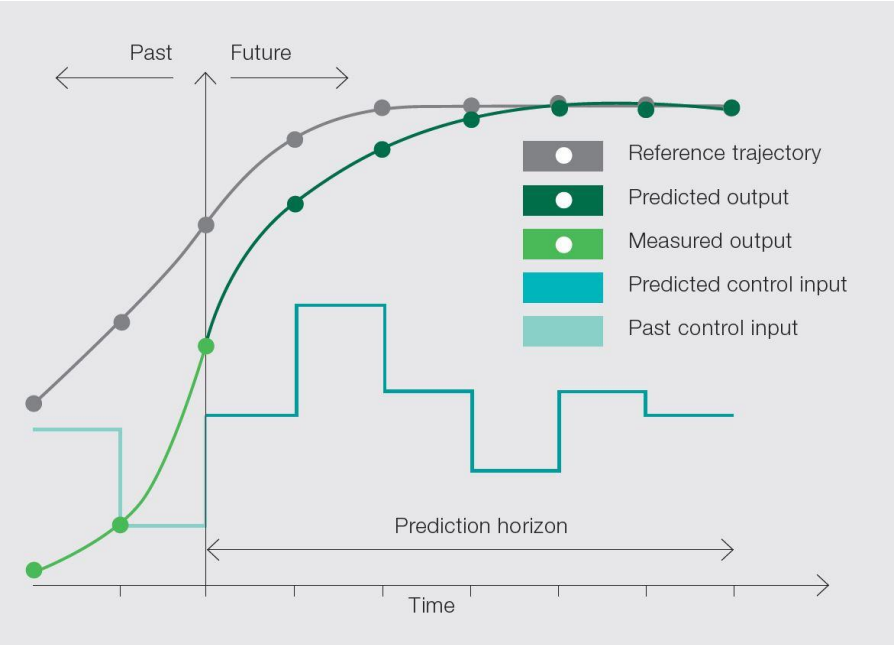



Figure 11: Visualization of a MPC problem.[1]

the future steps iteratively. This means that the calculation takes place with every step and in every step, the control output is optimized and taken in account. This way, if the person would open the window, the controller could get a signal either from a window-state switch or from temperature sensors (measurement feedback), that there is a disturbance and the controller could react accordingly to the change in state. For this reason, this algorithm is referred to as receding horizon control.

MPC has been designed for industrial applications in chemical facilities where the system is somewhat simple and predictable with little distrubance, however in the past years it has been involved in HVAC and heating systems thanks to advances in computation technology (receding horizon control needs high computational performance, because it optimizes and calculates the output for every time interval, "online") and in mathematical modelling. Its main advantage is that it enables engineers to control MIMO systems with predictability.

# 2

# HARDWARE

This chapter is dedicated to the PCB design with all the components. The control PCB is one of the key features in the system of a photovoltaic power plant on a residential house. It has these crucial responsibilities: data acquisition, data-logging, room temperature control and power-flow routing. The system has to be robust and resistant to failure, otherwise the consequences could be severe. The following sections shall present the design and components.

## CONTROLLER

The main control component is the controller. Its purpose is to drive all the peripherals, acquire and store data, synchronize all components, communicate with other devices and execute necessary calculations.

The controller consists of a microprocessor and IO peripherals. For the purpose of this application a special controller module (System on Module) has been chosen as the controller – the G120E System on Module created by GHI Electronics. It is a low cost SMDs processor module ideal for rapid prototyping of embedded applications. It runs on the .NET Micro Framework technology, which is compatible with the programming languages C# or Visual Basic.

It has a very wide array of connectivity possibilities, many General Purpose Input Output (GPIO) pins and a fair computational capacity. The key features are listed in Table 1. This module has been chosen because of its versatility and low cost, comfort of programming tools (C# in Visual Studio) and because of positive previous experience with this platform.


Figure 12: The G120E control board from GHI Electronics.[6]

The model of the G120E had to be created in Cadsoft Eagle as it couldn't be found in any official Eagle library. The following Figure 13 shows the schematic of the G120E module.

The pins on the left side include CAN, Analog Inputs, $I^2C$ and PWM outputs. The right side pins consist of the SD/MMC card interface, SPI bus, UART, Ethernet and PWM outputs plus 8 GPIO pins are programmed to control the switching relays. The top pins are used as PWM outputs for controlling of an electronic switching device. The purpose of the bottom pins is to drive a LCD monitor, but they are unused, since there is no LCD used in this project.

The processor module is powered by a 3.3V input, which drains power from the power supply. The module is connected to the ground plane at a few points. There is a 10kΩ resistor connecting the pin IO71/MODE to the ground, which defines the communication interface with the host computer (this means USB in our case) on startup. After startup, the pin gets reconfigured as a standard GPIO pin.

| | |
|---|---|
| Processor | 120MHz 32 – bit ARM Cortex-M |
| System Platform | .NET Micro Framework |
| User Available Flash | 2.87MB |
| User Available RAM | 13.67MB |
| GPIO | 80 |
| PWM | 12 |
| Analog Input | 7 |
| Analog Output | 1 |
| UART | 5 |
| SPI | 2 |
| $I^2C$ | Yes |
| Networking | Ethernet TCP/IP, WiFi, and SSL |
| CAN | 2 |
| $1-Wire$ | Yes |
| Real-Time Clock | Yes |
| Memory Cards | Yes |
| File System | FAT16/FAT32 |

Table 1: G120E features.[6]

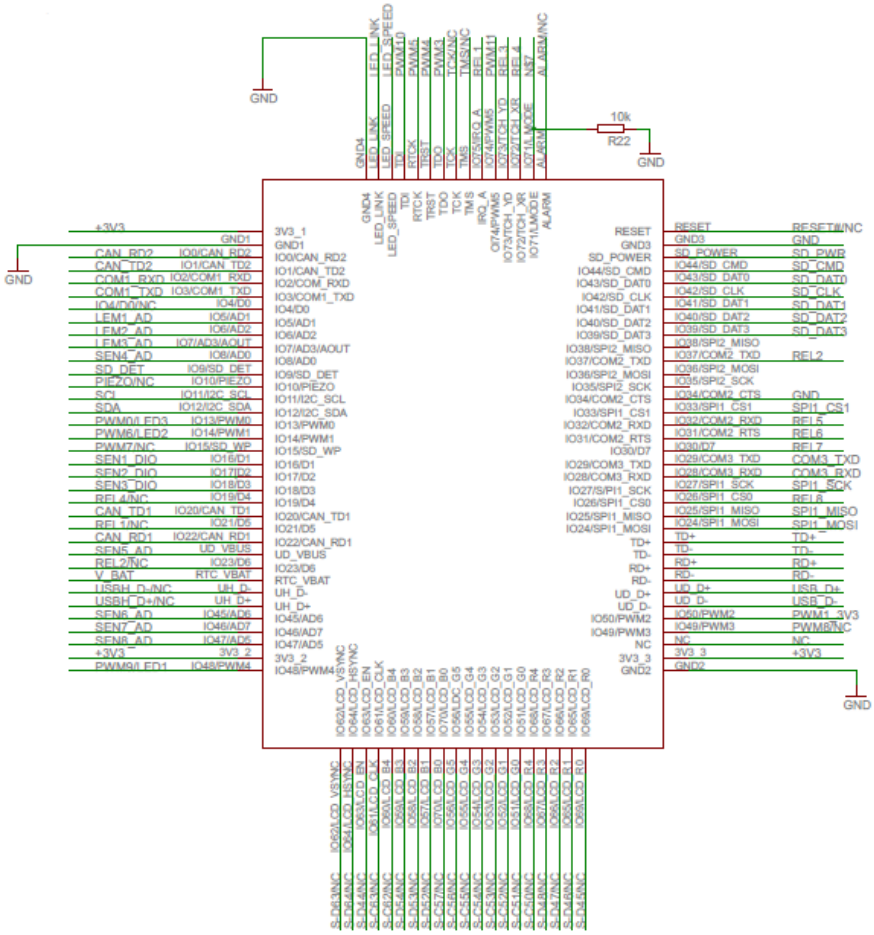



Figure 13: Schematic and pinout of the G120E module.

## POWER SUPPLY

A power supply has been designed to meet the needs of the hardware. The board is powered from a source of 12V via a connector. The 12 V are reduced to 5 V and 3.3 V, so there are 3 voltage levels.

The 12 V voltage level is used for powering of the DC/DC converter. This converter powers the current sensors. The 5V lines power the $One-Wire$ sensors, the USB connector, $RS-485$ bus and the CAN controller interface. As for the 3.3V supply, its main purpose is to power the G120E module and the Ethernet connector.

For stabilization and down-verting purposes, there are two ICs, LM2574N $-$ 5.0 which handles the 5V power line and the LM2574N $-$ 3.3 which does the same for the 3.3V line. Their outputs are filtered

and protected by the output inductors, capacitors and the anti-parallel Schottky diodes.

### DATA ACQUISITION

One of the important duties of the control board is to acquire data from sensors as input into the model of the system. This application has several sensor inputs:

- 8x One-Wire sensors (Analog Inputs)
- 3x LEM current sensors (Analog inputs)
- 8x relay states (GPIO)

The One-Wire sensors are used for temperature measurement at several places in the house. The One-Wire bus is a very simple, low data transfer speed, low power and low cost bus, which uses just 2 wires: ground and data. The sensors have capacitors which power the sensor while data is being transferred. Thanks to their characteristics, they are well suited for usage as thermometers, which don't require complex control and timing.

The relay state is defined by the control signal which opens the control transistor. As a matter of fact, there are no sensors, but states of the control relays are important for the model of the system.

Last, there are LEM sensors, which are used for measurement of current drawn by the house. These sensors are based on contactless measurement method with Hall Effect. The current flow through the mains induces a magnetic field which affects the flow of electrons inside the Hall sensor. The measured value is proportionate to the magnetic field of the current conductor. The LEM sensors are not integrated into the PCB, only the $\pm$15V power supply via the DC/DC converter is implemented and the input coming from these sensors.

The DC/DC voltage converter is powered with 12 V over a 47µF input capacitor. The output pins are connected in parallel (+ with

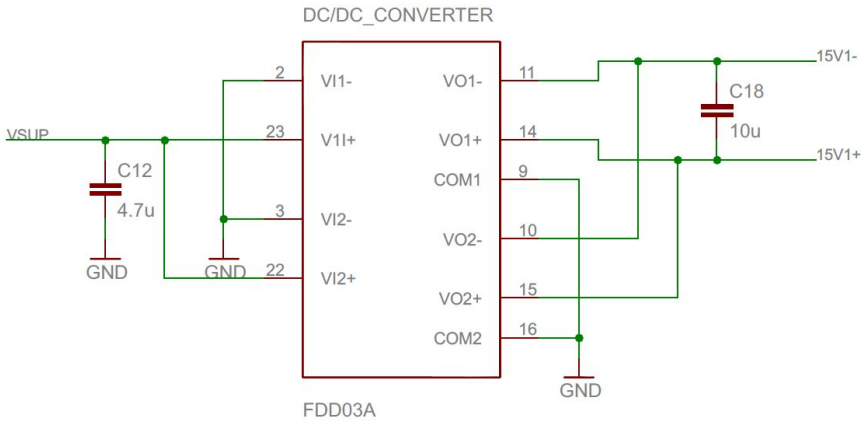
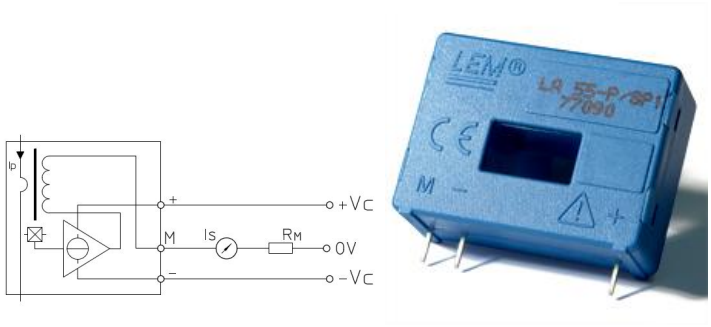



Figure 14: DC/DC converter schematic.




(a) (b)

Figure 15: The LEM sensor measuring circuit (a) and photograph (b).

+, - with -), the total output is ±15 V, which is filtered with a 10µF capacitor. The max power of the converter is 3 W.

The output voltage of ±15 V from the DC/DC converter is connected to the $+V_c$ and $-V_c$ leads to the LEM sensor. The M lead is the data line, which measures the actual current in the conductor, which is lead through the hole in the LEM sensor.

### DATA LOGGING

Now, that the data acquisition has been introduced, it is time to mention data-logging. All the measured data is being measured every 10 seconds and is saved into the memory of the G120E. At defined time instants, the data should be saved to some permanent media. In this

project, there are 2 methods of logging used: logging to SD card and to an internet server via Ethernet connection.

The SD card is used with a standardized slot. It is used for storage of the measured data. Its benefit is that it is portable and available whenever the user needs it; it is removable. However, it has a limited number of read/write cycles, so it is necessary to minimize the number of R/W actions (e.g. store the data only once a day from a buffer).

The other means of data storage is logging data to a server via Ethernet connection. The G120E module supports Ethernet connection on its own; no other device is required, so all that is needed is just an Ethernet connector. In practice, the data is sent to the server at the same time as it is stored to the SD card from the buffer.

### COMMUNICATION INTERFACE

There are several communication buses implemented in this design, it is necessary to communicate with several peripherals and devices in the system, this part shall be dedicated to the description of the means of communication.

Following is a list of used communication buses:

- RS − 485 bus
- CAN
- USB
- Ethernet

The Ethernet connection has been presented in the previous section; it is therefore omitted out of this section. Probably the most important bus in this project is the UART driven RS − 485 bus. Its purpose is to communicate with the inverter of the photovoltaic system, to control its output power and to collect data from the inverter. It consists

of an UART input from the G120E controller, a 74LVC1G57 configurable multiple-function gate and a MAX13487 half-duplex RS − 485 transceiver with auto direction control. Altogether, these components form the RS − 485 module. Basically it converts COM port signal to the RS − 485 compatible signal and opposite in the opposite direction. The RS − 485 bus is needed because it is compatible with the inverter.

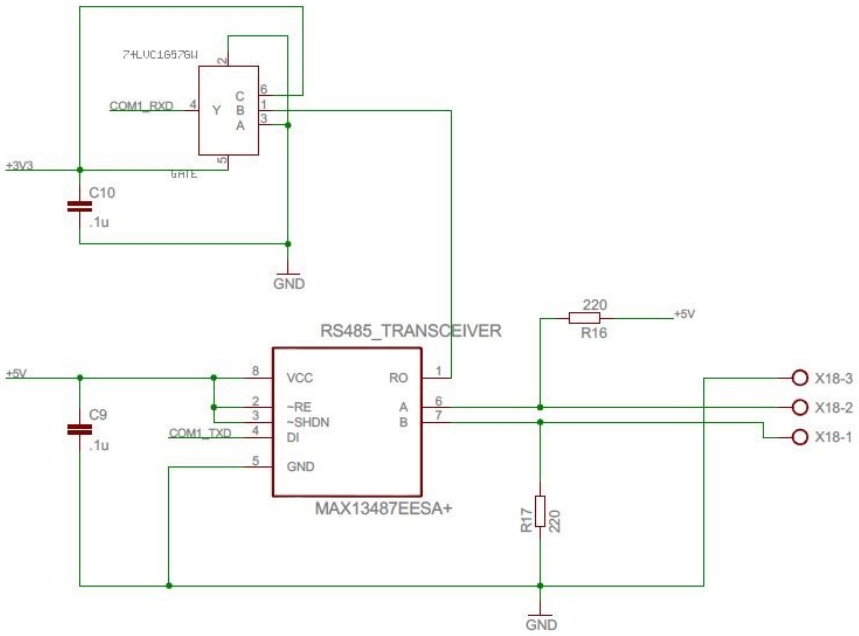



Figure 16: The RS − 485 module schematic.

Next is the CAN bus. It is used for communication with the rectifier controlling the power to the accumulation tank. The G120E is compatible with the CAN bus, a CAN controller interface has been added to the control board. It consists of a PCA82C250 IC and a parallel capacitor on the output side of the interface. It works as an interface between the CAN controller and the physical bus.

The USB is here for communication with the host computer, so its main purpose is programming of the G120E module. It is compatible with the micro-USB cable. It is selected as the active programming interface of the G120E with the signal MODE, which is connected to the ground via a 10kΩ resistor.

Last, there are 8 switching relays. Their purpose is to control the power flow in the house, some relays control the hot water gauges, some relays work as a switch between multiple power sinks. The control circuit (relay coil and transistor) is powered from the 12V source, while the switched circuit is connected to the 230V power grid in the house.

Figure 17: The switching relay.

## PCB LAYOUT

The final PCB layout has been implemented as in Figure 18. The layout was designed according to a set of rules for PCB design; the components have been placed into functional blocks for easier analysis of functionality, debugging and EMC compatibility.[11] For grounding purposes, copper pour technology has been implemented, as well as the solder mask. No silk screen layer has been applied.

The control board has 2 layers, the top layer traces and components are shown red, the bottom traces and components are blue. The top layer holds most of the bigger components, as well as the microprocessor module, the bottom layer holds mainly SMDs and the toroids.

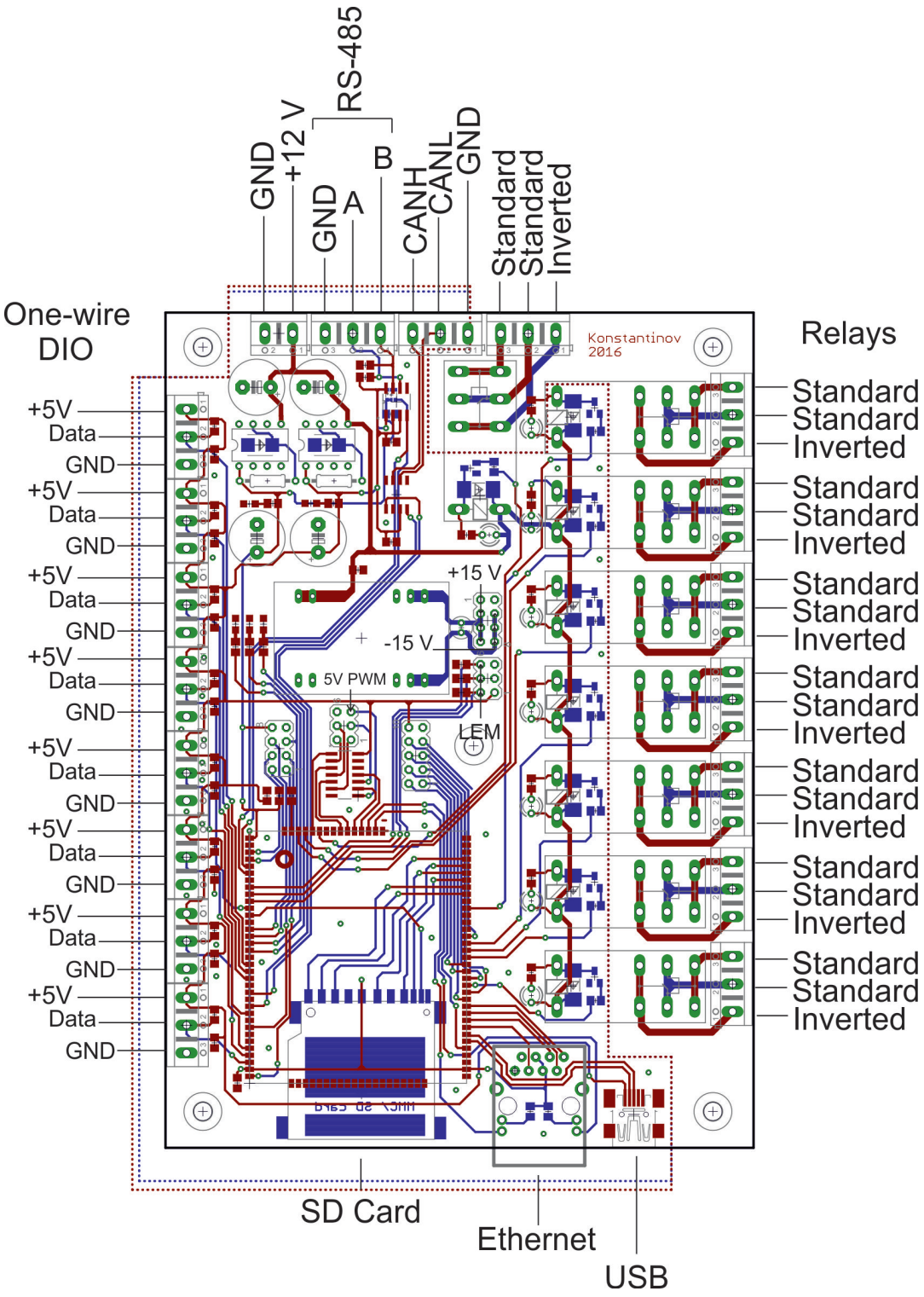



Figure 18: The PCB layout.

# 3

# SOFTWARE

Chapter 3 will cover the software part of this project. All the key features of the control and measurement program will be presented here, as well as some aspects of the usage of the .NET Micro Framework.

## SOFTWARE PLATFORM

As has been said in Chapter 2, the microprocessor module G120E runs the platform .NET Micro Framework, which is a module based on the full Microsoft .NET platform, but is adjusted to the needs of embedded systems. It is compatible with additional libraries enabling usage of various hardware components, but some features of the full .NET have been removed either due to irrelevance or to the limited resources of the embedded systems. There are several programming languages available to use (C#, Visual Basic, etc.). The C# programming language has been chosen for this project.

C# is a high level programming language based on C++ and Java. It has been developed along with the Microsoft .NET Framework. It is a simple and robust programming language mainly used for applications in networking and portable devices. C# enables full exploitation of OOP, which makes it a modern and effective language.[5]

### *.NET Micro Framework*

The NETMF platform has been optimalized for embedded systems and expanded with a number of libraries for usage with many hardware devices. It is designed to be compatible with C# programming

language for applications with constrained resources in embedded systems. It has a very large developer base, resulting with a community of developers where significant knowledge exchange takes place, therefore making it easy for beginners to start. NETMF is ideal for Rapid Prototyping, enabling developers to create, debug and deploy their applications in the matter of weeks.[6]

The main advantage of this framework, in comparison with other microprocessor platforms, is that it is basically an operating system which manages the application, assemblies and resources automatically, making it very simple to utilize various hardware-specific libraries. It has an automatic Garbage Collector, which takes care of unused resources and objects. The only drawback is difficult or almost impossible precise timing, since the system is not Real-time, making it unsuitable for applications with time-critical algorithms and precise timing (e.g. fast regulators, precise measurement, etc.). There is a way to overcome certain problems with timing by utilization of Runtime-Loadable Procedures, which have faster execution times, but it wasn't incorporated into this project.

*Microsoft Visual Studio*

The most suitable IDE is the Microsoft Visual Studio, which was designed to be compatible with .NET Framework. It is a user-friendly environment and enables the user to fully customize it in accordance to the users needs.

I have used the version Microsoft Visual Studio 2013 Community edition for this project, because it was available for free and its features fully satisfied the needs of the software part of this project. It has all the functions that a modern and effective IDE should have, e.g. debugger or emulator.

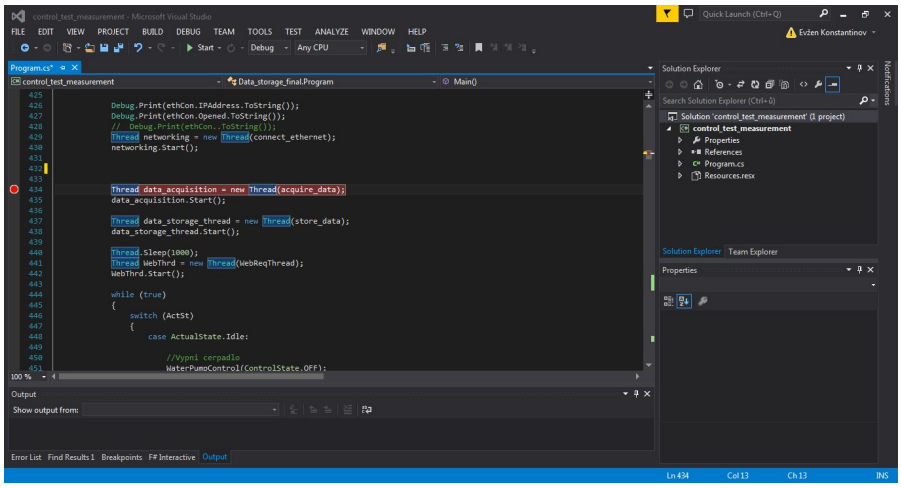



Figure 19: Screenshot of the Visual Studio IDE.

## PROGRAM

This section will be dedicated to the program which is flashed to the control PCB. It consists of several threads that are executed in parallel to each other and handle their own tasks. The following sub-sections will cover the main threads and functions.

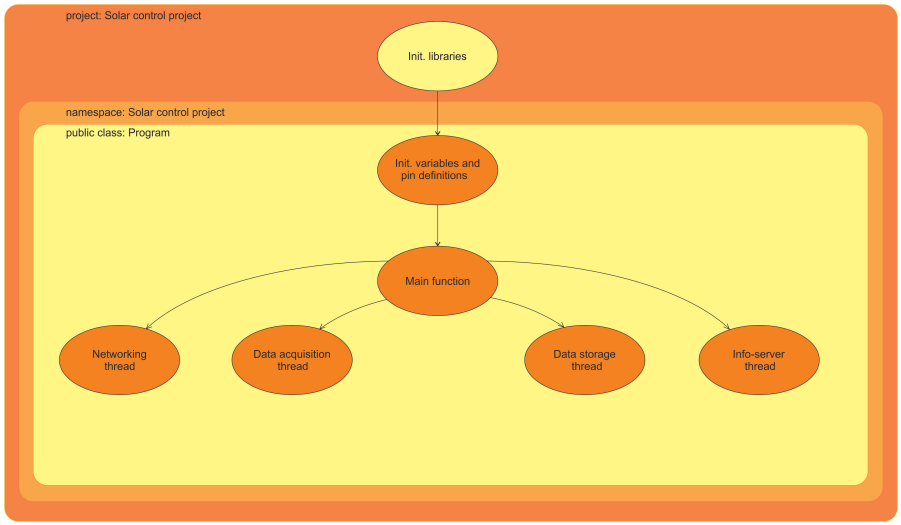



Figure 20: Flowchart of the program.

The main flowchart diagram of the upper-level of the program is visible in Figure 20. The program starts by initializing the *using* directives, which are similar to the plain C *define* statements. By initializing

the *using* directives, Visual Studio adds references to the used libraries to enable the usage of the various objects and methods included in those libraries. It is however necessary to add the references to the project in the Solution explorer (window to the right in Visual Studio).

Next comes initialization of global variables, pin definitions and hardware peripherals setup. This part includes:

- Global variables for the control model
- Global variables for data sharing among threads
- Digital Input/Output Ports
- Analog Channels
- $RS-485$ communication bus
- Enums for the control model

The $RS-485$ bus is setup as a standard *COM1* serial bus (UART) since it is transformed into $RS-485$ via hardware. The baudrate is set to the industrial standard of 19200 bps.

*Main method and Threading*

The method *MAIN* has 2 purposes: firstly it defines and starts other parallel threads and secondly it runs the control model of the heating system of the house.

At the beginning of the algorithm, Ethernet connection events and the *EthernetBuiltIn* object are defined. These events get fired every time the Ethernet cable is inserted or ejected from the port. The object grants access to the connection properties (e.g. IPAddress) and methods (e.g. *EnableDhcp()*). Next, the One-Wire sensor objects are defined and assigned to DIO pins.

Following are definitions of the threads. There are 4 threads running in parallel with the *MAIN* thread:

- Networking
- Data acquisition
- Data storage
- Info-server

These threads shall be described in the next subsections. After defining and starting the threads, the program enters the infinite while-loop. This loop is timed by 200 ms. The purpose of this loop is to drive the actuators of the system based on the knowledge of the system's parameters and variables (e.g. temperature, power, ...). The actuators are represented by DIO pins which drive electromagnetic relays on the control PCB. The relays drive contactors of the 230V mains. The model needs the following inputs: temperatures of the boiler, the accumulation tank and the pellet boiler and the power of the PV system. Based on these parameters, the state-machine enters predefined control states of the system. Figure 19 shows the diagram of the state-machine.

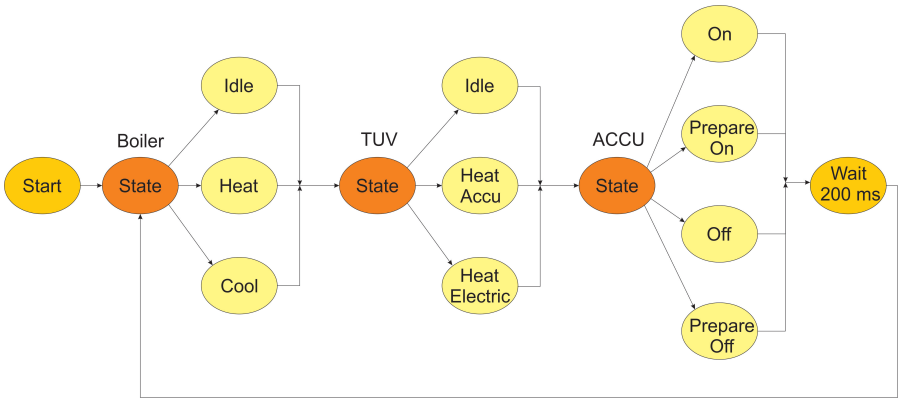



Figure 21: State-machine of the model control.

This state-machine evaluates current states of the Boiler, the TUV and the Accumulation tank. The boiler state is determined and controlled by the state of the thermostat switch which indicates whether it is necessary to heat or not. It can enter 3 states: Idle, Heat (the boiler is being heated electrically and water is pumped through it

to distribute the hot water) and Cool (the heater is switched off and the water is pumped throughout the house until it cools down below a temperature limit). The state-diagram of the boiler is visible in Figure 22.

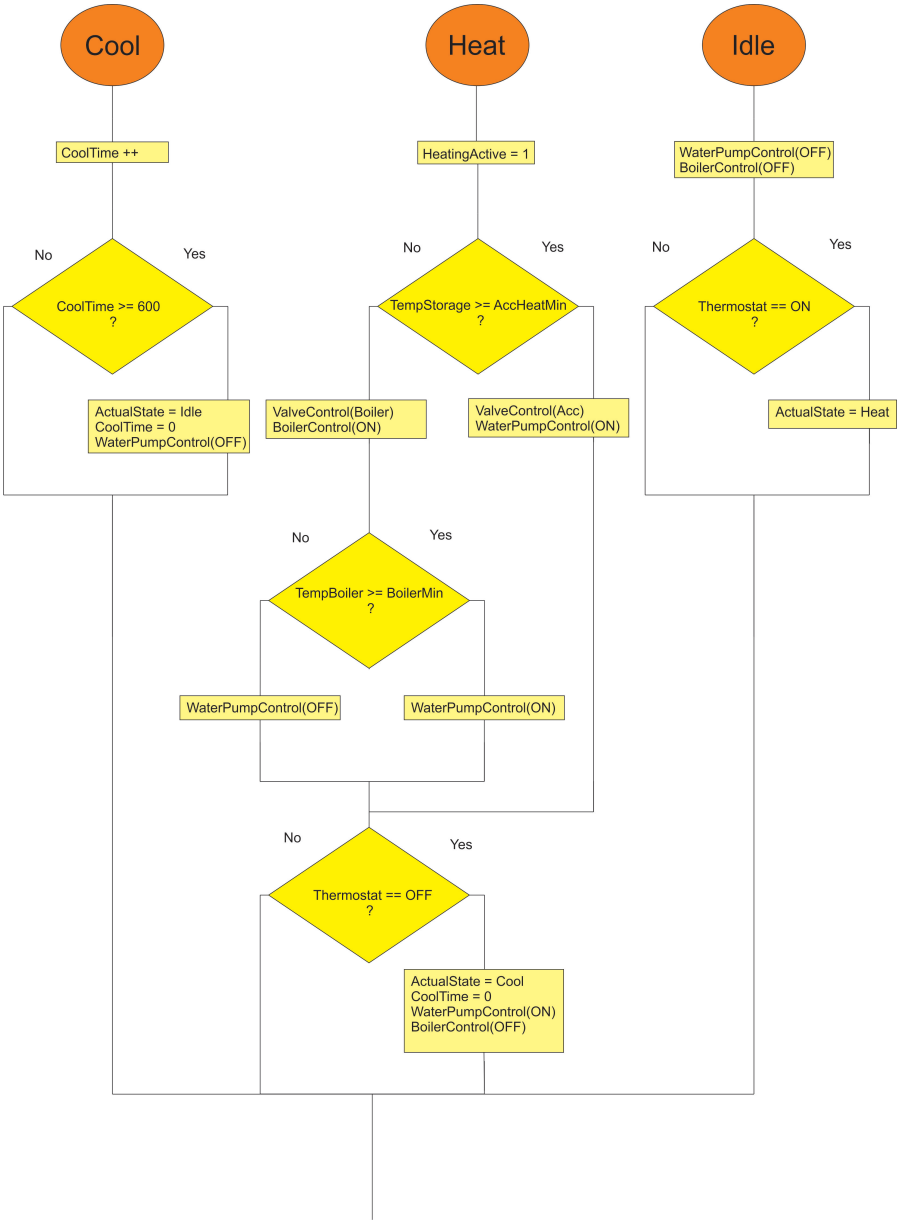



Figure 22: State-machine of the boiler control algorithm.

The algorithm begins by setting the initial state to *IDLE*. The water pump and boiler switch is turned off and the state-machine waits for the thermostat to signal the change of state -> when the thermostat triggers the switch *ON*, the machine transitions to the state *HEAT*.

This state says that the boiler should start heating. Until the storage temperature is higher than the minimal temperature of the accumulation tank, the valve of the boiler is set to control the boiler and boiler control is switched *ON*. When the temperature rises over the minimal boiler temperature, the waterpump is switched on to distribute the heat into the house. When the thermostat decides it is time to stop heating, the state-machine stops heating the boiler and switches on the water pump and transitions to the state *COOL*, in which it waits for a predefined amount of time with the water pump pumping water in the circuit, then switches the pump off and transitions to *IDLE*.

The algorithm executes certain function calls, for example *ValveControl(ON)* which is a function controlling the actuation of a regulation valve. For illustration purposes, imported code in Listing 2 is presented. The rest of the coded functions and state diagrams are placed in the Appendix.

Listing 2: Code: Valve control function

```
1  public static void ValveControl(ValveState ValveControl)
2          {
3              //Function, which sets the valve according to input
4              if (ValveControl == ValveState.Acc)
5              {
6                  //Selected accumulation tank
7                  ValveONPin.Write(true);
8                  ValveDirectPin.Write(true);
9                  VlvSt = ValveState.Acc;
10             }
11             else
12             {
13                 //Selected boiler
14                 ValveONPin.Write(true);
15                 ValveDirectPin.Write(false);
16                 VlvSt = ValveState.Boiler;
17             }
18         }
```

The algorithm of the TUV control starts in *IDLE* as well, with the switching relays of the TUVControl and electric heating of boiler switched off. The algorithm checks the temperatures, whether there is enough heat in the accumulation tank (then it transitions to the state *HEATACC*) or whether it is necessary to heat the water electrically (state *HEATELEC*). In the state *HEATACC* the algorithm switches on

the relay of the TUV to supply heat. If the system is heating as well, the valve needs to be set to control the accumulation tank. When the temperature of the water is high enough, the system returns to *IDLE*. The branch of the state *HEATELEC* is similar, except the actuators are different and if the temperature in the storage tank is higher, the system transitions to the *HEATACC* state to save energy (less electrical heating is needed). This algorithm's diagram is available in the Appendix in Figure 32.

The last part of the state-machine watches over the accumulation tank, which uses energy from the PV system to heat the water and store it. It consists of 4 states: *ON*, *OFF*, *PrepareON* and *PrepareOFF*. It begins in the state *OFF* and waits until the power of the PV system is higher than a certain constant value. Then it switches to the state *PrepareON*, which functions as a sort of delay before starting to heat the water in the tank. In the state *ON*, the relay controlling the contacter of the electric heating switches the circuit so that the current from the PV system supplies energy for heating. When the power of the PV falls below a certain value and stays under it (state *PrepareOFF*), the state changes to *OFF*. This diagram is also available in the Appendix in Figure 33.

These 3 parts of the state-machine are in series one after each other and are timed by 200 ms, which gives time for the processor to process other routines, faster response isn't needed anyway. This is where the main thread ends. This thread should be running by all circumstances, because its purpose is to supply energy for heating and warm water in a residential house. Exceptions in this thread would result in discomfort and a need to reconfigure or reset the system.

*Data acquisition thread*

The Data acquisition thread has 2 purposes. Firstly it provides data for the state-machine which controls the heating system of the house,

so the algorithm should be robust and reliable. Secondly it is used for data logging for further analysis of the system.

There are 4 types of inputs that need registration and logging. As has been discussed above, there needs to be an algorithm for temperature measurement. Next there is the current measurement which should provide us with information on the actual power of the PV system. We also want to know the states of all actuators (relays) for possible debugging and analysis of behaviour. Last, the control state-machine needs to know the precise power generation, which is given by the inverter of the PV panels over the $RS-485$ bus.

The Data acquisition thread begins with an initialization sequence of variable declarations. There is a semaphore which pauses the thread and waits for the system time of the PCB to be initialized. This is necessary because measured data without proper timestamps would render the measurement useless. After the time is synchronized, the thread jumps into an infinite loop.

Every iteration of the loop starts with reading the actual time from the Real-time Clock object and saving it into a variable. There are also other time objects - next_ log, next_ log_ server and next_ sample of type *DateTime*. Their values are defined in the initialization part of this thread. These variables hold the times of the next log, next log to server and next sample. The algorithm then compares these values with the actual time and if any of the conditions are met, the procedure is triggered.

The first branching of the algorithm comes when the condition for next_ log_ server is met. The algorithm signals into a global variable that it is ready for logging to the server. The Data storage thread registers this and picks up the data from a global array. The data-logging period is 20 minutes.

Next is the code for logging to the SD card. There is a slight difference here, because it has been defined that server logs always come first, so when the server logging and SD logging take place at the same time, first comes the log to the server and next is the log to the

SD card. The code signals to the Data storage thread that it is ready to log and waits for a signal, that the Data storage has been finished. Then the data is erased from the memory to prevent memory leakage and congestion (inspiration from [3]). Data is logged to a SD card every 40 minutes.

Last is the branch of the program which does the actual measurement. It runs with a period of 10 seconds, so it is the fastest part. First the values of the relays are read, next is the temperature measurement and last is the electric current measurement. These algorithms are described in the following subsections.

After receiving all the necessary data the thread stores them into one string with the timestamp at the end and adds the string into the global data array which is used by the Section 3.2.3.

*Temperature measurement*

Temperature measurement is a key part of the program as it provides essential information about the state of the system and determines future development of the state-machine. There are 3 temperature signals: temperature of the water in the storage tank, in the boiler and in the TUV. These values are obtained from *One-Wire* sensors which are placed at the desired locations (see Figure 1).

*One-Wire* sensors have a predefined service sequence for reception of the measured values (for precise information see the attached CD with datasheets). Listing 3 shows the procedure needed for correct reception of the temperature value. It has been written in accordance with the example from GHI Electronics of *One-Wire* handling.

The *One-Wire* sensor is basically an analog sensor with AD conversion circuitry and a parallel capacitor, so from the view of the processor, it acts as an ADC. The procedure works according to the following sequence: if the sensor has been reset, writing the value *OxCC* addresses all sensors on the *One-Wire* bus, after that writing the value *OxBE* enables the master (the G120E module) to read the

scratchpad of the sensor module. The next lines obtain the values from the data registers of the scratchpad and convert it to the desired value in °C. The rest of the algorithm handles unpredicted behaviour and limits the registered values. This procedure is called for every sensor seperately with a period of 100 ms.

Listing 3: Code: Temperature measurement procedure

```
if (SensorBedTemp.TouchReset() > 0)
{
    SensorBedTemp.WriteByte(0xCC);
    SensorBedTemp.WriteByte(0x44);

    Thread.Sleep(900);
    if (SensorBedTemp.ReadByte() != 0)
    {
        SensorTempSts = true;
        SensorBedTemp.TouchReset();
        SensorBedTemp.WriteByte(0xCC);
        SensorBedTemp.WriteByte(0xBE);
        temperature = (byte)SensorBedTemp.ReadByte();
        temperature |= (ushort)(SensorBedTemp.ReadByte() << 8);
        SensorBedTemp.TouchReset();
        TempStorage = (float)temperature / 16f;
        if (TempStorage > 150)
        {
            TempStorage = 0;
                                }
    }
    else
    {
        SensorTempSts = false;
                        }
    if (SensorTempSts == false)
    {
        TempStorage = 0;
    }
}
else
{
    TempStorage = 0;
                }
```

*Electric current measurement*

For validation and informational purposes, electric current measurement has been implemented into the program. The measurement schematic is in Figure 23.

The measured current $I_P$ flows through the main wire, the transducer is powered by the power mains $+15$ V and $-15$ V and the Hall

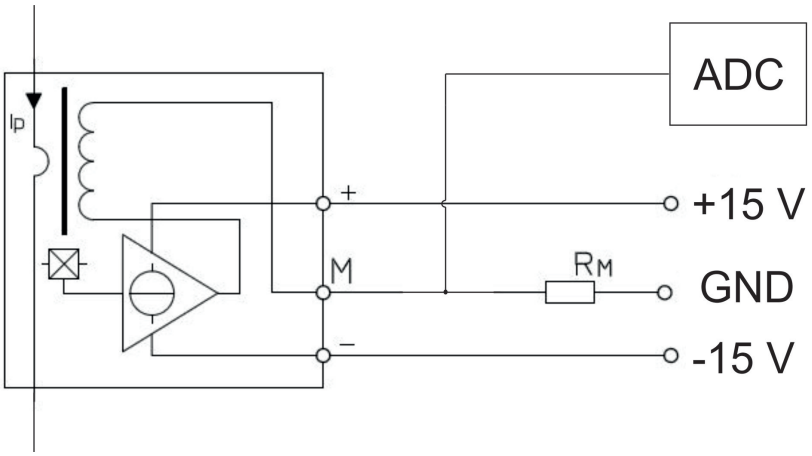



Figure 23: Schematic of measurement circuit.

sensor reacts to the magnetic field of the measured current. The transduced analog signal is then transferred to the ADC of the processor with a resistor connected in parallel. The resistor value is 68 $\omega$ and was chosen according to the datasheet of the LEM sensor. The sensor has a defined gain of $1 : 1000$, so after calculating the current value from the voltage across the resistor, it is 1000x smaller. The ADC measures voltage from 0 to 3.3 V with $10-bit$ precision and the digital input is a floating-point number from 0 to 1, so the signal needs to be scaled accordingly.

At this moment we know how to measure the values but we are facing the problem of measurement of an alternating current with a frequency of 50 Hz. Since usually alternating current is characterized by RMS, the algorithm calculates the RMS of the measured signal.

The algorithm begins by sampling of the measured signal with 1kHz frequency. The values are read from the analog channels $1, 2$ and 3 and are stored into arrays of 400 values (there are 3 arrays in total for 3 LEM sensors). For one period of the sine wave, we have 20 samples, which is OK for our purposes (Shannon-Kotelnik sampling theorem is satisfied). Since the ADC measures only positive values, the negative half-wave is equal to zero, so the obtained waveform looks like Figure 24.

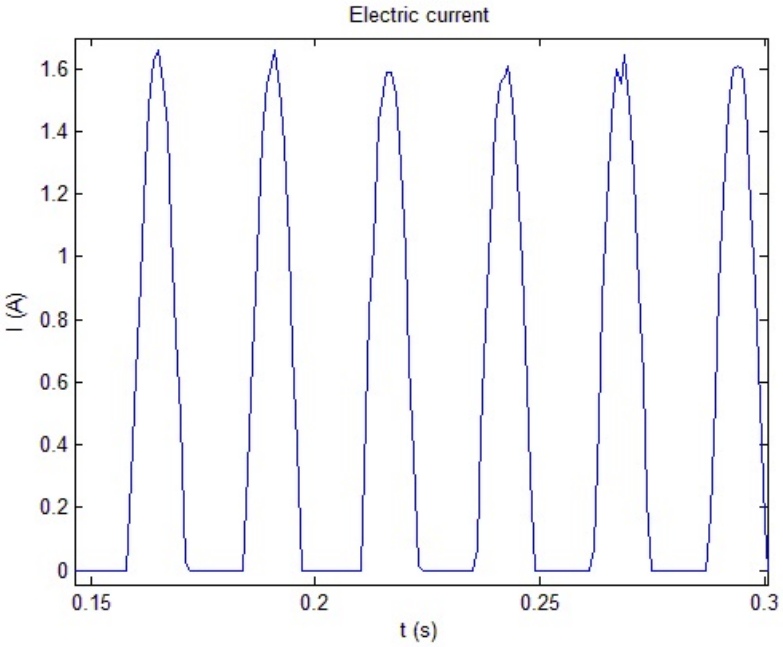



Figure 24: Graph of the typical measured waveform.

Since we don't know at what time instant of the waveform started the AD conversion, it is necessary to find the beginning of the first sine wave in the sampled signal as we want to calculate the RMS only from full half-waves. For this reason, the function *find_ zero()* is called. This function simply starts at the beginning of the array and finds the first zero-value. From there the algorithm finds the beginning of the first half-wave. Now it can start calculating the RMS, so the function *calculate_ RMS()* is called.

This function takes the measured data and calculates the RMS value of the signal. It first extracts 10 half-waves from the signal and then calculates the RMS of each half-wave. It calculates an average value of the 10 RMS values and returns it as an output parameter of the function. The RMS calculation itself consists of a numerical integration method of the square power of the signal and the square root

of the integral divided by the period of the signal, as in the following formulas.

$$I_{RMS} = \sqrt{\frac{1}{T}\int_0^T i(t)^2\,dt} \tag{1}$$

$I_{RMS}$ is the RMS value of the current, $i(t)^2$ is the actual current and $T$ is the period of the signal. This is the formula for a time-continuous signal, however we need to discretize the integral for the computer to be able to compute it, so a numerical method of integral calculation has been applied, namely the *trapezoidal method*. This method approximates the signal by replacing of the continuous signal with discrete values and integrating of the small segments (in the shape of a trapezoid) and performing a sum of these integrals across one period of the signal, as in the equations below.

$$\int_a^b f(x)\,dx \approx \sum_{i=1}^{n} \frac{f(x_i) + f(x_{i-1})}{2} \tag{2}$$

$$\sum_{i=1}^{n} \frac{f(x_i) + f(x_{i-1})}{2} = \frac{b-a}{n}\left(\frac{f(x_0)}{2} + \sum_{i=1}^{n-1} f(x_i) + \frac{f(x_n)}{2}\right) \tag{3}$$

The discretization itself is performed by the ADC which samples the values. The algorithm simply calculates the areas of all the trapezoids of the square power of the signal, performs a square root of the value divided by the period and calculates the average value of the 10 RMS values. The numerical method is implemented in C# and the code is documented in the Appendix. Measured data is documented in Chapter 4. The RMS value can be used for calculation of actual power flowing from the PV system or from the grid, but this is out of the scope of this project.

The value of the RMS is then scaled in compliance with the gain of the transducer. The electric current is measured every 10 seconds like the temperatures.

*Actual PV Power calculation*

This function is called after the electric current measurement. Its purpose is to acquire the actual power generated by the PV system. It is mostly a communicational procedure which obtains the value from the inverter via RS − 485 bus.

The function initializes an input buffer for reception of the data from the inverter. It then sends the defined *ComLynx* message as an array of bytes. After a short delay, the algorithm reads from the RS − 485 and saves the data to the input buffer. If the received message has a sufficient length, it means the message has been received correctly and the algorithm can decode the message to obtain the actual PV system power.

Listing 4: Code: Actual power reading function

```
1       public static int ActualPower()
2       {
3           int result = 0;
4           byte[] input_buffer;
5           RS485.Write(output_buffer, 0, output_buffer.Length);
6           Thread.Sleep(100);
7           input_buffer = new byte[RS485.BytesToRead];
8
9           if (RS485.BytesToRead != 0)
10          {
11              try
12              {
13                  RS485.Read(input_buffer, 0, RS485.BytesToRead);
14              }
15              catch (Exception)
16              {
17                  if (error_line < 10000)
18                  {
19                      Error_array[error_line] = "RS-485 reading error. " + RealTimeClock.GetDateTime().
                            ToString();
20                      error_line++;
21                  }
22              }
23          }
24          if (input_buffer.Length > 19)
25          {
26              result = input_buffer[18] << 32 + input_buffer[17] << 16 + input_buffer[16] << 8 +
                        input_buffer[15];
27          }
28          else
29              result = 0;
30          return result;
31      }
```

The algorithm has been coded in compliance with the datasheet to the *ComLynx* communication protocol from the company *Danfoss*, the manufacturer of the inverter. The protocol is $RS-485$ compatible with the inverter. More formation on the *ComLynx* protocol is in Chapter 1. The inverter has predefined return messages and is able to provide many parameters like total energy generated throughout the day, data from the past, data about the irradiation of the PV system and many more. This project utilizes just the parameter of actual power.

*Data storage thread*

After obtaining and storing the data into the RAM, it is desirable to log the data to the SD card and to the server. This is the duty of the Data storage thread.

The Data storage thread is basically a scheduler which awaits a signal to call the storage functions. There are 2 global variables acting as semaphores, which indicate that the data is ready for storage in the global array. When a semaphore gets unlocked, the functions *write_ to_ SD()* or *write_ to_ server()* are called. According to Section 3.2.2 the data is logged to the server every 20 minutes and to the SD card every 40 minutes. After execution of the storage function, the semaphore is locked again to prevent repeated unwanted data storage. When the 2 semaphores get unlocked at the same time, the server log is executed first and after it comes the SD card log. After logging to the SD card, the array-line pointer is reset to 0 again and Section 3.2.2 erases the data from the global array. The next two subsections describe the data storage functions. Both have been inspired by the examples from the GHI Electronics support web-page.

*Write to server*

This function uses *HTTP* communication protocol to communicate with the data storage server. The server has a public IP address, so the data is available on the web address:

http : //147.32.200.115 : 8000/SolarMonitor/WS/data/1

It has a database for storage of desired data, which is accessible from any web-browser. The *HTTP* command *GET* enables us to obtain information about the server, time or data. The command *PUT*, which is used in this function, is used for saving the data to the server.

First, the function saves all the data into one string. Then, declaration and initialization of the *HTTP* communication objects is performed. Properties like *ReadWriteTimeout* or *Method* are set here, the values are visible in the code in the Appendix in Listing 8. What is most important is the addition of the authentication line to the header of the *HTTP* message, beacause the server is secured with *HTTPS* and requires credentials (username, password). The credentials need to be entered in an encoded mode in *base64* code. I have used an online *base64* converter. The credentials need to be written in a string in the following scheme: "username : password" => convertToBase64 => encodedCredentials. The username and password is stated in the Appendix in the documented code.

After initialization of the *HTTP* objects, a StreamWriter object is declared and linked to the *HTTP* object. Now, it is possible to call stream-related methods like *stream.Read()*, *stream.Close()* etc. The *stream.Write()* method is used for saving the data to the server (measured data is the input).

Now the data is stored to the server and it is necessary to clean up. The open stream needs to be closed and declared objects need to be disposed to prevent memory issues. The Garbage Collector should take care of this, but it is good practice to do so.

It is important for the storage to the server to be successful that the data is in the correct format. Below is an example of a correct data string:

```
2016-01-01 19:27:21
21.850000;62.510000;38.510000;34.210000;0;0;0;1;1;1;1;1;6.490000;80.780000;1;2015-10-29 01:00:01
95.640000;56.330000;84.210000;97.810000;1;0;1;1;1;1;0;0;88.390000;20.670000;0;2015-10-29 01:00:06
```

Figure 25: Example of data structure.

If this structure isn't respected, the log to the server will most likely fail. It shouldn't endanger the program flow, but the data will be available only on the SD card.

*Write to SD*

This function was inspired by the example code as well. It begins similarly to the *write_ to_ SD()* function, first the output string is created by appending of all the strings from the data array. Then an instance of object *SDCard* is declared and the card is mounted. To obtain the file-system of the card, it is necessary to get the directories and files from the root directory of the card, which is done after mounting. After that a FileStream object (which is linked to a specific directory on the SD card) and a byte[] array is declared. The data string is encoded in UTF8 into bytes and stored in the byte[] array. The method *FileStream.Write()* writes the byte[] array to the desired directory.

Again, the objects are disposed and the *FileStream* is closed. The SD card receives the data in the form of a .txt file with the data stored in text. The data is stored to the card less frequently than to the server because SD cards have a limited number of Read/Write cycles before they break down - approximately 100,000 cycles is the average lifetime of a SD card.

*Info-server thread*

For the purposes of verification and information gathering, a simple server has been implemented into the program. Upon entering the IP address of the server into a web browser running on a device which is connected to the same LAN network, a web-page appears, providing information about the system. The page is visible in Figure 26.

Actual state of heating: Heating ON

Acutal state of heating TUV: Heating ON

---

Temperature of accumulation: 45 ° C

Temperature of pellet boiler: 60 ° C

Temperature of boiler: 42.5 ° C

---

Boiler: Minimal temperature TUV: 50 ° C

Boiler: Maximal temperature TUV: 52 ° C

Boiler: Minimal temperature of accumulation at the beginning of TUV heating: 60 ° C

Boiler: Minimal temperature of accumulation while heating TUV: 30 ° C

---

Heating: Minimal temperature accumulation: 45 ° C

Heating: Temperature for heating of pellet boiler at the start: 55 ° C

---

PV: Actual power: 2235 W

PV: Heating of accumulation tank: Accumulation heating OFF

Figure 26: Available web-page providing information on the system.

The page provides information about key parts of the system: state of the heating system, state of the TUV tank, current temperatures in all 3 tanks, the current power of the PV system and all the limit values of temperatures.

The algorithm declares a *HTTPListener* object and calls the *start()* method. After that, it enters an infinite while-loop, which makes the server available continuously. In every iteration of the loop, the server gets the context of a response, creates the *HTML* web-page with actualized parameters of temperatures etc. and writes the web-page to the *OutputStream* of the response. Last comes the flushing and closing of the *OutputStream*. The loop is timed by 10 ms.

An improved version of this server with a graphical user interface is planned in the future, but out of the scope of this thesis. This version of the server works merely for parameter verification.

*Ethernet connection thread*

The last thread to be covered is he Ethernet connection thread. The control system is supposed to communicate with a server and log data to that server, therefore an Ethernet connection must be implemented. The PCB itself is equipped with a connector for this purpose. The G120E is able to establish such a connection, so the hardware is prepared.

At the beginning of the Main function (Section 3.2.1) an *EthernetBuiltIn* object is declared. This object handles configuration and connection to the network. Then, the networking thread is declared and started with the function *connect_ ethernet()* as input.

This function is an infinite while-loop which handles the Ethernet connection by testing of the IP address of the PCB. The algotrithm tests, whether an IP address was assigned to the device (the *DHCP* function is disabled, because there were problems with the *DHCP* server in the house). If the IP address is "0.0.0.0", then a predefined static address "192.168.0.150" is assigned and the communication is opened, which renders the Ethernet connection available for usage. If the IP address is already assigned, the algorithm checks if the communication is opened. If it is not, the method *Open()* is called and the

network is enabled. When the connection is established, the function intialize_ clock() is called to find out the state of the system time.

If time is incorrect, the method synchronizes the time with a time server. The procedure for obtaining of the time is similar to logging of the data to the server (Section 3.2.3.1). The algorithm communicates with the server via *HTTP* commands, here the *GET* command is used. First, the time objects are declared and the state of the Real-time clock is obtained. If it is incorrect (the time is compared to the last known time) the *HTTP* objects (*HTTPRequest*, *HTTPResponse*) are declared and the header is defined (again with the ceredentials mentioned in Section 3.2.3.1 and the method GET). Then a *Stream* object is declared and linked to the *HTTPResponse*. Now, it is possible to read data from the server, for which the method *ReadByte()* is used for reading 30 bytes from the time-server. The bytes are saved into an array and converted to a string. The string is the first line of the webpage of the time-server, which contains the actual time. The string is then parsed to extract the values of date and time. These values are then used to reset the Real-time clock to the new values. After time-initialization, the Data acquisition thread (Section 3.2.2) can start (it needs information on the actual time to function properly).

### GRAPHICAL USER INTERFACE

The last section of the software chapter will describe the graphical user interface. There are many ways to create GUI for data visualization, I have been choosing between Matlab and NI DiaDem, both of which have their pros and cons, but Matlab has been chosen in the end for it enables to deploy the program as a standalone application.

The GUI was created for the purpose of visualization and analysis of the measured data in the system. The application should have two key features: visualization of the data and the possibility to export the data as a report. The application was programmed as a main

function with *nested function-callbacks* to the GUI components. There is no need of continuous execution of any algorithms or calculations, so the architecture of nested function callbacks reacting to the user input from the IO components (buttons, sliders etc.) seems to be a valid one. In the figure Figure 27 an overview of the GUI is visible.

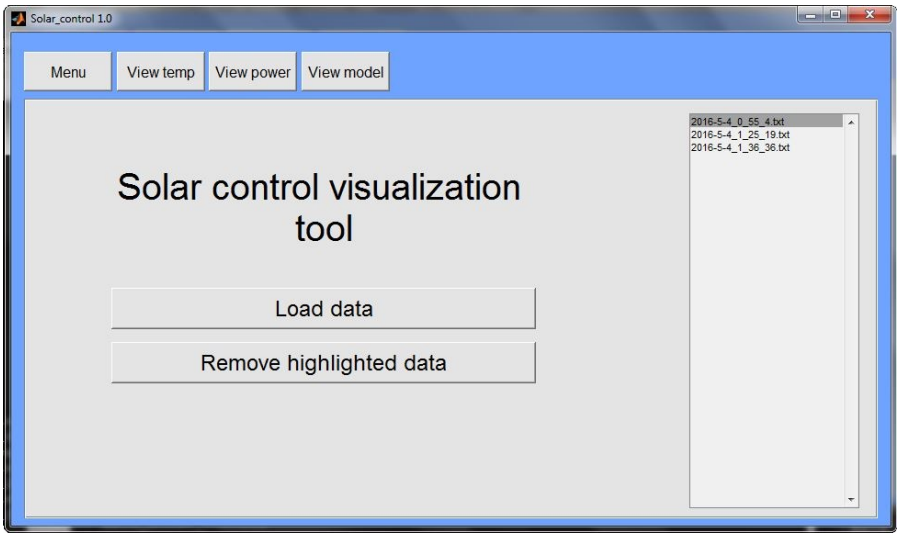



Figure 27: Matlab GUI.

The application starts on the *Menu* page. In the middle, there are buttons for operation with the data. The *Data portal* to the right shows the loaded data files. Above the main panel, there are selection buttons for visualization of different data (temperatures, power, electric current and model). Upon clicking on the button *View temp* the main panel is replaced with a graph which visualizes desired data selectable in the pop-up boxes at the bottom. The *Plot* button plots the chosen data in the range defined by the pop-up boxes.

The other select-buttons function the same way as the *View temp* button, except the visualized data is different. The *View model* button shows the schematic of the house and the parameter values are scrollable with a slider representing the time-axis.

The button *Report* creates a report of the desired data (graph and other data such as limit values and total power generated by the PV system). The report is saved as an image.

# 4

# CONCLUSION

## MEASURED DATA

The first section of this chapter will present the measured data. When the system was installed into the house a test variant of the connection was implemented. The PCB was connected to the inverter for power measurement, one temperature sensor was wired to it, one LEM current sensor was connected to measure current of one output phase of the inverter and one control relay was connected to a contacter. There are 3 measured parameters: temperature of TUV, electrical current and power fo the PV system. The data-logging was started 16.5.2016 and is continuously running.

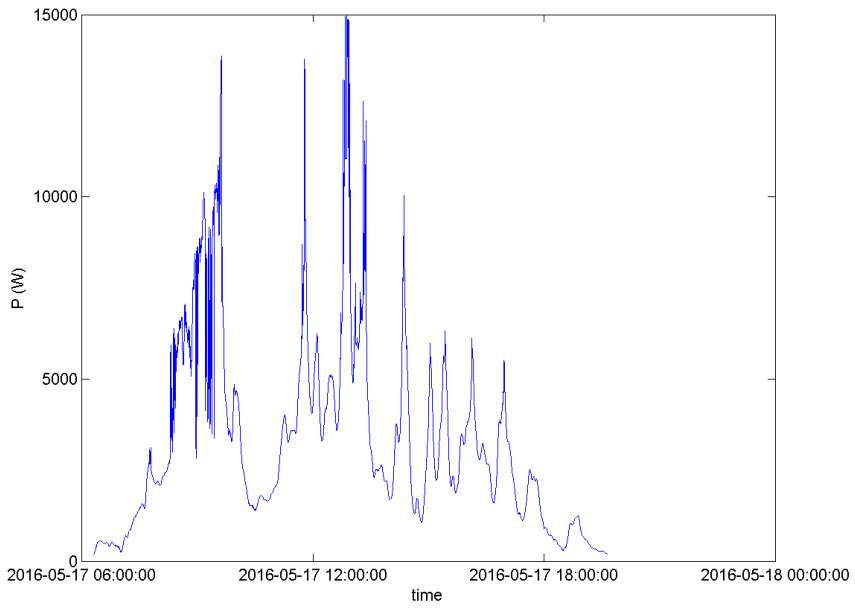



Figure 28: Data: Power of PV system 17.5.2016.

Following are figures depicting the power generation of the PV system. It is visible that there were different weather conditions each day resulting in different power generation.

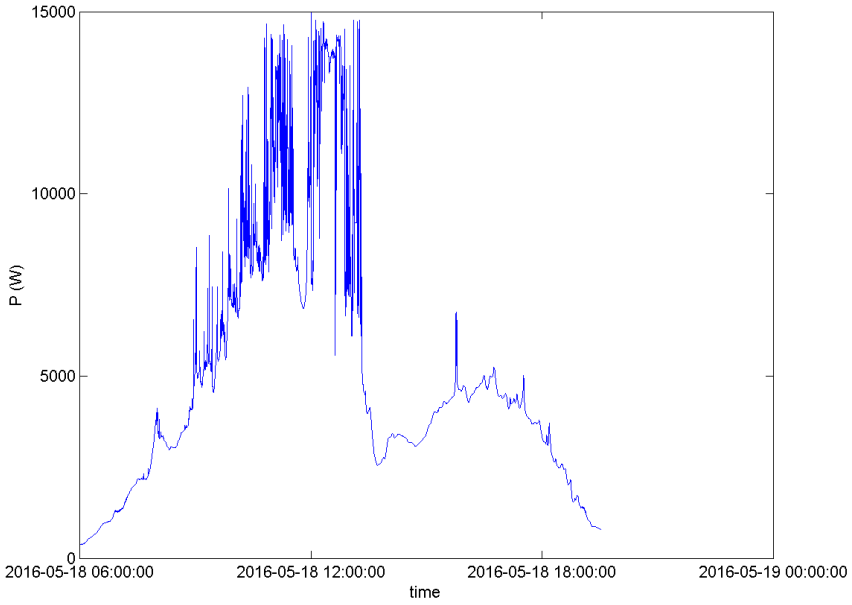



Figure 29: Data: Power of PV system 18.5.2016.

As for the total energy generated, the PV system generated 44.28 kWh on 17.5.2016, 68.26 kWh on 18.5.2016 and 86.54 kWh of energy - the values were measured by the inverter. Upon inspection of the graphs, it is clear that 17.5. it was cloudy, so the PV system generated the least power. On 18. 5. there was a lot of irradiation until noon and then the weather got worse, resulting in less total energy generated than the last day, 20.5., when the weather was sunny throughout the whole day.

The figure Figure 31 shows the electrical current drawn from the PV system. There are similar patterns in the curve of the elecrical current and in the curve of the power of the PV system that day, which shows that the measurement was valid.

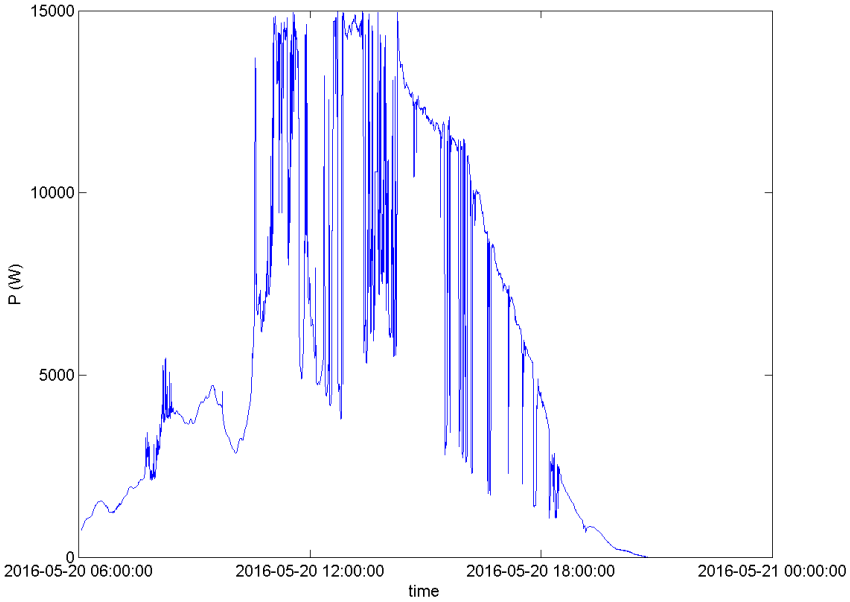



Figure 30: Data: Power of PV system 20.5.2016.

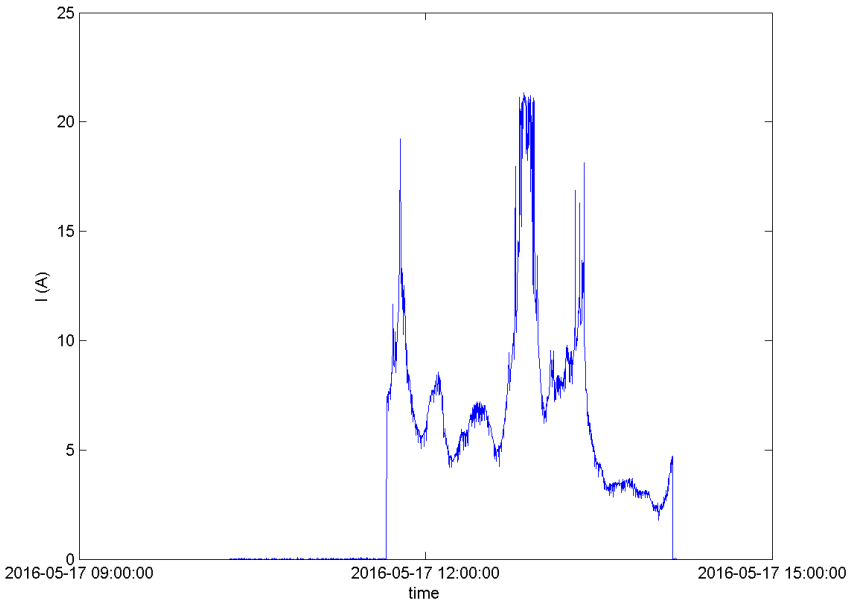



Figure 31: Data: Electrical current of PV system 17.5.2016.

## EVALUATION

The subject of the Diploma thesis is designing of the control system for the heating and PV system of an existing residential house for optimization of the energy balance of the system.

The first instruction was to design a PCB with the main control module G120E and the demanded peripherals (SD card, Ethernet connection, analog inputs for LEM current sensors, One-Wire temperature sensors and DIOs for electromagnetic relay operation). The board was designed in the program Eagle from Cadsoft and was fabricated by the company Pragoboard. I mounted the components on the board and tested the functionality, which was debugged and verified. The board is currently placed in the house and is logging data, so the functionality is proved.

The second instruction was to program the software of the data logging of the system (temperature, electric current, communication with the PV system). Temperature measurement was implemented flawlessly, the electric current measurement was implemented as in Chapter 3. There were some issues with noise in the current signal, but the final outcome was acceptable, the current RMS value was validated by a measurement with an oscilloscope. The communication with the PV system was implemented via $RS-485$ bus and functions as desired. Data logging to the SD card and to the server encountered difficulties with memory due to relatively big amount of data being handled at the end of each log interval, resulting in Garbage collector exceptions, but the problem was solved by reduction of the time interval between logs.

The third part was to design the control software to maximalize the usage of power generated by the PV system. This was implemented in the control state-machine by detecting the actual power of the PV system and electrical heating of the water in the accumulation tank whenever there was enough power.

The last instruction was to create a GUI for visualization of the measured data obtained from the log-files on the SD card. The GUI was created in Matlab and is described in Chapter 3. The files are on the CD added to the thesis.

The requirements were met and a functioning system was designed. Its main benefit is the low cost of the device and the level of integration of the system, which is integrated on a single board. The data logging and server communication was added to the existing system, as well as the current measurement and GUI.

For future development I would suggest implementing a web-based visualization tool for pseudo-real-time monitoring of the current state. There is also space for further optimization of the utilization of PV energy. An electrical accumulator could be used, since improvements in the application of batteries were made recently, enabling accumulation of energy for the usage in a residential house in cases of emergency or for optimization regarding the costs of electricity at certain times. There is also space for improvement of the control state-machine incorporating weather forecasts, which could be a difficult task in our temperature zone due to high unpredictability of the weather, but could lead to further optimization. Last, the electricity generated in the PV system could be used in the grid of the house, but it would need additional power electronics. The control board is equipped with several PWM outputs, so the expansion would be possible.

# A

# APPENDIX

The Appendix will be used for attaching additional figures and data concerning the thesis. Documented code and diagrams prevail. The last section is dedicated to important notes concerning the application of the hardware and software (e.g. IP addresses, credentials etc.).

## PROGRAM

Following are the state diagrams and algorithms implemented in C# code.

Listing 5: Code: State-machine of heating

```
                        // Heating state-machine
            switch (ActSt)
            {
                case ActualState.Idle:
                    //Switch off pump
                    WaterPumpControl(ControlState.OFF);
                    //Switch off boiler
                    BoilerControl(ControlState.OFF);
                    //If thermostat is true, start heating
                    if (ThmrSt == ControlState.ON)
                        ActSt = ActualState.Heat;
                    TopeniAktivni = false;
                    Message_Heat = "Topeni vypnuto";
                    break;
                case ActualState.Heat:
                    //If temperature is over 30 degC, switch on pump, switch off boiler
                    TopeniAktivni = true;
                    if (TempStorage >= ConstTempMinAccHeat)
                    {
                        //Set valve to Accumulation
                        ValveControl(ValveState.Acc);

                        //Start pump
                        WaterPumpControl(ControlState.ON);

                        Message_Heat = "Topeni z akumulace";
                    }
                    else
                    {
                        //Set valve to Boiler
                        ValveControl(ValveState.Boiler);
```

```
32                      //Start boiler
33                      BoilerControl(ControlState.ON);
34
35                      if (TempBoiler >= ConstTempMinBoilerStart)
36                      {
37                          //Boiler is heated, start pump
38                          WaterPumpControl(ControlState.ON);
39                          Message_Heat = "Topeni z kotle";
40                      }
41                      else
42                      {
43                          //Boiler is cold, stop pump
44                          WaterPumpControl(ControlState.OFF);
45
46                          Message_Heat = "Nahrivani kotle";
47                      }
48                  }
49                  //If thermostat false or water is heating, transition to Cooling
50                  if (ThmrSt == ControlState.OFF)
51                  {
52                      ActSt = ActualState.Cool;
53                      CoolTime = 0;
54                      WaterPumpControl(ControlState.ON);
55                      BoilerControl(ControlState.OFF);
56                  }
57                  break;
58              case ActualState.Cool:
59                  CoolTime++;
60
61                  Message_Heat = "Dochlazovani - cas " + CoolTime.ToString();
62                  if (CoolTime >= 600)
63                  {
64                      ActSt = ActualState.Idle;
65                      CoolTime = 0;
66                      WaterPumpControl(ControlState.OFF);
67                  }
68                  break;
69          }
70                      // TUV state-machine
71          switch (TUVSt)
72          {
73              case TUVState.Idle:
74                  RELElectricBojlerON.Write(false);
75                  RELWatterTUVON.Write(false);
76                  Message_TUV = "Ohrev TUV vypnut";
77                  if (TempTUVWatter < ConstTempMinTUV && TempStorage > ConstTempMinAccTUV)
78                  {
79                      TUVSt = TUVState.HeatingAcc;
80                  }
81                  if (TempTUVWatter < ConstTempMinTUV && TempStorage < ConstTempMinAccTUV)
82                      TUVSt = TUVState.HeatingElektro;
83
84                  if (TempTUVWatter == 0)
85                      TUVSt = TUVState.Idle;
86                  break;
87              case TUVState.HeatingAcc:
88                  RELElectricBojlerON.Write(false);
89                  RELWatterTUVON.Write(true);
90                  Message_TUV = "Ohrev z akumulace";
91
92                  if (TopeniAktivni == false)
93                      ValveControl(ValveState.Acc);
94
```

```
95                      if (TempTUVWatter > ConstTempMaxTUV || TempStorage < ConstTempMinAccTUVHeat)
96                          TUVSt = TUVState.Idle;
97                      break;
98                  case TUVState.HeatingElektro:
99                      RELElectricBojlerON.Write(true);
100                     RELWatterTUVON.Write(false);
101                     Message_TUV = "Ohrev elektrinou";
102                     if (TempTUVWatter > ConstTempMaxTUV)
103                         TUVSt = TUVState.Idle;
104                     if (TempStorage > ConstTempMinAccTUVHeat + 2)
105                         TUVSt = TUVState.HeatingAcc;
106                     break;
107             }
108                     // Accumulation tank state-machine
109             switch (ActAccHeatSt)
110             {
111                 case AccHeatState.OFF:
112                     Message_Acc = "Vypnuto";
113                     AccTime = 0;
114                     RELAccHeatON.Write(false);
115                     if (ActualFVEPower >= ConstPowerMinAccfHeatON)
116                         ActAccHeatSt = AccHeatState.PrepareON;
117                     break;
118                 case AccHeatState.PrepareON:
119                     if (ActualFVEPower >= ConstPowerMinAccfHeatON)
120                     {
121                         AccTime = AccTime + 2;
122                         Message_Acc = "Zapinani, cas = " + AccTime.ToString();
123                         if (AccTime >= ConstTimeAccHeat)
124                         {
125                             ActAccHeatSt = AccHeatState.ON;
126                             AccTime = 0;
127                         }
128                     }
129                     else
130                         ActAccHeatSt = AccHeatState.OFF;
131                     break;
132
133                 case AccHeatState.ON:
134                     Message_Acc = "Zapnuto";
135                     RELAccHeatON.Write(true);
136                     if (ActualFVEPower <= ConstPowerMinAccHeatOFF)
137                         ActAccHeatSt = AccHeatState.PrepareOFF;
138                     break;
139                 case AccHeatState.PrepareOFF:
140                     if (ActualFVEPower <= ConstPowerMinAccHeatOFF)
141                     {
142                         AccTime = AccTime + 2;
143                         Message_Acc = "Vypinani, cas = " + AccTime.ToString();
144                         if (AccTime >= ConstTimeAccHeat)
145                             ActAccHeatSt = AccHeatState.OFF;
146                     }
147                     else
148                     {
149                         ActAccHeatSt = AccHeatState.ON;
150                         AccTime = 0;
151                     }
152                     break;
153             }
```

Listing 6: Code; Ethernet connection thread function

```
1   public static void connect_ethernet()
2   {
3       while (true)
4       {
5           if (ethCon.IPAddress == "0.0.0.0")
6           {
7               if (ethCon.Opened == false)
8               {
9                   ethCon.Open();
10                  ethCon.EnableStaticIP("192.168.0.150", "255.255.255.0", "192.168.0.1");
11                  ethCon.EnableDynamicDns();
12              }
13              Debug.Print("Connceted with IP adress " + ethCon.IPAddress);
14          }
15          else
16          {
17              if (ethCon.Opened == false)
18              {
19                  ethCon.Open();
20                  ethCon.EnableStaticIP("192.168.0.150", "255.255.255.0", "192.168.0.1");
21                  ethCon.EnableDynamicDns();
22                  Debug.Print("Connceted with IP adress " + ethCon.IPAddress);
23              }
24
25          }
26          Thread.Sleep(100);
27      }
28  }
```

Listing 7: Code: Write to SD function

```
1   public static void write_to_SD(string output_data, string[] data_field, int line)
2   {
3       LED_SD.Write(true);
4       string data = "";
5       data = RealTimeClock.GetDateTime().ToString() + ";\r\n";
6       for (int ite = 0; ite < line; ite++)
7       {
8           data = data + data_field_glob[ite];
9       }
10      output_data = data;
11      try
12      {
13          DateTime today = RealTimeClock.GetDateTime();
14          SDCard SD = new SDCard();
15          Thread.Sleep(500);
16          SD.Mount();
17          Thread.Sleep(500);
18          string rootDirectory = VolumeInfo.GetVolumes()[0].RootDirectory;
19          string[] folders = Directory.GetDirectories(rootDirectory);
20          string[] files = Directory.GetFiles(rootDirectory);
21          FileStream FileHandle1 = new FileStream("" + rootDirectory + @"\"" + today.Year.ToString() +
                "_" + today.Month.ToString() + "_" + today.Day.ToString() + "_" + today.Hour.ToString()
                 + "_" + today.Minute.ToString() + "_" + today.Second.ToString() + ".txt", FileMode.
                Create, FileAccess.Write);
22
23          byte[] data_transfer;
24          data_transfer = Encoding.UTF8.GetBytes(output_data);
```

```
25              Debug.Print("Writing data to SD card at " + RealTimeClock.GetDateTime().ToString());
26              FileHandle1.Write(data_transfer, 0, data_transfer.Length);
27              VolumeInfo.GetVolumes()[0].FlushAll();
28              FileHandle1.Close();
29              FileHandle1 = null;
30              rootDirectory = null;
31              folders = null;
32              files = null;
33              data_transfer = null;
34              output_data = null;
35              SD.Unmount();
36              SD.Dispose();
37              SD = null;
38          }
39          catch (Exception)
40          {
41          }
42      }
```

Listing 8: Code: Write to server function

```
1       public static void write_to_server(string output_data, string[] data_field, int line, int prev_line)
2       {
3           string first_line = RealTimeClock.GetDateTime().ToString() + " \r\n ";
4           string output_data2;
5           output_data = first_line;
6           for (int ii = prev_line; ii < line; ii++)
7           {
8               output_data = output_data + data_field_glob[ii];
9           }
10          try
11          {
12              HttpWebRequest request_upload = HttpWebRequest.Create("http://147.32.200.115:8000/
                    SolarMonitor/WS/data/1") as HttpWebRequest;
13
14              request_upload.Headers.Add("Authorization", "Basic dGpfYXM6aGVzbG8xMjM=");
15              request_upload.Timeout = 10000;
16              request_upload.ReadWriteTimeout = 20000;
17              request_upload.Method = "PUT";
18              request_upload.ContentLength = output_data.Length;
19
20              Debug.Print(RealTimeClock.GetDateTime().ToString());
21              Debug.Print("Writing data to server at " + RealTimeClock.GetDateTime().ToString());
22              StreamWriter stream = new StreamWriter(request_upload.GetRequestStream());
23              stream.Write(output_data);
24              stream.Flush();
25              Debug.Print(RealTimeClock.GetDateTime().ToString());
26              stream.Close();
27              stream = null;
28              request_upload.Dispose();
29              request_upload = null;
30              Debug.Print("Data written to server.");
31          }
32          catch (Exception)
33          {
34          }
35
36      }
```

Listing 9: Code: Info-server thread

```
1   static void WebReqThread()
2   {
3       DateTime datum = new DateTime();
4       while (ethCon.IPAddress == "0.0.0.0")
5       {
6           Debug.Print("Waiting for network to start info-server");
7           Thread.Sleep(1000);
8       }
9       HttpListener listener = new HttpListener("http", 80);
10      Thread.Sleep(2000);
11      listener.Start();
12      initialize_clock();
13      while (true)
14      {
15          HttpListenerResponse response = null;
16          HttpListenerContext context = null;
17          try
18          {
19              context = listener.GetContext();
20              response = context.Response;
21              response.StatusCode = (int)HttpStatusCode.OK;
22              byte[] HTML = Encoding.UTF8.GetBytes(
23                  "<html><head><title>Sadilek topeni</title></head>" +
24                  "<body><p>Aktualni stav topeni: " + Message_Heat + "</p>" +
25                  "<p>Aktualni stav ohrevu TUV: " + Message_TUV + "</p>" +
26                  "<p>----</p>" +
27                  "<p>Teplota akumulace: " + TempStorage.ToString() + " C</p>" +
28                  "<p>Teplota kotle: " + TempBoiler.ToString() + " C</p>" +
29                  "<p>Teplota bojleru: " + TempTUVWatter.ToString() + " C</p>" +
30                  "<p>----</p>" +
31                  "<p>Bojler: Minimalni teplota TUV: " + ConstTempMinTUV.ToString() + " C</p>" +
32                  "<p>Bojler: Maximalni teplota TUV: " + ConstTempMaxTUV.ToString() + " C</p>" +
33                  "<p>Bojler: Minimalni teplota akumulace pri zacatku ohrevu TUV: " +
                        ConstTempMinAccTUV.ToString() + " C</p>" +
34                  "<p>Bojler: Minimalni teplota akumulace pri ohrevu TUV: " + ConstTempMinAccTUVHeat.
                        ToString() + " C</p>" +
35                  "<p>----</p>" +
36                  "<p>Topeni: Minimalni teplota akumulace: " + ConstTempMinAccHeat.ToString() + " C</p>
                        " +
37                  "<p>Topeni: Teplota pro nahrati kotle pri startu: " + ConstTempMinBoilerStart.
                        ToString() + " C</p>" +
38                  "<p>----</p>" +
39                  "<p>FVE: Aktualni vykon: " + ActualFVEPower.ToString() + " W</p>" +
40                  "<p>FVE: Ohrev akumulacni nadrze: " + Message_Acc + "</p>" +
41                  "</body></html>"
42                  );
43              response.ContentType = "text/html";
44              response.OutputStream.Write(HTML, 0, HTML.Length);
45              response.OutputStream.Flush();
46              Thread.Sleep(10);
47              response.Close();
48          }
49          catch
50          {
51          }
52      }
53  }
```

Listing 10: Code: RMS calculation procedure

```
1               for (int i = 0; i < wave_count; i++)
2               {
3                   for (int j = 0; j < waves[0].Length; j++)
4                   {
5                       power[j] = waves[i][j] * waves[i][j];
6                       power_monitor = waves[i][j];
7                       sum = sum + power[j];
8                   }
9                   // If too few samples (no current flowing, only some background noise) then dont
                        calculate
10                  if (period < 5)
11                  {
12                      sum = 0;
13                  }
14                  integ = sum / period;
15                  rms_array[i] = System.Math.Sqrt(integ);
16              }
17
18              //sum of 5 waves
19              for (int i = 0; i < wave_count; i++)
20              {
21                  average = average + rms_array[i];
22              }
23              average = average / wave_count;
24              voltage_meas = 2 * eff;
25              current_meas = voltage_meas / resistance_meas;
26              current_true = current_meas * 1000;
```

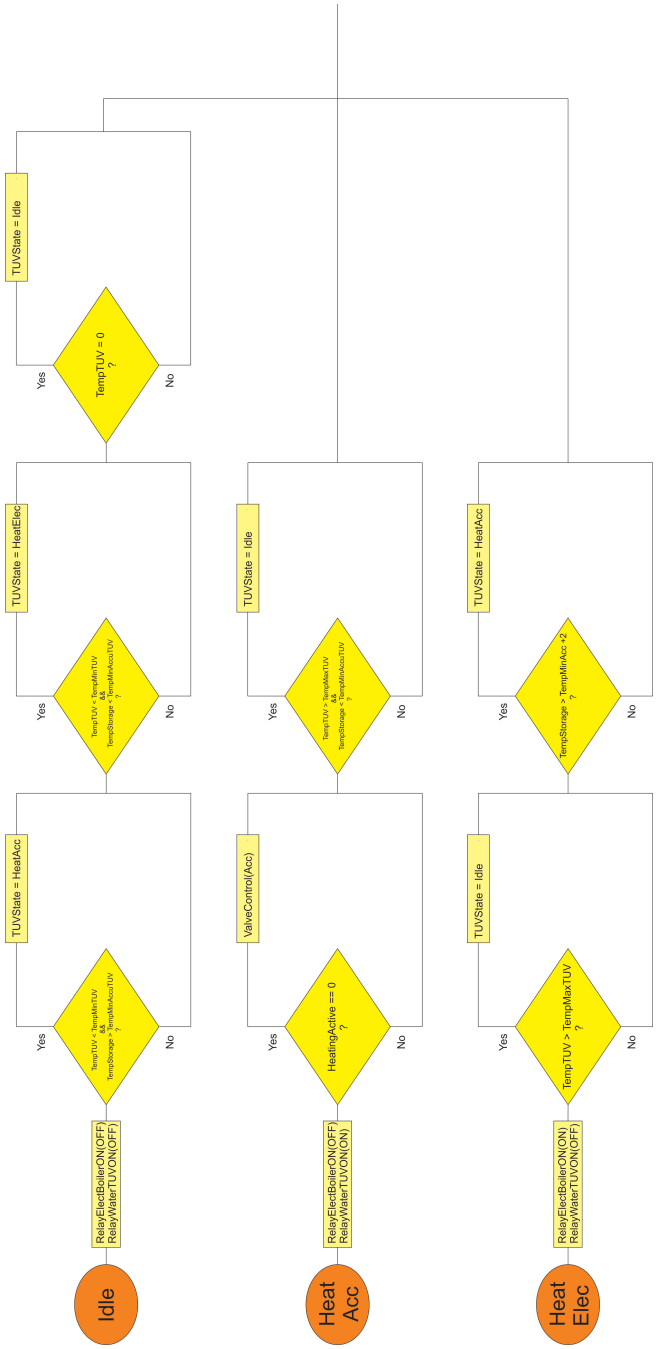



Figure 32: State diagrams of TUV.

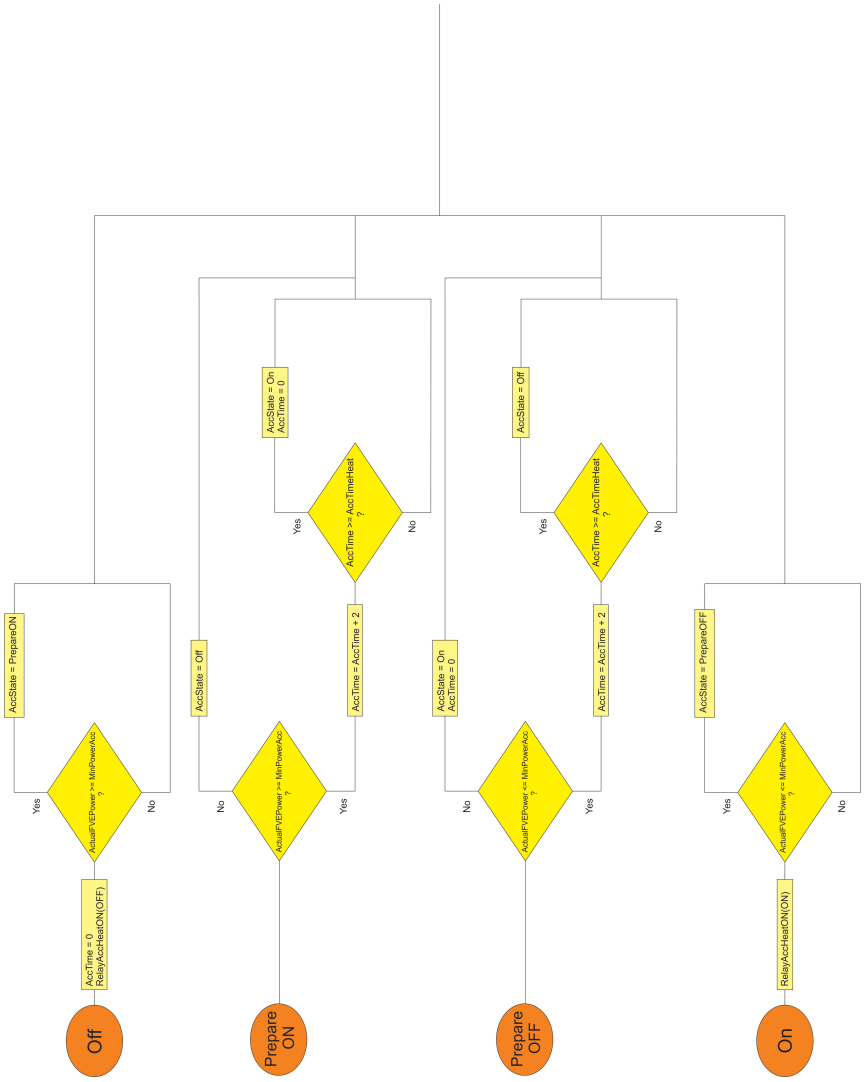



Figure 33: State diagrams of Acc.

## COMMUNICATION INFORMATION

This section provides important information concerning the networking.

**Web-page of the data server:**

http://147.32.200.115 : 8000/SolarMonitor/WS/data/1

**IP address of the server:**

147.32.200.115 : 8000

**Web-page of the time server:**

http://147.32.200.115 : 8000/SolarMonitor/WS/generator.php

**Credentials for connection to the data server:**

Username: tomas

Password: heslo123

**IP address of the info-server:**

192.168.0.150 : 80

**DHCP server:** Unavailable

**Dynamic DNS:** Available

**Email address usable by** G120E**:**

Username: g120esolarcontrol@seznam.cz

Password: SolarControl2016

**RS-485 settings:**

Baudrate: 19200 bps

Parity: None

Stopbits: 1

Databits: 8

# BIBLIOGRAPHY


[1] ABB. *Model predictive control technology demystified*. Jan. 2016. URL: http://new.abb.com/control-systems/features/model-predictive-control-mpc/.

[2] *Accessing Inverter Parameters via* RS−485 *using the ComLynx Protocol*. Danfoss Solar Inverters A/S. Apr. 2014.

[3] Jack Ganssle. *The Art of Embedded Design*. 2nd. Burlington, MA, USA: Newnes, 2008.

[4] Jiří Cigler Jan Široký Frauke Oldewurtel and Samuel Prívara. "Experimental Analysis of Model Predictive Control for an Energy Efficient Building Heating System." In: *Applied Energy* 8.9 (2011).

[5] Ben Albahari Joseph albahari. *C sharp* 5.0 *in a Nutshell: The Definitive Reference*. Sebastopol, CA, USA: O'Reilly, 2012.

[6] GHI Electronics LLC. *.NET Micro Framework for Beginners*. Nov. 2015. URL: https://www.ghielectronics.com/support/netmf.

[7] Kraig Mitzner. *Complete PCB Design Using OrCAD Capture and PCB Editor*. Burlington, MA, USA: Newnes, 2009.

[8] Jiří Tichý. "Řídící systém rodinného domu." MA thesis. Czech Technical University in Prague, Czech Republic, 2014.

[9] S. Tumanski. *Principles of Electrical Measurement*. Boca Raton, FL, USA: Taylor & Francis Group, 2006.

[10] Vladimír Král Zdeněk Hradílek Ilona Lázničková. *Elektrotepelná technika*. 1st. Prague, Czech Republic: The MIT Press, 2011.

[11] Vít Záhlava. *Návrh a konstrukce desek plošných spojů*. 1st. Prague, Czech Republic: BEN - technická literatura, 2011.

[12] Electronics hub. *Current transformer*. June 2015. URL: http://www.electronicshub.org/current-transformer/.