Rotary Clipper Diagnostics

Bachelor’s thesis
Degree Programme in Electrical and Automation Engineering
Valkeakoski, Spring 2017

Andrey Doremidov
The aim of this thesis was the renovation of the diagnostics program code of the rotary veneer clipper with Raute Oyj. Today's code was done fifteen years ago, so the reason is obvious. The main code, the safety mode, the call back functions, and visualization, which are already done, need the increase of the functionality. The thesis consists of issues of automation and electrical engineering. TwinCAT 3.1 and Microsoft Visual Studio are the most used programs in this project. The thesis includes theoretical and practical parts, software and hardware work, such as programming and emulating the possible equipment.

Besides the coding of the diagnostics for the clipper, I also need to learn other issues of the veneer preparation process so I could fully familiarize with the goals of the clipper and renovate it in the best way, combining the today's technologies and techniques.

Suggested ideas will increase the precision and reliability of the veneer clipper. Software changes mostly concern functionality and HMI details, increasing the efficiency and speed of reaction and maintenance actions. Hardware ideas are more about new interfered technologies and the way it can help the product lines.

The software programming samples are ready and presented in the thesis, hardware functions, although it is not possible to get and test practically, are explained and emulated.

**Keywords**  PLC, TwinCAT, programming, internship, automation

**Pages**  42 p. + appendices 28p.
## CONTENTS

1 INTRODUCTION ........................................................................................................... 1
   1.1 Thesis objectives ................................................................................................. 1
   1.2 List of abbreviations ............................................................................................ 1
   1.2 Company ............................................................................................................. 2
   1.3 Veneer consumption ........................................................................................... 3
   1.4 Veneer production .............................................................................................. 3

2 THE TOOLS ................................................................................................................... 5
   2.1 Software .............................................................................................................. 5
      2.1.1 TwinCAT 3.1 ............................................................................................. 5
      2.1.2 Standard programming languages .......................................................... 6
      2.1.3 Microsoft Visual Studio ........................................................................... 8
   2.2 Programming ..................................................................................................... 10
      2.2.1 Used software functions ........................................................................ 14
      2.2.2 Simatic Step 7 ........................................................................................ 15
   2.3 HARDWARE ....................................................................................................... 16
      2.3.1 CX2020 ................................................................................................... 16
      2.3.2 SIMATIC S7-1500 ................................................................................... 16

3 VENEER CLIPPING ...................................................................................................... 17
   3.1 Processing .......................................................................................................... 17
   3.2 Veneer clipper ................................................................................................... 19
   3.3 Activation inputs and inside processes. .............................................................. 21

4 PROJECT ..................................................................................................................... 22
   4.1 Principles of process .......................................................................................... 22
   4.2 Function ............................................................................................................. 22
      4.2.1 Mistake table ......................................................................................... 23
      4.2.2 Sensor’ answer check ........................................................................... 25
   4.3 Concepts ............................................................................................................ 27
      4.3.1 Knife twist check .................................................................................... 27
      4.3.2 Integrity of the blade ............................................................................. 33

5 CONCLUSION ............................................................................................................. 36

6 REFERENCES AND APPENDICES ................................................................................. 37
1 INTRODUCTION

1.1 Thesis objectives

The issues of the thesis are to renovate and increase the efficiency of the rotary veneer clipper for the maintenance and operating workers of the rotary veneer clipper in Raute Oyj. Functionality needs to be more practical and elucidating, so the conveyor can be more autonomous and substantive. The renovations consist of both, software coding and hardware suggestions and one of the main points is to create the sample, which can be appended into the already working machinery as far as into the future projects.

1.2 List of abbreviations

CNC: Computer Numerical Control
CAD: Computer Aided Design
AC: Alternating Current
DC: Direct Current
PC: Personal Computer
CPU: Central Processor Unit
ADS: Automation Device Specification
PLC: Programmable Logic Controller
HMI: Human Machine Interface
ISO: International Organization for Standardization
I/O: Input / Output
FBD: Function Block Diagram
POU: Program Organization Unit
LD: Ladder Logic Diagram
VAR: Variable
ST: Structured Text
OS: Operating System
IDE: Integrated Development Environment
VS: Visual Studio
IEC: International Electro Technical Commission
1.2 Company

The purpose of this thesis was to examine and develop a rotary veneer clipper which could satisfy industrial requirements. The thesis theme was proposed by my internship mentor at Raute Oyj.

Raute is a technology a company supplying the wood industry worldwide with the wood processing machinery. Plywood and LVL industries are their most important customers. (About Raute, 2017)

Raute's customers are companies managing with the wood product industry, manufacturing veneer, plywood and LVL, what stands for - Laminated Veneer Lumber. The increase of popularity of the revolving raw materials and products, for example in construction, means an expansion of the wood products industry. The value of the global timber production industry commerce is approximately EUR 150 billion. (Raute in value chain, 2017)

The summary investments of the manufacturing technology investments in the global veneer, plywood and LVL industry is approximately around EUR 600 million per year in a regular economic situation. (Raute in value chain, 2017)

Raute is the worldwide market leader in its significant client sector, the plywood industry, owning a 15–20 percent market share. Raute's positioning in the LVL industry is also strong, as on average more than a half of the manufactured LVL worldly is produced with equipment supplied by Raute. (Raute in value chain, 2017)

Raute's technology’s offerings cover machinery and equipment for the customer's whole production process. As a supplier of mill-scale projects, Raute is a worldwide market leader both in the plywood and LVL businesses. In addition, Raute's full-service project includes technology services ranging from spare parts deliveries to regular maintenance and equipment modernizations. Common feature of a project-focused organization are sensitivity to periodic economic variations and low timing predictability.

Raute’s customers are served by close to 650 Raute professionals from ten countries. Their production factories are situated in the Nastola area of Lahti and Kajaani in Finland, in the Vancouver area in Canada and in the Shanghai area in China. The company's sales network has a global coverage. Maintenance of the equipment and spare part supply services are increasingly offered locally. (About Raute, 2017)

The company's series shares have been announced on the Nasdaq Helsinki Ltd since 1994. (About Raute, 2017)
1.3 Veneer consumption

The timber production industry uses the enginey production of Raute company to create the veneer, plywood, and LVL, out of the wood, which are mostly in use in constructing and furniture fabrication as well as in the transport vehicle and packing industries. (Raute in value chain, 2017)

Plywood is produced by gluing veneers’ layers to each other with the fiber crisscross. Due to its strength characteristics and the appearance, provided by a continuous wood design, plywood can be used in a wide field of applications. Plywood has preserved its strong niche in applications, where structural reliability, moisture and impact resistance, and an appealing appearance are necessary. Significant variety of coatings adds to the flexibility of the applications. (Plywood consumption and production, 2017)

A veneer production, where the veneers’ layers are glued together with the grains in one direction, is called LVL, the abbreviation of Laminated Veneer Lumber. LVL's main usage is in construction, - there it is an environmentally conducive alternative to such materials as steel and concrete. Load-bearing LVL structures can sustain the natural disasters, like earthquakes, provide with a special range of application capabilities. LVL is used to produce reliable crossbars, applied in load-bearing structures and large wood-based sheets as panels. Various species of wood are used as raw material. (Plywood consumption and production, 2017)

1.4 Veneer production

As the focus of global economic performance is switching, the construction of new capacity for the plywood producing industry is also widening on the new emerging markets with new goals.

Europe is an important market area niche for Raute, although the European sector of manufacturers operating in the field creates a inconstant client base. The European plywood manufacturers are concentrated on memory usage creating the highly-qualified end production. A considerable amount of smaller factories operate, particularly, in southern Europe. (Plywood consumption and production, 2017)

The Finnish plywood industry is continuously stable, which has helped Raute in founding and developing its current position. For the past time, the investment focus has changed from Finland to the Baltic countries and Russia. The plywood sector continues to develop also in many niches in other countries, for example, France and Poland. (Plywood consumption and production, 2017)
Africa is a primary supplier of wooden logs. The restrictions on wood production exportation in some main producing countries have increased the necessity for local further processing of wood. The African commerce has traditionally been repressed by the European machinery suppliers. In the nearest future, the customers and investors will develop more operative process technologies and techniques – an idea which will create some new possibilities for Raute. (Plywood consumption and production, 2017)

The major forest area, almost the quarter of Earth’s wood is located in Russia. Other good issues of the plywood industry are the availability of experienced working personal and low labor costs. Plywood is mostly an export production in Russia, however as the country's economy has increased itself marketing characteristics, the domestic market has also gained reliability. Birch is the main raw production, also spruce, pine, and larch in demand. The plywood production situation is similar to other ex-Soviet countries. (Plywood consumption and production, 2017)

Most of the plywood material produced in the Asia-Pacific area is in use by the closer local area. The plywood of Asian production lines is based on the use of tropic wood species, for example, meranti, as raw materials. The usage of smaller, comparing to mainly used, diameter plantation wood, such as acacia and albizzia, is increasing during nowadays according to demands. Various types of pines’ and eucalyptuses’ species are used for plywood producing in the area of Pacific. There is a large number of planting forests in Australia and New Zealand. Plantation forests perform a sufficient role also in China and South-East Asia. (Plywood consumption and production, 2017)

Northern America is being the major area in the world' producing softwood plywood and LVL. Moreover, the area also has a significant veneer and plywood industry of utilization of hardwood. The serviced area is quite wide and the production is mostly concentrated on major forestry industries. Requirement for the plywood and LVL production is primarily encouraged by the US economy, specifically housing construction. (Veneer consumption and production, 2017)
2 THE TOOLS

2.1 Software

Nowadays there is a significant list of possible simulating and coding programs, which offer dozens of ways to automate processes. For this project, I needed to use software for PLC programming.

Beckhoff Automation GmbH is a German manufacturer of automation. The company focuses on PC-compatible control equipment, industrial PCs, embedded PCs, Bunterminals, I/O modules, Drive technology and automation software. The headquarters of Beckhoff are located in the east of North Rhine-Westphalia and consist of the central parts of development, production, management, sales, marketing, support, and service. (Beckhoff, 2017)

2.1.1 TwinCAT 3.1

TwinCAT stands for Total Windows Control and Automation Technology, which is a technology of automatization and control for the Microsoft Windows. TwinCAT is a complex of hardware and software systems, operating equipment to complete the programmed one or more issues. (TwinCAT System Overview)

TwinCAT works only with Microsoft Windows OS’s XP, 7, 8, 8.1, and 10. The launch of the program under Linux, Unix or MacOS is not possible, although virtual machines might be used. Still, the program is complex of wares – there are certain system requirements. (Beckhoff Automation GmbH, 2017)

Aside from the hardware segment, TwinCAT is a complex of software, creating a real-time operating system out of Windows OS, so it is suitable for the industrial usage. At the same moment the PC, as the host of the OS, turns into a full-fledged industrial controller. (Beckhoff, 2017)

Thus, both, the developer and the user, are acquiring two devices in one: one is a default PC, which can work, as usual, running basic programs, such as Word, Excel, Photoshop, and another one, running the programs, controlling the manufacturing processes.

In order to become the complete industrial controller, the PC needs the “Runtime” – one of the main parts of TwinCAT - real-time environment for creating, developing and controlling the industrial process. (Banks, 1998)
2.1.2 Standard programming languages

IEC 61131 is an IEC standard for programmable controllers. There are several parts of this standard, - I am going to concentrate on third publication. (PLCopen)

IEC 61131-3 deals with basic software architecture and the programming languages of the control program within the PLC. They are done so that the program can read and transform data from one language to another. (PLCopen)

List of programming languages of IEC 61131-3:

- ST - Structured Text, textual
- LD - Ladder diagram, graphical
- FBD - Function Block Diagram, graphical
- IL - Instruction List, textual
- SFC - Sequential function chart, sequential and parallel control processing

Structured Text (ST) – is a standardized programming language that is used in TwinCAT. It is designed to configure PLCs. It is a high-level language which is block structured and recalls Pascal and Assembler, on which it is based. Widely used in SCADA/HMI/SoftLogic toolkits. (PLCopen)

Benefits:

- Users, who have experience of Pascal and Basic can accustom easily with ST.
- Easy to read due to symbolic representation.
- The program, based on ST, can be written in any text redactor. (Structure Text; Национальная библиотека им. Н.Э.Баумана; Bauman National Library, 2016)

Commands Structured Text consists of several operators:

- Cyclic operator WHILE.
- Cyclic operator FOR.
- An operator of cyclic transactions IF-THEN-ELSE.
- An operator of cyclic transactions CASE. (Operators - Structured Text, 2016)

Ladder Diagram – a language relay, ladder logic, is created for the programming industrial controllers - PLCs. Originally this was a written method to document the design and construction of relay racks as used in manufacturing and process control. Every device in the relay rack is represented with a symbol on the ladder diagram with the connection between those devices displayed. Moreover, devices, which are external to the relay rack also are illustrated in the diagram. (Ladder Diagram (LD), 2017)

Ladder logic has developed into a programming language that explains a program or code by a graphical chart based on the circuit diagrams of relay logic hardware. (PLCopen)

Function Block Diagram (FBD) – a graphical language for PLC design that can describe the ongoing processes between inputs’ and outputs’ variables. A function is described as a cluster of elementary blocks. Input and output variables are connected to blocks by connection lines. (Function Block Diagram (FBD) Beckhoff Information System, 2017)

Inputs and outputs of the blocks are set together using wired connections, such as lines or links. Also, on the diagram, two logical points can be connected by the single line. (IEC, 2007-2008)

Instruction List (IL) – list of instructions, is a low-level language, resembles assembler. Program control, also called as control flow, is achieved by jump instructions and function calls, subroutines with optional parameters. IL is a low-level programming language, and it is not tied to a specific architecture.

IL allows explaining and visualizing functions, function blocks, and programs, as well as steps and transitions in the SFC language. One of the key advantages of the Instruction List is a simplicity and the ability to achieve highly optimized code for the implementation of critical software sectors. However, IL is inconvenient to describe complex algorithms with a lot of branching.

The basis of the Instruction List programming language, as in the case of Assembler, is the label transitions and the battery. The value of the variable is loaded into the accumulator, and the further realization of the algorithm is the extraction of the values from the battery and committing operators over it. (IEC, 2007-2008)
Sequential Function Chart (SFC) – is a graphical language. The SFC standard is defined as, - Preparation of function charts for control systems. Its’ purpose is to program processes that can be split into steps.

Main components are:

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

Steps in the SFC diagram can be active or inactive. Actions are only executed for active steps. A step can be active for one or two motives:

- It is an initial step as specified by the programmer;
- It was activated during a scan cycle and not deactivated since. (IEC, 2007-2008)

2.1.3 Microsoft Visual Studio

Microsoft Visual Studio in an integrated development environment (IDE) from Microsoft. It is used to develop computer programs for Microsoft Windows, as well as websites, web applications, and web services. Visual Studio uses Microsoft software development platforms as Windows API, Microsoft Silverlight, etc. It can produce both - native and managed code.

Microsoft Visual Studio is a set of tools to create software: designing user interface, coding, testing, debugging, analyzing the performance and quality, deploying in the customer environment and collecting telemetered data about usage. These tools are designed to maximize teamwork effectiveness, all of them are available in integrated development environment – IDE, which is Visual Studio.

Specifically, for TwinCAT there is engineering version - XAE. Programs will synchronize and create the environment for programming PLC.

Inside the VS there are different types of data possible to use and code:

- BOOL: the signal is TRUE or FALSE, ON and OFF state. Uses 8 Bits of memory, originally done to manage variables with two states: TRUE or FALSE, 1 or 0. (TwinCAT 3 Tutorial: Structured Text, 2017)

- BYTE: integer which uses values from 0 up to 255. Uses 8 Bits of memory, contains binary data and perform the values between 0 and 255. (TwinCAT 3 Tutorial: Structured Text, 2017)
- **WORD**: integer which uses values from 0 up to 65535. Uses 16 Bits of memory. Manage text variables, mostly used for HMI representation. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **DWORD**: integer which uses values from 0 up to 4294967295. Uses 32 Bits of memory. Specific representation of the unsigned integer, used to store long numbers and acts with UINT without the considerable loss of data. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **SINT**: short signed integer which uses the number from -128 up to 127. Uses 8 Bits of memory. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **USINT**: unsigned short integer data type which uses values from 0 up to 255. Uses 8 Bits of memory. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **INT**: signed integer which uses values from -32768 up to 32767. Uses 16 Bits memory. Mostly used to store numeric data and managing calculations. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **UINT**: unsigned integer which uses values from 0 up to 65535. Uses 16 Bits of memory. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **DINT**: signed integer which uses values from -2147483648 up to 2147483648. Uses 32 Bits of memory. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **UDINT**: unsigned integer which uses values from 0 up to 4294967295. Uses 32 Bits of memory. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **REAL**: floating point which uses numbers from $\sim -3.402823 \times 10^{38}$ up to $\sim 3.402823 \times 10^{38}$. Uses 32 Bits of memory. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **STRING**: the variable that consists of any string data type. The usage of the controller's memory is defined according to the amount of the characters listed there. Storing text variables. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **TIME**: definition of time from T#0ms up to T#71582m47s295ms. Uses 32 Bits of memory. (TwinCAT 3 Tutorial: Structured Text, 2017)
- **TIME_OF_DAY**: time of day from TOD#00:00 up to TOD#1193:02:47.295. Uses 32 Bits of memory. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **DATE**: date, month, and year. The time starts from D#1970-01-01 to D#2106-02-06. Memory use 32 Bit. (TwinCAT 3 Tutorial: Structured Text, 2017)

- **DATE_AND_TIME**: daytime, date, month, and year from DT#1970-01-01-00:00 to DT#2106-02-06-06:28:15. Uses 32 Bits of memory. (TwinCAT 3 Tutorial: Structured Text, 2017)

Some of the veneer clipper's visualizations are programmed in SQL and closely connected to TwinCAT software. It is not used to program the listed concepts, but must be introduced since it was necessary to go through it to understand to full command line.

SQL - Structured Query Language (wikipedia.org, 2017)- is a domain-particular language used in programming and designed to manage databases held in a relational data bank management system or for stream processes in a relational data stream management system.

Originally founded on relational algebra and some tuple relational calculations, SQL consists of data definitions for the programming language, data controlling and managing language, and data controlling language. The scope of SQL includes data insertion, querying, updating and deleting, schematic creation and modification, also databases access controlling. Although SQL is mostly described as a declarative language, it also consists of the procedural elements. (wikipedia.org, 2017)

SQL is one of the first merchant programming languages for Edgar F. Codd's relational example, as described in his research, "A Relational Model of Data for Large Shared Data Banks". Although of not entirely following the relational example as described by Codd, it appeared to be the most used language of databases. (wikipedia.org, 2017)

The SQL software appeared in a list of standards of the American National Standards Institute, ANSI, in 1986, and of the International Organization for Standardization, ISO, in 1987. (IEC, 2007-2008) Since then, the standard has been renovated, so at this moment it also considers a larger range of functions. Despite the existence of such standards, most SQL code is not completely portable among different database systems without adjustments. (wikipedia.org, 2017)
2.2 Programming

Most of the work was done using TwinCAT 3 developing environment. The environment is provided via Microsoft Visual Studio. In this case Microsoft Visual Studio (VS 2013).

TwinCAT software is the master application for EtherCAT. It consists of three different programs: System Manager – central tool for the configuration of the TwinCAT System Manager; PLC controller – complete development environment for the PLC; and Scope View – analysis tool providing the graphical display of the variables related to various PLC- and NC tasks. (Kiela, 2017)

TwinCAT substitute the regular PLC and NC/CNC controllers as well as operating machinery with:

- Open, consonant personal computer hardware.
- Embedded IEC 61131-3 software PLC, software NC and software CNC in the most of Windows versions, NT/XP Embedded, CE.
- Programming and run-time systems optionally operating together on one PC or discrete.
- Combination with all regular field buses.
- PC HMI functions are supported.
- Data communication with the HMIs and other programs by aids of open Microsoft standards: OPC, OCX, DLL etc.

TwinCAT is the fully-featured and de-facto reference EtherCAT master implementation. (EtherCAT Technology Group, 2017)
Next pictures represent the user interface of Visual Studio with Integrated TwinCAT.

![Figure 2. Initial Visual Studio Page](image)

Depending on your purpose you can start rather by scanning the connected devices by right clicking on “I/O” and then “devices” then “scan”. Scanning can be done only in “configure mode”. “Scan” will show you the connected devices and possible inputs and outputs for further configuring.

However, TwinCAT gives the possibility to create programs even without the necessary hardware. In order to make it possible we need to separate and emulate CPUs of the computer so the program process will not affect the computer functions. In solution explorer click on “System” and then “Real-time”.

![Figure 3. CPU setting’ page](image)

We need to click on “Read from target” and change the marks from Windows CPU to two isolated CPUs. CPU limit will change to 100% and will not affect the performance of PC. Then click “set on target”. Program and computer will be rebooted, so after these actions, your computer will become PC and the developing environment in one. Mind that for the
right processing Visual Studio needs active Intel VT-x – check your BIOS options for this function.

![Figure 4. Splitting the CPUs for proper processing](image)

After the exampled actions, we can start programming. Right clicking on “PLC” in Solution Explorer, “Add new item”. The program will create a raw program for you. It consists of:

- **DUT - Data Unit Type** - User is able to specify own datatypes: structures, enumeration lists, and references can be designed as Data Type Units in a DUT editor (Data Type Unit (DUT) Beckhoff Information System, 2017).

![Figure 5. Standard project’s content](image)
• GVL - Global Variable List - GVLs are used to claim the global variables for the program. If GVL exists, the variables will affect the whole project and not just a certain part (Beckhoff 2016).

• POU - Program Organization Units - functions, function blocks and programs are POU, which can be supplemented by actions. (Beckhoff 2016).

• VISU - Visualization - For making the project more visualized like adding virtual buttons and gauges for showing information (Creating a Target Vusualization Beckhoff Information System, 2017).

• References – function and processes referenced to internal libraries and integrated into TwinCAT, which precede inner system actions and self-actions such as the timer or getting time.

In POU you will find a program called “Main”. This program is one of the key elements, it is calling - switching on - other processes. (TwinCAT PLC Control: Data Types, 2017)

2.2.1 Used software functions

Database of TwinCAT already has the wide range of functions which are written inside the specified library repositories. Some of them are added by the program in the very beginning while building up the project template, for example, FBD functions. Some of the function libraries are less used and are not added unless programmer needs them, so they supposed to be added manually, for example – getting the time of the computer into the project. It can be found in the solution explorer, it is presented on the Figure 6.

Figure 6. Libraries in Visual Studio
Beside the libraries there are some extra programming environments, more suitable in the different situation and goals, such as connectivity or motion environments. They are specified according to the performance need to be produced in the program. Rather the online control or simulation can be imagined inside the TwinCAT. (Project Components Beckhoff Information System, 2017)

2.2.2 Simatic Step 7

Simatic Step 7 is also one of the programs used in parallel with TwinCAT. It is considered in some future projects. Cooperation of these two software units is necessary for the proper processing between higher and lower level of automatization.

Simatic Step 7 – software by Siemens, developed for the automation engineering, based on the principles of PLC. Using this program, the set of creations, improving and maintaining automation based on Simatic S7-300 and Simatic S7-400 systems of Siemens Company. Processes of the project are procured with main utility Step 7 – Simatic Manager. Step S7 provides the possibility to configure PLCs and networks with such utilities as HWConfig and NetPro. During the process of configuring composition of the equipment, modules, connections networks and setting for the modules are determined.

The system checks the correct use and connection of individual parts. The configuration is completed by loading the selected settings into the equipment. The configuration utilities allow you to diagnose the hardware, detect improperly installed hardware or hardware errors.

Programming of controllers is made by the editor of programs, providing writing programs in three languages:

- LAD - the language of ladder-contact logic;
- FBD - the language of functional block diagrams;
- STL - the language of the instruction list.

In addition to the three main languages, four extra versions can be added:

- SCL – structured management language, syntax similar to Pascal;
- GRAPH7 – the language of sequential technological processes management;
- HiGraph7 – control language based on the state graph of the system;

- SFC – the language of the state diagrams.

The ability to monitor the current state of the program, available with any programming language, provides not only the debugging of the software, but also diagnosing the mistakes of connected equipment, even if it does not have own self-diagnostic.

In Siemens production, Step 7 performs the integration functions. The integration of projects for HMI in the Step 7 project facilitates the automatic linking of projects for the controller and the operator interface, speeds up the design and avoids the errors, associated with the separate use of programs.

Step 7 also allows you to design network settings. Connections and data transfer between automation devices, for example, the Master-Slave system for data exchange via the Profibus bus using the DP protocol.

2.3 HARDWARE

2.3.1 CX2020

CX2020 is a present module, used in machinery of Raute Oyj.

CX2020 is a basic CPU module that carries out the instructions of a programmer, performing the basic arithmetical, logical, control and I/O operations.

The CX2020 has a 1.4 GHz Intel® Celeron® CPU, it is fanless and has no rotating parts inside. The CX2020 also contains the main memory, as the addition to the CPU and chipset, with 2 GB RAM as defined standard. 4 GB option is possible to be included into the system. The controller switches on by the CFast flash memory card. (Beckhoff, 2017)

The basic setup of the CX2020 consists of a CFast memory card, two autonomous Gbit Ethernet interfaces, four USB 2.0 interfaces and a DVI-I interface. (Beckhoff, 2017)

The CPU has a 128 kB NOVRAM persistent data memory for situations where no UPS is used.
2.3.2 SIMATIC S7-1500

SIMATICS7-1500 is a planned CPU for the renewing of the machinery. All actions are done in the way so the inconsistencies could be minimized.

This controller is planned to be used in the closer times, supposed to change the previous Siemens' production in the conveyor. The Advanced Controller SIMATIC S7-1500 can be used in the entire automation industry in applications that are medium-sized to high-end machines with high demands on performance, communication, flexibility and technological features.

Within the SIMATIC S7-1500 Portfolio, there are various CPU types for multiple performance classes available. Besides the standard CPUs and the compact CPUs, the Portfolio will be enlarged by the S7-1500 T-CPU and CPU 1518 ODK. The technology-CPUs are offering a wide range of motion control function such as cam disks and constant velocity as well as the possibility to execute C/C++ code on the CPU 1518 ODK.

This model is planned to be used in Raute’s technology lines.

3 VENEER CLIPPING

Veneer is a thin layer of wood, produced according to defined thickness by peeling, slicing, or sawing the incoming log. The easiest way to imagine this process is to sharpen the pencil – a thin platen of wood is separated by slicing it with the sharpener, and after that there is a blade, cutting the layer into pieces and excising the bad parts. Traditionally the veneer was sawn, although nowadays, veneer is obtained either by "peeling" the wooden log or by slicing large rectangular blocks of wood called flitches.

The appearance of the grain and figure in wood comes from the method of slicing through the growth rings of a tree and also depends upon the angle at which the wood is sliced.

The very first idea needed to be considered in the veneer clipper is to make the code easier for understanding since the processes done by the veneer clipper are controlled not by the programmers, but by the people, who do not know the PLC coding. So, the more user-friendly the program is, - the better it will work.
3.1 Processing

First of all, I would like to explain the whole manufacturing process from the very beginning so it will be easier to understand. There are different ways to produce veneer and LVL, - in Raute Oyj the peeling method is used and the following example considers some specific actions, - Figure 7.

Figure 7. Schematic processing

Wood is bought from the cutting companies. Numbered and stocked in the queue and according to needs is taken from storage. When the log is...
coming to the line it is being cleaned from bark and dirt, then being dried until the required humidity level. After preparing procedures log enters saw where being sawn as needed. Then the logs come to the slicing machine, which creates a thick cloth of log, called veneer cloth.

The line of veneer comes to the clipper, which divides the cloth with a knife into the required size and excises defective elements out of the veneer. If there is defective element veneer’s knife is chopping it into smaller fragments for further use.

After the clipper, good veneer fragments are stocked into pallets. Then, the batch of veneer slices come to the conveyor, where they are glued, pressed, dried and sawn. The product after these actions is called - plywood. It is packed and stocked until the order.

3.2 Veneer clipper

Veneer clipper – machine of production lines, clipping the upcoming veneer into pieces and excising the defective elements. In this paragraph, I would like to explain its working processes deeper.

As far as the log is sliced, the veneer is continuing its’ way on the production line so it is being scanned by cameras which diagnose the quality of upcoming cloth and giving commands to the clipper CPU about clipping timing. The quality of the wood is one of the most key factors of profitable production. If there are no defects, the clipper processes follow the given criteria about the length of the clipped piece. In case there is flaw the clipper cuts it off and continues processing as supposed to.

Here is the order of actions of the regular processing:

1. The log is sliced.
2. Veneer comes to the line.
3. Clipper motors start working.
4. Cameras check the veneer’s quality and control the fire signals of clipper.
5. Clipper’s knife takes the initial position.
6. Veneer comes to the clipping place.
7. Clipper’s knife turns around its’ axis chopping the veneer cloth.
8. Clipper’s knife moves to the next given position.

9. Veneer is continuing its way on the line.

Figure 8. Representation of action order inside the clipper system.

All these processes are done at the high speed and within the millisecond timing, which requires precise calculations and high equipment quality. Due to the high-speed processes and a large number of repetitions, the whole system must be on time checked and serviced in order to continue correct work. Small inaccuracy can increase the wear of equipment and
lead to inflated consumption of veneer and financial expenses. All the stages are shown in the figures A to H below.

3.3 Activation inputs and inside processes.

Cutting mode and processing are controlled by the Beckhoff controller and programmed by TwinCAT.

At first, there is a camera, which is detecting the presence and quality of incoming veneer. The presence of veneer activates the knife so it adjusts itself into the initial position. Depending on quality, the camera sends the signal rather that veneer is good enough or not. In the first case, the knife
is clipping the cloth using the defined values of length. If there is a substandard element of veneer, - camera defines the length and sends the necessary data about the situation to knife controller. This data is about the length of defective veneer, so the knife can speed up its rotational speed so the element is going to be cut into strips and sorted into another storage for different purposes, example: particle, - chip, board).

Apart from external devices, internal processes are mostly indicated by the program code. Due to the continual acting with high speed, the clipper is being self-diagnosed by the program each period of time, from $1 \mu s$ up to $100 \mu s$.

Since the processing speed is extremely high the values are always checked and compared to the assigned values to detect any inconsistencies and warn the service staff in advance.

4 PROJECT

The idea of the project is to update the veneer clipper diagnostics and increase the reliability. The program and technique are already in use and experienced by the developers, so the local modifications of the system are preferable than a complete change. In this paragraph

4.1 Principles of process

I will explain the programming of a separate block since I needed to insert an extra code element to add functions and new paths. All described features are focused on the increasing the precision of diagnostics. Software changes can be done directly and take less setup time, while hardware ideas are only concepts since everything must be precisely adapted and this takes more time, so it cannot be done in short terms.

4.2 Function

In this paragraph, the possible functional renovations are presented. There are software changes, made of programming in TwinCAT using mostly Structured Text programming language. First idea, mistake table, is required by the personal of Raute Oyj, to ease the cooperation of the PLC and maintenance group. The second function, sensor’s answer check, is a quite basic function, however, was not used in the nowadays clipper’s code, also is helpful, increase the speed and effectiveness of the maintenance of the system.
4.2.1 Mistake table

This program is basic, consisting only of Twincat commands. It is a table of 4 columns writing down the events and its’ time for future possible use. Then, in the case of demand worker can open this table and check all the changes. The main reason is to make the service and maintenance more comfortable for use.

This kind of table uses basic functions such as:

- Get current time from the computer
- Timer
- Time structuring
- Writing and storing the data

Everything starts with the updating of the conditions and mistakes of the clipper. In this case, it is done as easy as possible to be visualized. There is the timer, called fbTimeTimer starting every 100 milliseconds, unless its’ own output will not be activated. Obviously, the output, called fbTimeTimer.Q, is not going to be activated, since we need the constant respond from the program. Down here the variables of the program are presented.

Example code in TwinCAT:

```plaintext
fbTimeTimer(IN:= NOT fbTimeTimer.Q, PT := T#100ms);
```

Figure 11. Structure of the variables for the program

Figure 9. Possible example of the event table
The variable called “stMistake” is the string unit, keeping the text data; DT variables write down the Day Time, taken from the PC by the standard TwinCAT function called “fbNT_GetTime”. Variable “tMistake” compares the time between to upper variables and get the difference. All these data units form the cluster of one condition. Then, an array of the necessary amount of strings create the massive out of this units, and the function “FW_MemMove” keeps the data stored in the array until the next change of condition. When the conditions are changing, the function rewrites the previous data to the following string and puts the current information into the first line. In case if all lines occupied with data the function will erase the oldest records. The program element which rewrites the memory of the variables in the array is shown below.

```plaintext
FUNCTION F_FIFO_WRITE : BOOL
VAR_INPUT
    cbData : UDINT;//Stuff coming in
    pData : POINTER TO BYTE;
    cbBuffer : UDINT;//Where all the stuff lives
    pBuffer : POINTER TO BYTE;
END_VAR
VAR
END_VAR

FW_MemMove(pDest := pBuffer + cbData, pSrc := pBuffer, chLen := cbBuffer - cbData);
FW_MemMove(pDest := pBuffer, pSrc := pData, chLen := cbData);
```

Figure 12. Rewriting function; standard TwinCAT option

After programming this part, the controller gets the universal sample, where the only one thing needs to be defined, - which data to store. It could be mistakes, times of servicing, changes of the setup of the machine, and etc. The idea is to add the variable to be stored into this function, and then the controller will record it by itself.

4.2.2 Sensor’ answer check

As the previous example, basic programming commands can provide the machine with a useful function, which will check the condition of sensors with the defined table of conditions. After this, the defined structure of data supposed to be included into the program.
The idea consists of the possibility to create the list of all variables, to number them and control with a couple of clicks. Down here you can see the element of the code to perform the function. There is a variable declaration of the array’s data, because we need to number the conditions; a timer, to switch off the commands for the next usage; a string variable to inform the operator; two boolean variables to start the actions; the operating function FOR...DO. On the figure 14 below, there are two declared arrays, one multiplying another.

```fortran
VAR_GLOBAL PERSISTENT
(* Mistakes list*)
arr_Mistakes : ARRAY[1.. &] OF BOOL;
END_VAR

VAR_GLOBAL CONSTANT
(* Info *)
arr1_Mistakes : ARRAY[1.. & ] OF Mist_Def :=[
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 )
];
END_VAR
```

Figure 13. Declared arrays in global variable list

While the operator activates the Boolean variable the program starts numbering and comparing the variable value with the defined one and if it is different – equalize the data. For performing these actions we need two arrays, one of them must be of a structure, here “Mist_Def”.

```fortran
TYPE Mist_Def:
  STRUCT
    def: BOOL;
    rw: BOOL;
  END_STRUCT
END_TYPE
```

Figure 14. Structure of the array
Next figure consist the variable list of the answer check program. That element consists of the quite small amount of variables. That is due to the simplicity of the controlling function. Despite the simplicity it can control the whole system.

```
PROGRAM DTInitDefaults
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
iIndex : UINT;
giLastBoo : UINT;
B1 : bool;
B2 : bool;
ton : TON;
END_VAR
```

Figure 15. List of local variables for the program

This function has two Boolean variables defining the state of array's variables according to the commands. There are three possibilities:

1. Variables' values are defined without considering the array.

2. Variables are defined according to "def" values.

3. Variables are defined according to "rw" values.

So, by using one of the possible cluster of data we can operate the device. In this project, I decided to use this function as a call operator so that the proper condition of sensor could be checked. By using one the variable, here "B1" "B2", the program equalizes the variable for two seconds so we can check the reaction. By this we can test the new equipment and features, also can be used as a diagnostics' "call-answer" test when the operator needs. Text data is also done for more comfortable usage. Figure 16 below shows the program code, which controls the state of the variables. Due the fact that those variables are Boolean, there are two states, rather 0 or 1, FALSE or TRUE. On the same figure 16, on the top of it, there is an element, numbering the variables of the array of Figure 15, so they are synchronized.
4.3 Concepts

The most problematic element of the veneer clipper is the main element of the machine - the knife, cutting the veneer. The idea is to improve the failure preventive mode. Due to the high speed and multiple repetitions, the veneer knife is susceptible to breakage. Beside blunting, the knife can over twist or can be nicked.

Although the veneer clipper is being under control by the operator, the condition of the clipper cannot be precisely checked all the time.

4.3.1 Knife twist check

Now, the angle is checked by comparing the values of two encoders which are controlling the motors, moving and positioning the blade. Although this method is working, deviations or hysteresis can appear and there are a couple of values which cannot be checked by the encoders.
As a concept, I would like to introduce the way which can be appended into the project using infrared laser module. The principle of processes is abridged to the very basics of automation.

There are two ways to manage with sensors, both are quite easy and programmed with basic manner.

Laser ray is being sent to the receiver, which is giving the variable to the controller. As long as the knife is not crossing the ray there is a continuous signal if knife crosses the ray, signal disappears, meaning rather that positioning is bad or the knife has twisted. Since the blade is rolling according to necessities the lasers will be active only during the breaks, when the knife is coming back to horizontal position.
This system is not that functional as following, though easy to build up and adjust. Since a lot of the conveyors are already in use this device can be installed on any of the production versions. This system is not the that flexible although the efficiency is high due to the simplicity.

Figure 19. Horizontal lasers’ positioning (2)

Beside from the level checking, the height can be measured and provide reliable measuring without the global changes. Declared number of rangefinders will be adjusted, and provide simple data to the controller, where the values will be compared and the angle of the knife can be defined. This method is the fastest in adaptation to the ready machines.

Figure 20. Vertical positioning of the lasers

The second option is to use vertical sensors with similar principals. There will be from 6 and more rangefinders, the accuracy increases with the amount, periodically measuring the distance between the blade, comparing the data and reacting as decided. In that case, if one of the
rays is shorter it will be detected and decided the grade of the twist. On the figure 21 below, there are variables, controlling the data of the rangefinders and comparing the values to each other and to the standard value.

```
PROGRAM Rangefinder
VAR
    Laser     : ARRAY [0.. 5] OF UINT;
    StandardLenght : UINT;
    NewValueForLasers : UINT;
    PermissionToChange : BOOL;
    DeviationAlarm : BOOL;
    strConditionS   : STRING;
    strConditionA : STRING;
END_VAR

Figure 21.List of the variables for the rangefinders
```

Here we can see the element of the code, which I suppose to use with this method. First of all, we need to declare program variables for this function. The list of them will consist of an array, unsigned integer values and a couple of string and Boolean values. The names for the variables can be decided by the user, in this example, I did the most obvious names so it can be easy to understand for everyone. Normally, variables are called shorter, for example with abbreviations, but naming depends mostly on the programmer and his goals.

According to the program, there will be three level of warning about the twisting. Due to the geometrical characteristics of the blade, it is better to adjust the sensors so that they will be closer to the edge of the knife, since the twist there will be more detectable. This element of the code is presented in the figure 22.
In case if the distance between the blade and rangefinder will change, the value, given by the sensor will be changed and be different from the given standard value. Then, depending on which sensor detected the twisting the controller will inform the operator about the type of deviation.

The rays are numbered from left to right, six at all, although you check can that in the program I've used the array from 0 to 5. That is because Twincat also considers zero as a number. So, if rays number 0 and 5 will change their data, the controller will show that there is a "diminutive twist" of the blade. At this point, the blade can still continue its work unless the company needs it to be maximally precise. Rays number 1 and 4 detect the "small twist", Rays 2 and 3 -- "significant twist". The positioning of the rays can vary. Moreover, this program does not need any kind of mathematical or geometrical calculations by the reason that the ray will produce an insignificant area on the blade, so there is no reason and no possibility to evaluate the angle precisely. (e.K., 2016)
Also, different features, such as permissible variation, pre-signal counter, the automatic list of changes can be added to the program and will be explained further in the text.

![Figure 23. Rays’ function](image)

Figure 23. Rays’ function

![Figure 24. 3D model of twisted blade](image)

Figure 24. 3D model of twisted blade

This is the model of what is happening with the blade if the rotating motors have different rotational phases. Beside the decrease of quality of production, it decreases the guarantee time, can cause other machine breakage and harm the workers.

4.3.2 Integrity of the blade

Processing is insanely fast and due to that fact, the risk of breakage is quite high. Nicks, scratches, cracks are happening not that often as twisting or blunting, but all these situations can be detected by next
example. It is the composition of the camera, rangefinder, and protractor.

The idea is to adjust the scanners on both angels of the clipper so that the whole area of the blade will be under the camera’s vision. Blade’s surface will be interfaced as a pattern into scanner controller and remembered as the initial ideal condition for future compare. By this way, all the damage will be detected and updated on the operator’s computer. Two sensors are needed for an increased sensitivity, since the quality of the picture depends on the length of the ray.
The checked area of the blade could be slimmer and closer to the edge, although in the exampled method the precision will increase due to the increased comparative surface.

The idea of this project depends on the functional outputs from the scanner, but due to the functional range of Twincat 3.1, any extra function can be added by coding. The list of the functions of scanners varies respectively according to the price so it depends on the decision commission which type to use.
Despite the fact, that wood dust remains and clutter all equipment the blade is staying clean due to the high speed and vibration, so we the false signal is improbable. For the better guarantees, the counter could be programmed so that program will react after defined amount repetitions.

Next picture shows how the scanner discerns the damage of the blade such as twisting or being nicked.

![Scanner’s twisting and nick view](image)

According to the Figure 29 the scanner will produce the scanning surface and memorize the ideal state of the blade. Each time after the one round of cutting it will switch on and scan the blade again, checking for the changes. If there is any damage the program will get the signal from the scanner, warning the operator and the CPU about the corruption of the blade. Depending on the stage of damage the program will decide whether to stop the conveyor at all or give the operator the possibility to decide the following actions. Also, system will be tolerant to the minor changes, such as little twist less than 0.5 degrees, which always can happen and will not affect the blade.
5 CONCLUSION

The main objective of this thesis project was to design possible renovations to the software and hardware elements for better diagnostics of the rotary veneer clipper in Finnish wood and LVL equipment company called Raute Oyj. Both, the software and the hardware, ideas are given here in the way, that they can be included into the already completed machinery and future projects there.

In this thesis, two main issues are explained – software and hardware updates. Software renovations mostly consider the already done element of coding and, such functions as (BeckhoffTwinCAT 3 Basics, 2017) the table of mistakes and sensors' answer check, are based on the already processing and properly working program code, so they are done for the observation of operators and easier maintenance. Hardware concepts mainly present the possible improvement of precision of the machinery and respect the needs of the already processing machine as far as the oncoming machines.

At this moment, the software programming elements, such as answer check function, resetting the default values, and mistake table function, memorizing the states of the system, are done and can be interacted into the nowadays and future processing machinery. The universalism of those functions is provided by the capacity of the programming languages and the world standards, giving the possibility to cooperate between several PLCs.

Hardware issues need to be examined and tested so the most proper functional way and the manufacturer will be identified. There are lots of fabrics producing the possible hardware with the wide range of functions. Using any production means ordering a significant amount of elements which cost money and such actions must be considered and organized particularly carefully by the knowledgeable people.

In the nearest future there are planned changes to the machinery, mostly considering the controlling units, so TwinCAT is planned to cooperate with the new extra PLCs and the possible next project will be to connect Beckhoff controllers with the Siemens production, so the cooperation will be improved, for example, by controlling one CPU by another.
REFERENCES AND APPENDICES


APPENDICES

Down here the full elements of code are presented, and also the elements which are not used in the explanation due to the less importance.

```plaintext
PROGRAM Rangefinder
VAR
  Laser : ARRAY [0..5] OF UINT;
  StandardLength : UINT;
  NewValueForLasers : UINT;
  PermisionToChange : BOOL;
  DeviationAlarm : BOOL;
  bOperatorAlarm : BOOL;
  bEmergencyStop : BOOL;
  strConditionS : STRING;
  strConditionA : STRING;
  B1 : bool;
END_VAR

IF (Laser[0] AND Laser[5]) <> StandardLength THEN
  strConditionS := 'Diminutive twist';
END_IF

  strConditionS := 'Small twist';
  bOperatorAlarm := TRUE;
END_IF

  strConditionS := 'Huge twist';
  bEmergencyStop := TRUE;
END_IF

IF PermisionToChange THEN
  StandardLength := NewValueForLasers;
  strConditionS := 'New values saved';
END_IF

  DeviationAlarm := TRUE;
  strConditionA := 'Knife damaged';
ELSE
  strConditionA := 'Knife is okay';
  strConditionS := 'No twist';
  bOperatorAlarm := FALSE;
  bEmergencyStop := FALSE;
END_IF

Reference 1. Main element for the rangefinders
```
PROGRAM DIIInitDefaults
VAR_INPUT
END_VAR
VAR_OUTPUT
END_VAR
VAR
iIndex : UINT;
giLastBoo : UINT; // Last index in Boolean Data Table
B1 : bool;
B2 : bool;
ton : TON; // offtimer
END_VAR

/* Initialize Data Tables with default values */
giLastBoo := SIZEOF(arr_Mistakes) / SIZEOF(arr_Mistakes[1]) - 0;
ton(IN:= B1 OR B2, PT:= T#2S);
IF ton.Q THEN
    B1 := FALSE;
    B2 := FALSE;
END_IF

IF B1 THEN
FOR iIndex := 1 TO giLastBoo (* Set boolean defaults *)
    DO arr_Mistakes[iIndex] := arr1_Mistakes[iIndex].def;
END_FOR
END_IF

IF B2 THEN
FOR iIndex := 1 TO giLastBoo (* Set boolean defaults *)
    DO arr_Mistakes[iIndex] := arr1_Mistakes[iIndex].rw;
END_FOR
END_IF

IF B1 OR B2 THEN
    GVL.smistType := 'Updating mistake data';
END_IF

Reference 2. Main element for the answer check
Reference 3. The array of the declared values for the variables for the answer check

```plaintext
VAR_GLOBAL PERSISTENT
(* Current Values *)
arr_boo : ARRAY[0..16] OF BOOL;
END_VAR

VAR_GLOBAL CONSTANT
(* Info *)
arr1_boo : ARRAY[0..16] OF Bool_Def := [
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 1 ),
  (def:= 0 , rw:= 0 ),
  (def:= 0 , rw:= 0 ),
  (def:= 0 , rw:= 0 ),
  (def:= 0 , rw:= 0 ),
  (def:= 0 , rw:= 0 ),
  (def:= 0 , rw:= 0 ),
  (def:= 0 , rw:= 0 ),
  (def:= 0 , rw:= 0 )
]
END_VAR
```
Reference 4. The list of the variables for the clipper simulation program

```plaintext
// TIME
fbNT_GetTime : NT_GetTime;
ST_CurrentTime : TIMESTRUCT;
dtCurrentTime : DT;
fbTimeTimer : TON;
bVisTable : BOOL;
bVisTableCommand : BOOL;
dtTimeOfMistake : DT;

// TABLE

// MISTAKE SOLVING
bMistSolved AT %I* : BOOL;
iMistProcessState : UINT;
iLastMistProcessState : UINT;
sStateDesc : STRING;
Mist : STRING;
callST_MIST : ST_MIST;
arrHistoryOfMistakes : ARRAY [0..100] OF STRING;
arrHistoryOfMistakesTime : ARRAY [0..100] OF DT;
// arrHistoryOfMistakesTime : ARRAY [0..100] OF DT;
T : TON;
T1 : TON;
T2 : TON;
T3 : TON;
bChange : BOOL;
bc : BOOL;
bcl : BOOL;
bc2 : BOOL;
bcl2 : BOOL;
bM1 : BOOL;
bM2 : BOOL;
bM3 : BOOL;
bM4 : BOOL;
bM5 : BOOL;
bM6 : BOOL;
bM7 : BOOL;
bM8 : BOOL;

END VAR
```
Reference 5. The first part of the clipper simulation program

```c
bm1 := arr_Mistakes[bad1];
bm2 := arr_Mistakes[bad2];
bm3 := arr_Mistakes[bad3];
bm4 := arr_Mistakes[bad4];
bm5 := arr_Mistakes[bad5];
bm6 := arr_Mistakes[bad6];
bm7 := arr_Mistakes[bad7];
bm8 := arr_Mistakes[bad8];

ftTimeTimer(IN: NOT fTimeTimer.Q, PT := T#100ms);

ftNW_GetTime(
  DUTID:= ,
  START:= fTimeTimer.Q ,
  TMOUT:= T#5s ,
  TIMESRET=stCurrentTime );

dtCurrentTime := SYSTEMTIME_TO_DT(stCurrentTime);

T[IO := bc , PT := T#5s ];
T1[IO := bc1 , PT := T#5s ];
T2[IO := bc2 , PT := T#5s ];
T3[IO := bc3 , PT := T#5s ];

IF bVisTableCommand THEN
  bVisTable := FALSE;
ELSE
  bVisTable := TRUE;
END IF

IF (bm1 OR bm2 OR bm3 OR bm4 OR bm5 OR bm6 OR bm7 OR bm8) AND NOT (DTInitDefaults.B1 OR DTInitDefaults.B2) THEN
  iMissProcessState := IC;
  bc := TRUE;
END IF

IF DTInitDefaults.B1 OR DTInitDefaults.B2 THEN
  sStateDesc= "Updating mistakes";
END IF

IF I.Q THEN
  bm1 := FALSE;
END IF
```
Reference 6. The second part of the clipper simulation program

```plaintext
CASE iMistProcessState OF

/////////////////////////////////////////////////////////////////

0: //no mistakes

/////////////////////////////////////////////////////////////////

10: sStateDesc := 'Mistake happened';
    dtTimeOfMistake := dcCurrentTime;
    IF iLastMistProcessState <> iMistProcessState THEN
        F_FIFO_WRITE(chData := SIZEOF(sStateDesc),pData := ADR(sStateDesc),
                     cbBuffer := SIZEOF(arrHistoryOfMistakes), pBuffer := ADR(arrHistoryOfMistakes))
        );
        F_FIFO_WRITE_1(chData2 := SIZEOF(dtTimeOfMistake),pData2 := ADR(dtTimeOfMistake),
                        cbBuffer2 := SIZEOF(arrHistoryOfMistakesTime), pBuffer2 := ADR(arrHistoryOfMistakesTime))
        );
    iLastMistProcessState := iMistProcessState;
    END_IF
    
    IF T.Q THEN
        bc1 := TRUE;
        bc := FALSE;
        iMistProcessState := 20;
    END_IF

    /////////////////////////////////////////////////////////////////

20: sStateDesc := 'Repair Started';
    dtTimeOfMistake := dcCurrentTime;
    IF iLastMistProcessState <> iMistProcessState THEN
        F_FIFO_WRITE(chData := SIZEOF(sStateDesc),pData := ADR(sStateDesc),
                     cbBuffer := SIZEOF(arrHistoryOfMistakes), pBuffer := ADR(arrHistoryOfMistakes))
        );
        F_FIFO_WRITE_1(chData2 := SIZEOF(dtTimeOfMistake),pData2 := ADR(dtTimeOfMistake),
                        cbBuffer2 := SIZEOF(arrHistoryOfMistakesTime), pBuffer2 := ADR(arrHistoryOfMistakesTime))
        );
    iLastMistProcessState := iMistProcessState;
    END_IF
    
    IF T1.Q THEN
        bc1 := FALSE;
        bc2 := TRUE;
        iMistProcessState := 30;
    END_IF
```
Reference 7. The third part of the clipper simulation program.
Reference 8. Global list of the variables of main inputs and outputs

```
{attribute 'qualified_only'}
VAR_GLOBAL

//outputs

bLamp1 AT %Q* : BOOL;
bLamp2 AT %Q* : BOOL;
bLamp3 AT %Q* : BOOL;
bLamp4 AT %Q* : BOOL;
bLamp5 AT %Q* : BOOL;
bLamp6 AT %Q* : BOOL;
bLamp7 AT %Q* : BOOL;
bLamp8 AT %Q* : BOOL;
bFire : BOOL;

//mistakes

bError : BOOL;
bMistake1 AT %Q* : BOOL;
bMistake2 AT %Q* : BOOL;
bMistake3 AT %Q* : BOOL;
bMistake4 AT %Q* : BOOL;
bMistake5 AT %Q* : BOOL;
bMistake6 AT %Q* : BOOL;
bMistake7 AT %Q* : BOOL;
bMistake8 AT %Q* : BOOL;

//visualization units

bVis1 : BOOL;
sMistType : STRING;
bPowerOFF : BOOL;

//speed

iSpeedVal : INT;
```

END_VAR