Technical University of Crete
Department of Electrical & Computer Engineering

DEVELOPMENT OF SMART BUILDING MANAGEMENT SYSTEM

Angelos Isoufi

EXAMINING COMMITTEE

Associate Professor Kanellos Fotios (Supervisor)
Associate Professor Minoas Petrakis
Associate Professor Tsekouras Georgios

Chania Crete, March 2025

Abstract

When it comes to HVAC systems installed in buildings where human beings are present, comfort, ease of use and power consumption are the topics that matter the most. Advances in technological inventions have led to HVAC systems that meet even the most demanding requirements. However, there is always room for more improvements.

One aspect of HVAC systems that has gathered significant attention in recent years, is power consumption and its associated economic impact to the users. As stated by European Union, energy prices have reached all-time highs in 2022 as they were directly affected by the price of gas. New measures to cut down energy bills have been introduced. One of them, is the obligation for EU countries to reduce consumption by at least 5% during peak hours.

In this work, taking in consideration the needs of the consumers, the energy crisis that EU and the whole world is facing as well as the need to reduce emissions to preserve a better environment for the future generations, we try making the first steps of transitioning from theory to practice aiming to bring a solution to these problems. This research focuses on implementing an algorithm that aims to provide better control and less power consumption, thereby lowering energy bills, to a real-world controller with the aim to create a smart Building Management Systems (BMS).

The development of the BMS involves several steps. It begins with understanding the basic concepts of the algorithm, then finding out how it can be implemented in a real controller. Next steps involve gathering the appropriate equipment and knowledge background, focusing on specific parts since implementing a fully functional BMS system requires a lot of time, effort and resources. The final steps include conducting tests, gathering results and proposing further improvements.

Acknowledgments

I would like to express my gratitude to my supervisor, Professor Kanellos Fotios, for providing me the opportunity to get involved in this valuable and interesting project and get theoretical as well as practical knowledge on a subject of great interest. His contribution throughout this journey was vital in making the whole process easier and more efficient.

I could not have undertaken this journey without the assistance of Mr. George Bintarchas, the administrator and owner of ELCON Systems & Components. His extensive expertise in the fields of electronics, industrial automations, electromechanical and information systems as well as his willingness to address all my questions and theoretical gaps, and provide the necessary equipment whenever it was needed, has been crucial and immensely helpful throughout my research.

In the same vein, I would like to express my sincere thanks to Professor Dimitra Kyriakou. Her amazing work in clearly and extensively providing all the theoretical background knowledge when it comes to the algorithm used in this thesis, was invaluable.

Finally, I'd like to praise my friends Christos Stouraitis and Mikel Saltchai, mechanical engineers from NTUA, for contributing to my thesis and being always available to analyze topics within their expertise that were needed in my research and provided me with valuable data.

Chania, June 2024

Angelos Isoufi

Table of Contents

Introduction

Transitioning from traditional HVAC systems to innovative ones that incorporate advanced mechanisms and algorithms to address the climate crisis, reduce operational costs and enhance user experience is a complex yet essential task. That is why it should be on top priority and steps must be taken towards this goal.

This thesis combines a new approach on controlling HVAC systems with the concept of Building Management Systems (BMS) and aims to develop a smart controller. This smart controller takes into consideration parameters related to the thermal behavior of a building, thermal energy generated by appliances and occupants, current energy prices from the electricity supplier as well as the preferred temperature setpoints defined by the administrator. By focusing on keeping the room temperature between the setpoints as well as minimizing electricity costs, the system seeks to optimize power delivery to HVAC systems.

Moving from theory to practice involved several challenging steps. The initial one was understanding the provided algorithm. The second one was choosing the right controller and equipment for meeting all the prerequisites. After thorough investigation, a programmable logic controller (PLC) with integrated human machine interface (HMI) was proposed. This choice introduced the need to learn how to program and operate these machines. After learning the basics and gathering all the appropriate knowledge on how to use them, the final step was testing the controller and comparing the results with the expected outcomes.

Following this introduction, we will dive into each step in more depth and provide all the knowledge gathered until this point. We will start with the theoretical background needed followed by an analysis of the algorithm. After that, we will discuss about HMI and PLC implementation. At the end, there will be some discussion about possible future improvements and what is yet to be done for the BMS system to be truly functional.

BMS System - Abstract Representation

Before moving on with the theoretical knowledge gathered throughout this project, it would be wise to present an abstract model of the building management system (BMS) and its interaction with a building. For educational purposes, since this term might not be familiar to everyone, BMS is a sophisticated control system implemented in a building to monitor and regulate its mechanical and electrical components, including ventilation, lighting, electrical infrastructure, and HVAC systems [1]. This research will mainly focus on HVAC systems.

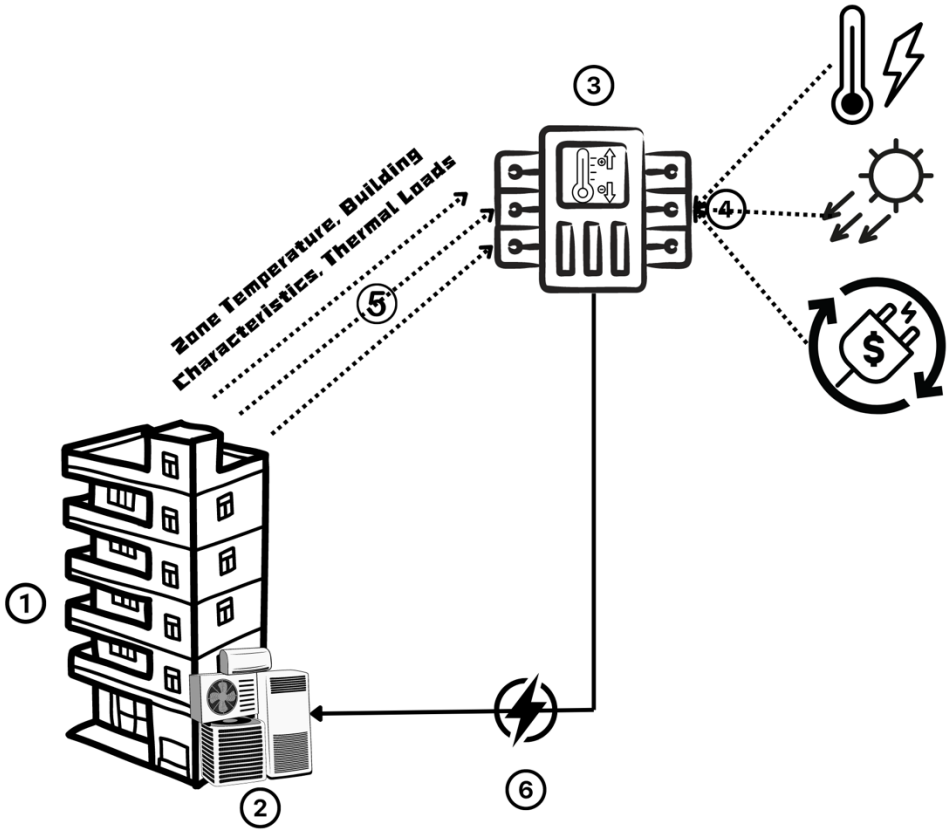


Figure 1: BMS & Building HVAC System Control

Let's break down the model into six parts (they are highlighted in the picture):

1. **Building**: Can vary in size and can consist of multiple rooms and floors.
2. **HVAC System**: Installed in the building.
3. **Controller:** Consists of HMI and PLC running the power delivery algorithm.
4. **External Factors**: Outdoor temperature, solar radiation, electricity price.
5. **Building Parameters**: Includes thermal loads from people inside, electronic devices and any other heat sources, as well as indoor temperature of each thermal zone.
6. **Controller Output**: Power supplied to the HVAC system.

The main idea is to create a system that controls the building's HVAC system as cost-effectively as possible. To achieve this, predictions of ambient temperature, solar radiation, electricity price are made. Minimum as well as maximum temperature setpoints are set by the system's administrator. Additionally, there is continuous monitoring of the building's temperatures in various areas sharing the same thermal behavior (thermal zones). Lastly, a model of the building is provided. This model encapsulates valuable information regarding its thermal behavior and characteristics.

Now, the focus will move on the theoretical foundations for understanding the whole concept.

Theoretical Background

1.1 Model of Building Thermal Load Agent

This research focuses on the behavior of thermal zones in various buildings. Understanding how this behavior can be manipulated and parameterized is crucial to finding the best methods of controlling it. By considering the following thermal equilibrium equations [2] [3], we can associate each zone's temperature with specific parameters such as internal temperature, thermal gains, thermal loads and ambient temperature.

As you can see from the first equation, the temperature rate of change in a thermal zone with density p_z, heat capacity C_z and air volume V_z is directly affected by the heat exchanges and thermal gains.

$$p_z \cdot C_z \cdot V_z \cdot \frac{dT_{in,z}}{dt} = \dot{Q_{ex,wall,z}} + Q_{in,wall,z} + \dot{Q_{win,z}} + \dot{Q_{SW,z}} + \dot{Q_{sg,z}} - Q_{HVAC,z} \quad (1)$$

For better comprehension, it is essential to establish the appropriate nomenclature:

$Fwall,Fwin$: wall/window surface area
τwin: window glass transmission coefficient
$Uwall,Uwin$: factor of the heat exchange of the external wall/window
Rse: heat resistance of the external surface
aw: the external wall absorbance coefficient
Rse: heat resistance of the external surface
IT,z: total solar radiation
SC: the shading coefficient of the windows
Ib,Id,I: beam, diffuse and total radiation on horizontal surface, respectively
pz: the density of the zth thermal zone
Cz: specific heat capacity
Vz: volume of the air of the zth thermal zone

$\dot{Q_{ex,wall,z}}$: It refers to the power transferred between the internal environment of the zone and the external environment through the external walls. It is affected by the size of the walls and their factor of heat exchange.

$$\dot{Q_{ex,wall,z}} = \sum_{y \in \mathcal{E}} U_{wall,y} \cdot F_{wall,y} \cdot \left(T_{out} - T_{in,z}\right) \quad (2)$$

$\dot{Q_{win,z}}$: It refers to the heat transferred between the internal environment of the zone and the external environment through the windows. It is affected by the size of the windows and their factor of heat exchange.

$$\dot{Q_{win,z}} = \sum_{y \in \mathcal{E}} U_{win,y} \cdot F_{win,y} \cdot \left(T_{out} - T_{in,z}\right) \quad (3)$$

$\dot{Q_{SW,z}}$: It refers to the heat gained from the solar radiation reaching the external walls.

$$\dot{Q_{SW,z}} = \sum_{y \in \mathcal{E}} a_w \cdot R_{se} \cdot U_{wall,y} \cdot F_{wall,y} \cdot I_{T,z} \quad (4)$$

$\boldsymbol{\dot{Q}_{sg,z}}$**:** It refers to the heat gained from the solar radiation entering the zone through the windows.

$$\dot{Q_{sg,z}} = \sum_{y \in \mathcal{E}} t_{win} \cdot SC \cdot F_{win,y} \cdot I_{T,z} \quad (5)$$

$\boldsymbol{Q_{in,wall,z}}$**:** It refers to the heat transferred between the internal walls of the available zones.

$$Q_{in,wall,z} = \sum_{y \in \mathcal{E}} U_{wall,y} \cdot F_{wall,y} \cdot \left(T_{in,nz} - T_{in,z}\right) \quad (6)$$

By appropriately modifying (1)-(6), the state space system of equations is obtained for each building.

$$\frac{\boldsymbol{dT_{in}(t)}}{\boldsymbol{dt}} = \boldsymbol{A_b} \cdot \boldsymbol{T_{in}(t)} + \boldsymbol{B_b} \cdot \boldsymbol{U} \quad (7)$$

$$\boldsymbol{Y(t)} = \boldsymbol{C_b} \cdot \boldsymbol{T_{in}(t)} + \boldsymbol{D_b} \cdot \boldsymbol{U} \quad (8)$$

The input vector U is of dimension $((2N_z + 2) \times 1)$. It is given in (9).

$$U = \begin{bmatrix} Q_{EC,1}(t) \\ \vdots \\ Q_{EC,NZ}(t) \\ Q_{in,1}(t) \\ \vdots \\ Q_{in,NZ}(t) \\ T_{out}(t) \\ I_T(t) \end{bmatrix}$$

With:

$\boldsymbol{I_{T,Z}}$: Total solar radiation incident on a tilted surface, thus the calculation of R_b

$$I_{T,Z} = I_b \cdot R_b + I_d \cdot \left(\frac{1+\cos\beta_Z}{2}\right) + I \cdot p_g \cdot \left(\frac{1-\cos\beta_Z}{2}\right) \quad (10)$$

$$R_b = \frac{\cos\theta}{\cos\theta_Z} \quad (11)$$

$\boldsymbol{Q_{EC,NZ}(t)}$**:** internal heat gains for zone N

$\boldsymbol{Q_{in,NZ}(t)}$**:** colling power production for zone N

$\boldsymbol{T_{out}(t)}$**:** ambient temperature

The elements of the matrix A_b with dimension $(N_Z \times N_Z)$ are calculated in (12)-(13).

$$A_{j,j} = -\sum_{y\in\mathcal{E}} U_{wall,y} \cdot F_{wall,y} - \sum_{x\in\mathcal{I}} U_{wall,x} \cdot F_{wall,x} - \frac{1}{p_Z \cdot C_Z \cdot V_Z} \cdot \sum_{y\in\mathcal{E}} U_{win,y} \cdot F_{win,y} \quad (12)$$

$$A_{j,i} = \begin{cases} \frac{1}{p_z \cdot C_z \cdot V_z} \cdot U_{wall,x} \cdot F_{wall,x}, & \text{if } j,i \in \mathbb{N} \\ 0, & \text{if } j,i \notin \mathbb{N} \end{cases} \quad (13)$$

The dimension of B_b is $N_Z \times (2N_Z + 2)$ and it is calculated as follows,

$$B_b = \frac{1}{p_z \cdot C_z \cdot V_z}\left[-I_{(N_Z \times N_Z)}\ I_{(N_Z \times N_Z)}\ B_{ex,w}\ B_{rad}\right] \quad (14)$$

with

$$B_{ex,w(N_Z \times 1)} = \begin{bmatrix} B_{ex,w,1} \\ \vdots \\ B_{ex,w,NZ} \end{bmatrix} \quad (15)$$

$$B_{rad(N_Z \times 1)} = \begin{bmatrix} B_{rad,1} \\ \vdots \\ B_{rad,NZ} \end{bmatrix} \quad (16)$$

The elements of the submatrices $B_{ex,w}$ and B_{rad} of matrix B_b are determined as follows.

$$B_{ex,w,z} = \sum_{y\in\epsilon} U_{wall,y} \cdot F_{wall,y} + \sum_{y\in\epsilon} U_{win,y} \cdot F_{win,y} \quad (17)$$

$$B_{\mathrm{rad},z} = \sum_{y\in\mathcal{E}} a_W \cdot R_{\mathrm{se}} \cdot U_{\mathrm{wall},y} \cdot F_{\mathrm{wall},y} + \sum_{y\in\mathcal{E}} \tau_{\mathrm{win}} \cdot SC \cdot F_{\mathrm{win},y} \quad (18)$$

Considering as output the internal temperatures of the thermal zones then the matrices $\mathbf{C_b}$ and $\mathbf{D_b}$ are defined as in the following,

$$\mathbf{C_b}_{(N_Z \times N_Z)} = \mathbf{I}_{(N_Z \times N_Z)} \quad (19)$$

$$\mathbf{D_b}_{(N_Z \times 2N_Z + 2)} = \mathbf{0}_{(N_Z \times 2N_Z + 2)} \quad (20)$$

The system of continuous time equations (7)-(8) is converted to discrete time equations (21)-(22).

$$T_{\mathrm{in}}(k+1) = \mathbf{A_{b,d}} \cdot T_{\mathrm{in}}(k) + \mathbf{B_{b,d}} \cdot \mathbf{U} \quad (21)$$

$$Y(k) = \mathbf{C_{b,d}} \cdot T_{\text{in}}(k) + \mathbf{D_{b,d}} \cdot \mathbf{U} \quad (22)$$

From all the equations presented above, the ones that the reader should focus to are (7) and (8), especially the first one. To better understand why, let's focus on the parameters $\boldsymbol{A_b, B_{b,} T_{in}(t), U}$ and find out how they affect the controller and the behavior of the BMS system.

- $\boldsymbol{A_b, B_{b,}}$ **:** These values are directly affected by the characteristics of each building and the materials used in its construction. These values can be considered constant for each building and help make the controller adaptable to any new environment.

- $\boldsymbol{T_{in}(t)}$ **:** This is the internal temperature of each zone. This value is measured by sensors connected to the BMS. Our goal is to keep it inside a desirable range.

- $\boldsymbol{U}$**:** This vector is the heart of the controller. It consists of all thermal loads as well as solar radiation and ambient temperature. By trying to keep the zone temperature within some preferred setpoints and making predictions about solar radiation, outdoor temperature and people inside the building, the controller makes the necessary changes to the power output of the HVAC systems. Another crucial parameter, not presented here but also playing a critical role in the final output, is the price of electricity. More on this will be discussed in the following pages.

1.2 HMI

1.2.1 Introduction to HMI

Figure 2: HMI

A Human-Machine Interface (HMI) is a user interface that combines specialized hardware and software, allowing a person to control a machine system or device. HMIs are commonly used for automation and industrial purposes [4].

When it comes to industrial use, HMI can serve multiple roles like:

- **Displaying data**
- **Gathering data and analyzing them**
- **Monitor machine input and outputs**

Practical uses of HMI include controlling the movement of a crane as well as monitoring temperatures in the thermal zones of an industrial building.

Industries using HMI include:

- **Energy**
- **Oil and gas**
- **Power**
- **Transportation**
- **And many more**

HMIs screens find great use when combined with PLCs. In real time, users can configure the PLC, access input and output data, and take immediate action in case of an emergency.

1.2.2 Connecting with HMI

Connecting with HMIs is straightforward, as they come with various ports to serve all needs. The most common connection ports include:

- **Ethernet** (Modbus TCP protocol)
- **Serial Port RS485** (Modbus RTU protocol)
- **USB**
- **Micro SD** (In order to boot the machine with your custom program)

1.2.3 HMI programming – Design of Graphical User Interface (GUI)

When designing a user interface for industrial purposes, it should be clear, easy to use, functional and leave no room for errors.

In this respect, most companies like Siemens, Schneider deliver their products with custom software specifically designed for this purpose. These software tools offer a variety of pre-designed and pre-programmed functionalities. For example, in case you want to add a switch to the screen to control a valve, you can simply search for the icon, add it to the main screen and adjust a few basic configurations. This process requires no programming knowledge, such as C or similar languages.

Additionally, there are third-party software options like Ignition from Inductive Automation, that support multiple PLCs and offer the same functionality.

Beyond the basic functionalities that all HMIs offer, in some cases, such as those offered by Kinco, HMIs provide the ability to write C code. This allows for custom operations. However, it is important to keep in mind, that writing complex logic is not recommended, since the resources are limited, and the operations should be executed in reasonable time frames.

1.3 PLC

1.3.1 Introduction to PLCs

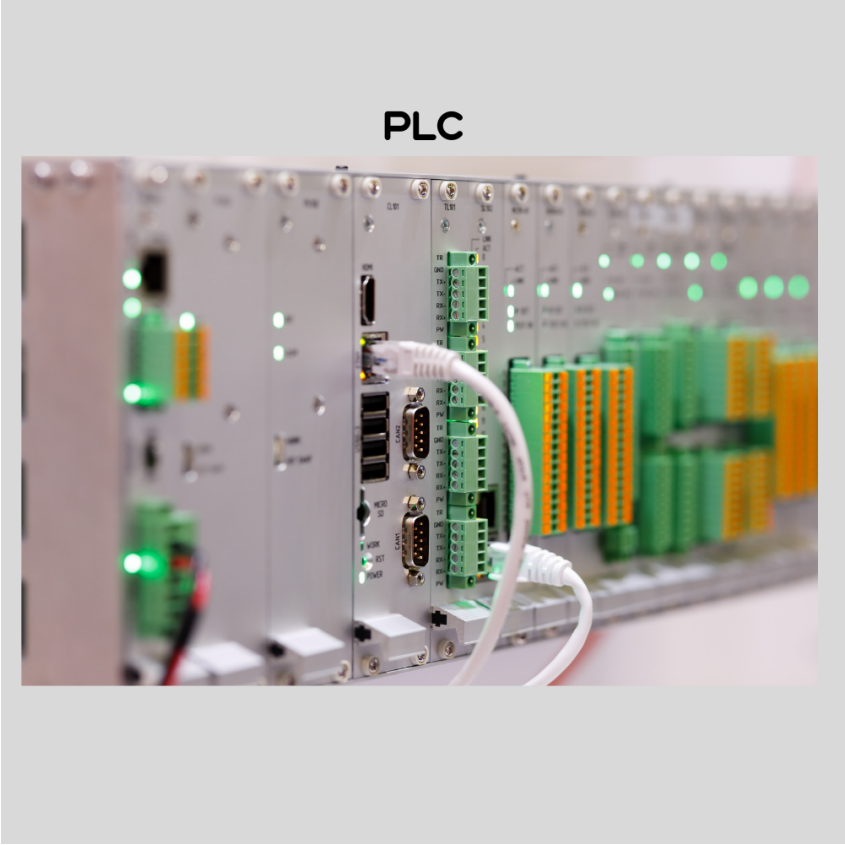


Figure 3: PLC

What is a PLC?
A programmable logic controller (PLC) is a device that [5]:

- You wire it with your machine equipment and sensors.
- Write algorithms inside it.
- It will control whatever signal connected to it.

Most Common Manufacturing PLCs Companies [5]**:**

- Allen Bradley
- Siemens
- Delta
- Schneider

Practical Use Case for a PLC [5]**:**

Consider the example shown in Figure 4, where a PLC is used to control the amount of product in a tank that feeds onto a conveyor belt, which is then transported to a specific location.

To achieve this, we need to integrate the mechanical parts (red) as well as sensors (blue) that will enable the PLC to make the right decisions and give the appropriate outputs (purple).
The PLC hosts the control software, receiving feedback from the sensors that measure the weight and level of the product inside the tank. By utilizing the correct algorithm and processing the data, PLC controls the mechanical parts and performs the needed actions like the valve opening plus the control of the conveyor band speed.

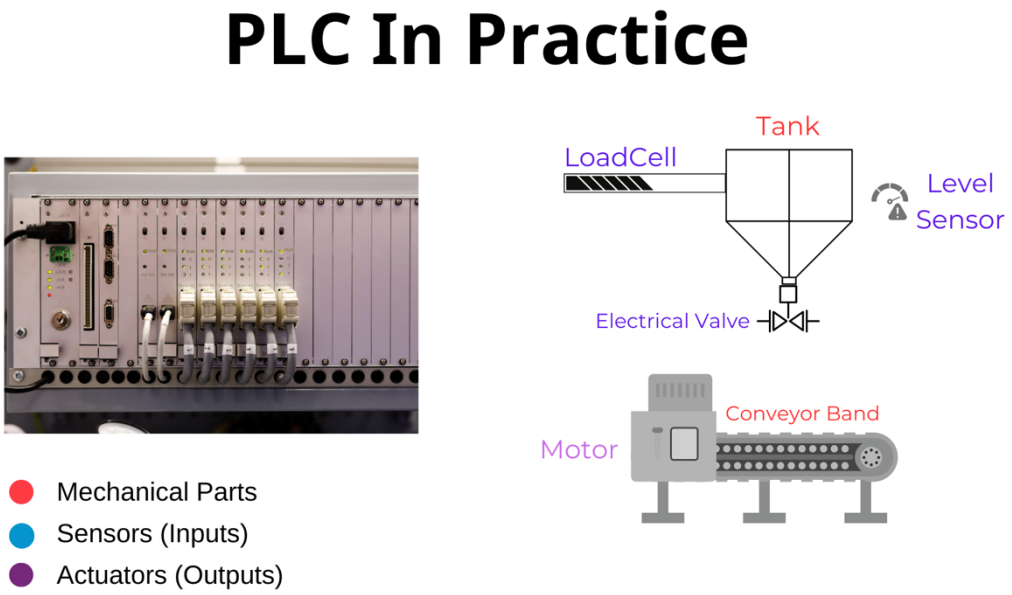


Figure 4: PLC controll of tank-conveyor band operation

PLC Extension Modules [5]**:**

- Digital Inputs
- Digital Outputs
- Analog Inputs
- Analog Outputs
- Communication Protocols Extensions
- Remote IO

PLCs come equipped with multiple connection ports for different types of communications. Most of the times, there is support for analog, digital I/O and Modbus. If additional ports are needed, there is the capability of adding extension modules in order to process more signals.

1.3.2 Basic Components of PLCs

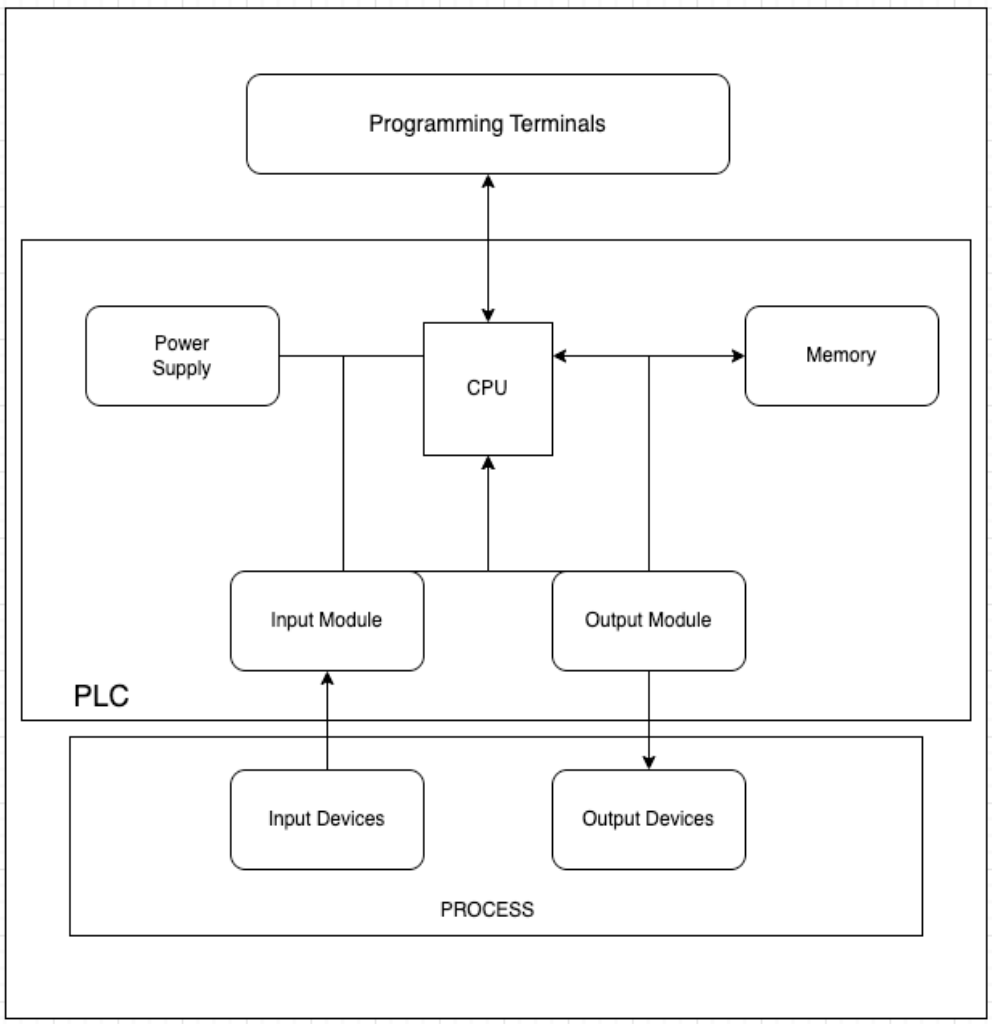


Figure 5 : Block Diagram of PLC with I/O

The PLC mainly consists of the following components [6]**:**

1) **Power Supply**: handles the power supply of the PLC
2) **Processor (CPU):** performs all the operations by executing programmed logic.
3) **Memory**: it is divided into program memory and data memory and stores all the necessary data
4) **Input/Output Cards**: interface between PLC and external environment
5) **Programming terminals**: used to enable users set their own software configurations
6) **Communication Interface**: communication between PLC and other devices like HMIs, computers even PLCs via protocols such as Modbus and Ethernet.

1.3.3 Modbus

Modbus [7] is a communication protocol developed by Modicon published in 1979 for use with its programmable logic controllers (PLCs). It is used to establish **serial communication** among various devices, primarily in the automation industry.

As an open protocol, Modbus allows manufacturers to integrate it into their machines, achieving in this way compatibility across different devices.

Versions of Modbus:

- ASCII
- RTU
- TCP/IP
- PLUS

Master Slave Architecture:

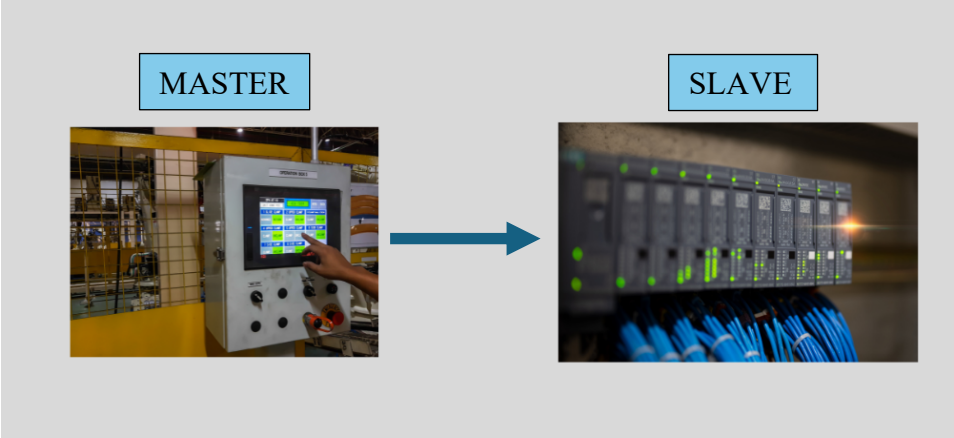


Figure 6: Master-Slave Architecture

The Modbus Protocol follows a master-slave architecture. In this architecture, the **Master** is the device which initiates a **request** and **Slave** is the one sending the **response**.

Serial Communication Protocols Used by Modbus:

- **RS-232**
- **RS-485**
- **Ethernet**

Nowadays, among RS-232 and RS-485, the latter one is mostly preferred [8]. There reasons are presented below:

	RS-232	**VS**	**RS-485**
Data Rate:	**1.42 kbps**		**20 kbps**
Transmission Range:	**15 meters**		**1200 meters**
Support For Multiple Senders:	**No**		**Yes**
Wiring:	**Four wires**		**Two wires**

When to use TCP/IP (Ethernet) over RS-485?

Using Modbus TCP with Ethernet communication over Modbus RTU with RS-485 connection offers several advantages. Ethernet allows for longer distances. Also, the same network can support various devices that not necessarily support the Modbus Protocol. This is because messages are encapsulated in TCP/IP headers. However, this is not the case with Modbus RTU and RS-485 where messages are sent in plain form, requiring the network to be dedicated to compatible devices only.

More about Modbus TCP [9]

We will focus more on Modbus TCP protocol since this the one used in this implementation.
This protocol, as its name suggests, is used for communications over TCP/IP network with default port of 502.

Establishing Connection Between Master and Slave Devices:

1) The master device must be informed about the **IP address** of the slave.
2) Communication should happen on the **correct port**, preferably the default port 502
3) Master should set the **correct register type**. More about the registers below.
4) Configure the **register address**.
5) Set the **function code**

Modbus Data Model

Primary tables	Access	Size	Features
Coil (discrete output)	Read-Write	1 bit (ON/OFF)	Controls external devices like relays, indicators by setting them ON or OFF
Discrete input	Read-Only	1 bit (ON/OFF)	State of physical input (e.g physical button) that can be read and not modified
Input register	Read-Only	16 bits (0–65,535)	Stores sensor readings, statuses or measurements such as temperature or pressure
Holding register	Read-Write	16 bits (0-65,535)	Stores configuration parameters,

			setpoints or any other user-defined data that can be read and modified

Above, you can find a brief representation of the available data types together with their access rights and their size.

Function codes [10]

The following function codes define the actions the master will take when connecting with the slave (read/write coil/register). The codes are two bytes long or one word (16-bit) of hexadecimal numbering with the most significant byte first or "big-endian".

Function Code	Description	Maximum # of coils / registers
01	Read Coils	2000
02	Read Discrete Inputs	2000
03	Read Holding Registers	125
04	Read Input Registers	125
05	Write Single Coil	1
06	Write Single Holding Register	1
15 (0x0F)	Write Multiple Coils	800
16 (0x10)	Preset Multiple Holding Registers	100

These are the most common ones. There are also less common ones targeted for other operations like diagnostics and communication analysis.

1.3.4 PLC Programming

PLC programming is achieved with custom manufacturer software or third-party ones.

PLC can be programmed using several languages standardized by the IEC 61131-3 standard [11]:

- **Ladder Diagram** (LD), graphical
- **Function Block Diagram** (FBD), graphical
- **Structured Text** (ST), textual
- **Instruction List** (IL), textual
- **Sequential Function Chart** (SFC)

We will dive more into the available programming languages [5] to get ourselves comfortable with PLC programming.

Ladder Logic & Ladder Diagram

Ladder logic is one of the most popular PLC programming languages. Its name derives from the way that the program is structured resembling the shape of a ladder with two vertical lines on the left and right and horizontal rugs between them. Ladder logic introduces contacts and coils. Contacts function like buttons controlling the occurrence of an action. Coils act like relays that start or stop the execution of a specific operation. The ladder diagram is read from left to right and from top to bottom.

Example of ladder diagram [12]:

Figure 7: Ladder Diagram

When programming with LD, there is a great variety of components that can be used to achieve different needs. Below is a small introduction to the basic ones.

1) Contact – Coils

Contacts:

- -| |- Normally Open (NO). It lets the flow of execution when the signal is set to 1
- -|/|- Normally Closed (NC). It lets the flow of execution when the signal is set to 0

Coils:

- -( )- An action is executed only when the Input is 1
- -(/)- An action is executed only when the Input is 0

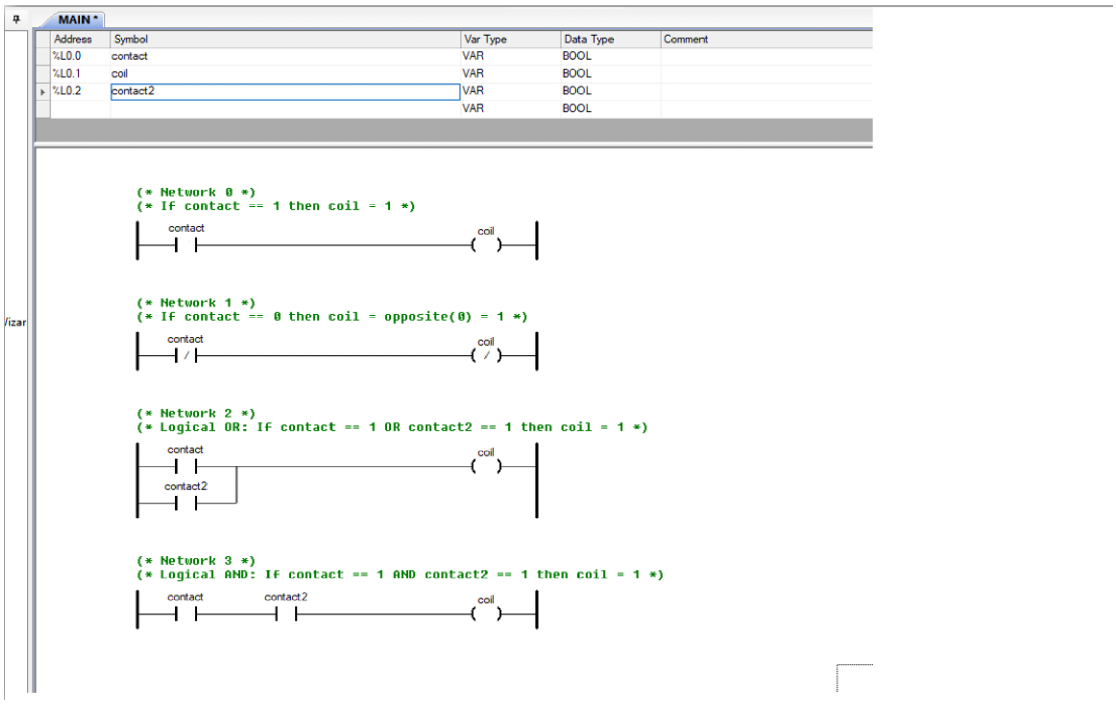


Figure 8: Ladder Logic Example

The ladder diagram above consists of networks 0 to 3. Each network is a separate line of execution with its own logic. The execution of these networks begins with the network 0 and finishes with network 3. Each network is executed from left to right.

- **Network 0**: When the Boolean variable "contact" is set to 1, the "coil" one is also set to 1. If for example, the contact was a push button, and the coil was a relay controlling a motor, if the push button was pressed, the relay would start the motor.

- **Network 1**: If the contact is set to 0 the coil is set to the opposite value of the input, which is 1.

- **Networks 2 and 3**: Logical OR and logical AND operations are achieved by using contacts in parallel and series respectively.

These examples illustrate only few of the available combinations of contacts and coils in ladder logic.

2) Counters

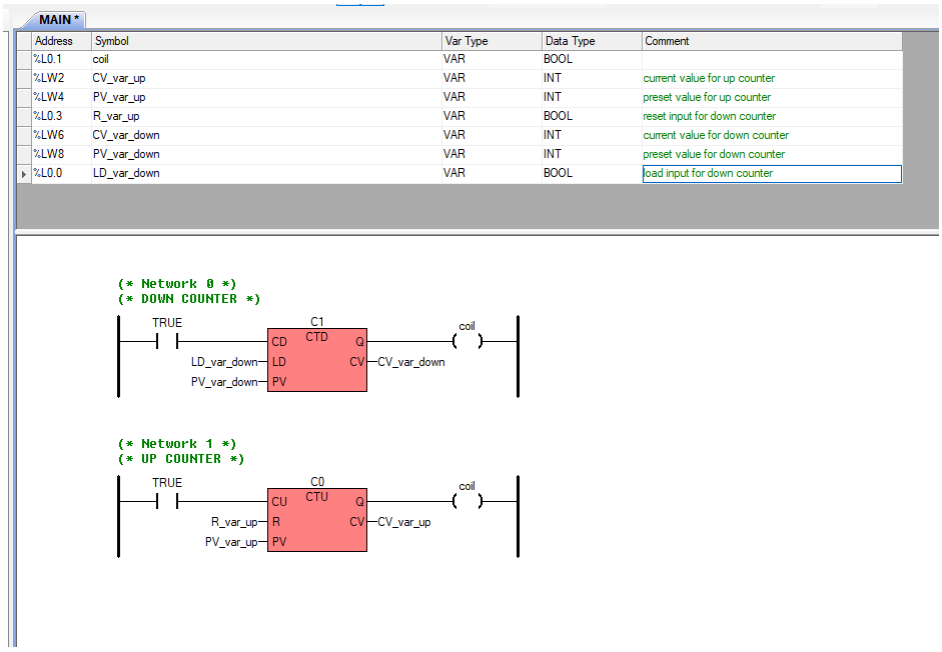


Figure 9: Counters

LD programming comes with a variety of counters. Two of the most common ones are up counters and down counters.

Up Counter: The CTU starts counting when CU input is set to 1 (rising edge). When CV reaches the integer value set to PV, Q is set to 1. The R input is responsible for resetting the counter.

Down Counter: The CTD starts counting down when CU is set to 1 and a value is loaded to counter. The loading is achieved with setting LD to 1. When CV gets to 0, the coil is set to 1.

3) Timers

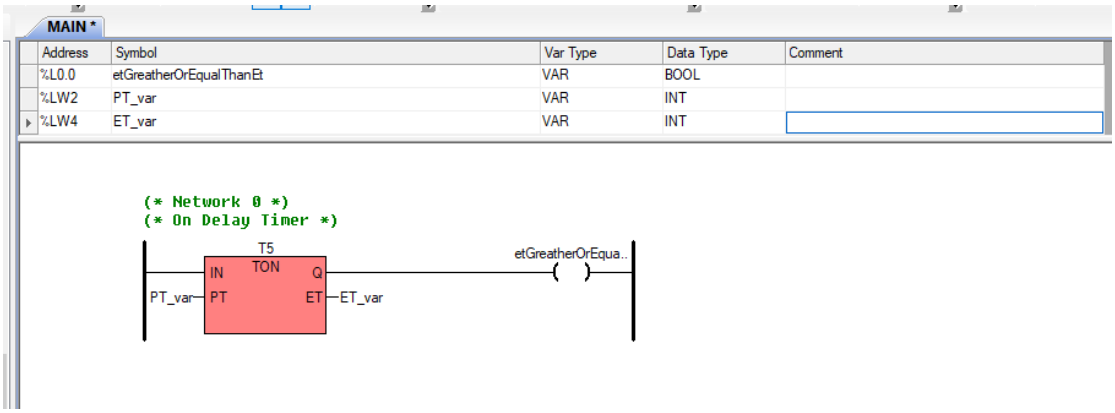


Figure 10: Timer

Timers are also an integral part of ladder logic programming. One of them is the on-delay timer.

On-Delay Timer: Starts working when the IN value is set to 1. When the elapsed time (ET) is greater or equal than the preset time (PT), Q becomes 1.

4) Boolean Operations

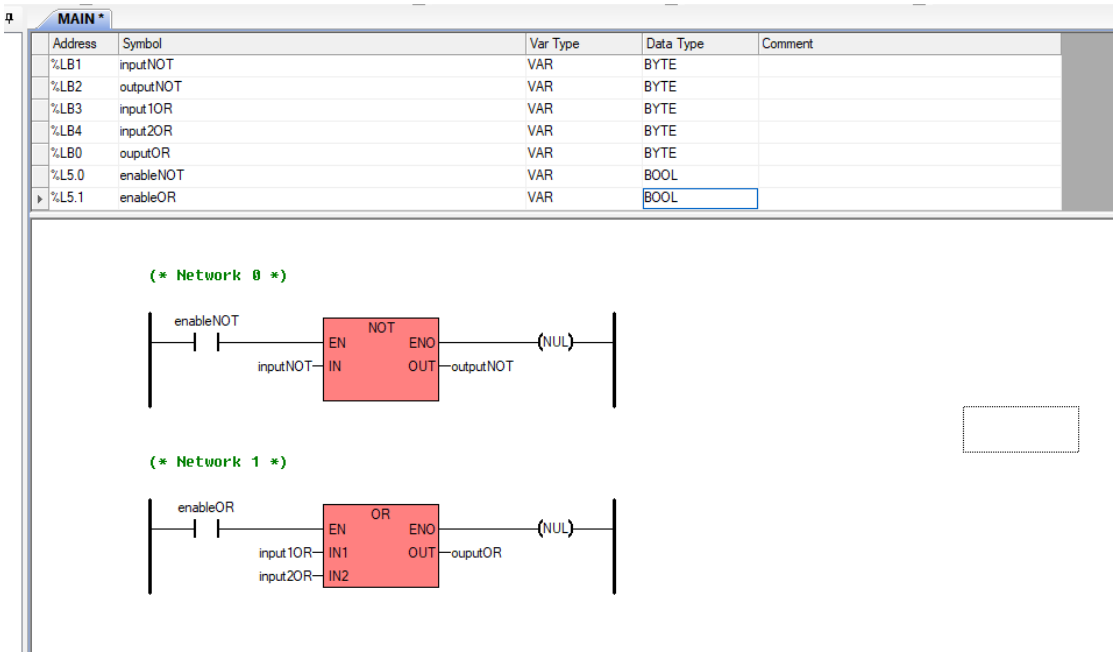


Figure 11: Boolean Operations

Boolean operations like AND, OR, NOT, XOR are feasible with the help of predefined components like the ones presented above

5) Move blocks

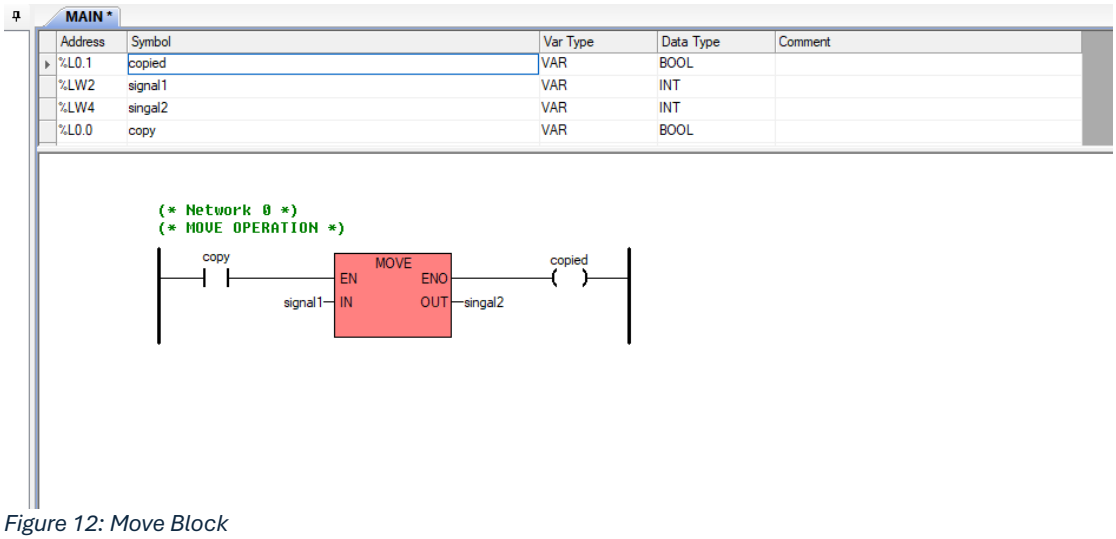


Figure 12: Move Block

MOVE block: When the EN signal is set to 1, the signal in the input is copied to the output one. Signals must be of the same type.

6) Conditional blocks

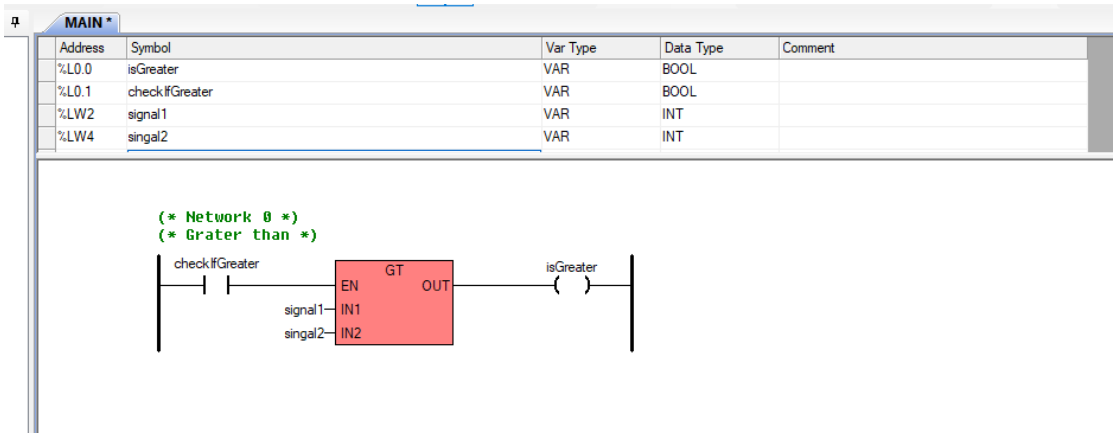


Figure 13: Conditional Blocks

Types of conditional blocks: greater than, greater or equal, equal, less than, less or equal.

In Figure 13 (Greater Than Block), if IN1 is greater than IN2, output is set to 1.

7) Arithmetic blocks

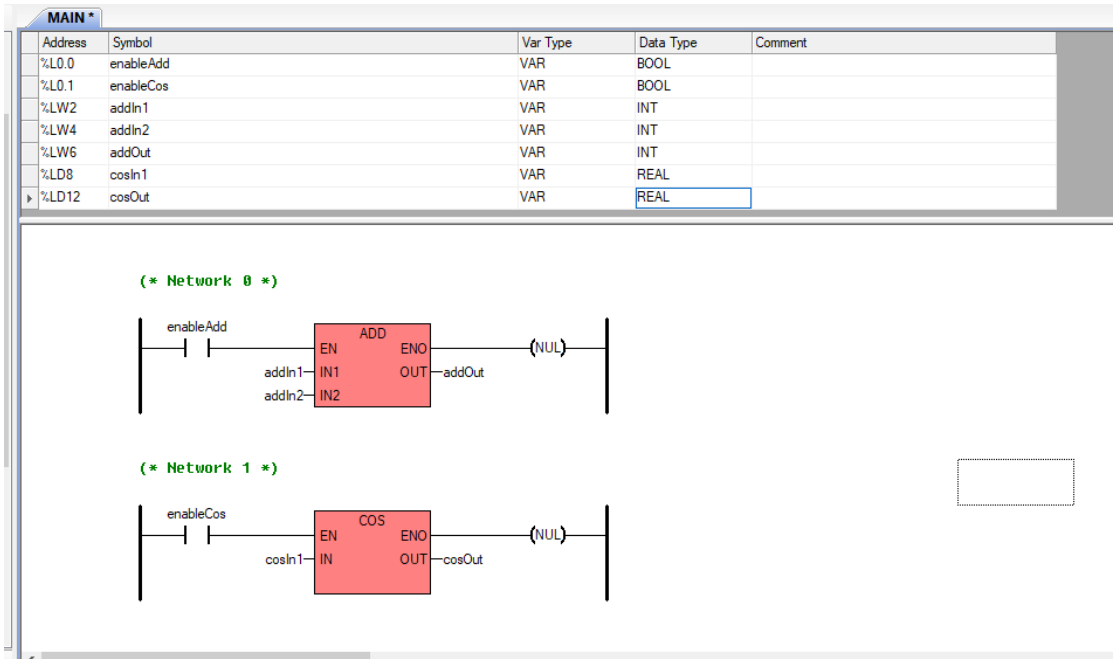


Figure 14: Arithmetic Blocks

Arithmetic operations like adding, subtracting, dividing, multiplying are available by using the appropriate arithmetic blocks.

Functional Block Diagram

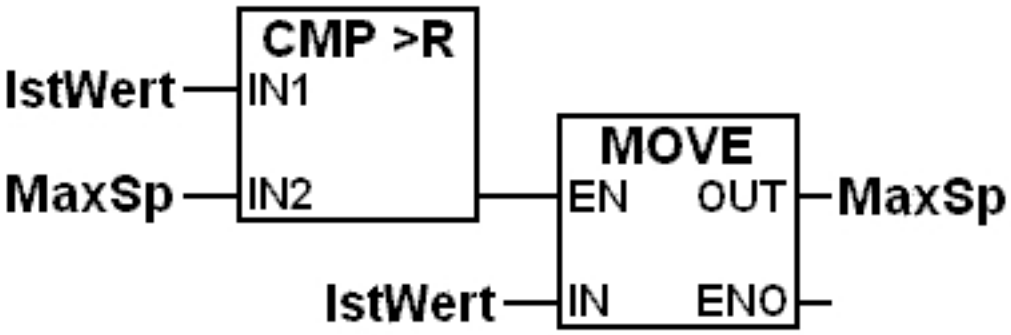


Figure 15: Simple Functional Block Diagram [13]

The function block diagram (FBD) is a graphical language for programmable logic controller design, that can describe the function between input variables and output variables. A function is described as a set of elementary blocks. Input and output variables are connected to blocks by connection lines.

Structured text

```
PROGRAM POU
VAR
MotorOverload : ARRAY[1..3] OF BOOL;
MotorOverCurrent : ARRAY[1..3] OF BOOL;
MotorPBSTart : ARRAY[1..3] OF BOOL;
MotorPBStop : ARRAY[1..3] OF BOOL;
MotorCoil : ARRAY[1..3] OF BOOL;
i : INT;

END_VAR
```

```
FOR i:=1 TO 3 BY 1 DO

IF  MotorPBStop[i] OR MotorOverload[i] OR MotorOverCurrent[i] THEN
    MotorCoil[i]:=0;
ELSIF MotorPBSTart[i] THEN
    MotorCoil[i]:=1;
END_IF
END_FOR
```

Figure 16: Structured Text Example [5]

Structured Text programming is not graphical as you can see in Figure 16. It is a scripting language which makes the creation of conditions and boolean decisions an easy task. Structure text is suitable for complex logic and can combined with ladder logic when input/output control is needed. [14]

Instruction List

```
   LD      Speed
           GT      2000
           JMPCN   VOLTS_OK
           LD      Volts
VOLTS_OK   LD      1
           ST      %Q75
```

Instruction List is an assembly like programming language where program control is achieved by jump instructions and function calls [14].

Sequential Function Chart

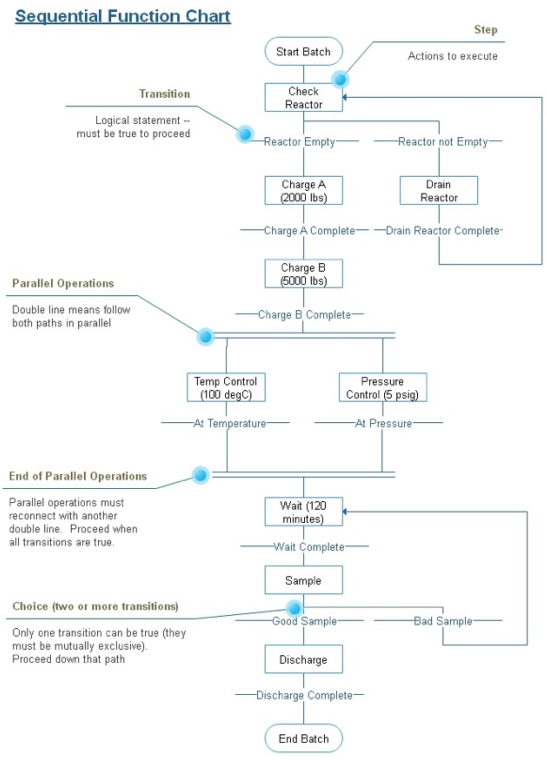


Figure 17: SFC Example [15]

SFC is used to program processes that can be split into steps.

Main components of SFC are [15]:

- Steps with associated actions.
- Transitions with associated logic conditions.
- Directed links between steps and transitions.

1.4 HVAC Systems – Heat Pump

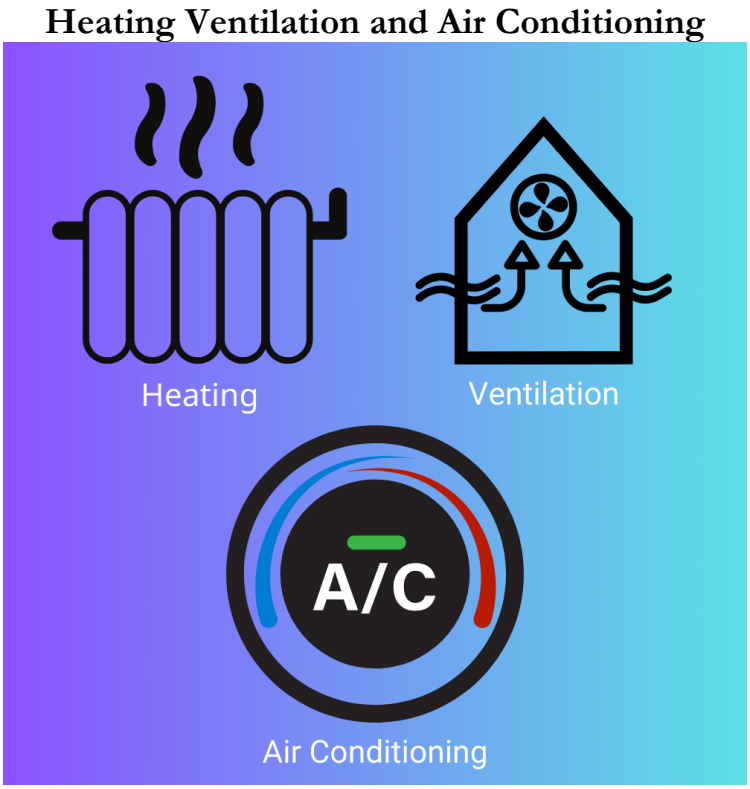


Figure 18: HVAC

Heating, ventilation, and air conditioning (HVAC) systems have the primary goal of preserving a comfortable environment by keeping parameters like temperature, humidity and air flow at certain acceptable comfort ranges. Normally, these are set by the user and in most cases include temperatures between 20 – 24 °C during winter and 23 – 26 °C in summer [16].

HVAC Components [17]

- **Thermostat:** It is responsible for getting temperature setpoints from the user and keeping the temperature to these values.
- **Heat Generator:** A device that converts heat to electricity in case of an electrical outage.
- **Heat Exchanger:** It is used for transferring thermal energy from one source to another
- **Blower Motor:** Manages airflow by changing the fan speed.
- **Condenser Coil / Compressor:** Compresses gas to greater pressure, increasing also its temperature.
- **Evaporator Coil:** Evaporates compressed liquid with low boiling point to cool it down.
- **Air Ducts and Vents:** Control the direction and speed of airflow.
- **Combustion Chamber:** Ignites fuel and turns it into energy.

HEAT PUMPS

One particularly famous component of modern HVAC systems is the heat pump. A heat pump extracts heat from sources like the ground or air and transfers it to where it is needed. It is used for both cooling and heating, relying mainly on components like compressors, valves, evaporators and heat exchangers.

What makes them particularly suitable for modern standards is their high coefficient of performance (COP). They outperform conventional heating technologies, such as boilers or electric heaters by 3-5 times. This means they consume less power, they have smaller CO2 footprint, and they reduce running costs [18].

Heating Mode:

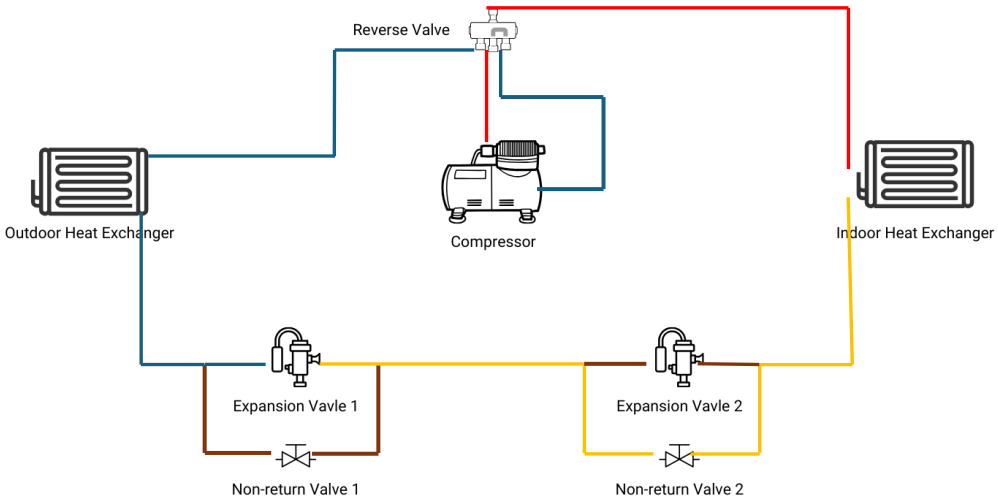


Figure 19: Heating Mode of Heat Pump

How heating mode is achieved:

1) Refrigerant leaves the outdoor heat exchanger with low pressure and low temperature.
2) It heads to the compressor where is compressed to higher temperature and transforms to superheated vapor.
3) Then it exchanges thermal energy with cool air passing through the indoor heat exchange. It keeps its high pressure, slightly decreasing its temperature and transforming to liquid.
4) It passes through expansion valve 1 becoming part a mix of vapor and liquid. Temperatures is reduced and it is dissipated through the outdoor heat exchanger.

Cooling Mode:

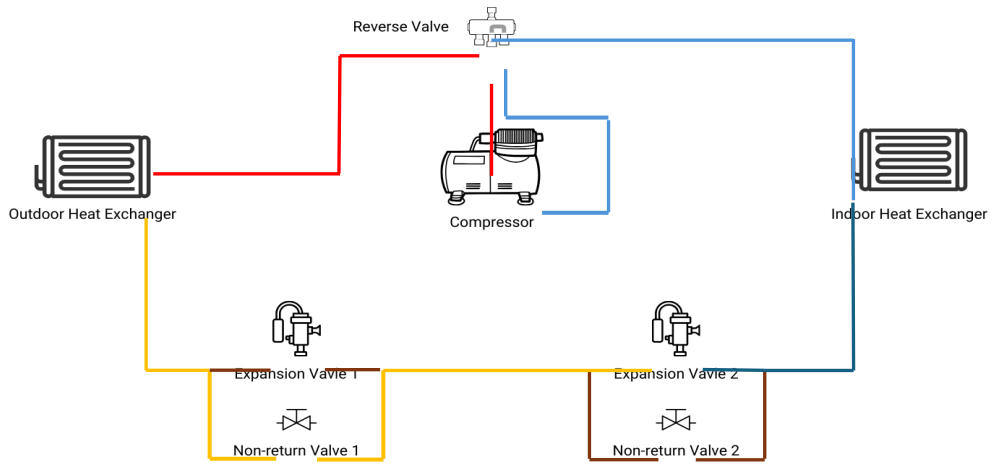


Figure 20: Cooling Mode of Heat Pump

How cooling mode is achieved:

1) Refrigerant leaves compressor with high pressure and high temperature.
2) It heads to the outdoor heat exchanger where it loses some heat, turning into liquid and keeping its high pressure.
3) Then it passes through the second expansion valve becoming a mixture of liquid and vapor, dropping its temperature as well as it pressure.
4) With the help of a fan, warm air is blown through the indoor heat exchanger, slightly dropping the temperature of the refrigerant.
5) Finally, the liquid is transferred to the compressor which is responsible for increasing its temperature and pressure.

1.5 Heat Transfer

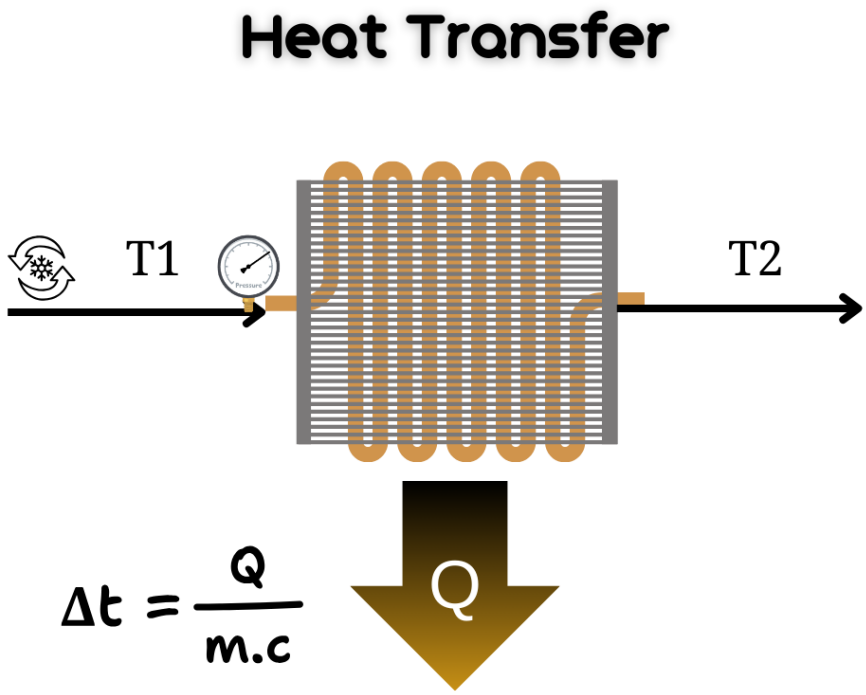


Figure 21: Heat Transfer

Based on the First Law of Thermodynamics for a closed system, heat transfer equation takes the following form [19]:

$$Q = Wk + \frac{dU}{dt} \quad (23)$$

Where Q is the heat transfer rate and Wk the work transfer rate both expressed in joules per second (J/s) or watts. The derivative is the rate of change of internal thermal energy.

The above equation, when applied to a close system with constant pressure and liquid flowing through a means like the one in figure 21, transforms to the following format:

$$Q = \dot{m} \times Cp \times \Delta T. \quad (24)$$

Q: Heat transfer rate
$\dot{m}$: mass flow rate of the fluid in kg/s
Cp: specific heat capacity of the fluid under constant pressure
ΔT: temperature change

MATLAB Implementation

One of the key components of the Building Management System (BMS) built in this thesis is the implementation of various control sub-systems using MATLAB. MATLAB handles several crucial tasks like:

1) **Thermal Model Creation**: Creates the building's thermal model.
2) **Initialization Data Storage:** Stores multiple initialization data sets including those needed for the building model.
3) **Forecasts Handling:** Manages the temperature, solar radiation as well as electricity price forecasts
4) **Modbus Communication:** Establishes communication via Modbus TCP with the real time controller implemented in the HMI-PLC and handles every aspect of it like the Modbus connection initialization, read-write operations, Modbus address mapping and more.

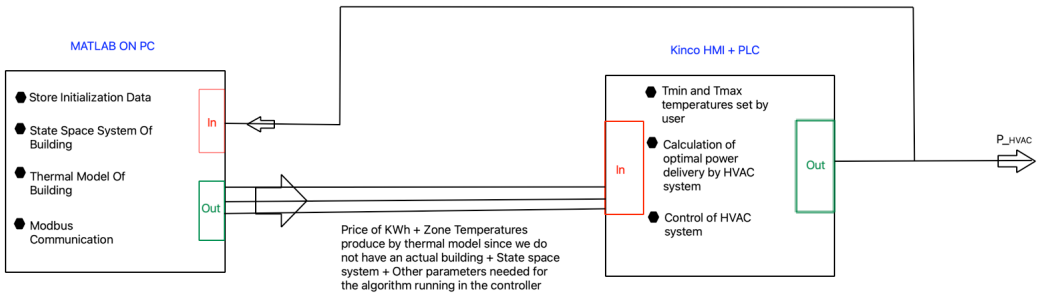


Figure 22: MATLAB Integration With HMI-PLC And HVAC Control

Figure 22 presents an abstract view of the main BMS components and their interaction, focusing mainly to the one established between MATLAB and the controller. This interaction is achieved with the use of Modbus protocol. The controller manages the HVAC systems and provides feedback to MATLAB, which uses this information to generate theoretical zone temperatures.

MATLAB Implementation Breakdown

The main functionalities programmed on MATLAB are presented below:

a) Building Model Initialization

- This involves an executable that produces the state space system as discussed in section 1.1 and expressed by equation (7):

$$\frac{d\boldsymbol{T_{in}(t)}}{\boldsymbol{dt}} = \boldsymbol{A_b} \cdot \boldsymbol{T_{in}(t)} + \boldsymbol{B_b} \cdot \boldsymbol{U}$$

b) Building Thermal Model

This is also an executable which takes the following **parameters**:

- **N_z**: Number of the total building's thermal zones.
- **N_z_f**: Number of thermal zones per floor.
- **Q_HVAC_z**: Power consumption of HVAC systems set by the output of the controller.
- **Q_in_office**: Internal heat gains from people, lightning and appliances.
- **Ad,Bd**: State space system parameters produced by the building model initialization executable.
- **Tin_z**: Internal temperature of each thermal zone
- **Tout_ambient**: Ambient temperature. Prediction from previous day or actual temperature measured at real time.
- **IT**: Total solar radiation.

And produces the following **output**:

- **Tin_z**: Temperatures of available thermal zones based on the model of the building.

It acts as a simulation of the thermal behavior of a building with the parameters specified above. It solves the state space system with the pre-calculated A, B and calculates the theoretical temperatures of the thermal zones.

c) Initialization File

- There is also an initialization file that keeps track of all the parameters that could affect the thermal behavior of a building, ranging from the heat transfer coefficient of the walls to the number of people and their heat load. Apparently, this file is subject to many changes especially when migrating from one building to another or dynamically monitor the ambient temperature.
  Various static values like the solar radiation or energy prices could be replaced by real-time values obtained from Web API's or sensors.

d) HMI Register Addresses

- When connecting to a PLC or HMI through Modbus, it is necessary to identify which Modbus addresses correspond to which HMI or PLC internal register addresses. This is clarified in the documentations provided by the manufacturer. When it comes to Kinco, the Modbus address range of 1-9000 corresponds to the local address range 0-8999 of the HMI. So, a mapping of the correct addressing has been implemented on MATLAB with the following specifications.

	Used HMI Register Addresses	Corresponding Modbus Registers
EpMin	**LW 16**	**LW 16 + 1**
EpMax	**LW 18**	**LW 18 + 1**
Ep	**LW 12**	**LW 12 + 1**
TinBuildAvg	**LW 14**	**LW 14 + 1**
TinZones	**LW 20 - 109 (for 90 zones)**	**LW [20 + 1… 109 + 1] (for 90 zones)**

NumOfZones	LW 200	LW 200 + 1
Q_HVAC_z_plc	LW 110 - 199 (for 90 zones)	LW [110 + 1 … 199 + 1] (for 90 zones)
executeMacro	LB 1000	LB 1000 + 1

LW: Word – 16bit register. Used for signed and unsigned integers.
LB: Bit register.

TinZones requires the occupation of 90 HMI registers for 90 zones. If an additional of 10 zones are needed (bringing the total to 100), ensure the registers 110 to 119 are not occupied by any other variables.
While addresses 110 to 119 are the most consistent choice, any other available addresses-such as 300 to 309 or any other unassigned combination-can also be used.

e) Modbus Initialization

Modbus initialization function creates a Modbus object by specifying the protocol type, server and port.

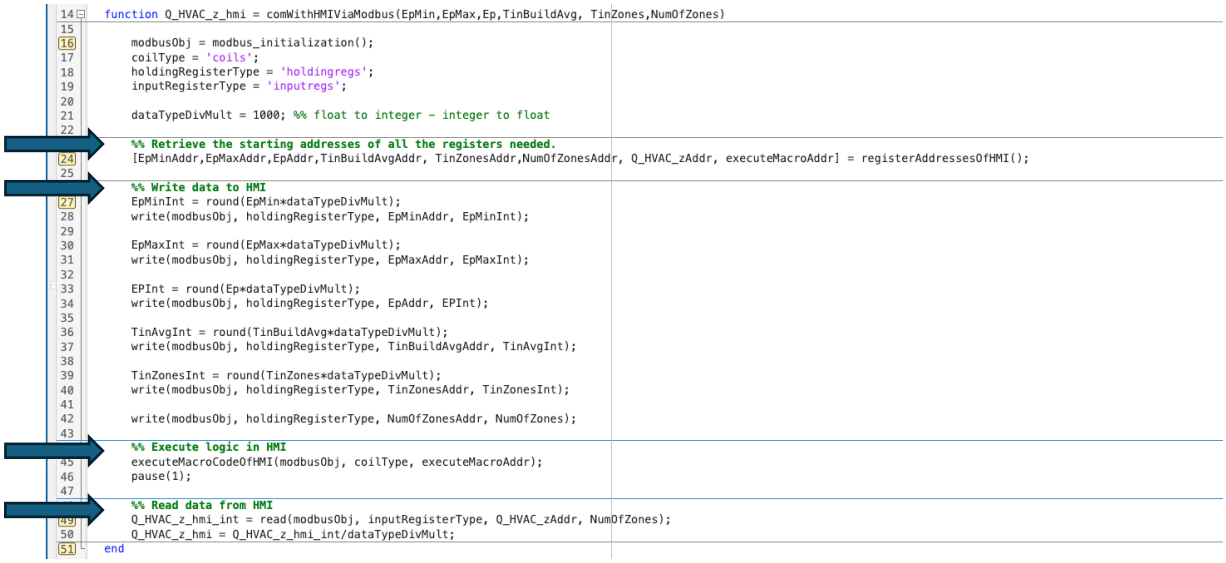
```
function m = modbus_initialization()
    modbustype = 'tcpip';
    modbusserver = '192.168.0.100';
    modbusport = 502;

    m = modbus(modbustype, modbusserver, modbusport);
    return;
end
```

In this implementation:

- The **Modbus port** is set to **502**
- The **server address** is configured by the **HMI-PLC**
- The **protocol type** used is **TCP/IP**

Communication with HMI via Modbus:

When connecting MATLAB to HMI, MATLAB acts as the master and HMI as the slave. This means that MATLAB initiates the communication, sends data and retrieves feedback. When new data, such as temperature measurements or electricity prices, become available, the master (MATLAB) performs the following steps:



```
function Q_HVAC_z_hmi = comWithHMIViaModbus(EpMin,EpMax,Ep,TinBuildAvg, TinZones,NumOfZones)

    modbusObj = modbus_initialization();
    coilType = 'coils';
    holdingRegisterType = 'holdingregs';
    inputRegisterType = 'inputregs';

    dataTypeDivMult = 1000; %% float to integer - integer to float

    %% Retrieve the starting addresses of all the registers needed.
    [EpMinAddr,EpMaxAddr,EpAddr,TinBuildAvgAddr, TinZonesAddr,NumOfZonesAddr, Q_HVAC_zAddr, executeMacroAddr] = registerAddressesOfHMI();

    %% Write data to HMI
    EpMinInt = round(EpMin*dataTypeDivMult);
    write(modbusObj, holdingRegisterType, EpMinAddr, EpMinInt);

    EpMaxInt = round(EpMax*dataTypeDivMult);
    write(modbusObj, holdingRegisterType, EpMaxAddr, EpMaxInt);

    EPInt = round(Ep*dataTypeDivMult);
    write(modbusObj, holdingRegisterType, EpAddr, EPInt);

    TinAvgInt = round(TinBuildAvg*dataTypeDivMult);
    write(modbusObj, holdingRegisterType, TinBuildAvgAddr, TinAvgInt);

    TinZonesInt = round(TinZones*dataTypeDivMult);
    write(modbusObj, holdingRegisterType, TinZonesAddr, TinZonesInt);

    write(modbusObj, holdingRegisterType, NumOfZonesAddr, NumOfZones);

    %% Execute logic in HMI
    executeMacroCodeOfHMI(modbusObj, coilType, executeMacroAddr);
    pause(1);

    %% Read data from HMI
    Q_HVAC_z_hmi_int = read(modbusObj, inputRegisterType, Q_HVAC_zAddr, NumOfZones);
    Q_HVAC_z_hmi = Q_HVAC_z_hmi_int/dataTypeDivMult;
end
```

Figure 23: Matlab Communication Steps With HMI

1. **Initiates Communication**: MATLAB starts the communication and retrieves the register addresses of the required ones.

2. **Sends Data**: MATLAB sends data to HMI using the **write()** method. Since the Modbus protocol primarily communicated through 16-bit registers, but floating-point numbers require 32 bits, a transformation is applied before transmission.

    - **In MATLAB**:
        - **Floating-point** values are **multiplied by** a **base value** to **convert** them to **integers**
        - **Base value selection**: For example, if the min and max values of a variable are 0.00 and 100.00 and **precision of two decimals** is required, the base value would be **100**.
        - The transformed integer value values are then transmitted to the HMI
    - **In the HMI**
        - Upon receiving the integer values, the original floating-point values are constructed by **dividing** with the **base value**

3. **Signals HMI**: MATLAB instructs the HMI to process the new data, execute its logic and generate outputs. To achieve this, it changes the logical value of a bit register triggering in this way an event. This event is interpreted by HMI following the execution of the written logic. In another words, changing the bit register from 0 to 1, triggers an event and the event executes the macro code.

4. **Requests Output**: After the output is created, MATLAB requests this data using the write method and passes it to the thermal model of building for the new zone temperatures to be generated.

f) Macro Code Execution

The following method changes the logical value of a bit register in the HMI. Such changes can be expressed as events in HMI. An event can trigger the macro code execution, where macro code is a special feature in HMI that allows programmers to write their own logic in C.

```
function executeMacroCodeOfHMI(modbusObj,coilType,executeMacroAddr)
    write(modbusObj, coilType, executeMacroAddr, 0);
    write(modbusObj, coilType, executeMacroAddr, 1);
end
```

g) Control of the main logic

A main function has been created to pack all the logic in one place and manage the flow of actions:

1) Initializes variables by calling initialization functions.
2) Calls Modbus communication function
3) Gets the output results from HMI and runs the thermal building model to get the zone temperatures
4) Stores them
5) Runs again step 2-4

```
Q_HVAC_z(:,t) =
comWithHMIViaModbus(EP_min,EP_max,EP(t),Tin_b_avg(t),Tin_z(:,t),N_z).';

Tin_z =
building_thermal_model(N_z,N_z_f,Q_HVAC_z,t,Q_in_office,Ad,Bd,Tin_z,Tout_ambient,IT);

Tin_b_avg(t+1) = sum(Tin_z(:,t+1))/N_z;
```

HMI Implementation

3.1 Basics

The HMI plays the main role of controlling the HVAC system. It takes care of registering the minimum and maximum preferable temperature setpoints by the administrator of the BMS system. Also, it informs the user in real time about various parameters like the electricity price, the zone temperatures and handles the optimization of power delivery. It communicates directly with the PLC unit and sends the necessary data for all thermal zones.

The HMI unit that is used in this thesis, is the MK043E-27DT from Kinco. It has the advantage of integrating in one unit both the HMI and PLC. So, there is no need for further connections.

Main Touch Screen:
This is the main touch screen of the HMI. It can be configured to meet the necessary requirements.

Figure 24: Touch Screen

Back Side (Used Both from HMI and PLC):

- Ethernet port for Modbus TCP connection (1)
- USB Type C for programming (2)
- COM Port: RS-485 (3)
- P, W, R: Power delivery (4)

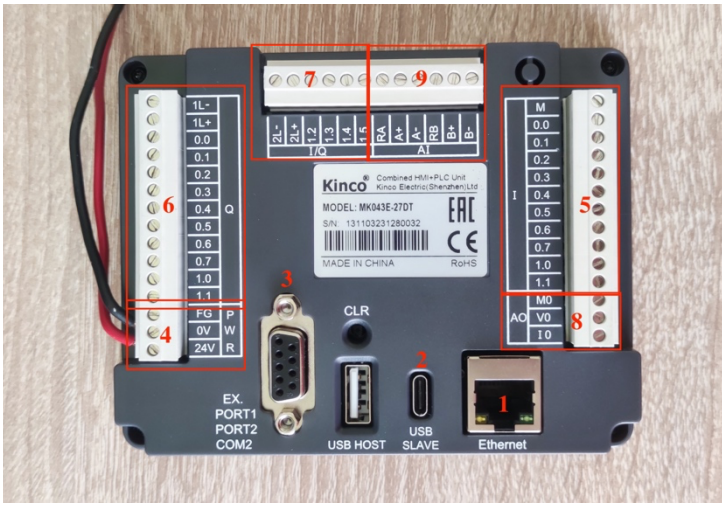


Figure 25: Back Side of HMI & PLC

The remaining connection ports are used only by the **PLC**.

- Digital Inputs (5)
- Digital Outputs (6)
- Digital I/O (7)
- Analog Outputs (8)
- Analog Inputs (9)

HMI Specifications:

HMI Specification											
Display	Color	Resolution	Backlight	Brightness	Port	Touch Panel	Backlight life	Memory	Recipe memory	Expandable memory	Program download
7" TFT	256K Colors	800*480	LED	250cd/m²	--	4-wire precision resistance network	50000 hours	128M Flash +64M DDR	256KB+RTC	1 USB Host	1 USB Slave /Ethernet
4.3" TFT		480*272		400cd/m²	1*RS232 (Only MK043E-27DT supported)						

Figure 26: HMI Specifications [20]

The HMI is programmable by using the *DTools* software provided by Kinco. More about the Dtools software in the next pages.

The HMI implementation is split into two parts: **Graphical User Interface & Macro Code**

3.2 Basic Configurations

When creating a new project using DTools software there are some steps that should be followed to make things easier.

1) **Read the user manuals**
2) **Manage the connection configurations for Modbus, RS-485 etc.**
3) **Get comfortable with the app interface.**
4) **Understand the basic idea of registers and their addressing.**
5) **Create a basic graphical user interface using all the tools provided by the app**
6) **Add functionalities like event handling, login security, macro code execution if necessary.**
7) **Compile, download the project to the HMI and test functionality.**

Below you can see the main configuration page which lets you choose the preferable HMI and PLC and manage their connections. You can navigate to this page by clicking to the name of the project in the Project structure window.

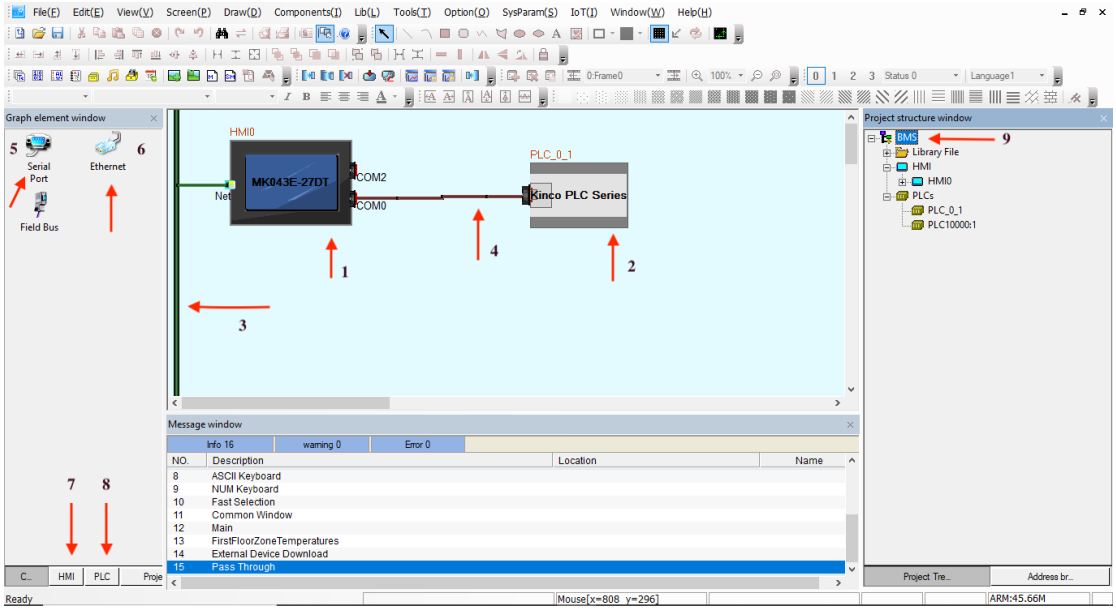


Figure 27: HMI-PLC connections page

1. **HMI**
2. **PLC**
3. **Ethernet Connection**
4. **Serial Communication**
5. **Serial Port option**
6. **Ethernet Connection option**
7. **List of HMIs**
8. **List of PLCs**
9. **Click there to open the connections page**

- **How to add your HMI to the connection page:**

Click at option 7, search for the appropriate HMI and drag and drop in the main window.

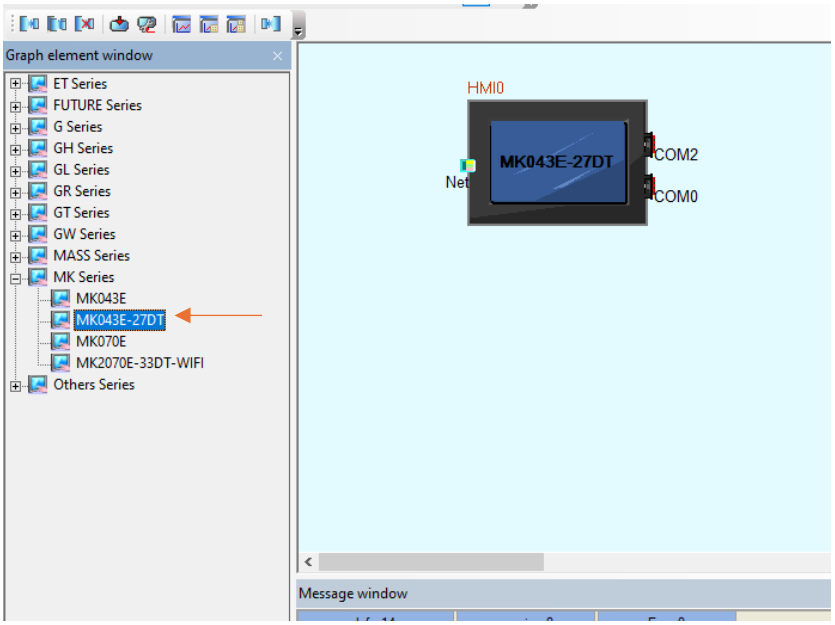


Figure 28: HMI Import

- **How to add your PLC to the connection page:**

Click at option 8, search for the appropriate PLC and drag and drop in the main window.

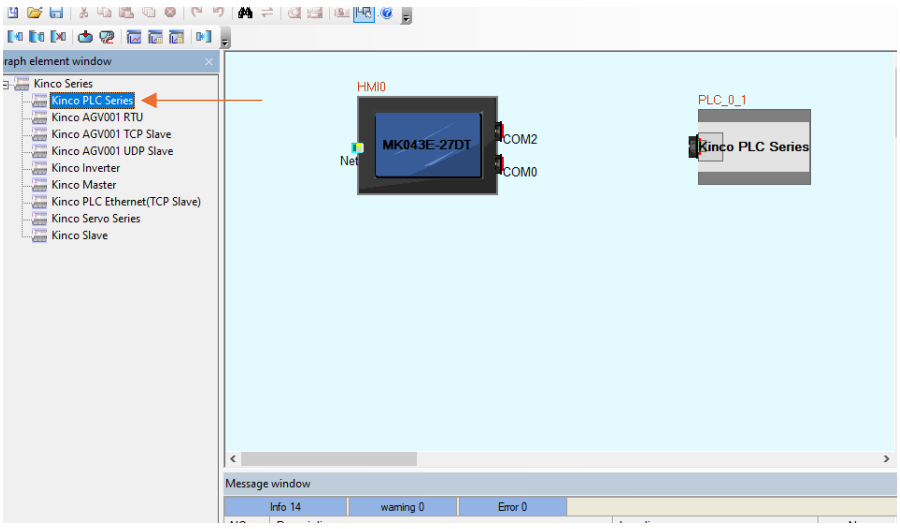


Figure 199: PLC Import

- **How to add Ethernet Connection support to the HMI:**

Click at option 6 and drag and drop next to the HMI. Then set the preferred connection configurations. In our case, HMI is the slave and MATLAB-PC the master. The connection protocol is Modbus TCP with the default port of 502. The IP address is 192.168.0.100.
An important side note: master and slave must be in the **same subnetwork**.

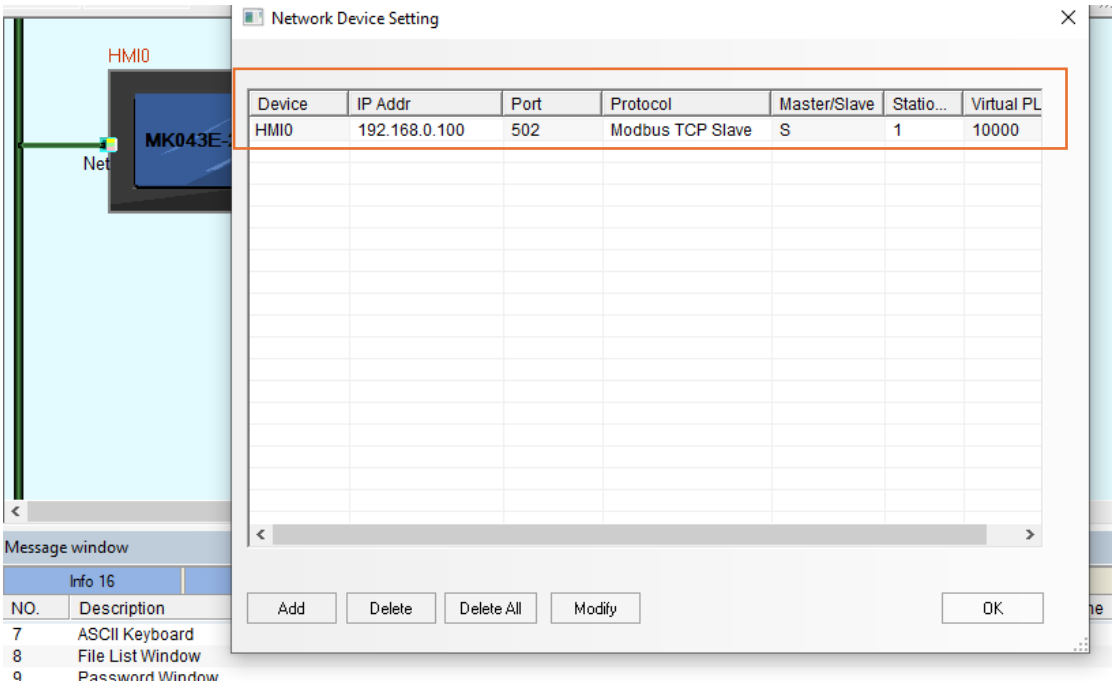


Figure 30: Ethernet Connection Configuration

- **How to connect HMI with PLC:**

Click at option 5 and drag and drop in the main screen. Then connect the two edges of the line to the HMI and PLC.

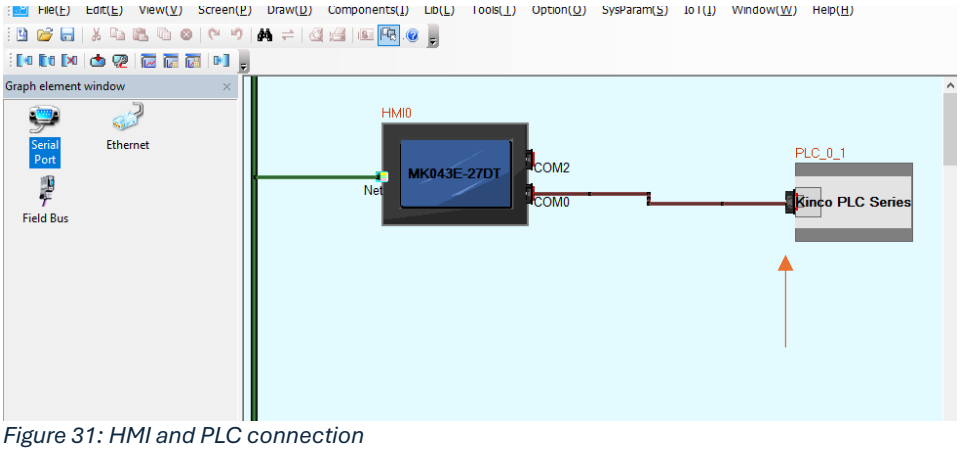


Figure 31: HMI and PLC connection

- **How to access general HMI configurations:**

  Double click on the icon of the HMI and the HMI Attribute window will open. There you can make several adjustments like for example setting security levels.
  Security levels are groups of functionalities and pages that meet the same security requirements and may need or not password verification.

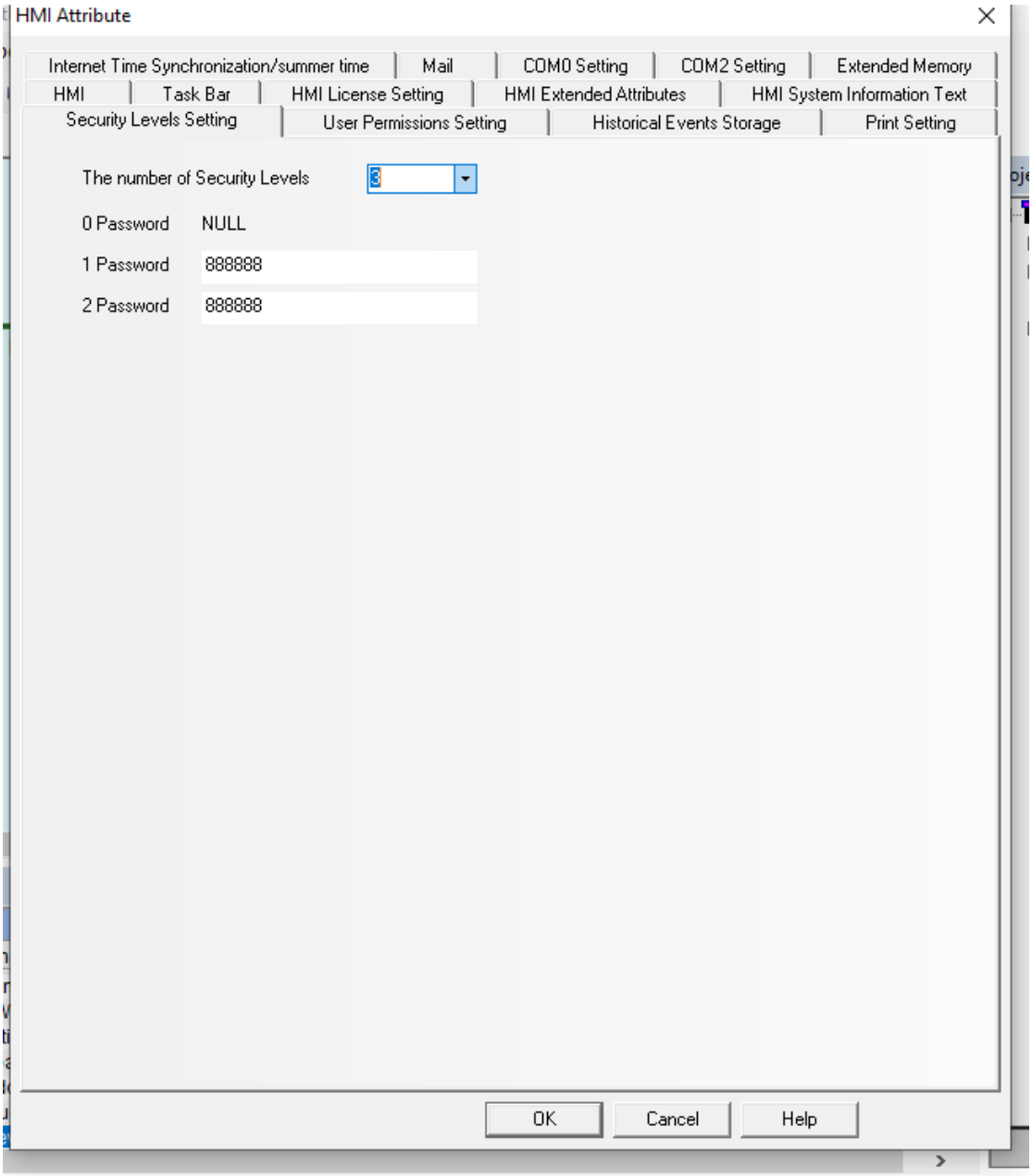
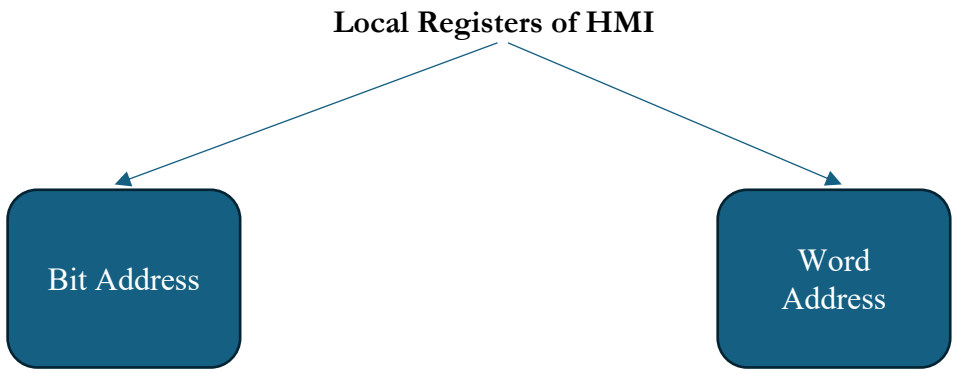


Figure 32: General HMI configuration page

3.3 Data Types – Address Registers

Kinco HMIs like the one used in this project, support the following data types when dealing with programming in C language (Macro Code).

Data Type	Data Length	Description
Bit	1bit	Bit variable, 0 and 1
Signed Short	1word (16bits)	Signed short integer variable, -2^{15}~$(2^{15}-1)$
Unsigned short	1word (16bits)	Unsigned short integer variable, 0~$(2^{16}-1)$
Signed int	2word (32bits)	Signed integer variable, -2^{31}~$(2^{31}-1)$
Unsigned int	2word (32bits)	Unsigned integer variable, 0~$(2^{32}-1)$
Float	2word (32bits)	Single float variable
Double	4word (64bits)	Double float variable



Local HMI registers are slit into these two categories. The user can also import external registers (**M, D, S from PLC**) to the DTools environment by simply making use of the Import functionality in the Address Tag page.

Bit Address registers are also categorized in **LB, LW.B, ELW.B, RB, RBI, FRB, FRBI**. Same with the Word Address registers which are categorized in **LW, ELW, RW, RWI, ERW0~ ERW2, ERWI0~ ERWI2, FRW, FRWI**.

From all these registers, the ones that are the most important are **LB** and **LW** which stand for Local Bit and Local Word respectively. They are used to reference the control devices and do not save data after power off. In simple terms, for each device used in HMI that must be controlled and get referenced by other devices, you must assign a register address that will store any value needed for each operation.

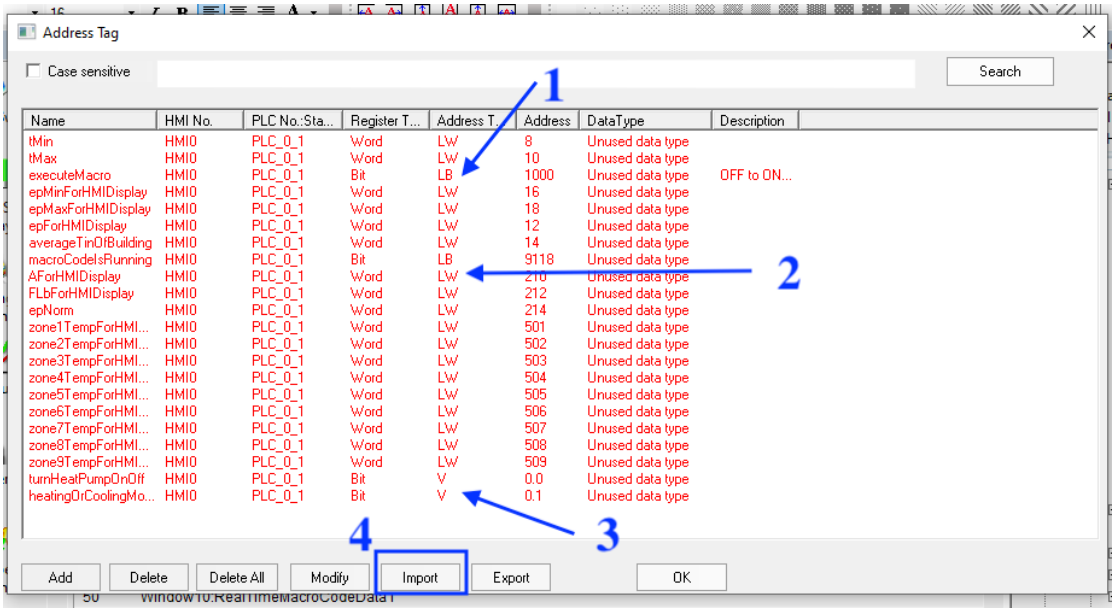


Figure 33: Address Tag Page

The figure above displays the **Address Tag** page in the DTools environment where you can assign names to specific registers for easy reference throughout the current project. For each register, a name and an address type should be provided with the option of adding a description.

1. The first vector in the figure points to the variable "*executeMacro*" which has an address type of LB. This means it can only take values of 0 and 1.

2. The second vector points to the "*AForHMIDisplay*" register, which is of type LW, indicating that it can support data types up to 16bits.

3. The third vector highlights a V variable imported by the PLC.

4. At the bottom of the picture (part 4), there is a button for importing external registers. This is done by selecting the appropriate .CSV file exported by the PLC software.

There is also a variety of *System Special* registers already reserved to serve special purpose like increasing the LCD contrast, displaying the screen brightness and many more.

How to link an element to a register:

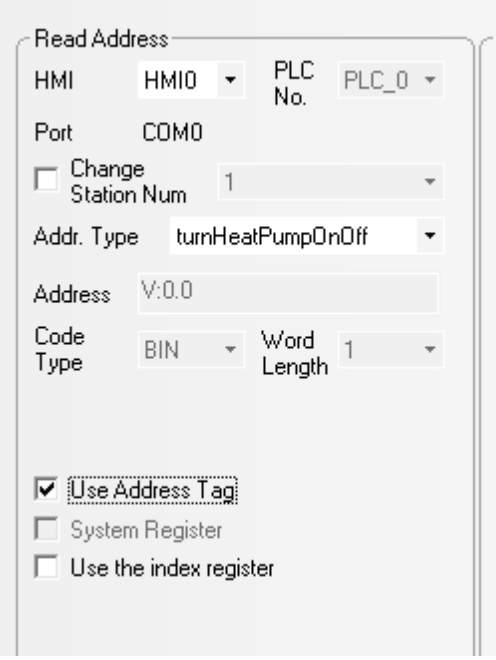


Figure 34: Address Reference

Everything comes to this point where, by clicking to any element, for example a button, you come across this pop-up window. This window gives you the opportunity to set an address to the selected element and store its produced value inside the referenced register. In this example, the checkbox "Use Address Tag" is selected to make use of the registers set in the Address Tag and more specifically of the "*turnHeatPumpOnOff*" one. In the Address section, the type of register is also displayed (V: external PLC register). You can also define a custom register address by selecting the "Use the index register" option.

So, if the selected element is a button, the "*turnHeatPumpOnOff*" is a PLC variable for controlling the start and stop of a heat pump, when the button is pressed, the V:0.0 register turns to 1 and the heat pump starts operating. When the button is not pressed, the register turns to 0 and the heat pump stops working.

3.4 Graphical User Interface

DTools comes equipped with various functionalities to make GUI design easier. It is organized in five main parts.

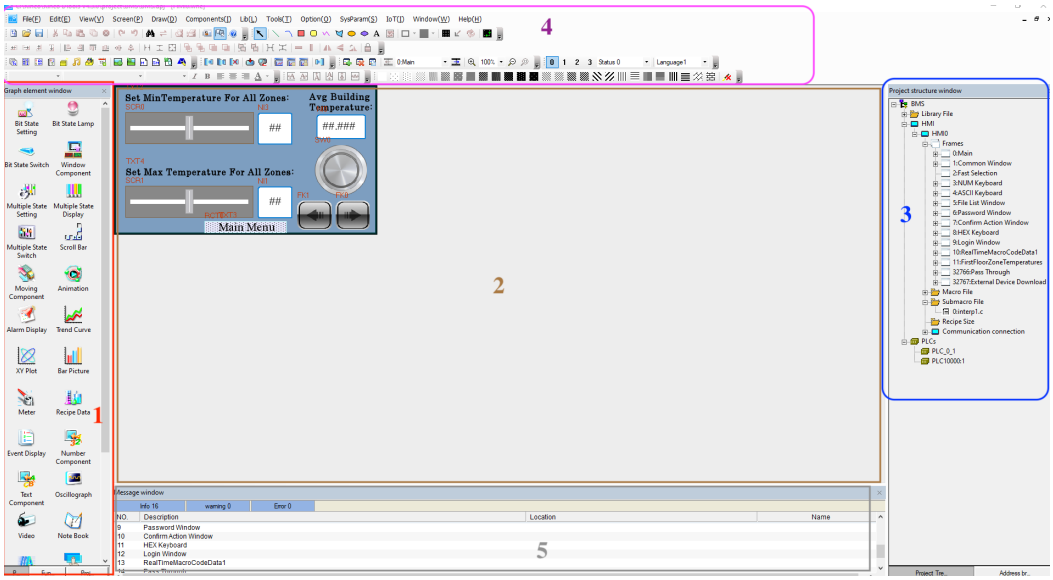


Figure 35: DTools GUI Design Window Overview

Main parts of GUIS design window:

1. **Graphical element window**

The graphical element window is consisted of three sub-windows. Each window hosts a variety of GUI design elements as well as PLC specific functionalities for data management. The most common GUI design elements are buttons, input fields, function keys, images, plot windows and many others. To use them just drag and drop in the main screen (2).

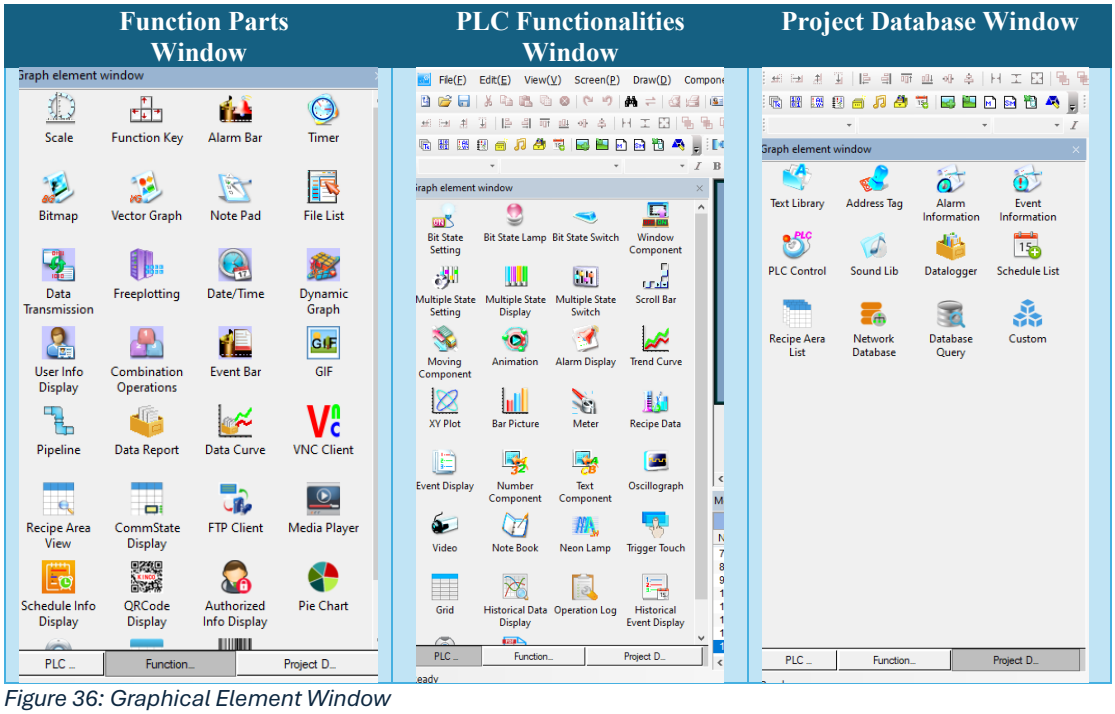


Figure 36: Graphical Element Window

2. Main Design Window

This is the main design window where the user can drag and drop any item and create the desired graphical user interface.

3. Project Structure Window

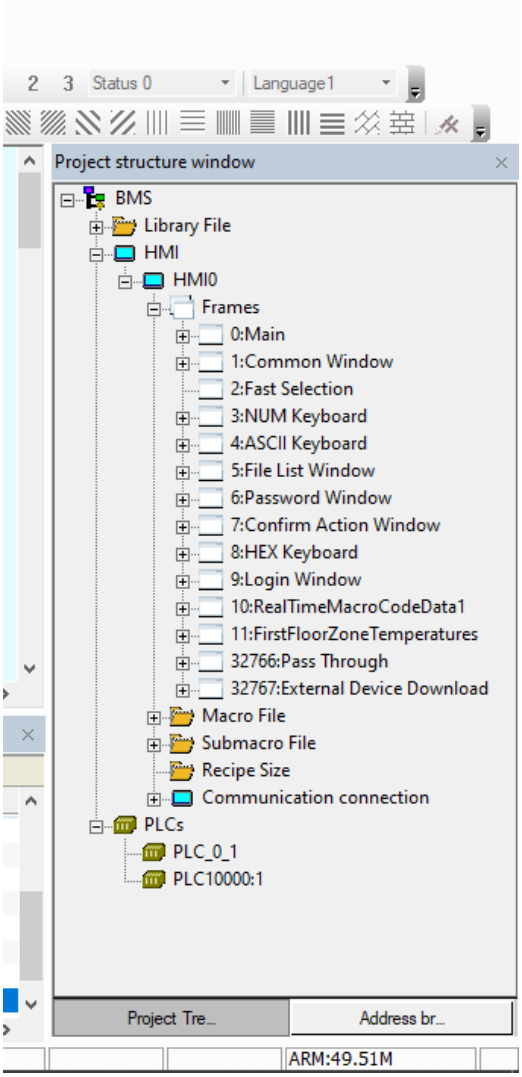


Figure 37: Project Structure Winow

The project structure window presents the structure of the project and helps the user to easily navigate through the HMI configurations, the connection panel, the PLC and the macro code. As shown in the figure above there is also a section called Frames. These are the main HMI pages and GUI components commonly used across various pages.

4. Task Bar

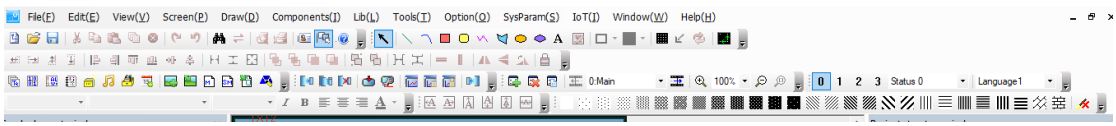


Figure 38: Task Bar

The task bar offers a variety of functionalities like creating custom GUI items, zooming in or out, compiling the project and downloading it to the connected HMI.

5. Message Window

The message window displays valuable message like errors, warnings and success messages during the compilation and download process of the created project.

Implemented Frames in this Thesis

1) Main Menu

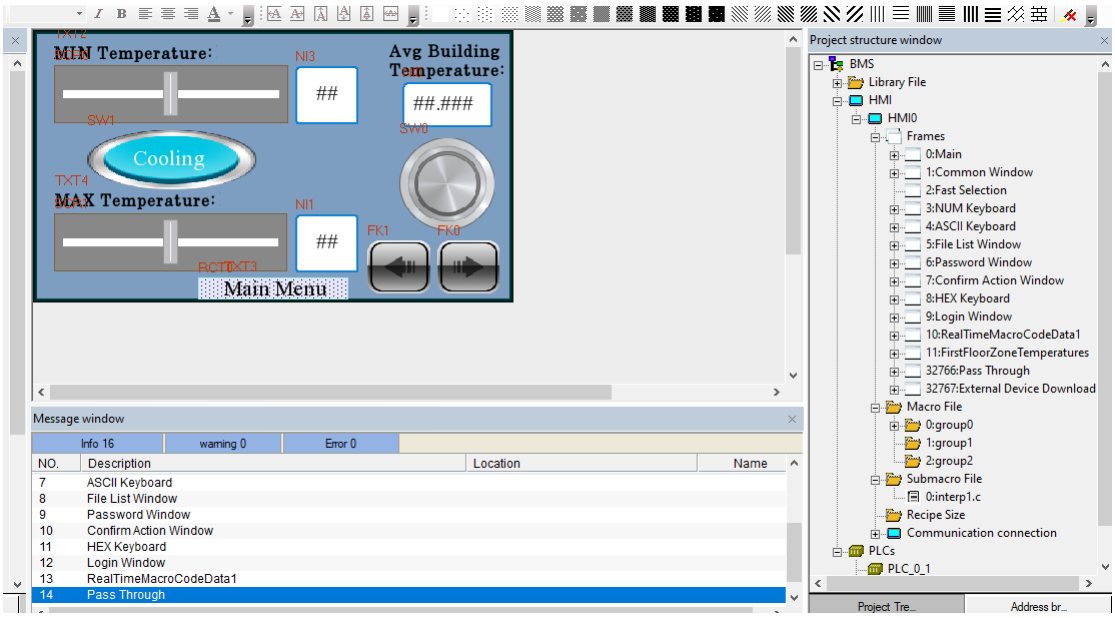


Figure 39: Main Menu Page

The main menu page illustrated above, is designed to facilitate the user's control of the building's HVAC system. This page features two scroll bars for controlling the setpoints of the minimum and maximum temperature values inside the building. These scroll bars are accompanied by input boxes allowing the user to manually enter the desirable temperature and clearly see the values entered.

Additionally, the interface includes an average building temperature indicator. Users can easily start or stop the HVAC system and toggle between heating and cooling modes.

At the bottom right corner of the screen, there are also two buttons that enable the user to navigate through the available windows.

2) Macro Code Parameters

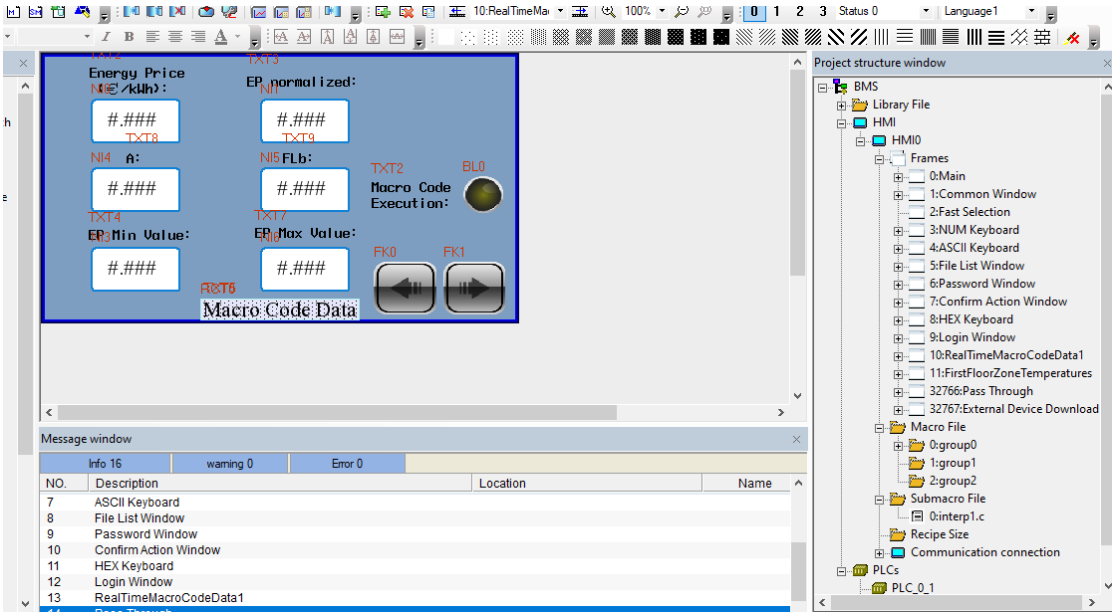


Figure 40: Macro Code Page

This page is dedicated to the macro code implementation and to parameters affecting the decisions of the power management algorithm like the current energy price.

There is also an indicator which turns green each time the macro code execution is invoked by MATLAB.

At the bottom right corner of the screen, there are also two buttons that enable the user to navigate through the available windows.

3) Zone temperatures

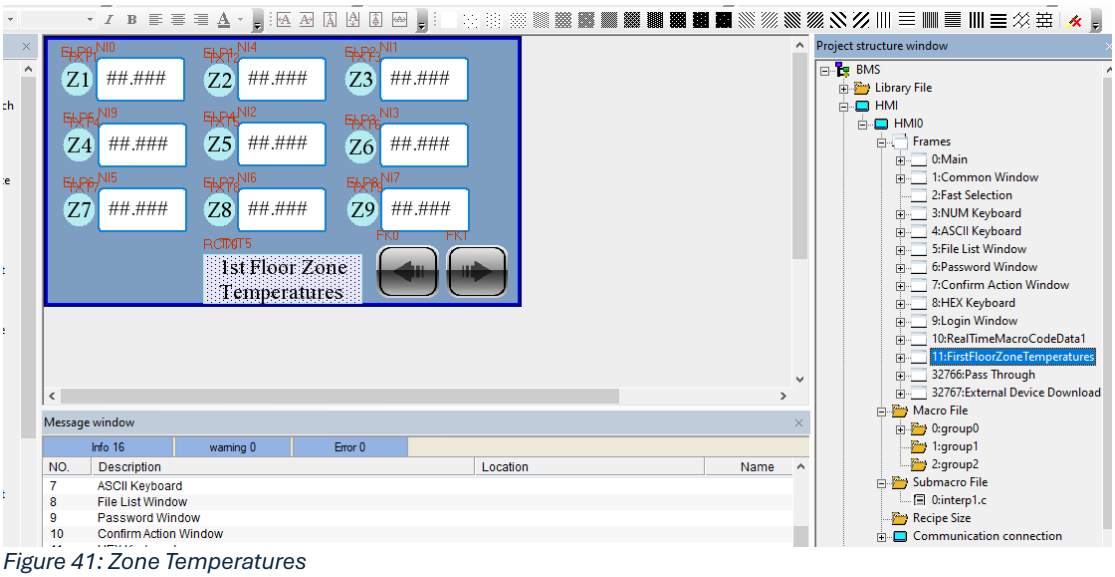


Figure 41: Zone Temperatures

This screen displays the real-time temperatures of the first floor's thermal zones.

3.5 Macro Code

The HMI used in this thesis supports programming in C which is useful for implementing custom logic. However, it is recommended to avoid complex logic due to limited resources and the need for quick decision-making. The Macro Code window in DTools is consisted of three main parts which are displayed below:

1) Editor

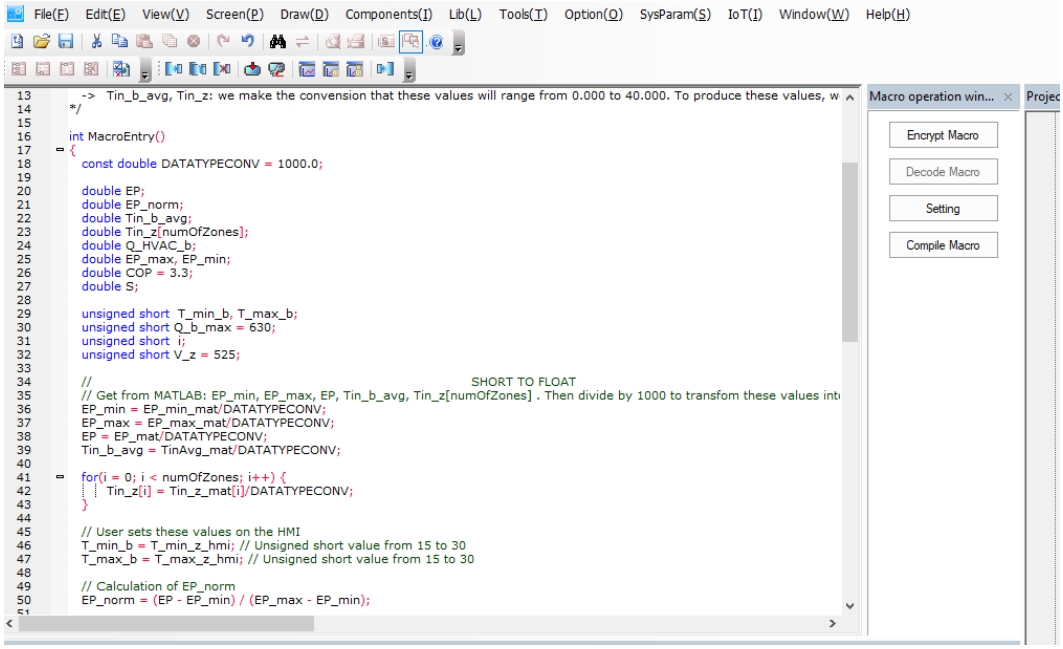


Figure 42: Macro Code Editor

C programming happens in the macro code editor as seen above.

2) Parameters Declaration

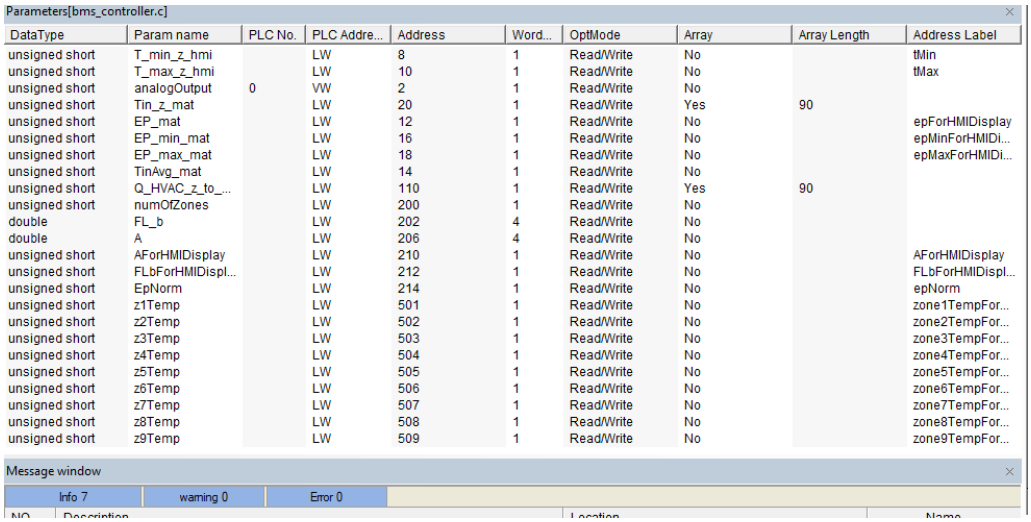


Figure 43: Macro Code Parameters

The previous figure displays the parameters page where global variables used by the C program can be declared, receive a name, a data type, an address and a description.

3) Submacro

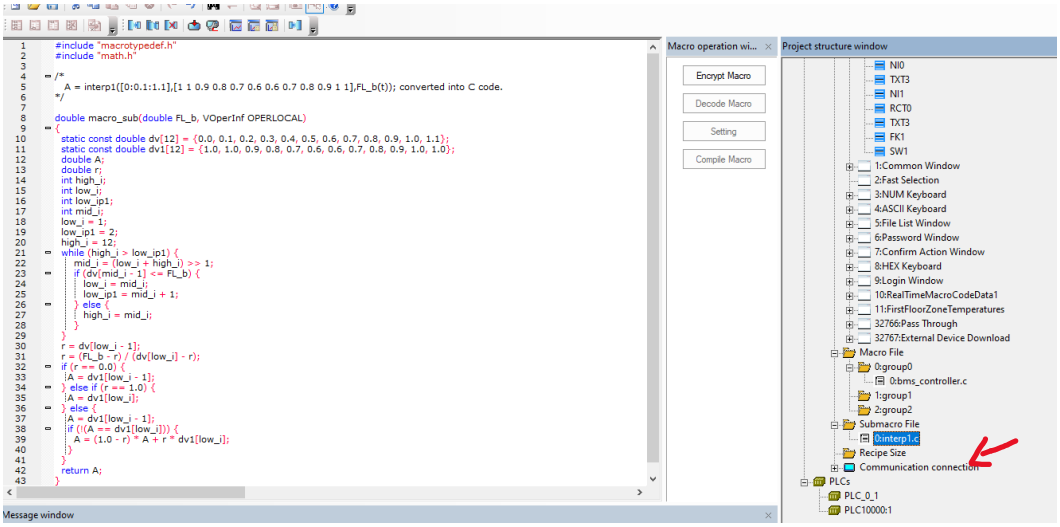


Figure 44: Submacro Code

Submacro files serve the same role as functions used in programming languages. In the figure above, you can see the interp1.c submacro file which implements in C the interp1 MATLAB function. Then, this function is called by bms_controller.c

4) Macro Code Logical Flow

- The process begins when MATLAB, running on the computer, triggers an event by changing a register value from 0 to 1. This event initiates the execution of the macro code.
- Data from MATLAB are received (min and max energy price, current energy price, average building temperature, zone temperatures).
- Integers converted to floats.

```
// Get from MATLAB: EP_min, EP_max, EP, Tin_b_avg, Tin_z[numOfZones] . Then divide by 1000 to transfom t
EP_min = EP_min_mat/DATATYPECONV;
EP_max = EP_max_mat/DATATYPECONV;
EP = EP_mat/DATATYPECONV;
Tin_b_avg = TinAvg_mat/DATATYPECONV;

for(i = 0; i < numOfZones; i++) {
        Tin_z[i] = Tin_z_mat[i]/DATATYPECONV;
}
```

- Minimum and maximum building temperatures set by the user in the HMI are recorded.

```
// User sets these values on the HMI
T_min_b = T_min_z_hmi; // Unsigned short value from 15 to 30
T_max_b = T_max_z_hmi; // Unsigned short value from 15 to 30
```

- Calculation of normalized values for the electricity price by taking into consideration the minimum and maximum values.

```
// Calculation of EP_norm
EP_norm = (EP - EP_min) / (EP_max - EP_min);
```

- Calculation of FL_b. This variable defines the flexibility of each thermal zone to increase or decrease its thermal power.

```
// Calculation of FL_b
FL_b = (Tin_b_avg - T_min_b) / (T_max_b - T_min_b);
```

- Performs interpolation to the previous value by calling the appropriate sub macro function and saves the results to variable "A" (this is the equivalent MATLAB code which was implemented in C). This variable is a weighting factor between the need to increase or decrease the temperature in a building and the energy price

```
/*
  A = interp1([0:0.1:1.1],[1 1 0.9 0.8 0.7 0.6 0.6 0.7 0.8 0.9 1 1],FL_b(t)); converted into C code.
*/
```

- Calculates variable "p". This is a variable which takes into consideration the variable "A", the flexibility of the thermal zone and the normalized electricity price. By calculating it with the maximum HVAC capacity, the total HVAC power allocated to the building is calculated.

```
// Calculation of p
double p = A * FL_b + (1 - A) * (1 - EP_norm);

// Calculation of Q_HVAC_b
Q_HVAC_b = Q_b_max * p;
```

- Calculates the thermal power dispatch for each zone with respect to the flexibility, the volume of the thermal zone and the building needs for thermal power delivery.

```
// Thermal power dispatch for each zone
for(i = 0; i < numOfZones; i++) {
    S += ((Tin_z[i] - T_min_z_hmi) * V_z) / (T_max_z_hmi - T_min_z_hmi);
}
```

- Calculates thermal power for each zone and returns the values to MATLAB.
- The thermal power is then propagated to the PLC for the HVAC control.

PLC Implementation

4.1 Basics

As previously mentioned, the MK043E-27DT HMI-PLC was chosen for monitoring the BMS system and managing the power delivery to the HVAC system in the building. The integration of a PLC controller in a single unit simplifies the implementation and adaptation process. The programming of this PLC unit produced by Kinco, is achieved by using the custom software Kinco Builder. This software is specifically designed for Kinco products and enables users to program in ladder logic and instruction lists. In this thesis, LD programming was selected due to its convenience and ease of understanding.

Before taking a deeper look in the actual PLC implementation, it would be beneficial to take a brief look in the basic pages and functionalities of the Kinco Builder application.

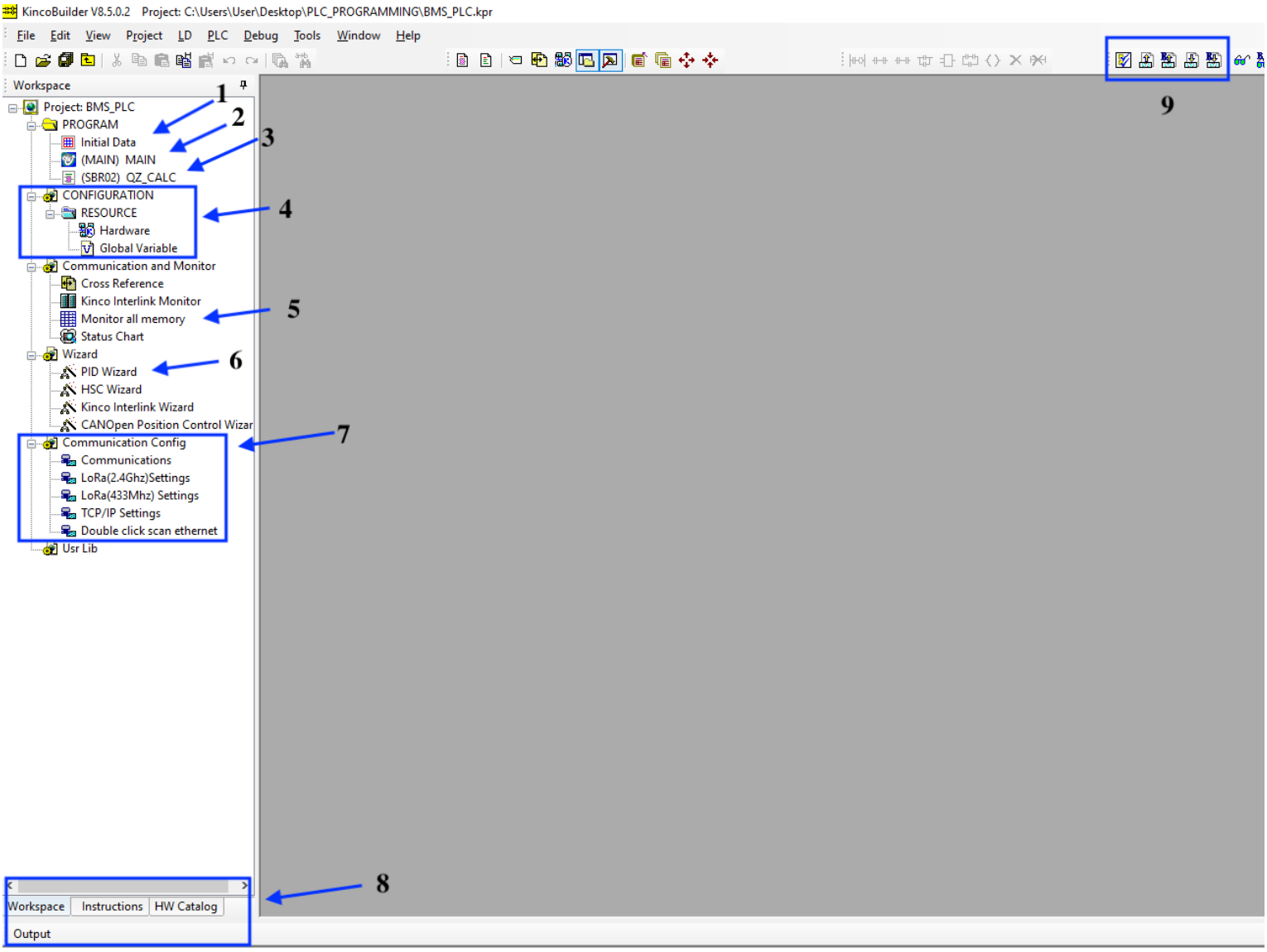


Figure 45: Kinco Builder Overview

In the figure above, there are nine parts of the Kinco Builder app highlighted.

1) **Initial Data**: This page allows users to set default values to registers, ensuring preset values are available when the PLC starts operating.
2) **MAIN**: This is main environment where you can program the PLC and set parameters.
3) **SBR**: This is a subroutine where you can define your custom logic and use it (like functions in programming) in other subroutines or the MAIN app.
4) **Configuration Options**: This is a list of configuration options regarding the hardware and global variables.

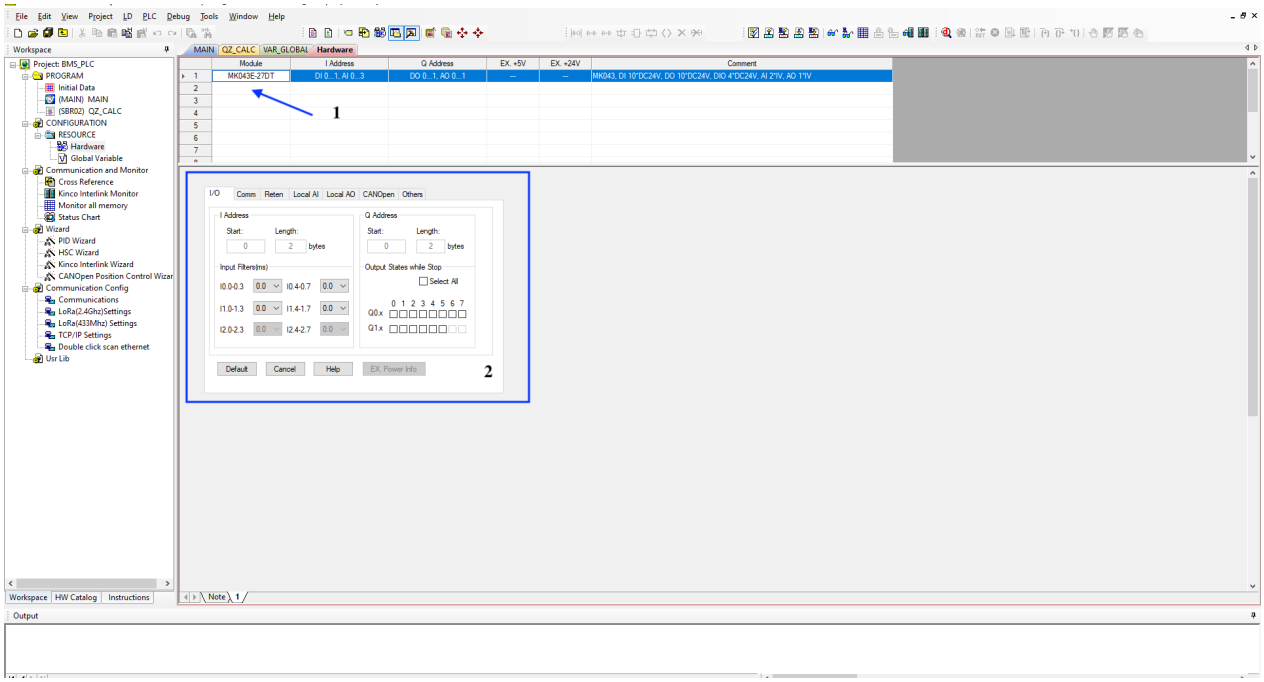
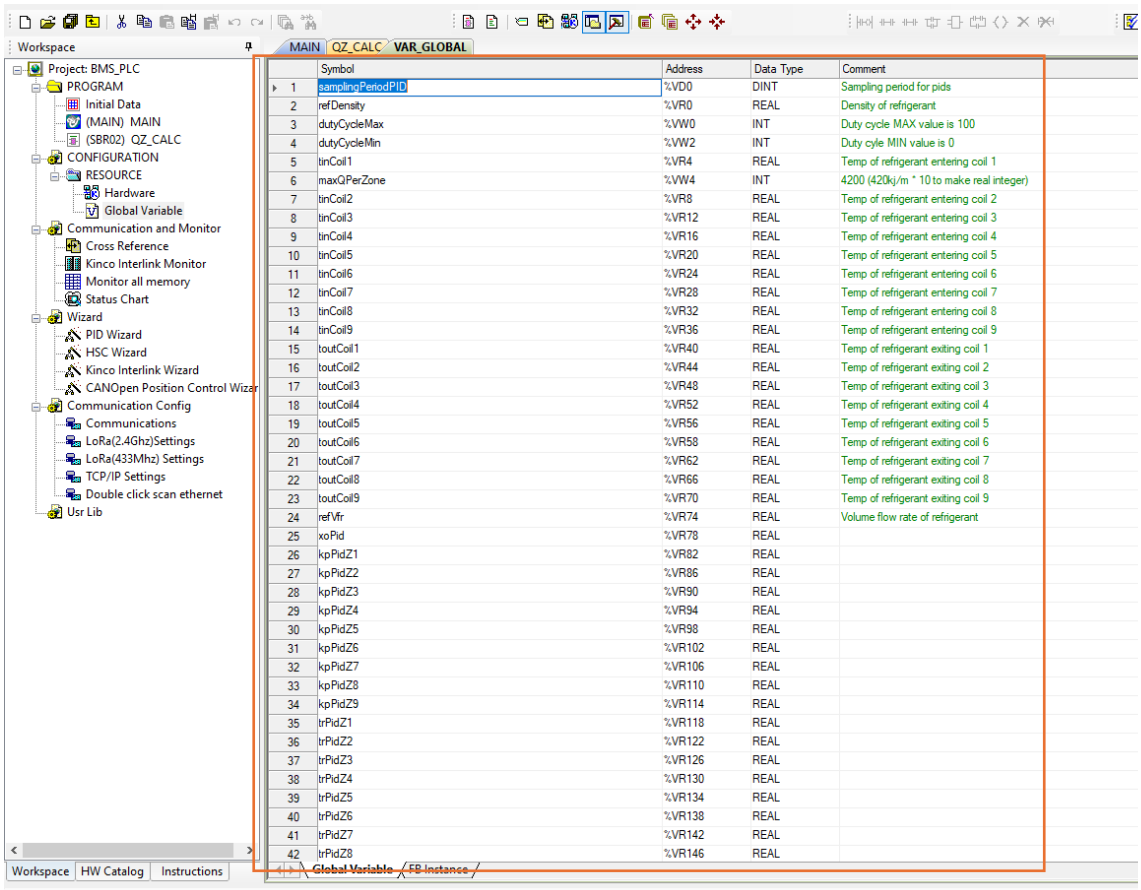


Figure 46: HardwareConfigurations

Section 1 points to the actual PLC used and programmed. By selecting the correct one, the Kinco Builder software informs the user of the available input and outputs, takes care of the correct register selection and use. It also opens a window (2) where there are options for changing the default behavior of the I/O ports, configure serial ports and change the analog I/O range values.



	Symbol	Address	Data Type	Comment
1	samplingPeriodPID	%VD0	DINT	Sampling period for pids
2	refDensity	%VR0	REAL	Density of refrigerant
3	dutyCycleMax	%VW0	INT	Duty cycle MAX value is 100
4	dutyCycleMin	%VW2	INT	Duty cyle MIN value is 0
5	tinCoil1	%VR4	REAL	Temp of refrigerant entering coil 1
6	maxQPerZone	%VW4	INT	4200 (420kj/m * 10 to make real integer)
7	tinCoil2	%VR8	REAL	Temp of refrigerant entering coil 2
8	tinCoil3	%VR12	REAL	Temp of refrigerant entering coil 3
9	tinCoil4	%VR16	REAL	Temp of refrigerant entering coil 4
10	tinCoil5	%VR20	REAL	Temp of refrigerant entering coil 5
11	tinCoil6	%VR24	REAL	Temp of refrigerant entering coil 6
12	tinCoil7	%VR28	REAL	Temp of refrigerant entering coil 7
13	tinCoil8	%VR32	REAL	Temp of refrigerant entering coil 8
14	tinCoil9	%VR36	REAL	Temp of refrigerant entering coil 9
15	toutCoil1	%VR40	REAL	Temp of refrigerant exiting coil 1
16	toutCoil2	%VR44	REAL	Temp of refrigerant exiting coil 2
17	toutCoil3	%VR48	REAL	Temp of refrigerant exiting coil 3
18	toutCoil4	%VR52	REAL	Temp of refrigerant exiting coil 4
19	toutCoil5	%VR56	REAL	Temp of refrigerant exiting coil 5
20	toutCoil6	%VR58	REAL	Temp of refrigerant exiting coil 6
21	toutCoil7	%VR62	REAL	Temp of refrigerant exiting coil 7
22	toutCoil8	%VR66	REAL	Temp of refrigerant exiting coil 8
23	toutCoil9	%VR70	REAL	Temp of refrigerant exiting coil 9
24	refVfr	%VR74	REAL	Volume flow rate of refrigerant
25	xoPid	%VR78	REAL	
26	kpPidZ1	%VR82	REAL	
27	kpPidZ2	%VR86	REAL	
28	kpPidZ3	%VR90	REAL	
29	kpPidZ4	%VR94	REAL	
30	kpPidZ5	%VR98	REAL	
31	kpPidZ6	%VR102	REAL	
32	kpPidZ7	%VR106	REAL	
33	kpPidZ8	%VR110	REAL	
34	kpPidZ9	%VR114	REAL	
35	trPidZ1	%VR118	REAL	
36	trPidZ2	%VR122	REAL	
37	trPidZ3	%VR126	REAL	
38	trPidZ4	%VR130	REAL	
39	trPidZ5	%VR134	REAL	
40	trPidZ6	%VR138	REAL	
41	trPidZ7	%VR142	REAL	
42	trPidZ8	%VR146	REAL	

Figure 47: Global Variables

The global variables page shown above, as its name suggests, gathers all the global variables that can be referenced anywhere in the app and exported for further use with the HMI unit. To export these variables, select the preferred ones, right click and select export.

5) **Monitor All Memory**: This is the page to monitor the values of all registers available by the selected PLC.
6) **PID Wizard**: Kinco Builder offers automated functionalities with step by step guidelines for implementing various functionalities and modules. One of them is the PID wizard which guides you through the process of creating a PID and selecting the correct configurations for any case.
7) **Communication config**: This is a list of options to help with the connectivity of the PLC with other devices like a computer or HMI. For example, the TCP/IP settings sections let the user set the proffered IP and Port address for establishing connectivity with a computer using the Modbus TCP protocol.
8) Navigate through the three available windows: **Workspace, Instructions, HW Catalogue**

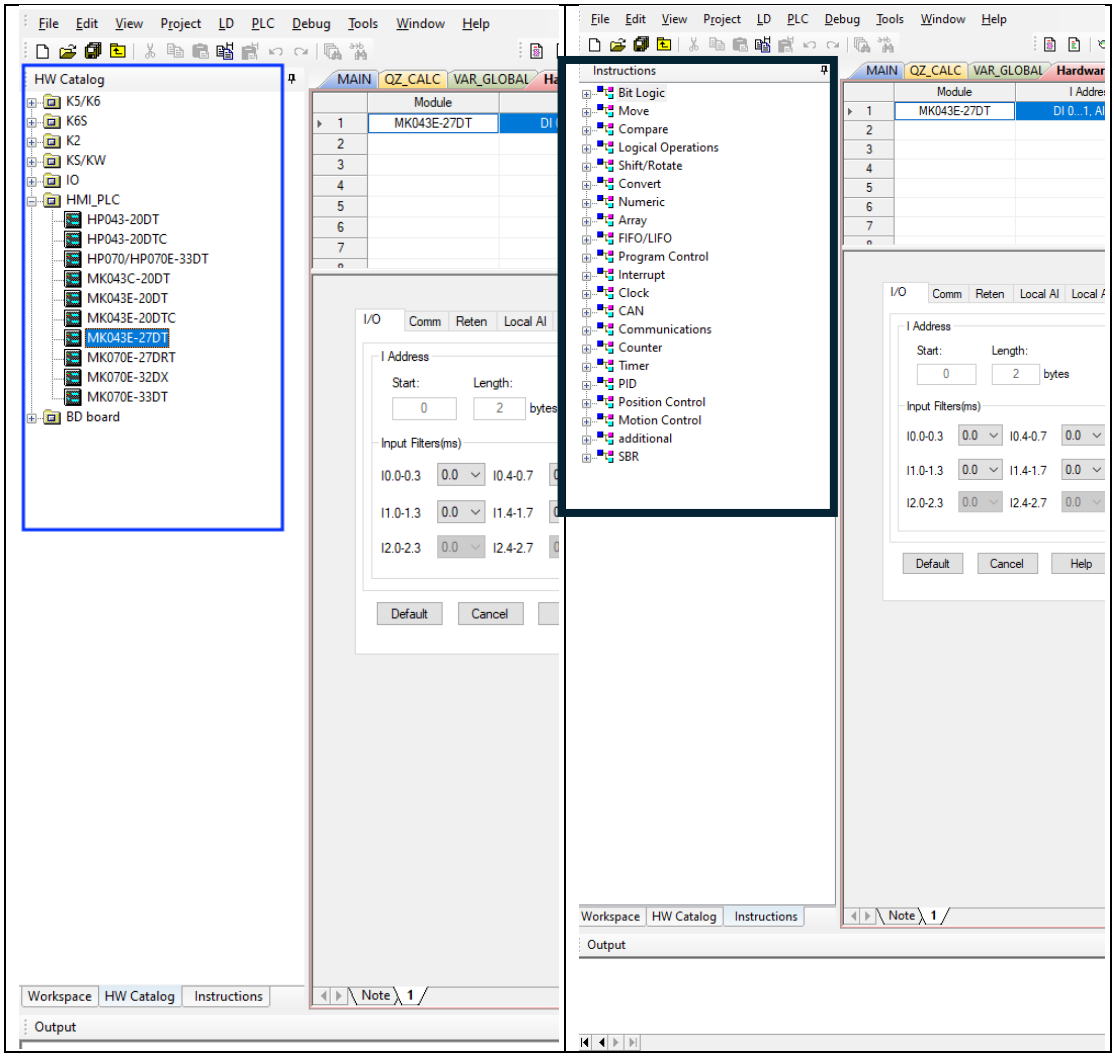


Figure 48: HW Catalog and Instructions

HW Catalog: This is where the preferred HMI-PLC can be selected and drag and dropped in the hardware page.

Instructions Page: It offers all the available components for programming in PLC with LD programming like Move Operator, PID component, Timers etc.

9) This part of the toolabar is responsible for compiling and downloading the changes in the PLC.

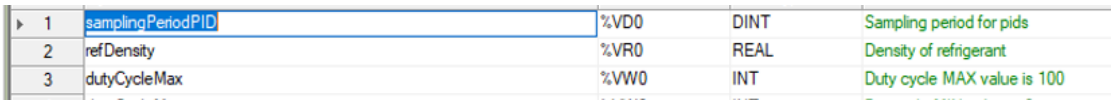
4.2 Data Types – Addressing

Kinco PLC comes with support for the following data types. These data types determine the type of data stored in the registers of the PLC and can be used for defining variables in PLC programming.

Keyword	Description	Size in Bits	Range of Values	Default Initial Value
BOOL	Boolean	1	true, false	false
BYTE	1word (16bits)	8	0 ~ 255	0
WORD	1word (16bits)	16	0 ~ 65,535	0
DWORD	2word (32bits)	32	0 ~ 4,294,967,295	0
INT	2word (32bits)	16	-2^{15} ~ $(2^{15}-1)$	0
DINT	2word (32bits)	32	-2^{31} ~ $(2^{31}-1)$	0
REAL	4word (64bits)	32	$1.18*10^{-38}$ ~ $3.40*10^{38}$, 0, $-3.40*10^{38}$ ~ $-1.18*10^{-38}$	0.0

The next table gives a brief overview to the available addressing areas. Registers are divided in categories depending on their type of use. For example, registers for the digital inputs occupy the "I" address space. Each address space has a specific number of available registers and gives support for a specific selection of data types.

Address Area	Description
I	Digital Input Area. Read only access
Q	Digital Output Area. Read/Write access
AI	Analog Input Area. Read only access
AQ	Analog Output Area. Read/Write access
HC	High-speed C counter Area. Used for storing the current counting value of high-speed counters. Read only
V	Variable Area. It is relatively large and can be used to store large quantity of data. Read/Write access.
M	Internal Memory Area. Used to store the internal status or other dat. Compared with V area, M area can be accessed faster and more propitious to bit operation
SM	System Memory Area. System data are stored here. You can read come SM addresses to evaluate the current system status, and you can modify some addresses to control some system functions
L	Local Variable Area. Not recommended for direct access.



1	samplingPeriodPID	%VD0	DINT	Sampling period for pids
2	refDensity	%VR0	REAL	Density of refrigerant
3	dutyCycleMax	%VW0	INT	Duty cycle MAX value is 100

Figure 49: Addressing Area & Data Types

As shown above, in case it is needed to store three data types of DINT, REAL and INT in the V address area, three variables can be declared by defining their data types and allocating the correct addresses in the V area. For example, for the refDensity variable, which is of type REAL, %VR0 is used. Addresses start with the symbol % followed by the type of data type, in this case is R – Real, and ending with the position in the address area. 0 stands for the start of the address area. In case the data type is BOOL, the address position should be of type: X.X starting with 0.0.

4.3 Main PLC Implementation in Depth

The PLC implementation in this thesis is designed based on the specific HVAC system used in the building. Here, we assume the building utilizes a heat pump connected to a network of pipes that circulate hot or cold liquid through fan coil units. This setup is commonly found in industrial applications, where each fan coil unit regulates the temperature within is respective thermal zone.
Unlike conventional HVAC systems, which rely on a single temperature sensor per zone to provide feedback for maintaining the desired temperature, this implementation introduces an advanced approach. Since the goal is to control the thermal power Q supplied to each zone, additional temperature sensors and a flow meter are incorporated. These sensors enable the real-time calculation of the thermal power transferred from the coils to the thermal zones.
By continuously measuring flow rates and temperature differentials, the PLC dynamically adjusts the system to maintain the required thermal power (Q), ensuring precise thermal regulation for each zone.

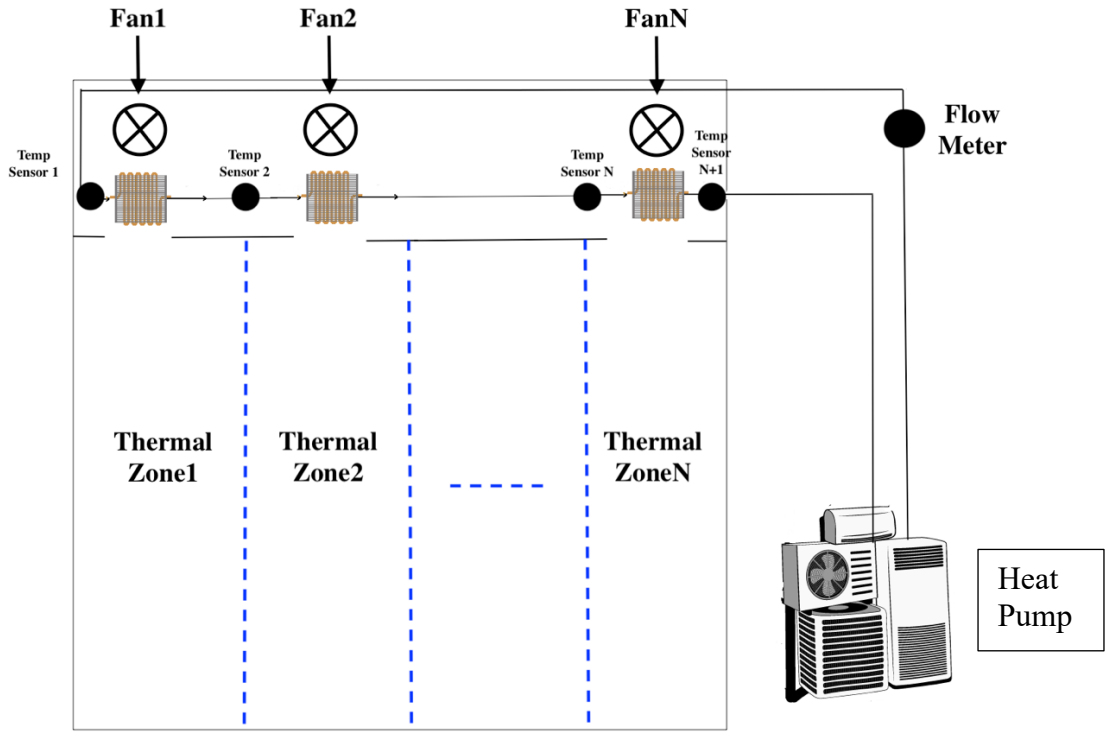


Figure 49: HVAC system model

HVAC System Main Components:

1) Heat Pump
2) Fans
3) Coils

Components needed by the PLC:

1) Temperatures sensors
2) Flow meter sensor

As it can be shown above, a central heating pump is used to send cool or hot fluid to coils connected in series. In this respect, by making use of fans lying over the coils and

producing air flow which passes through the coils, hot or cold air is entered in each thermal zone.

The controller inside the HMI, produces a thermal power setpoint (Q: Watt) which is the ideal one to achieve temperatures inside the thermal zones that meet the minimum and maximum values set by the administrator of the system and minimize the electricity consumption cost.

This setpoint (Q), is then passed to the PLC which is responsible for reaching and preserving this value. For better understanding, it is necessary to go back to the equation (24) regarding the heat transfer.

$$Q = \dot{m} \times Cp \times \Delta T. \quad (24)$$

Also, we must make the following assumptions:

1) The Q, when it comes to the HVAC system showed in figure 49, refers to the thermal energy transferred from the coils to the thermal zone during the operation
2) The operation and consumption of the heat pump will be considered constant
3) The fluid passing through the coils in series will have the same flow (not taking into consideration changes in static pressure)
4) The only way to change the Q is by changing the speed of fans above each coil

So, the final requirement for controlling the HVAC system is to find a way to calculate the Q produced by the coils in respect to the change of fan speed.

The equation (24) can be transformed in the following one:

$$Q = \rho \times VFR \times Cp \times \Delta T. \quad (25)$$

- **Q** is the rate of heat transfer (W)
- **ϱ** density of fluid flowing in the coils (kg/m^3)
- **VFR** is the volumetric flow rate (m^3/s)
- **Cp** is the specific heat capacity (J/kg*T)
- **ΔT** is the temperature difference in the fluid entering and exiting each coil

In this respect, to calculate the Q in real time, we need temperature sensors in the entry and exit point of each coil and a flow meter sensor to give us the VFR value in real time. So, by using a PID which takes the Q setpoint and the Q produced by the coil as inputs while adjusting the fan speed as the control output, we can achieve precise thermal regulation.

In this case, the PID will be manually configured, meaning the proportional (Kp), integral (Ki) and derivative (Kd) gains will be adjusted through testing to ensure stable and responsive control. This approach is commonly used in industrial HVAC systems.

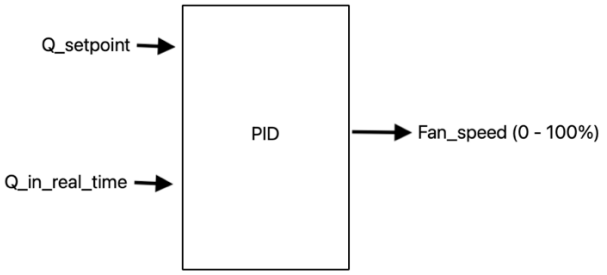


Figure 50: PID for controlling the fan speed

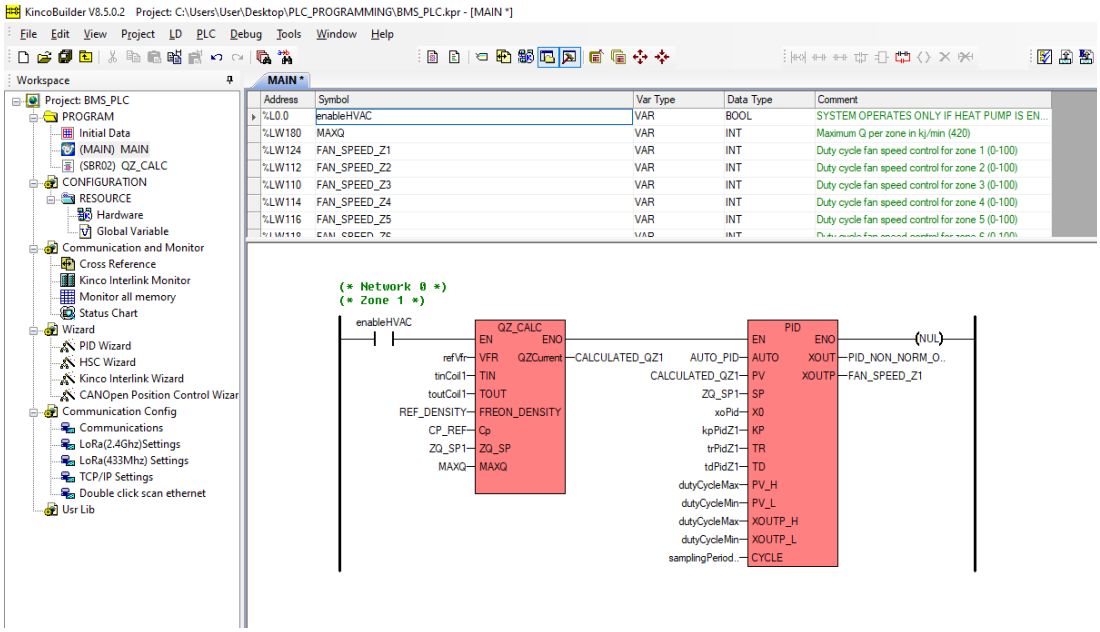


Figure 51: PLC controller black box

Above you can observe a "black box" of a possible PLC implementation for handling the thermal power transfer to each thermal zone. The implementation is consisted of two parts. The first one is the **QZ_CALC** subroutine and the second is a **PID**.

1) **QZ_CALC:** calculates the Q in real time. It takes as parameters the density of the fluid, the heat capacity of the fluid, the flow rate, the temperature of the fluid entering the coil and the temperature of the fluid exiting the coil. Also, it takes as input the Q setpoint and the maximum Q allowed to conduct a security check before proceeding with calculating the Q in real time
2) **PID** is responsible for taking as input the setpoint (ZQ_SP1) and the calculated Q (CALCULATED_QZ1) and producing as output (XOUT: 0.0 – 1.0 and XOUTP: 0 - 100) the fan speed percentage.

PID INPUTS AND OUTPUTS

Operands	IN/OUT	Data Type	Memory Areas	Comment
AUTO	INPUT	BOOL	I, Q, V, M, SM, L, T, C	Manual/Auto. 0=Manual, 1=Auto
PV	INPUT	INT	AI, V, M, L	Process Variable
SP	INPUT	INT	V	Setpoint
XO	INPUT	REAL	V	Manual value, range [0.0, 1.0]
KP	INPUT	REAL	V	Proportionality constant
TR	INPUT	REAL	V	Reset time, which determines the time response of the derivative unit. (Unit: s)
TD	INPUT	REAL	V	Derivative time, which determines the time response of the derivative unit. (Unit: s)
PV_H	INPUT	INT	V	The upper limit value of PV
PV_L	INPUT	INT	V	The lower limit value of PV
XOUTP_H	INPUT	INT	V	The upper limit value of XOUTP
XOUTP_L	INPUT	INT	V	The lower limit value of XOUTP
CYCLE	INPUT	DINT	V	Sampling period. (Unit: ms)
XOUT	OUTPUT	REAL	V	Manipulated Value, range [0.0, 1.0]
XOUTP	OUTPUT	INT	AQ, V	Manipulated Value Peripheral. The value is the normalizing result of XOUT

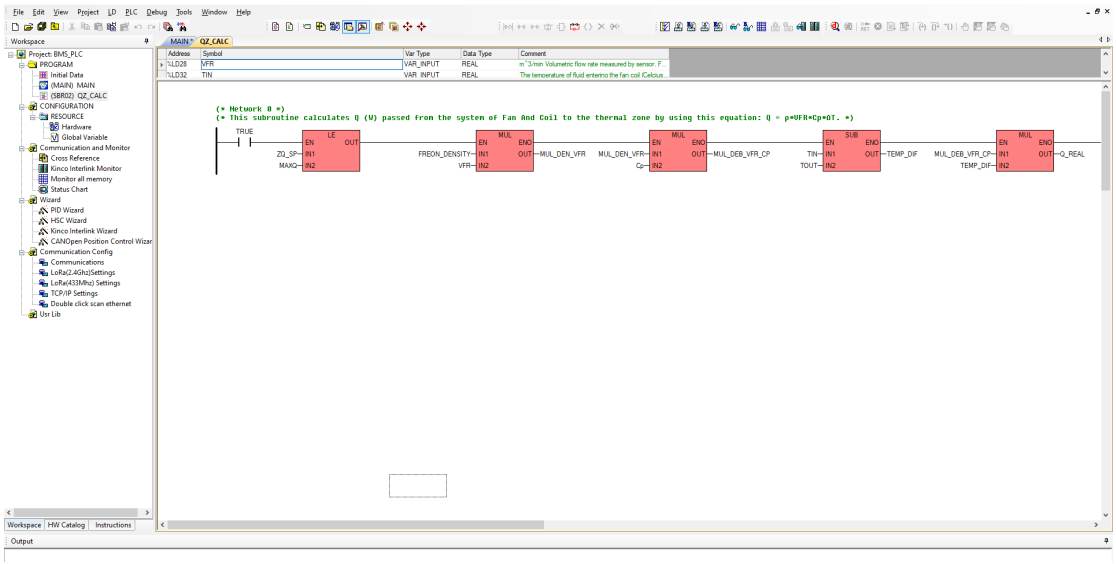


Figure 52: Real time calculation of Q (1)

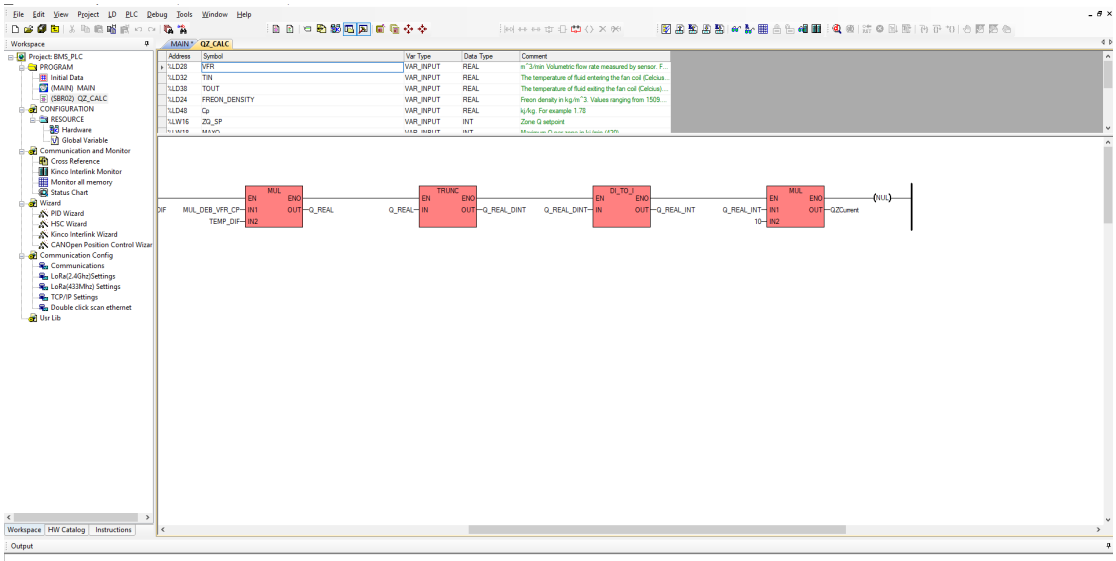


Figure 53: Real time calculation of Q (2)

Figures 52 and 53 take a deeper look to the sub-routine **QZ_CALC.** As it can be shown above, the logic starts with a LE block which checks to see of the Q_setpoint is less than the maximum allowed Q. Then it proceeds with calculating the following equation:

$$Q = \rho \times VFR \times Cp \times \Delta T. \quad (25)$$

The calculation is achieved by using the MUL, SUB, TRUNC, DI_TO_I blocks. TRUNC block is used to truncate a float value to DINT and the DI_TO_I to convert the DINT value to INT. The PV input of the PID is an integer. That is why this conversion is needed.

Results

After finishing with the implementation of the GUI interface and the logic inside the HMI, it was time to connect the computer, hosting the MATLAB implementation, with the HMI via Modbus TCP. Also, the aim was to run the algorithm for a 24-hour time frame to get the results and compare with the ones produced while running the whole implementation on MATLAB.

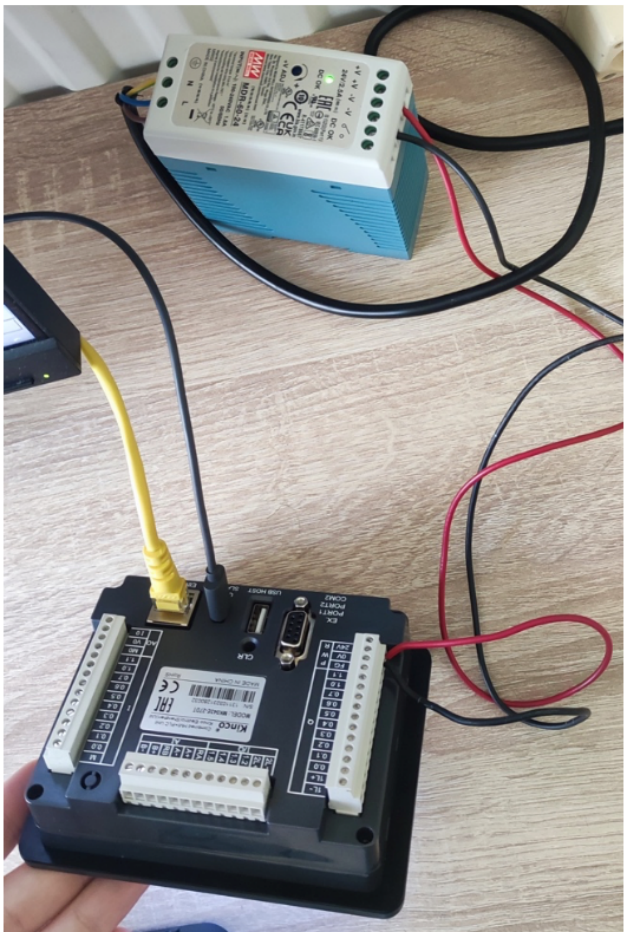


Figure 54: Connection of HMI with PC

The figure above shows the connection of the HMI with the ethernet cable in order to make use of the Modbus TCP protocol. Also, in the same figure, the power source of the PLC is displayed. The next step is to set the right minimum and maximum temperature values in the main screen of the HMI as shown in the next figure.

Figure 55: Main Menu Screen

After setting the setpoints and turning on the HVAC system with the button to the right edge of the screen, it is time for the MATLAB logic to be executed and start sending data to HMI and trigger the Macro Code execution. This is done by clicking the run button of MATLAB in the main function. When the communication is established and data are sent correctly, the "Avg Building Temperature" box starts displaying values.

Figure 56: Macro Code Data Screen

The screen above presents data related to the macro code execution and the energy price. Whenever a signal is sent to the HMI to produce new power consumption setpoints for the HVAC, the macro code execution light is turned on.

Figure 57: Zone Temperatures Screen

The screen displayed above shows the temperatures of individual thermal zones, providing a clearer understanding of the conditions in each one.

Although the controller (HMI Macro Code) processes calculations for all 90 zones across 10 floors, the screen above displays only the results for the first floor for simplicity. The pages for other floors, will follow the layout and logic, with the only difference being the registry values assigned to each thermal zone.

After the execution is over, the following results are gathered from the MATLAB operation which verify that the connection was correct, and no information was missed out.

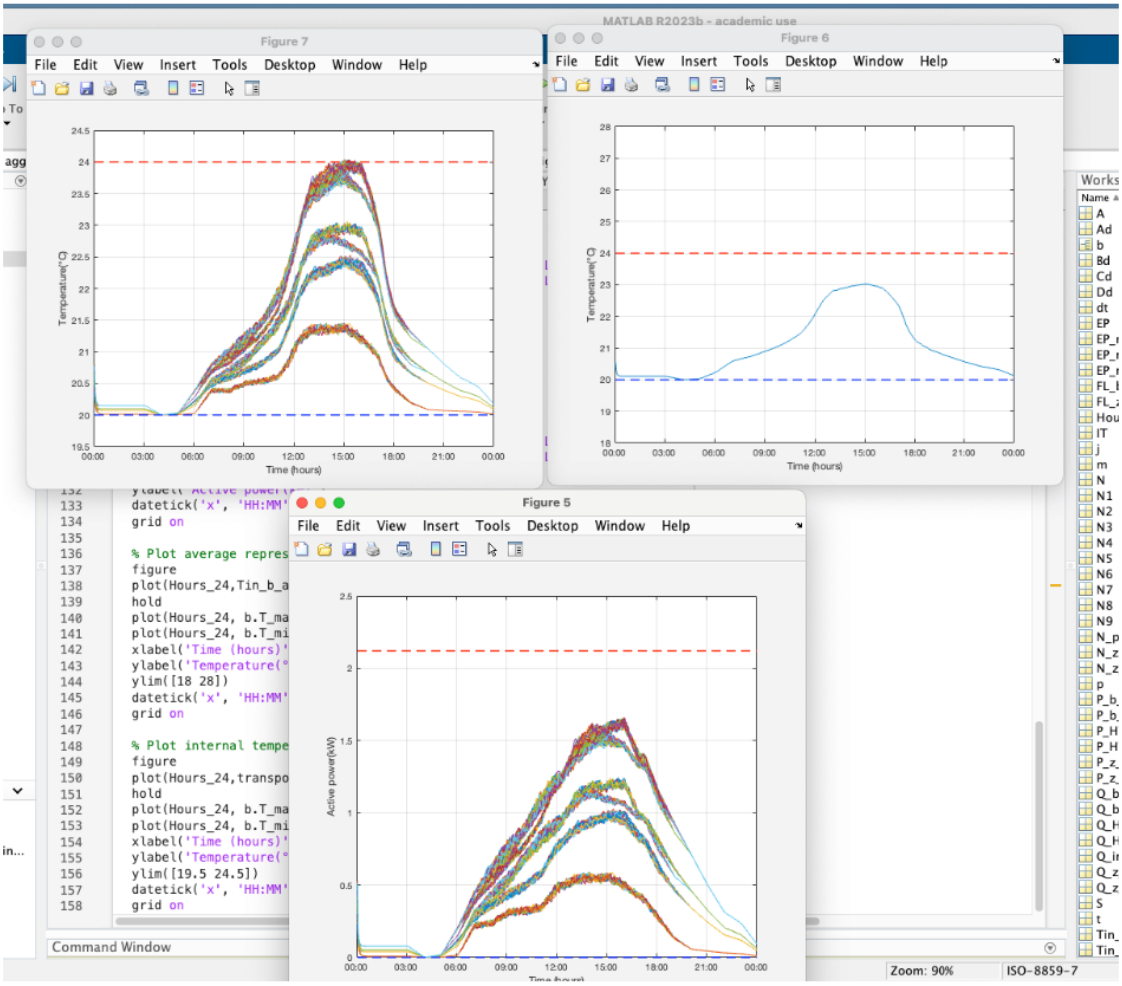


Figure 58: Results gathered from MATLAB

The figure above displays the temperatures of each zone, the average building temperature and the active power consumption of the HVAC system. It is apparent that the temperatures were kept with inside the 20-24 range and active power consumption was restricted below the max value.

So, the main conclusion of the whole experiment was the successful communication of the MATLAB with the HMI controller and the ability to interact with parameters like the min and max temperature setpoints.

Improvements

It is obvious that building a fully functional BMS system requires more resources, overcoming obstacles, testing the implementation in real scenarios and ensuring the testing is happening in the right environment. This thesis is just the base for future improvements and implementations.

The future improvements and necessary implementations can be organized in three categories:

1) **Predictions – Real time data streaming – Decoupling of MATLAB**

It is vital to find new ways of keeping track of data like the ambient temperature, solar radiation, thermal dissipation from the electrical devices as well as the number of people inside the building in real time and the electricity price each hour of the day. It is obvious, that some data cannot even be provided by current systems, take as example the hourly electricity price. New sensors directly connected to the building management system could be introduced to give accurate and real time data. For data that depend on third party application, a new software could be created to gather all these data in a computer, visualize them to the user and send them directly to the BMS.

2) **Improvements in HMI GUI**

When it comes to the HMI implementation, security options could be introduced, new menu items and functionalities could be added to address all the needs of the user like for example taking manual control of the system or producing graphs about the thermal power consumption.

3) **Improvements in PLC implementation**

For the case of the PLC, it is vital to set the correct requirements when it comes to the HVAC system used in the tested building and the algorithm that will control the actual thermal energy sent to the thermal zones. The number as well as the type of devices (fans, sensors, heat pump etc.) used for controlling the temperature of the building can determine the number of PLCs used in each floor and building, the actual needs for digital and analog I/O ports (analog ports are significantly more expensive than the digital ones) and much more.

Conclusion

In my thesis, the integration of the MK043E-27DT HMI-PLC with the MATLAB hosted in an ordinary computer, for monitoring and controlling the Building Management System (BMS) and managing the HVAC system was explored. The primary objective was to streamline the implementation and adaptation process of a thermal power delivery optimization algorithm into a system that would be consisted of ready to use devices with the possibility of scaling up and adding more functionalities in the future.

The findings of this research indicate that HMIs and PLCs can be used to easily host any algorithm for controlling HVAC systems, effectively adapt to the needs of any new environment, meet strict requirements, scale up and communicate effectively with other devices via protocols like Modbus TCP.

While the results are promising, there are limitations to this study. The research was restricted to specific set of parameters and conditions. For example, no current support for gathering live electricity prices. Also, more logic and more data added to the algorithm could result to unavailability of the HMI to meet responsiveness requirements. Besides these, further functionalities could be added to the HMI and the PLC implementation should be tested in real environment.

In conclusion, building a smart BMS system can be a complicated task. However, by sticking to the right tools and the right design the implementation process can be made significantly easier. Combining the ability of computer applications to gather and store information from multiple resources with the ease of use of HMIs and the reliability, effectiveness of PLCs, could be the way to go.

References

[1] A. Borodinecs, A. Paļčikovskis and A. Krūmiņš, "Assessment of HVAC Performance and Savings in Office Buildings Using Data-Driven Method," *ResearchGate,* p. 803, June 2024.

[2] D. G. Kyriakou and F. D. Kanellos, "Optimal Operation of Microgrids Comprising Large Building Prosumers and Plug-in Electric Vehicles Integrated into Active Distribution Networks," *Energies,* vol. 15, no. 17, p. 6182, 2022.

[3] X. Jin, J. Wu, Y. Mu, M. Wang, X. Xu and H. Jia, "Hierarchical microgrid energy management in an office building," *Applied Energy,* vol. 208, no. 480-494 , 2017.

[4] InductiveAutomation, "inductiveautomation," 10 08 2018. [Online]. Available: https://inductiveautomation.com/resources/article/what-is-hmi.

[5] K. H. Mouhammad Hamsho, "Udemy," July 2024. [Online]. Available: https://www.udemy.com/course/from-wire-to-plc-a-to-z-compilation.

[6] M. Hudedmani, "Programmable Logic Controller (PLC) in Automation," *ResearchGate,* p. 40, July 2017.

[7] S. E. USA, "Schneider Electric," USA, Schneider Electric, 03 March 2013. [Online]. Available: https://www.se.com/us/en/faqs/FA168406/.

[8] N. Malviya, "INFOSEC," INFOSEC, 04 March 2020. [Online]. Available: https://www.infosecinstitute.com/resources/scada-ics-security/rs-232-and-rs-485/.

[9] Wikipedia, "Wikipedia Modbus," [Online]. Available: https://en.wikipedia.org/wiki/Modbus.

[10] G. Vernova, "GE Vernova," [Online]. Available: https://www.ge.com/digital/documentation/cimplicity/version2022/oxy_ex-2/device_communications/topics/g_cimplicity_device_communications_function_codes_supportedby_modbus_tcp_server.html.

[11] Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/IEC_61131-3.

[12] M. Hamsho, "EverTutorial," [Online]. Available: https://evertutorial.com/articles/PLC/PLC-Ladder-Logic-Tutorial-Motor-Control.

[13] Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Function_block_diagram.

[14] Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Instruction_list.

[15] Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Sequential_function_chart.

[16] J.C. Solano, E. Caamano-Martin, L. Olivieri, D.Almeida-Galarraga, "HVAC systems and thermal comfort in buildings climate control: An experimental case study," *ScienceDirect,* pp. 269-277, September 2021.

[17] R. Aresco, "ERIE INSTITUTE OF TECHNOLOGY," ERIE INSTITUTE OF TECHNOLOGY, 28 April 2023. [Online]. Available: https://erieit.edu/components-of-an-hvac-system-and-how-they-work/.

[18] Wikipedia, [Online]. Available: https://en.wikipedia.org/wiki/Heat_pump.

[19] J. H. L. V. JOHN H. LEINHARD IV, A HEAT TRANSFER TEXTBOOK THIRD EDITION, Cambridge, Massachusetts: JOHN H. LEINHARD V, 2001.

[20] Kinco. [Online]. Available: https://en.kinco.cn/productdetail/mkserieshmiplc99.html.