MODEL BASED DESIGN OF HEAT RECOVERY VENTILATION SYSTEM

A MASTER'S THESIS

in

Electrıcal And Electronıcs Engıneerıng

Atilim University

by

YOUSSEF H. MUSSA DABAS

JANUARY 2018

MODEL BASED DESIGN OF HEAT RECOVERY VENTILATION SYSTEM

A THESIS SUBMITTED TO

THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

OF

ATILIM UNIVERSITY

BY

YOUSSEF H. MUSSA DABAS

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

IN

THE DEPARTMENT OF ELECTRICAL AND ELECTRONICS

ENGINEERING

JANUARY 2018

Approval of the Graduate School of Natural and Applied Sciences, Atılım University.

_____

Prof. Dr. Ali KARA

Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

_____

Asst. Prof. Dr. Kemal Efe ESELLER

Head of Department

This is to certify that we have read the thesis "MODEL BASED DESIGN OF HEAT RECOVERY VENTILATION SYSTEM" submitted by "YOUSSEF H. MUSSA DABAS" and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

_____	_____
Asst. Prof. Dr.Kutluk Bilge ARIKAN	Asst. Prof. Dr. Mehmet Efe ÖZBEK
Co-Supervisor	Supervisor

Examining Committee Members

Asst. Prof. Dr. Hakan TORA	_____
Asst. Prof. Dr.Enver ÇAVUŞ	_____
Asst. Prof. Dr. Mehmet Efe ÖZBEK	_____

Date: 25.1.2018

I declare and guarantee that all data, knowledge and information in this document has been obtained, processed and presented in accordance with academic rules and ethical conduct. Based on these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Last name: YOUSSEF DABAS

Signature:

ABSTRACT

MODEL BASED DESIGN OF HEAT RECOVERY VENTILATION SYSTEM

Youssef H.Mussa Dabas

M.S., Electrical and Electronic Engineering Department

Supervisor: Assist. Prof. Dr. Mehmet Efe Özbek

January 2018, 43 pages

This thesis explores the suitability of using a Model-Based Design workflow for the development of an industrial automation application, namely a heat recovery ventilation system. MATLAB Simulink and Stateflow tool has been used to model and verify the application software. The Structured Text code generated by the PLC Coder Tool has been adapted to the TwinCAT3 environment and tested using the experimental setup developed in the study. The study demonstrates a methodology for the design and implementation of industrial controllers in a IEC61131-3 compatible language using a real-time environment. It was observed that there is not much knowledge in the literature about the procedure adopted, when the study was started. Through a practical research work, it was shown that the selected methodology and tools are appropriate for the Model-Based Design development of industrial automation systems.

Keywords: Model-Based Design, Automatic Code Generation

ÖZ

ISI GERİ KAZANIMLI HAVALANDIRMA SİSTEMİNİN MODELE DAYALI TASARIMI

Youssef H.Mussa Dabas

Yüksek Lisans

Elektrik ve Elektronik Mühendisliği Bölümü

Danışman: Yardımcı Doçent Dr. Mehmet Efe Özbek

Ocak 2018, 43 sayfa

Bu tezde, bir endüstriyel otomasyon uygulamasının, ısı geri kazanımlı havalandırma sisteminin, geliştirilmesi için model tabanlı tasarım iş akışının uygunluğu araştırılmıştır. Uygulama yazılımını modellemek ve doğrulamak için MATLAB Simulink ve Stateflow araçları kullanılmıştır. PLC Coder aracı tarafından üretilen Structured Text kodu, TwinCAT3 ortamına uyarlamış ve çalışma kapsamında geliştirilen deneysel düzenek kullanılarak test edilmiştir. Çalışma, gerçek zamanlı bir ortam kullanarak IEC61131-3 uyumlu bir dilde endüstriyel denetleyicilerin tasarlanması ve uygulanması için bir yöntem ortaya koymaktadır. Çalışmanın başlangıcında, kullanılacak yöntem hakkında literatürde fazla bilgi bulunmadığı tespit edilmiştir. Uygulamalı bir araştırma çalışmasıyla, seçilen metodolojinin ve araçların, endüstriyel otomasyon sistemlerinin model tabanlı geliştirilmesi için uygunlukları gösterilmiştir.

Anahtar Kelimeler: Model Tabanlı Tasarım, Otomatik Kod Üretimi

DEDICATION

To My Family,

ACKNOWLEDGMENTS

I express sincere appreciation to my supervisor Assist.Prof.Dr. Mehmet Efe Özbek for his guidance and insight throughout the research. Thanks also go to my wife, I offer sincere thanks for her continuous support and patience during this period.

TABLE OF CONTENTS

LIST OF TABLES

TABLE

LIST OF FIGURES

FIGURES

LIST OF ABBREVIATIONS

PLC - Programmable Logic Control

MBD - Model Based Design

TwinCAT - The Windows Control and Automation Technology

HSM - Heat Storing Material

CHAPTER 1

Introduction

In recent years there has been a major development in computing power which has contributed to more complicated system software requirements. Despite the rapid increase in the demands, the software development method is generally carried out by hand coding. However, recently several methods have been devised to do software by using computer-aided design (CAD) tools. Model Based Design (MBD) is one of the popular approaches used for this purpose. Figure 1.1 illustrates history of the usage of MBD approach in the company Cummins for the manufacture of diesel engines, which has adopted MBD [1]. In the absence of automatic code generators, modelling of controllers was done by hand coding which is slow and prone to specification, interpretation and implementation errors. The first available commercial MBD tools did not have simulation capability. Soon after, code generators capable of simulating the component behaviour and then the whole system's behaviour have emerged providing considerable savings in the development and testing processes.

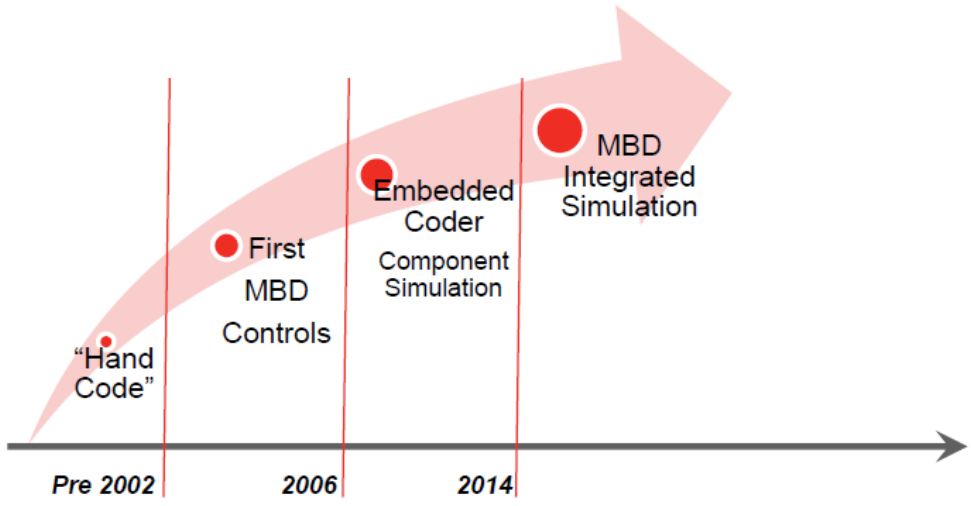

Figure 1.1 the development of using MBD approach in the company Cummins for the manufacture of engine

Development of an industrial automation system can be quite time-consuming. Virtual simulations are frequently used to decrease the development time especially the time spent for testing and validating the designed system. However, due to the lack of integration between simulation tools and the software development environments the same work is done more than one time increasing the labour and time spent for development. It is desirable to use, as much as possible, the system model which has been developed for simulating the system behaviour, also as a basis for application code development [2].

Today, the industry is experiencing a pressure to decrease the time-to-market and a trend to produce customized products. These call for flexibility in manufacturing systems, rapid development of new production lines [3]. MBD approach can help to achieve these goals by accelerating the development workflow.

The purpose of this thesis is to answer this question: Is it possible and feasible to automatically generate code starting from a system model, and then integrate this code in an Integrated Development Environment (IDE) that is commonly used for developing industrial automation software.

The main significance of this research is to contribute to the establishment of a software development workflow comprising system modelling, design simulation, automatic code generation commissioning and testing. The possibility of generating code from a physical model which is compatible with the TwinCAT3 IDE has been studied. Since the subject is relatively new, there are not many studies in the literature which help to identify and solve the problems that emerge during the process, so considerable time and effort was spent for troubleshooting during this study.

The remaining part of this thesis is organized as follows:
Basic information about Model-Based Design and literature review is described in chapter 2. Chapter 3 describes the research methodology used in this study. Chapter

4 illustrates the hardware implementation section of this thesis. Conclusion of the thesis is provided in chapter 5.

CHAPTER 2

A Review Concepts and The Literature

2.1 Model-Based Design

Model-Based Design is an approach to develop dynamic systems fast and inexpensively, including control, signal processing, and communications systems. In Model-Based Design, the system model is at the focus of the development procedure, starting from the requirements, through design, implementation, and testing. After the model development process is complete, simulation is beneficial to determine whether the model satisfies the system requirements. After it is verified that the model captures the system requirements, facilities for code generation directly from the model is available in MBD platforms, which lead to considerable time savings and avoid hand-coding mistakes [1].

2.1.1 Software Development Process

Development of software often include new ideas, but the main concept remains constant. During development procedure the integration level changes [4]. V design concept takes this into consideration, which is illustrated in Figure 2.1.

Classically, the system design stages are represented in the left side of the V-model, while the verification and validation stages are represented in the right side.

The first step is requirement analysis, which is about determining what the typical system has to execute, without defining how the software will be designed or built. This is the part where target requirements and specifications are made.

The second step is system design. At this stage, the system has been studied and analysed by studying and understanding the user's true needs. Then the system is tested by using Model-in-the-Loop (MiL) method until the system meets the requirements [5].

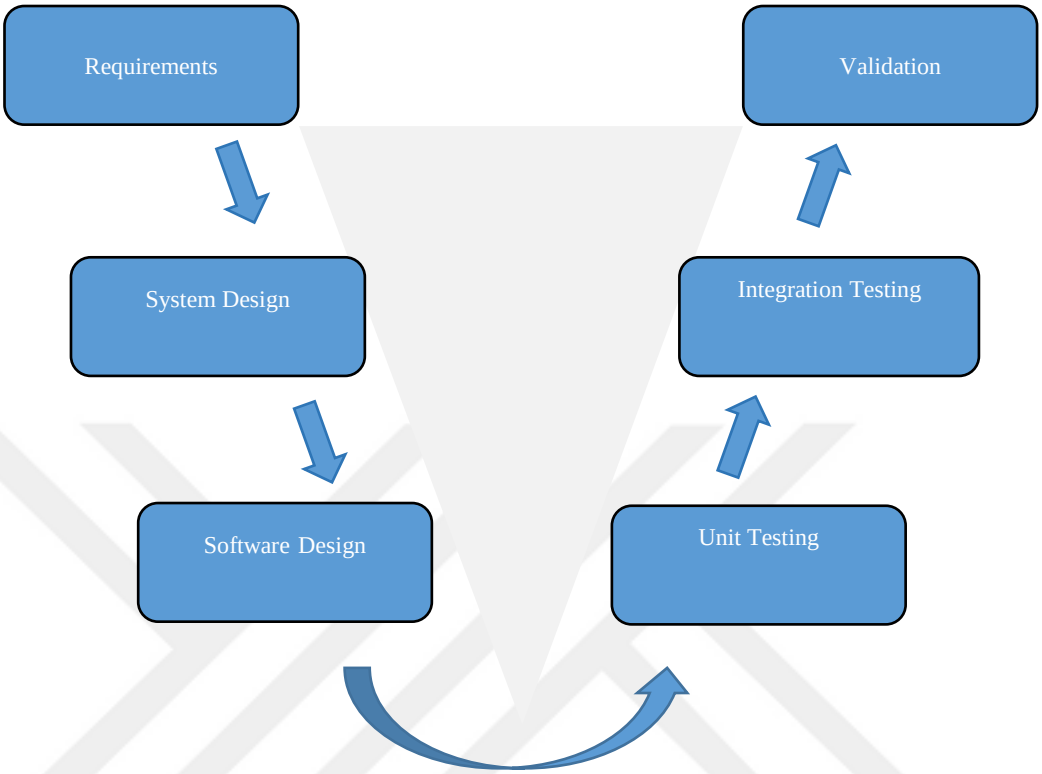

Figure 2.1 V design model

In the third stage, code is generated, and is tested by Software-in-the-Loop (SiL). Moving along the V-model the next step is unit testing which is for testing the processor to be used. This phase is not critical in this thesis since a PLC which is thoroughly tested by the manufacturer is available. Unit testing constitutes a large part in cases where the processors hardware development is a part of the process. The integration testing stage is where the generated code is executed with the hardware and the Hardware-in-the-Loop (HiL) test is done, which is made to sure if the controller created in the design stage works well within the system. Last step of the V model is validating that the outcomes satisfy the system's requirements [5].

2.1.2 Advantages of Model-Based Design:

One of significant features of Model Based Design is the simulation of the model instead of a performing a physical test. This contributes to reduce the cost and time and makes it possible for the designer to compare several designs and to choose the best one.

The weakness of programming skills is one of the important reasons that led several companies to choose the Model-Based Design approach. Experience in programming or hardware description language is a crucial component of traditional development methods. On the contrary, graphical design interface used in a model based design setting does not require expertise [6]. Moreover, this method takes advantage to reuse systems and code when facing similar problems with another design. This leads to reduce the time required for development [5]. The ease of understanding the graphical design, extends the circle of design participation by people that are not familiar with software development, as well as engineers who prefer to draw schemes to understand the design [6, 7].

Automated code generation is extremely useful to reduce the development time for an embedded application. Also it allows the designer to concentrate on the design instead of code writing [8]. By utilizing automated code generation, the functional correctness demonstrated in the simulation step is also preserved in the implementation.

Model-Based Design method also provides the advantage of automatic documentation, that helps to understand and revise the design, for future development and maintenance.

2.2 **IEC 61131-3 Standard**

IEC 61131-3 is the third part of The IEC61131 standard for programmable logic controllers (PLC), which specifies all the necessary hardware and software requirements to create a PLC system. Before IEC61131-3 standard, many program methods were used by PLC manufacturers. This was a big challenge from users' point of view and PLC markets. One of the essential criteria for this standard was to define clear and truly manufacturer-independent programming languages that are compatible to those used before.

In 1993 The official version was published. After that it has been refined twice in 2003 and 2013. Even though many trials were made with the collaboration of main PLC manufacturers, IEC61131-3 was the first to receive the necessary international recognition as a global standard for PLC programming [9] .

The IEC61131-3 defines common elements of five of Programmable Logic Controller(PLC) programming languages. There are two graphical languages, Ladder Diagram - LD and Function Block Diagram – FBD and two textual languages, Structured Text - ST and, Instruction List - IL. The standard also defines a graphical language Sequential Function chart - SFC.

Ladder diagram (LD) is similar to electrical circuit diagram, which are easy to with people who have an electrical knowledge. LD is fundamentally intended for describing a Boolean-type signal flow from left to right as shown in Figure 2.2.

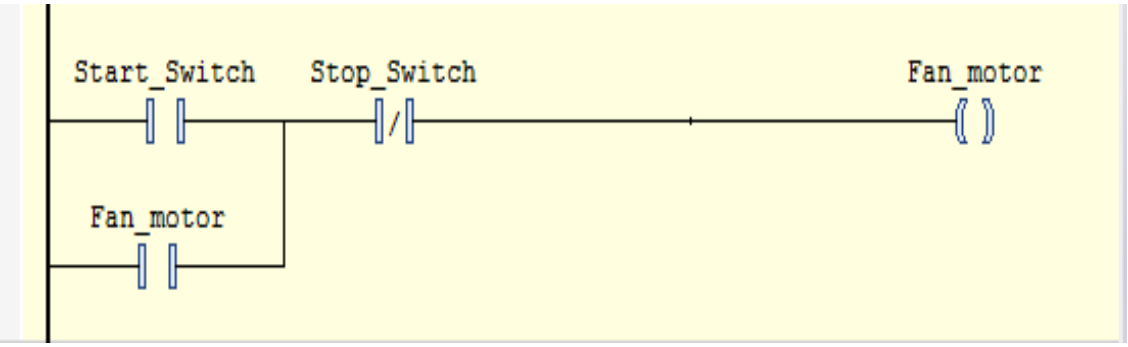

Figure 2.2 An example of Ladder diagram (LD)

Function Block Diagram – FBD represents the behaviour of elements of program as a group of graphic blocks, as in electronic schematics. It models the system as an collection of components and signals flowing from left to right among those components [10]. Figure 2.3 shows an example of the function block diagram language FBD.

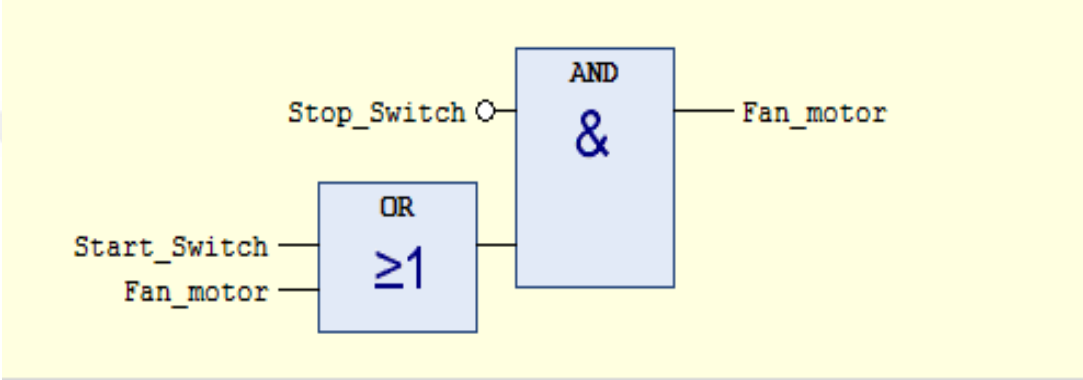

Figure 2.3 Function Block Diagram – FBD

Instruction List (IL) is classified among low-level languages and it has given good results when dealing with simple problems. IL can be executed directly by several IEC61131-3 compliant PLCs [11].

```
LDN     Stop_Switch
AND     Start_Switch
ST      Fan_motor
```

Figure 2.4 Instruction List - IL

Sequential Function chart (SFC) is a tool for defining the general state transition structure of a program, rather than a programming language. For applications including sequences of actions, SFC specifies transition conditions and step actions which are programmed in other standardized languages [12].

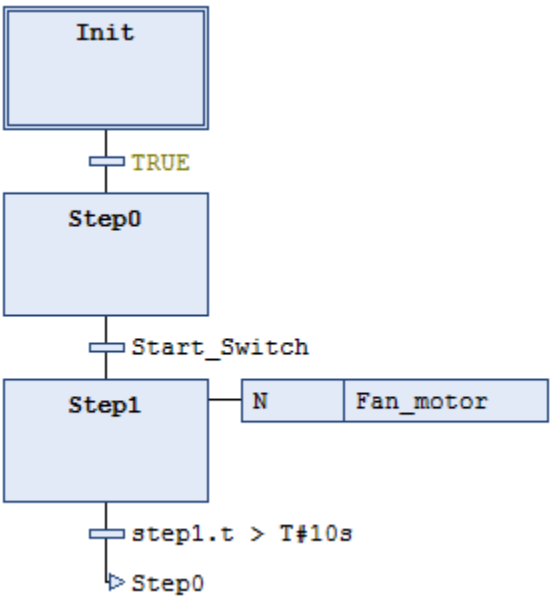

Figure 2.5 A sample of SFC

Structured Text (ST) is a high-level text-based language. An example of ST is shown in Figure 2.6. Although it has a language structure (syntax) that similar the syntax of a high-level programming language with variables, loops, operators and conditions, structure text is intended for industrial control applications. It is specifically beneficial for complex mathematical and logic operations. ST is easy to learn for people who already familiar with high-level programming languages [13].

```
PROGRAM MAIN
VAR
    Fan_motor :BOOL;
    Start_Switch : BOOL;
    Stop_Switch : BOOL;

IF Stop_Switch THEN
    Fan_motor:=FALSE;
    ELSE IF Start_Switch THEN
        Fan_motor:=TRUE;
        ELSE
            Fan_motor:=FALSE;
        END_IF
END_IF
```

Figure 2.6 Structured Text

2.3 **MATLAB**

MATLAB is one of a high-level programming language and it is a leading program in the engineering and mathematical applications which was produced by MathWorks. MATLAB allows mathematical manipulation with matrices, a mathematical chart, implements various algorithms, creates graphical user interfaces, and integrates with programs written in other programming languages. It has been used by millions of users through academia and industry fields. Although MATLAB was intended for manipulating with data visualization tools, nowadays many toolboxes are present for different purposes [14].

2.4 **Simulink**

Simulink is a model-based design environment integrated with MATLAB. It is a data flow graphical programming language tool where block diagrams can be built using

a wide array of blocks. Automatic code generation, simulation and test are among the features supported by Simulink environment [15].

2.5 Simulink PLC Coder

Simulink PLC Coder is a Simulink extension for generating Structured Text code from Simulink models, MATLAB functions and Stateflow charts. The codes are generated in PLCopen XML and formats used by a wide range integrated development environments (IDEs) are also supported. With PLC Code it is possible to compile and deploy an application to many programmable logic controller (PLC) and programmable automation controller (PAC) devices. [16].

2.6 TwinCAT 3

Beckhoff launched PC-based control technology in 1986 and developed TwinCAT (The Windows Control and Automation Technology) which is an automation suite that turns a PC-based system into a real-time controller with multiple virtual PLC systems inside. TwinCAT 3 is the latest version of TwinCAT, which is integrated with Microsoft Visual Studio. TwinCAT 3 supports many programming languages including C/C++ for real time applications and the object-oriented extension of IEC 61131-3 [17].

CHAPTER 3

Problem Definition and Methodology

3.1 An illustrative model

In this thesis, in order to study and demonstrate model-based design concept has been applied to the development of a real-life device. The model was decided for this example is called a heat recovery ventilation system. A model-based software design procedure using Simulink and TwinCAT has been applied for the design of the example system. The basic idea of a heat recovery ventilator is to ventilate a room or building conserving the heat inside the room as much as possible by heating the incoming fresh air using the heat recovered from the outgoing contaminated air.

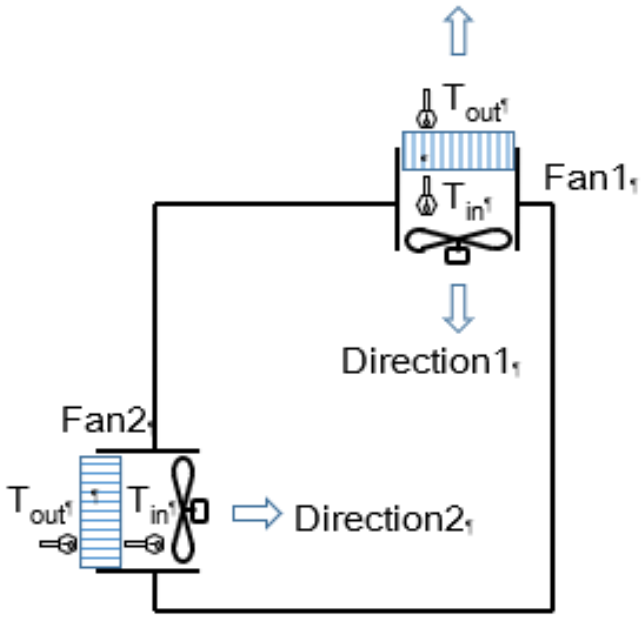

Figure 3.1. System diagram of a heat recovery ventilation system

3.1.1 Operation of System

The diagram presented in Figure 3.1 illustrates heat recovery ventilation system used in this study. Essentially heat recovery ventilator comprises two fans which can rotate in different directions under the control of a PLC program and a heat storing material (HSM) placed such that air flow passes through. The heat recovery ventilation cycle starts with the PLC operating the fan so as to blow the air in the outside direction. When the air flow is towards outside, the warm (contaminated) air shares some of its thermal energy with the HSM and the temperature of HSM increases. A temperature sensor is placed at outer side of the HSM to measure the temperature of the air leaving the HSM, namely T_{out}. This is necessary so that the controller (PLC) can figure out how much heat transfer has occurred. When T_{out} drops to a predetermined value $T_{out\text{-}lim}$ it is concluded that the recovery of heat from the outgoing air is satisfactory. After a waiting for the fan to come to a stop for 5 seconds, the PLC acts to change the fan direction in order to let fresh air come inside through the HSM. Since HSM has already been warmed by the outgoing air, its temperature is higher than the temperature of the air coming from outside, leading to a transfer of heat from the HSM to the incoming air. A temperature sensor placed at inner side the HSM measures the temperature T_{in} of the incoming air after passing through the HSM. The inward flow of air continues until T_{in} reaches a predetermine $T_{in\text{-}lim}$ value indicating that it has recovered sufficient heat from HSM. At this point the PLC pauses for 5 seconds and start the same cycle from the beginning. Within one heat recovery ventilation cycle, the thermal energy of the air inside is first transferred to the HSM and then to the cold air flowing in. The amount of heat recovered depends on temperatures inside and outside the room, durations of inward and outward air flows and the properties of the HSM. In this study the main parameter under our control is the duration of inward and outward air flows and is decided by the algorithm running on the PLC by measuring T_{in} and Tout.

3.1.2 Modelling the system

In the Model-Based Design and development of an industrial automation system, the controller is represented by a model. The model can be as simple as a collection of a few signals or can be much more complicated, depending on the context. The main idea is that the model should satisfy the performance requirements when it is converted to an executable application at the end of the Model-Based design workflow.

In the modelling phase the key point is capturing the requirements and translating them into an executable model. In the case of the heat recovery ventilation system, the controller is supposed to move between a number of states to perform its task. Therefore, a finite state machine is a good choice for modelling this aspect of the controller. In the next section we elaborate on the states finite state machine and define the states which represent the modes of operation.

3.1.3 Defining the operating States

States are modes of operation in a physical system. Each mode in which the system can operate should be identified, therefore, to determine the number of states needed for the system. Usually, a table is very useful for identifying each mode and determining dependencies between states [18].

Table 3.1 States of the controller and the dependencies between the states

Operating Mode	Description	Dependencies
Power Off	Turns off all power in the control system No fan can operate when power is off.	No fan can operate when power is off.
Power On	Turns on all power in the control system	Zero, or two fans can operate when power is on.
Direction 1	Activates Fan 1 & Fan 2	Direction of rotation of Fan1 is outward and Fan2 is inward. The transition to the next state occurs if value of outside temperature equal $T_{in\text{-}lim}$
Pause 1	Activates timer	Fan 1 is off & Fan 2 is off. Timer is on and the transition to the next state occurs if 5 secs passed.
Direction 2	Activates Fan 1 & Fan 2	Direction of rotation of Fan1 is inward and Fan2 is outward. The transition to the next state occurs if value of inside temperature equal $T_{out\text{-}lim}$
Pause 2	Activates timer	Fan 1 is off & Fan 2 is off. Timer is on and the transition to the next state occurs if 5 secs passed.

Stateflow is very useful tool form Simulink that is used to design and simulate models based on flow charts and state machines. One of the main features of Stateflow is that it performs animation illustrating the state machine operation. It also performs static and run-time tests for checking if the model is executable. In this study we have used the Stateflow tool for modelling and developing our target system. Figure 3.2 shows the heat recovery fan controller model that we have designed by using Stateflow.

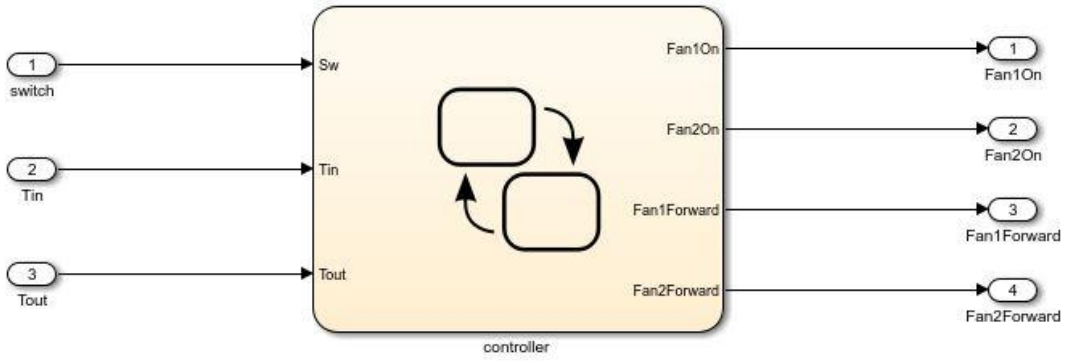

Figure 3.2 The Stateflow model of the heat recovery fan controller

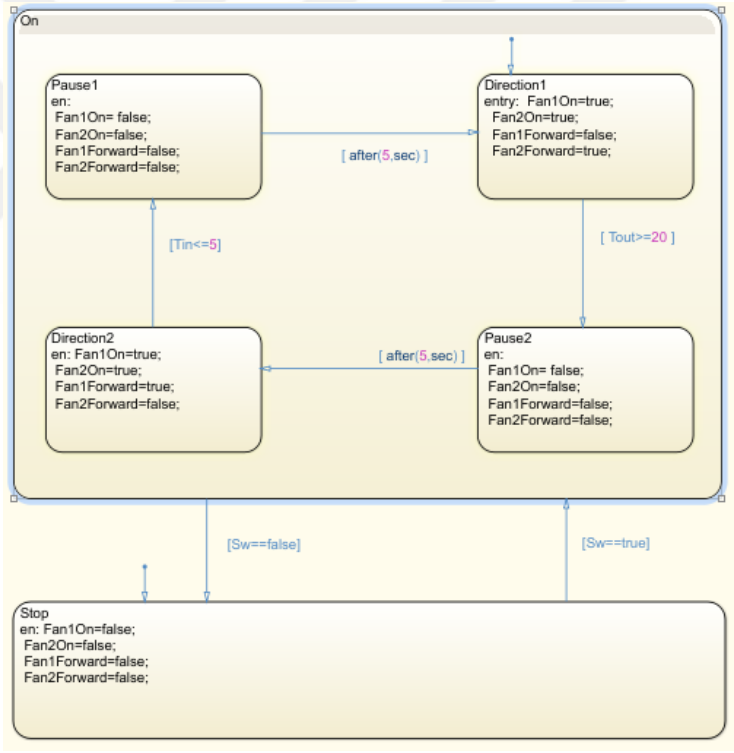

Figure 3.3 Heat recovery fan controller Model: Nested states shown.

3.2 **Code generation**

One of the significant feature of using Model-Based Design method is the ability to generate code automatically from Simulink models [4] . By using automatic code generation, hand coding mistakes and time of development can be minimized [19]. In this project Simulink PLC Coder was used for generating code. There are several steps that should be accomplished before the code can be generated. First the model should be encapsulated in one so-called one subsystem, since PLC Coder does not support generating and importing multiple subsystems into PLC software system to avoid serious overlapping problems. In this example all the target components are placed under one subsystem as shown in Figure 3.4.

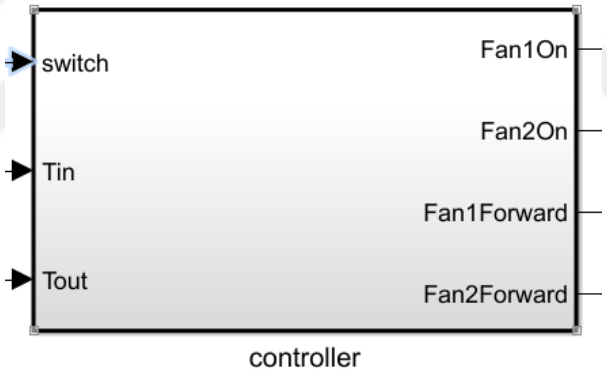

Figure 3.4 subsystem of the system's controller

The second step is to define the subsystem as an atomic subsystem, because PLC Coder integrates only this type of subsystems. Subsystem can be determined as atomic as shown in Figure 3.5.

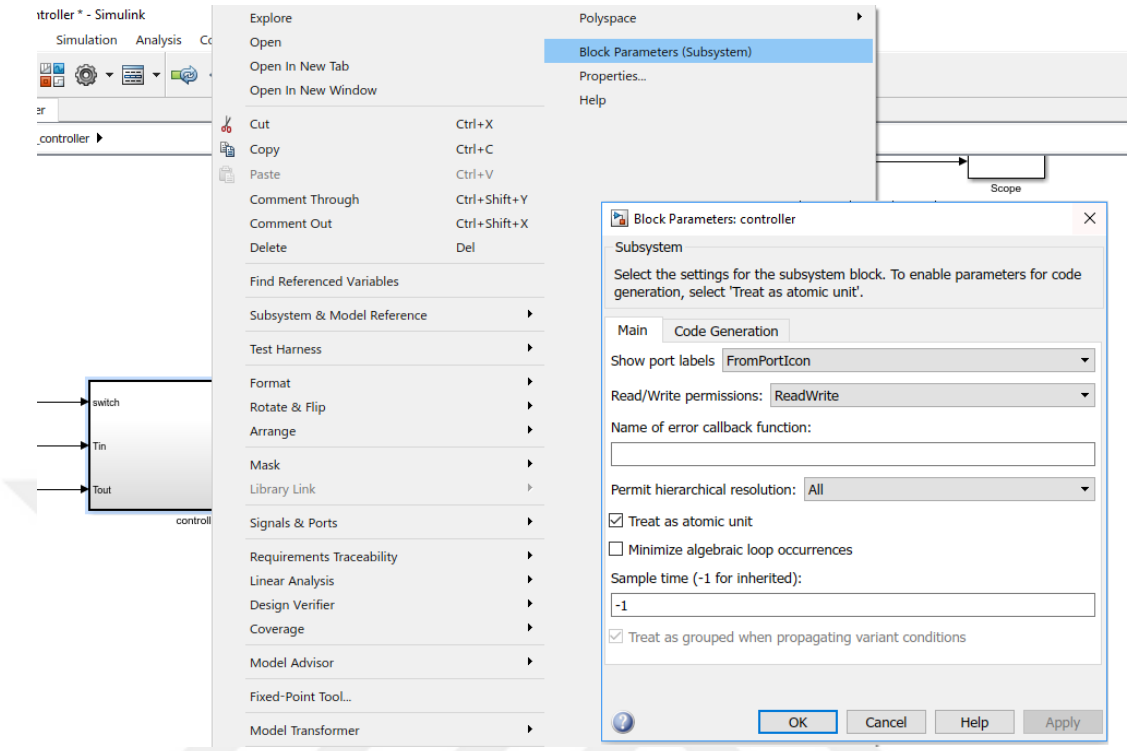

Figure 3.5 Enabling "Treat as atomic unit" to generate code

The third step is checking if the Subsystem is compatible by using PLC Coder's check compatibility tool. If PLC Coder identifies any errors in the compatibility check, code generation process cannot be done. Figure 3.6 shows activation of compatibility checker.

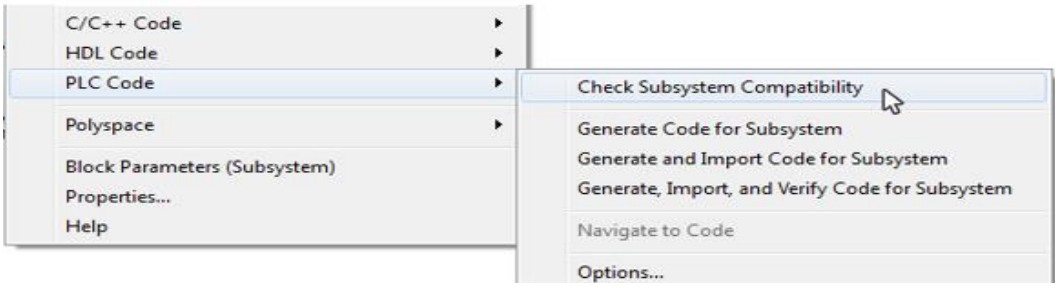

Figure 3.6 Compatibility check tool to check if subsystem is compatible with PLC Coder

At this point the target IDE software should be set up to generate Structured Text code. Even though the definition of the ST language is accurate, there are a little bit difference in implementation syntaxes and structures for each IDE [6].

The IDE in this thesis is TwinCAT3 software, which does not exist in the list of target IDE in Simulink PLC coder version R2016b. The solution for this obstacle is by choosing alternative target IDE which integrates with TwinCAT3 environment. So PLCopen XML was chosen as target IDE from the options menu showed in Figure 3.7. Which is complete compatible with TwinCAT3 software.

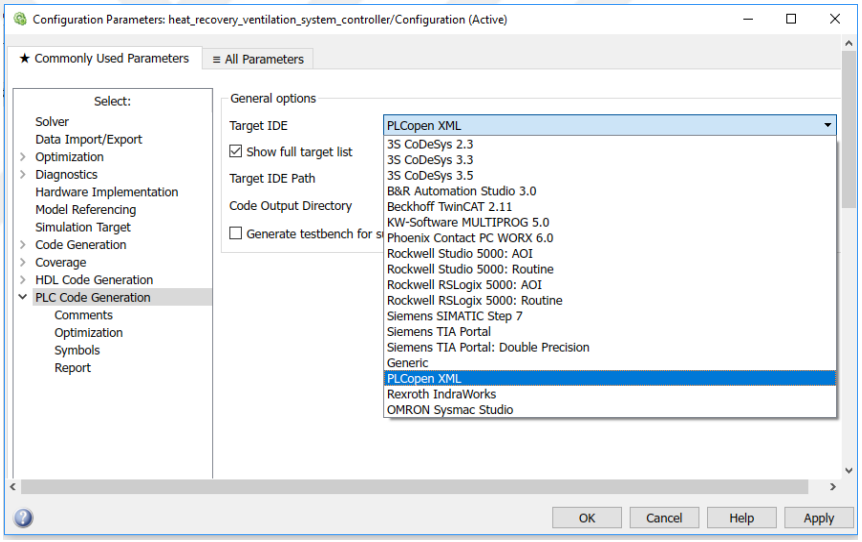

Figure 3.7 Determining the target IDE shapes the code to implementable form

There are three ways available for generating code in the PLC Coder's options the first one (Generate Code for Subsystem) generates an importable text file. The second (Generate and Import Code for Subsystem) the generated ST code file is imported automatically to IDE. The last one (Generate, Import, and Verify Code for

Subsystem) importing the code and in addition, performing verification by running the generated code automatically in the IDE [6].

Since automatic importing code options are not supported for TwinCAT 3 IDE in PLC Simulink Coder, the generated code was imported manually.

To import the generated code to TwinCAT3 IDE, we should open new PLC project then right clicking on POUs folder Then select 'Import PLCopen XML' from the menu and locate the generated xml-file as shown in Figure 3.8.

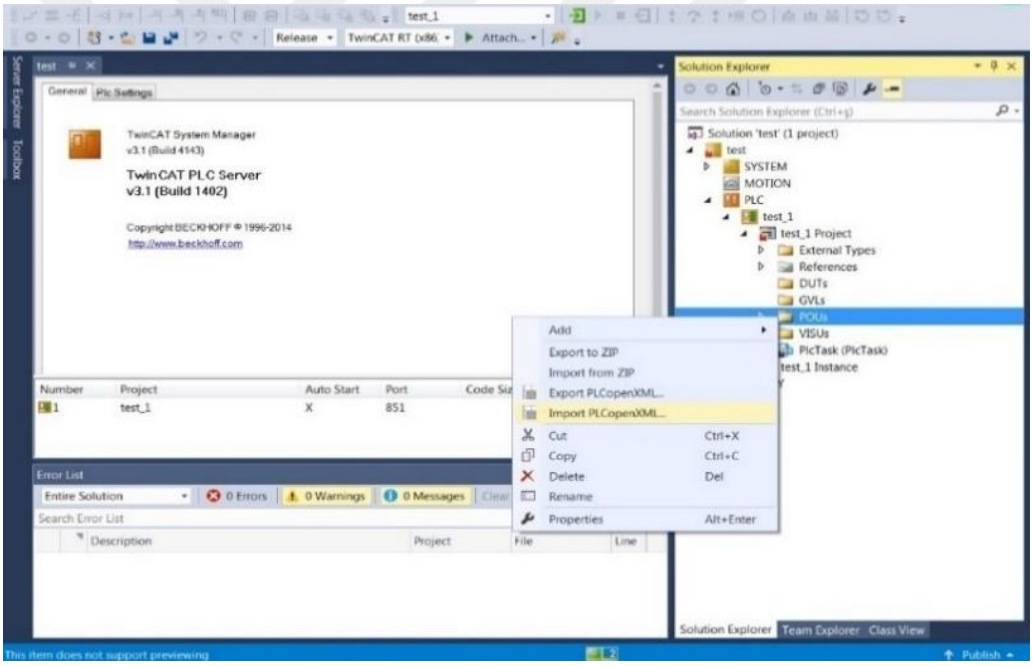

Figure 3.8 import generated code into TwinCAT3 IDE

3.3 **Integrating Generated Code with Beckhoff® TwinCAT3 IDE**

At this point, the code which is executable on the PLC is already. The code generated for the subsystem block in Simulink model is imported as a function block into TwinCAT 3 IDE as shown in Figure 3.9. The PLC code is simulated in the

TwinCAT 3 environment to validate that its syntax is compatible with the IDE and to verify that it performs the desired function. In this simulation step, rather than associating with the I/O hardware, the input values are entered manually from the IDE, using the utilities provided by the TwinCAT 3 IDE. This facility is a big advantage enabling early verification of the model and constitutes one step of the Model-Based design approach.

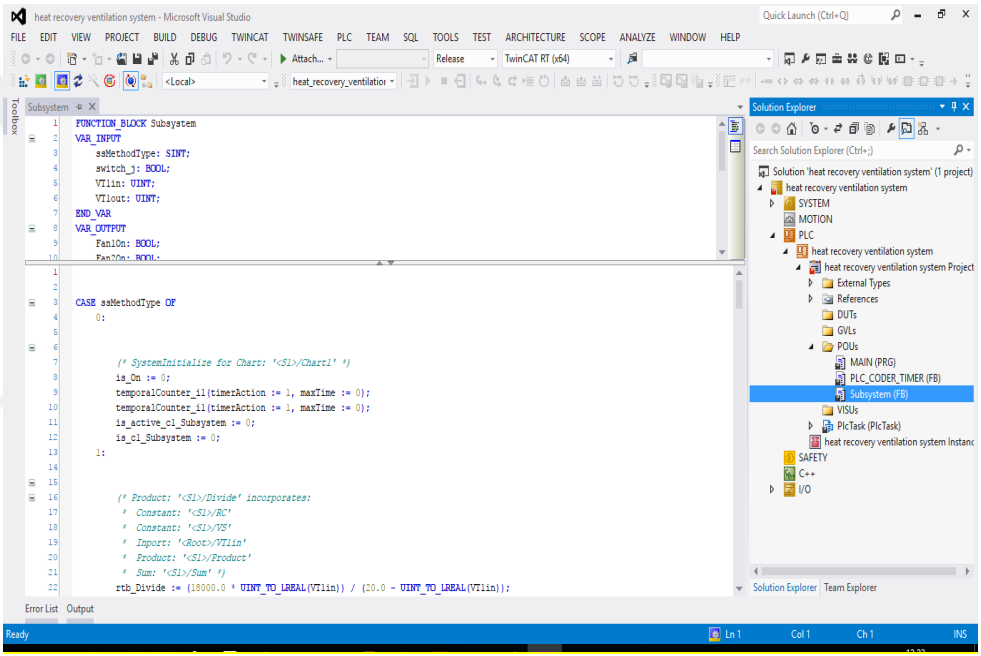

Figure 3.9 The generated code is imported as a function block into IED software

3.4 **Designating Inputs and Outputs of PLC**

After the code generation and model verification process are completed, provisions should be made so that the code can interact with the hardware. The generated code should be edited in order to associate input and output variable with the I/O hardware modules.

Designating inputs and outputs variables is done by modifying the declaration of variables. Figure 3.10 shows the original list of variables in the heat recovery ventilation PLC program.

```
VAR_INPUT
    ssMethodType: SINT;
    switch : BOOL;
    VTlin  : UINT;
    VTlout : UINT;
END_VAR
VAR_OUTPUT
    Fan1On : BOOL;
    Fan2On : BOOL;
    Fan1Forward : BOOL;
    Fan2Forward : BOOL;
END_VAR
VAR
```

Figure 3.10 Original variable declaration in the PLC code.

According to syntax in TwinCAT3 program, we should add (`AT %Q*`) directly after the variable name to designate a variable as an output and (`AT %I*`) after the variable name to designate variable as an input in program, as illustrated in Figure 3.11.

```
VAR_INPUT
    ssMethodType: SINT;
    switch AT %I*: BOOL;
    VTlin  AT %I*: UINT;
    VTlout AT %I*: UINT;
END_VAR
VAR_OUTPUT
    Fan1On      AT %Q*: BOOL;
    Fan2On      AT %Q*: BOOL;
    Fan1Forward AT %Q*: BOOL;
    Fan2Forward AT %Q*: BOOL;
END_VAR
```

Figure 3.11 Variable declaration after designating as an inputs and outputs in PLC program.

After the PLC project is compiled and built, PLC Task's input and output nodes will appear under "heat recovery ventilation system Instance" node in the Solution Explorer pane as shown Figure 3.12.

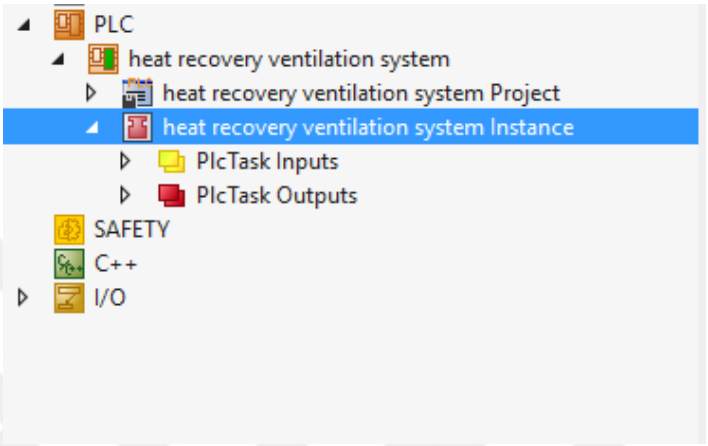

Figure 3.12 Plc Task Input and output nodes

By expanding the PLC Task inputs and outputs, the variables of inputs and outputs that were designated are available, which normally is mapped to physical I/O as shown in Figure3.13.

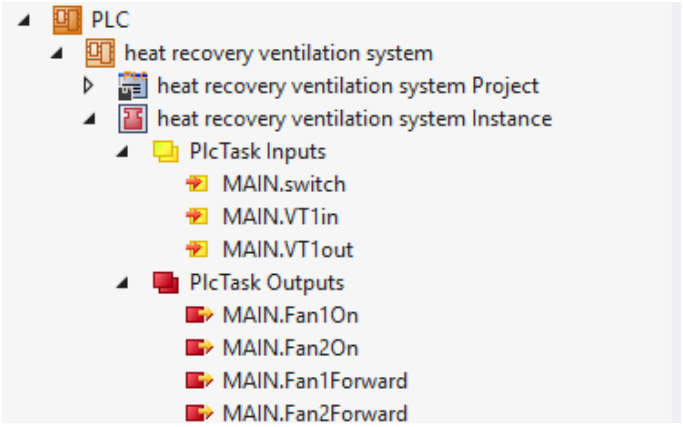

Figure 3.13 Designated inputs and outputs for mapping to hardware I/O

CHAPTER 4

Hardware Implementation

4.1 I/O Hardware

In the experimental part of this study we have used EtherCAT Terminals produced by Beckhoff Company as I/O modules. Beckhoff EtherCAT system contains modular I/O terminals which are connected to a EtherCAT fieldbus cable through the so-called Bus Coupler. The field bus cable is a standard Ethernet cable that terminates on the standard Ethernet port of the PC running TwinCAT 3. In **Error! Reference source not found.** and **Error! Reference source not found.** the EtherCAT Terminals are shown respectively, and the hardware setup used in this study.

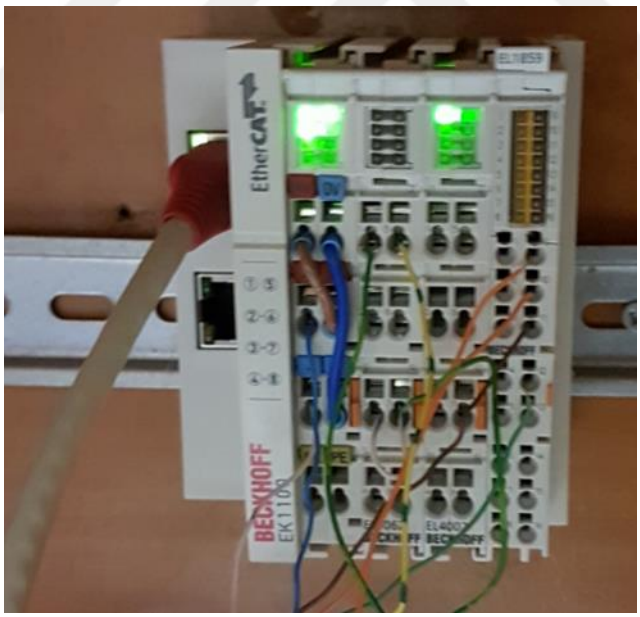

Figure 4.1 EtherCAT Coupler and I/O Terminals

The system developed contains two analog inputs for temperature measurement. The analog input terminal EL3062 has been used for the conversion of these signals. For each fan, two digital outputs are necessary, one for turning the fans on/off and the other for changing its direction of rotation. An EL1859 (8-channel digital input + 8-channel digital output) module was used to generate these digital control signals. An EK1100 EtherCAT Coupler is used for connecting these modules to the EtherCAT fieldbus.

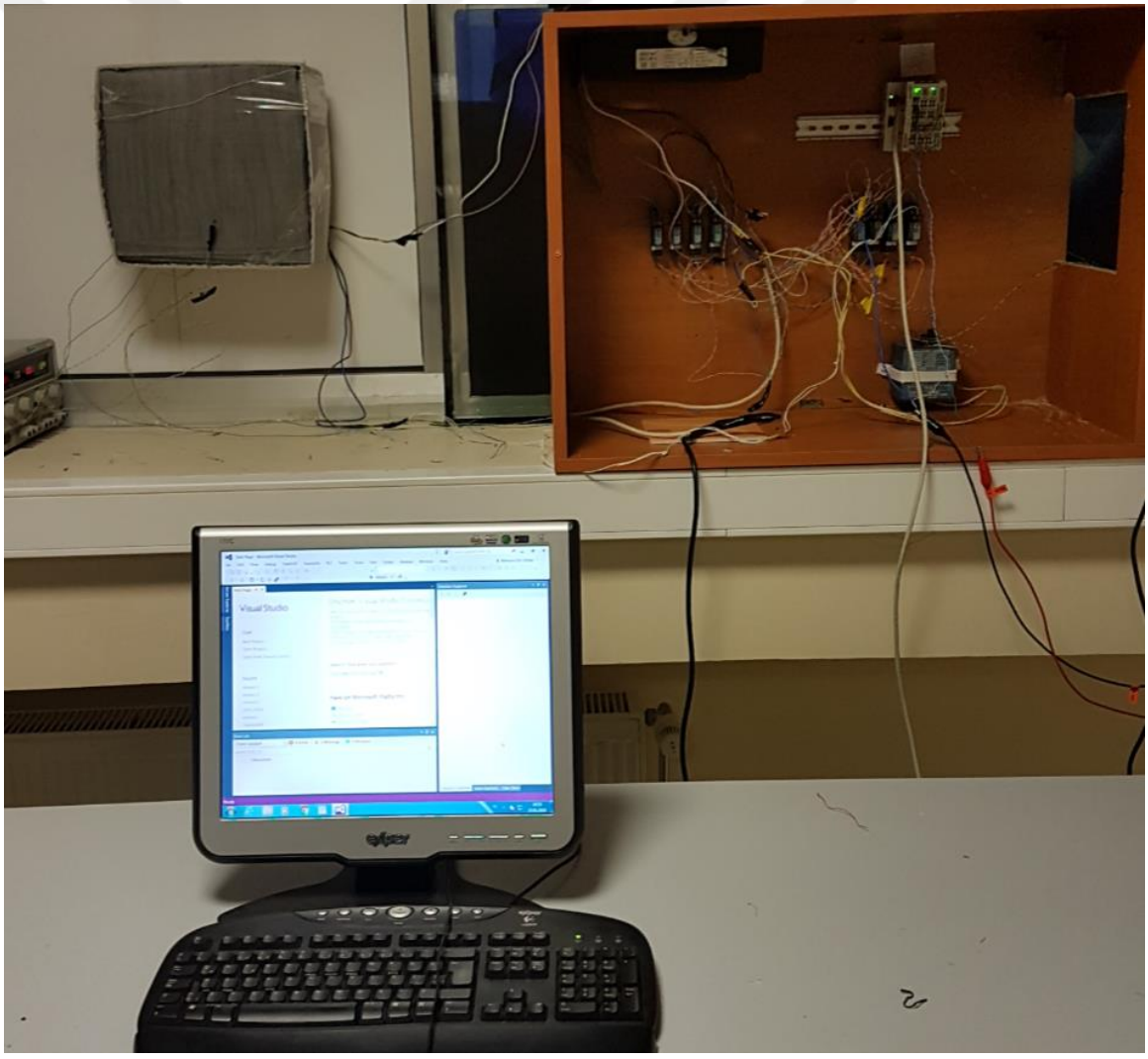

Figure 4.2 The Hardware Setup

4.2 **Temperature Measurement**

In industrial automation, temperature measurements are performed by temperature transducers which convert temperature to voltage. A temperature transducer typically contains a sensing element, namely a thermistor, the resistance of which changes significantly with temperature.

4.2.1 Thermistor

A thermistor is a type of resistor the value of which is dependent on temperature. Thermistors are widely used as temperature sensors and inrush current limiters.
There are two types of thermistors. Negative Temperature Coefficient (NTC) and Positive Temperature Coefficient (PTC). In our application, temperature readings are necessary for triggering transitions from one state to another within the state machine. Figure 4.3 shows NTC thermistors which use in this study for sensing the temperature .

Figure 4.3 10kΩ NTC thermistor

The relationship between the thermistor resistance and temperature is highly nonlinear. A typical resistance-temperature curve is shown in Figure 4.4.

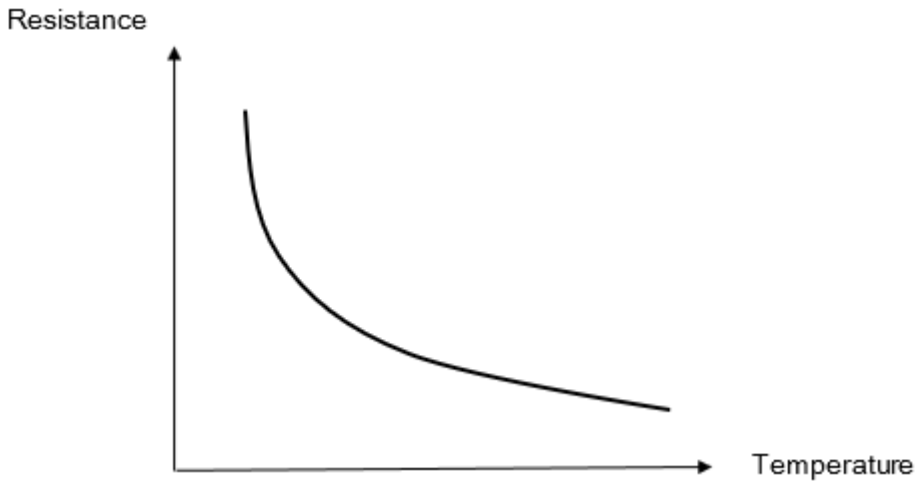

Figure 4.4 Resistance - temperature curve for NTC

Different approximations can be utilized in order express the resistance-temperature relationship mathematically. In this study we use the so-called beta formula which gives satisfying results in the range of 0°C to +100°C. The formula is dependent on a single material constant β and can be written as:

$$R_T = R_{T0} \cdot e^{\beta\left(\frac{1}{T}+\frac{1}{T_0}\right)} \qquad (4.1)$$

If two resistance measurements are made, the value of β can be calculated using the recorded data as

$$\beta = \frac{ln\left(\frac{R_T}{R_{T0}}\right)}{\frac{1}{T}-\frac{1}{T_0}} \qquad (4.2)$$

where R_{T0} is the resistance measured at T_0, and R_T is the resistance measured at T. In this study temperature measurements have been made at 273°K and 291°K and β values have been calculated.

4.2.2 Temperature Transducer Design

The resistance of the NTC should be converted into a voltage so that it can be measured by the analog input module. A typical circuit used for this purpose is a voltage divider shown in Figure 4.5, where the thermistor resistance R_T is in series with a fixed resistance R_C. The voltage across the thermistor is

$$V_T = V_S \frac{R_T}{R_T + R_C} \qquad (4.3)$$

In order to get more accurate temperature measurements, it is desirable to increase the sensitivity which can be defined as the change in V_T corresponding to a unit change in the temperature. This can be done by using an appropriate resistance value (R_C) in series with thermistor's resistance (R_T).

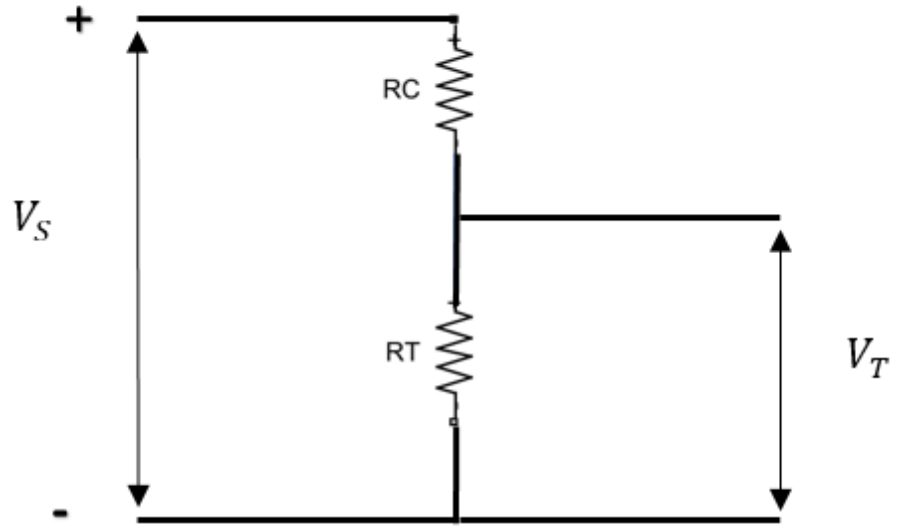

Figure 4.5 Diagram for transducer circuit comprising a thermistor.

In determining the optimum value of RC, our aim as to obtain the largest possible value of ΔVT, in the temperature range where the transducer will be operating. Considering the prospective temperatures of air flows through the heat recovery ventilator, the operating range of the transducer has been determined to be 273°K to

291°K. (See Figure 4.6) The lower and upper limits of this range correspond to indoor and outdoor temperatures of a building in winter time, respectively.

The following procedure has been followed for optimizing the value of RC: Using resistance values recorded at T = 291°K (indoor) and T = 273°K (outdoor) we calculate

$$\Delta V_T = V_T - V_{T0} \qquad (4.4)$$

where

V_{T0} is value of voltage on the thermistor at 273 K°

V_T is value of voltage on the thermistor at 291 K°

By substituting equation (4.3) in equation (4.4), we get

$$\Delta V_T = V_S\left[\frac{R_{T0}}{R_{T0}+R_C} - \frac{R_T}{R_T+R_C}\right] \qquad (4.5)$$

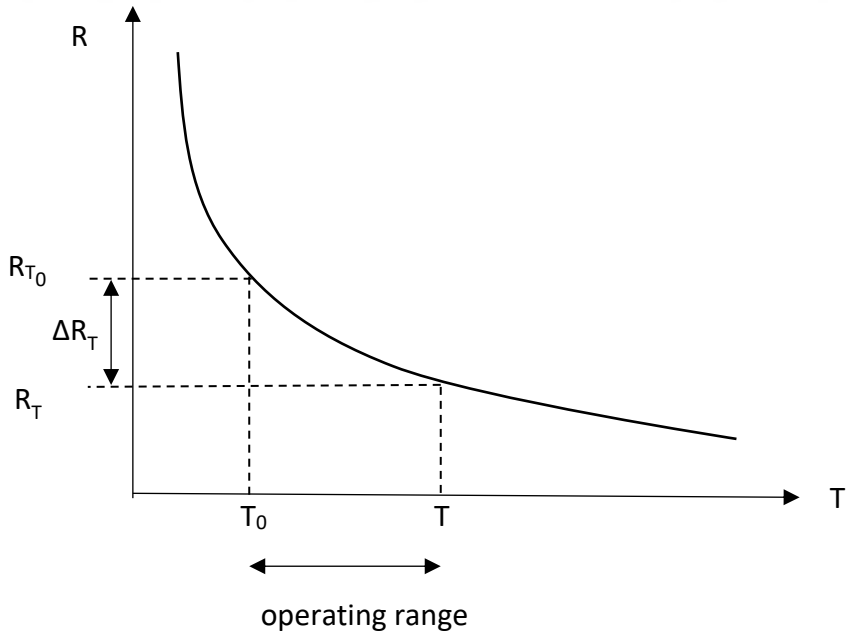

Figure 4.6 The operating range of temperatures

For a selected R_C value, ΔV_T is calculated using equation (4.5). This calculation is repeated for several values of R_c using MS Excel and the results are tabulated. The value of R_C which gives the maximum ΔV_T is identified as the optimum R_C value (R_{Cmax}).

The optimum R_C value is calculated for thermistors with different nominal values (10kΩ, 50kΩ ,100kΩ). ΔV_T is plotted as a function of R_C in Figure 4.7,

Figure 4.8 and

Figure 4.9 so as to illustrate the optimum value of R_C.

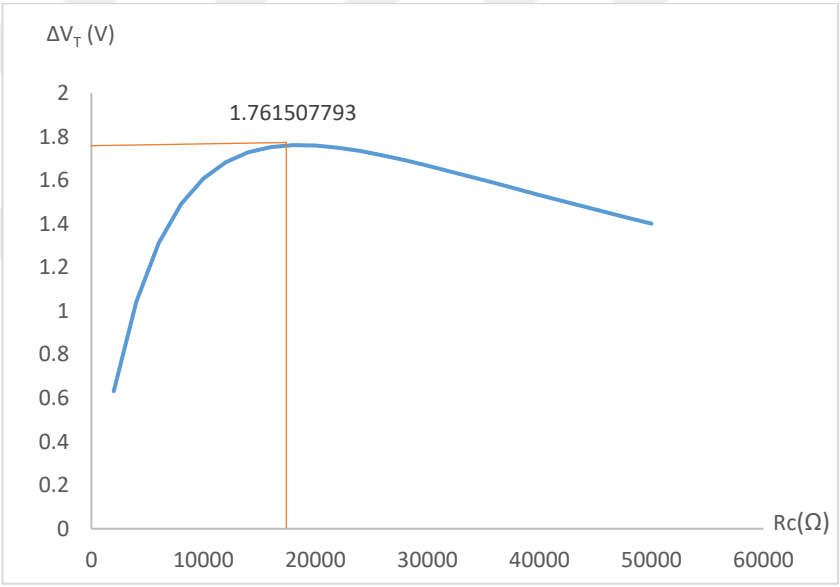

Figure 4.7 Optimum R_c value for thermistor (10kΩ)

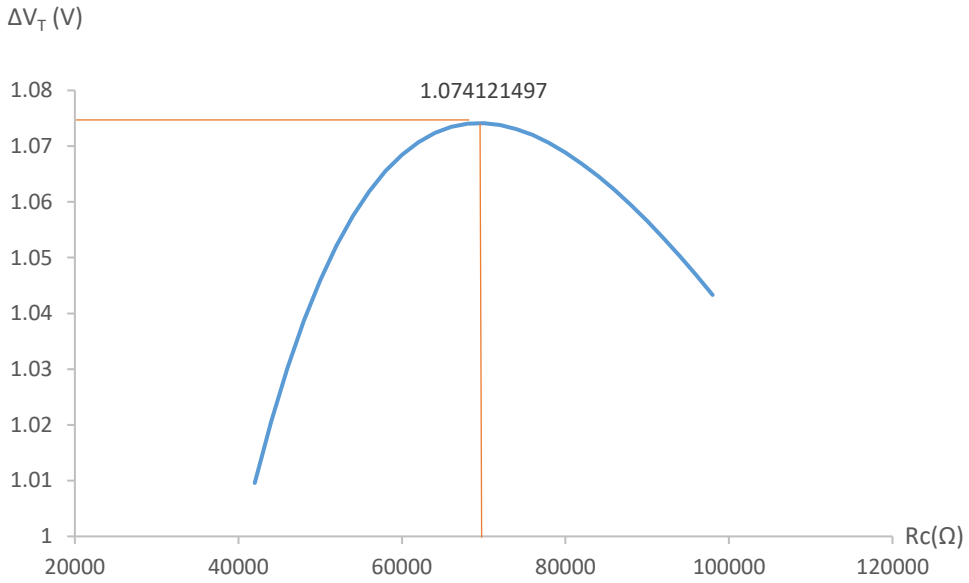

Figure 4.8 Optimum R_c value for thermistor (50kΩ)

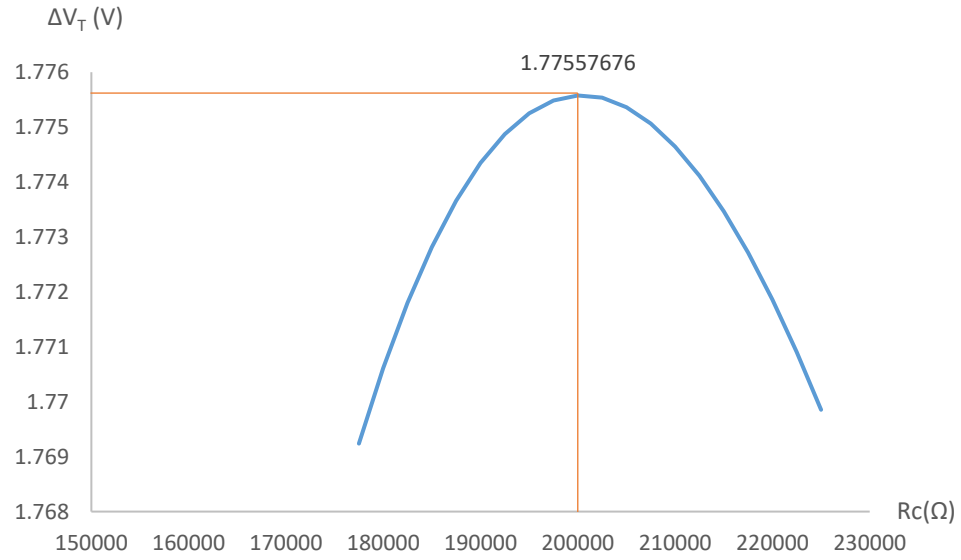

Figure 4.9 Optimum R_c value for thermistor (100kΩ)

Inspection of equation 4.5 reveals that the value of the supply voltage V_S is also effective on the sensitivity of temperature measurements. By increasing V_S, the sensitivity can be increased as desired, as long as the transducer output V_T does not go beyond the input range (0-10V) of the analog input module. In order to avoid such an overflow, we set the maximum value of V_T (which occurs at the minimum temperature) to 8V which is sufficiently below the maximum permissible voltage (10V) and calculate the corresponding V_S from equation 4.6.

$$V_S = V_T \frac{R_{T0}+R_C}{R_{T0}} \qquad (4.6)$$

where R_{T0} is the thermistor resistance recorded at 273°K.

Equation 4.5 should now be recalculated using the updated values of V_S. The calculations are repeat for the 10kΩ, 50kΩ and 100kΩ thermistors and optimum values of V_S and R_C obtained are shown in table 4.1.

Table 4.1 Comparison of the optimum ΔV_T for (10kΩ, 50kΩ ,100kΩ) thermistors

Thermistor's resistance	Rc(kΩ)	Vs(volt)	ΔV_T(volt)
10kΩ	18	13.4	2.36
50kΩ	70	14.5	1.55
100kΩ	200	13.57	2.4

The objective of the previous steps is to get a higher value of ΔV_T, and in view of the above table, 10kΩ-thermistor and 100kΩ-thermistor are preferable since their ΔV_T values of are greater.

4.2.3 Temperature Calculation

In this section we explain the steps followed for choosing among thermistor (10kΩ) and thermistor (100kΩ) for reasonably accuracy in calculation of temperature. In

order to determine which one is better, we compare the calculated temperatures with the measured ones.

To calculate the temperature in NTC thermistor, we should first determine value of by using equation (4.2) for every thermistor as follows:

- Calculate Beta(β) in the thermistor (10kΩ)

  Where

  RT = 13kΩ is recorded at T = 291°K

  RT0 = 26.5kΩ is recorded at T0= 273°K

  Then

  β is found to be = 3110.9

- Calculate Beta(β) in the thermistor (100kΩ)

  Where

  RT = 142 kΩ is recorded at T = 288.2 °K

  RT0 = 292 kΩ is recorded at T0= 273 °K

  Then

  β is found to be = 4009.1

After finding β for each thermistor and calculating V_S which maximizes ΔV, as shown in a table 4.1, we now are ready to calculate V_T for each by using equation (4.7), which re-written here for convenience.

$$T = \frac{1}{\frac{1}{T_0}+\frac{1}{\beta}ln\left(\frac{R_T}{R_{T0}}\right)} \qquad (4.7)$$

Where

R_{T0} is the resistance of thermistor at T_0 which equal 273 °K. T is calculated for several thermistor resistance values (R_T) within the range used in this study. Also, output voltage of thermistor (V_T) should be calculated in order to compare the value of the corresponding temperature with the temperature measurements performed with a multimeter with temperature measurement capability. These calculations were performed using MS excel for both thermistors and the results are shown in tables (4.2) and (4.3).

Table 4.2 shows the calculated values of RT, VT and T in thermistor (10kΩ)

RT(KΩ)	VT(V)	T(°K)	T(°C)
50	3.571053	309.8728	36.9
100	5.654167	294.1156	21.1
150	7.018966	285.6196	12.6
200	7.982353	279.8833	6.9
250	8.698718	275.5902	2.6
300	9.252273	272.1789	-0.8
350	9.692857	269.36	-3.6
400	10.05185	266.9649	-6.0
450	10.35	264.8874	-8.1
500	10.60156	263.0561	-9.9
550	10.81667	261.4213	-11.6
600	11.0027	259.9464	-13.1
650	11.16519	258.6043	-14.4
700	11.30833	257.374	-15.6
750	11.43539	256.239	-16.8
800	11.54894	255.1864	-17.8
850	11.65101	254.2055	-18.8
900	11.74327	253.2875	-19.7
950	11.82706	252.4252	-20.6
1000	11.90351	251.6126	-21.4
1050	11.97353	250.8445	-22.2
1100	12.0379	250.1165	-22.9
1150	12.09729	249.4248	-23.6
1200	12.15224	248.7661	-24.2
1250	12.20324	248.1376	-24.9
1300	12.25069	247.5367	-25.5
1350	12.29497	246.9612	-26.0
1400	12.33636	246.4092	-26.6
1450	12.37516	245.8789	-27.1
1500	12.41159	245.3687	-27.6
1550	12.44586	244.8773	-28.1
1600	12.47816	244.4033	-28.6
1650	12.50866	243.9457	-29.1
1700	12.5375	243.5034	-29.5
1750	12.56481	243.0754	-29.9
1800	12.59072	242.6609	-30.3

Table 4.3 shows the calculated values of RT, VT and T in thermistor (100kΩ)

RT(KΩ)	VT(V)	T(°K)	T(°C)
50	3.571053	309.8728	36.9
100	5.654167	294.1156	21.1
150	7.018966	285.6196	12.6
200	7.982353	279.8833	6.9
250	8.698718	275.5902	2.6
300	9.252273	272.1789	-0.8
350	9.692857	269.36	-3.6
400	10.05185	266.9649	-6.0
450	10.35	264.8874	-8.1
500	10.60156	263.0561	-9.9
550	10.81667	261.4213	-11.6
600	11.0027	259.9464	-13.1
650	11.16519	258.6043	-14.4
700	11.30833	257.374	-15.6
750	11.43539	256.239	-16.8
800	11.54894	255.1864	-17.8
850	11.65101	254.2055	-18.8
900	11.74327	253.2875	-19.7
950	11.82706	252.4252	-20.6
1000	11.90351	251.6126	-21.4
1050	11.97353	250.8445	-22.2
1100	12.0379	250.1165	-22.9
1150	12.09729	249.4248	-23.6
1200	12.15224	248.7661	-24.2
1250	12.20324	248.1376	-24.9
1300	12.25069	247.5367	-25.5

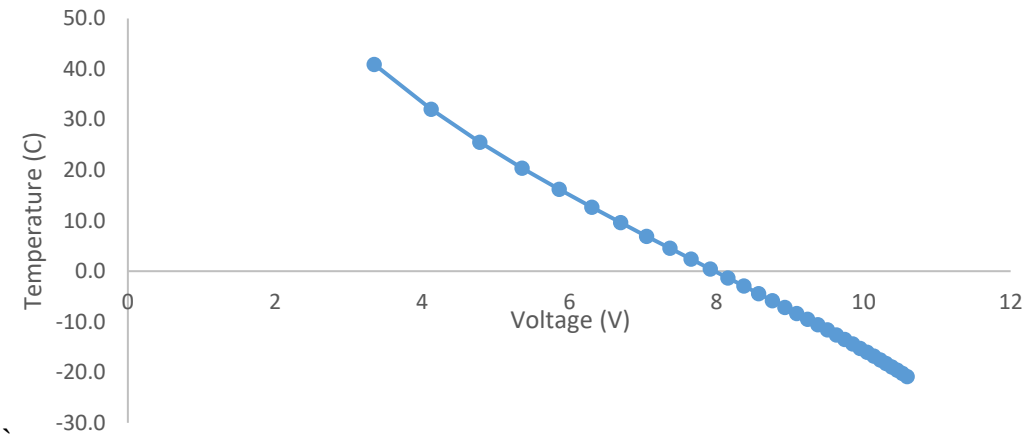

Figure 4.10 The relation between output voltage and temperature in thermistor (10kΩ)

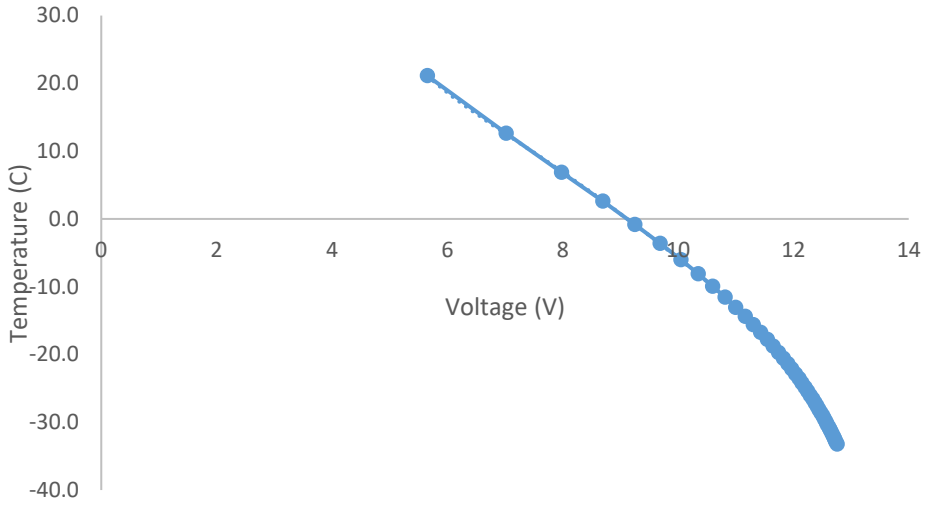

Figure 4.11 The relation between output voltage and temperature in thermistor (100kΩ)

At this point we decided to choose thermistor (10kΩ) as the most accurate and appropriate for this study. The decision was taken after making the necessary comparisons between the calculated and measured temperature values for the two thermistors.

4.3 **Problems during the study**

Within the PLC code that is automatically generated by PLC coder, a global variable named `ssMethodType` is present. The function of this variable is to indicate if the PLC is executing the code for the first time or not. In the first cycle of execution `ssMethodType` should have the value of 0 (zero) so that the generated codes perform the necessary initializations. The portion of the code that runs in the first cycle is called the creates base layer or definition layer code. If the Simulink model contains structures that have initial states, for instance unit delay or integrator, PLC coder creates base layer code. (Also called definition-layer or 0-layer code.) Base layer code loads the initial values into memory before running the system.

In the second and subsequent cycles `ssMethodType` should be set to 1 (one) to let the cyclic portion of the code run. This portion is also named 1-layer code.

If we look at figure 4.10, the code generated in this study has both the 0-layer and the 1-layer. What we want to do is to run the 0-layer in the first round and the 1-layer in the following rounds. Since the execution of 0-layer code is not critical for our application, we set `ssMethodType` to 1 permanently so that 0-layer code does not ever run, but 1-layer code runs in all the PLC cycles.

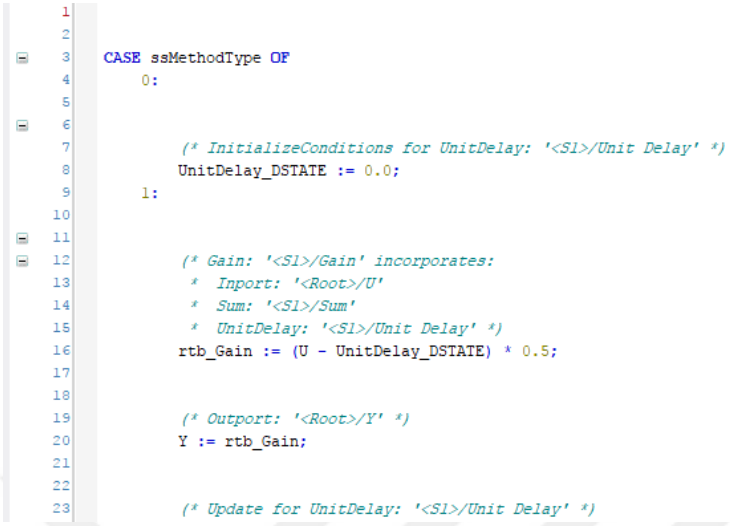

Figure 4.12 initial state ssMethodType, the PLC Coder creates a definition layer.

Practical part during this thesis has proved that importing code without correct definition of the date type causes some errors with the TwinCAT3 IDE. This should be considered in the stage of designing the Simulink model, since not defining a variable correctly leads to an incorrect variable definition in the code generation process. This is exactly what happened to us with some variables during code generation attempts. For instance, in this study the function of the switch is turning the system on and off and the associated variable should be of type Boolean. ın our first attempt of code generation, we did not explicitly define the type of this variable as Boolean in the Simulink model. In the generated code the type of the switch variable turned out to be LReal which is not appropriate for this application. We deduce that if not defined explicitly by the programmer, Simulink assigns types to variables by inheriting types from other signals in the model.

Another problem observed was related to conversion of the polynomial block in Simulink model in to PLC code. Polynomial block was intended to be used to obtain value of a temperature in the thermistor. The value of temperature is considered very critical in the process of transition from state to another in the controller system. To overcome this dilemma, instead of dealing with the temperature obtained from the

polynomial equation block, we used directly the voltage value corresponding to the desired temperature value in the program.

CHAPTER 5

Conclusion

The purpose of this thesis is to study the feasibility and possibility to automatically generate code starting from Simulink model, and then integrate this code in an Integrated Development Environment (IDE) which is generally used for developing industrial automation software. For this purpose, a sample industrial control system has been designed using MATLAB Simulink modelling tool. Then automatically code generation process was done with the aid of the Simulink PLC-Coder add-on, which showed high efficiency in code generation with fewer problems than expected. The generated code was imported into TwinCAT 3 IDE imported, tested by simulation and then implemented using the hardware setup available.

During this search, it became clear that there is possibility of generating code from a physical model which is compatible with the TwinCAT3 IDE, and the problems that were not planned in the beginning of the study could be solved. The model that was created illustrates the advantages of using Model-Based Design approach.

To further test the capabilities of this MBD approach it would be interesting to try a more complicated physical model containing various types of blocks as much as possible.

References

1. MathWorks, *Model-Based Design.* R2017b Retrieved from https://www.mathworks.com/help/simulink/gs/model-based-design.html?s_tid=srchtitle
2. Falkman, P., E. Helander, and A.G. MikaelAndersson, *A way of Ensuring PLC and HMI Standards, Aerospace Conference.* IEEE ETFA, 2011 Retrieved from
3. Flordal, H., et al., *Automatic model generation and PLC-code implementation for interlocking policies in industrial robot cells.* Control Engineering Practice, 2007. **15**(11): p. 1416-1426 Retrieved from
4. Lambersky, V. *Model based design and automated code generation from Simulink targeted for TMS570 MCU*. in *Education and Research Conference (EDERC), 2012 5th European DSP*. 2012. IEEE.
5. Bergström, D. and R. Göransson, *Model-and Hardware-in-the-Loop Testing in a Model-Based Design Workflow.* MSc Theses, 2016 Retrieved from
6. Haapala, O., *Application Software Development via Model Based Design.* 2014 Retrieved from
7. Kirstan, S. and J. Zimmermann. *Evaluating costs and benefits of model-based development of embedded software systems in the car industry-Results of a qualitative Case Study*. in *Proc. ECMFA 2010 workshop C2M: EEMDD-from Code Centric to Model Centric: Evaluating the Effectiveness of*. 2010.
8. Mozumdar, M.M.R., et al. *A framework for modeling, simulation and automatic code generation of sensor network application*. in *Sensor, Mesh and Ad Hoc Communications and Networks, 2008. SECON'08. 5th Annual IEEE Communications Society Conference on*. 2008. IEEE.
9. John, K.H. and M. Tiegelkamp, *IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids*. 2010: Springer Science & Business Media.
10. ABB, *Overview of the IEC 61131 Standard* 2007 Retrieved from https://library.e.abb.com/public/81478a314e1386d1c1257b1a005b0fc0/2101127.pdf
11. Baresi, L., et al., *PLC programming languages: A formal approach.* Proc. of Automation, 1998. **98** Retrieved from
12. Muˇsiˇc, G.s., D. Gradiˇsar, and D. Matko. *Iec 61131-3 compliant control code generation from discrete event models*. in *Intelligent Control, 2005. Proceedings of the 2005 IEEE International Symposium on, Mediterrean Conference on Control and Automation*. 2005. IEEE.
13. Huang, L., W. Liu, and Z. Liu. *Algorithm of transformation from PLC ladder diagram to structured text*. in *Electronic Measurement & Instruments, 2009. ICEMI'09. 9th International Conference on*. 2009. IEEE.
14. MathWorks, *MATLAB.* R2017b Retrieved from https://ch.mathworks.com/help/matlab/index.html?s_tid=srchtitle
15. MathWorks, *Simulink.* R2017b Retrieved from https://ch.mathworks.com/help/simulink/
16. MathWorks, *Simulink PLC Coder.* R2017b Retrieved from https://ch.mathworks.com/help/plccoder/index.html?s_tid=srchtitle

17. Beckhoff, *TwinCAT 3.* 2017 Retrieved from http://www.beckhoff.com/english.asp?twincat/twincat-3.htm
18. MathWorks, *Implementing the States to Represent Operating Modes.* R2017b Retrieved from https://www.mathworks.com/help/stateflow/gs/implementing-the-states-to-represent-operating-modes.html
19. Ahmadian, M., et al., *Model based design and SDR.* 2005 Retrieved from
20. resistorguide.com, *NTC thermistor.* 2017 Retrieved from http://www.resistorguide.com/ntc-thermistor