People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed BOUGARA – Boumerdès

Institute of Electrical and Electronic Engineering
Department of Power and Control

Final Year Project Report Presented in Partial Fulfilment of

the Requirements of the Degree of

Master

In **Electrical and Electronic Engineering**

Option: **Control Engineering**

Title:

Information Archiving of remote HMI TP900 comfort In a PLC based networked plant using ODK Development Kit and TIA Portal

Presented By:

- **BOULOUDENE Walid**

Supervisor:
Dr. A. OUADI

Co-Supervisor:
Mr. S. GALOU

Registration Number: M201531033530/2015

Abstract

In a plant composed of S7-1217Cs, connected to an HMI TP900 comfort at a remote location having disconnection problems, we will be developing a solution to store Process Data of the plant PLCs in an Archiving PLC chosen to be an S7-1507S installed in an IPC227E. The archiving will be done in a CSV file that will then be accessed by the HMI after its re-connection. This development will be through ODK Development Kit and TIA Portal. The solution obtained performs efficiently the task and fulfills the criteria of our customer.

Dedication

"What we know is a drop, what we don't know is an ocean."

ISAAC NEWTON

First of all we thank god the almighty for his protection through our life and guidance to follow the right way and pray for him to show us the path to success.

I dedicate this work

To my parents, brother and sister. May this humble work make you proud and hopefully reassure your hearts that the son and sibling you have, lives out to be the image you have of him and hoped he grows into. May God preserve you and keep you as the beacon and light of my life.

To all my friends, in particular those who will recognize themselves here, for all the support and laughs during the long years leading to this moment.

Acknowledgement

First and above all, we thank God for the well-being and health in these hard times, and for providing us this opportunity and granting us the capability to complete this experience at IGEE.

We would like to thank our supervisor, Dr. Abderrehmane OUADI, for his precious advice, supervision and support.

We would like to thank also all the professors and staff who helped us at IGEE from close or afar, in particular Pr. Harriche, Pr. AitKaid and Pr. Yelles (may god rest his soul) for being the image of INELEC we will always remember and cherish and Dr. Dalila CHERIFI for her support and advice throughout the years.

I would also like to express my sincere and complete gratitude and thankfulness to my co-supervisor Mr. Sofiane GALOU for the opportunity to work on a project at SIEMENS and for all his guidance, assistance and monitoring during the past months.

We also thank our parents for the support and encouragement they offered us and all the resources and time they put so that we reach this point in life.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

The automation of systems has been growing more and more in the industry and the reason for it is the multiple benefits it has brought to the manufacture process. The main reason it has invaded so many fields is all thanks to the economical benefit it has generated, from economy of scale by reducing the cost of manufacture by increasing the production and also from the economy of scope by providing a versatility in production and so having less machines to buy to create different products.

The programmable logic controller or PLC is an example of automation system, we can describe it as an industrial computer used to perform a various number of tasks that correspond the manufacturing process it was assigned to. It corresponds perfectly as a basis for our solution, thanks to its many technological properties such as:

1. Fast scan times
2. Small volume and high density of Inputs/Outputs.
3. Efficient data handling and intercommunication with different peripherals.

We can understand the large capabilities of the PLC when we realize how many ways can be used to program it, either by:

- Ladder Diagram (LD): it is a programming language similar to circuit drawing, using contacts and switches. It is more appealing and used by those who have electrical backgrounds.
- Instruction Lists (IL): it is a programming language similar to circuit drawing, using contacts and switches. It is more appealing and used by those who have electrical backgrounds.
- Function Block Diagram (FBD): it is a programming language similar to circuit drawing, using contacts and switches. It is more appealing and used by those who have electrical backgrounds.

For our solution we will be basing our work on not just one of the past languages but concatenate the project using FBD, LD and even create new functions using a development kit we will introduce later in our report.

But the most important characteristic that led us to choose it as our ground to create the solution is its ability to create functions from High-level languages, knowing our solution will be developed in a windows environment, giving us the liberty to program the function in C++.

This flexibility will also permit other engineers to upgrade, extend and even enhance the program as they desire or as the problem at hand requires.

The communication abilities of the PLCs will be useful in our solution implementation, we will be using the Transmission Control Protocol (TCP) to perform the Data transfer between the plant PLCs and the PLC used for Archiving, the TCP protocol is a complement of the Internet protocol (IP), the pair is commonly mentioned as TCP/IP. This protocol is used for small distance between the plant PLCs and the Archiving PLC, through an Ethernet switch and Profinet cables (do not exceed 100m). As we will see further in our work, the IP addresses of the PLCs will be used as connection parameters to establish the data transfer.

The communication between the HMI and the Archiving PLC safely transferred using TCP/IP protocol in form of webpage that access to the PLC. The HMI will be the access point to our saved files in the PLC by reading the CSV files in the directory we will be setting as storage folder of the generated files.

The main and interested process data which is recorded when the HMI disconnects, and then send back once the connection is back to its normal state, are stated as follow:

a. Process Alarms: The process alarms are activated either by a bit changing from 0 to 1 or vice versa, this corresponds to discrete alarms in the HMI, or it can be a value that either goes higher or lower than a certain threshold, this corresponds to analogue alarm.

b. Process Messages: Process messages are programmed according to the plant to display certain messages corresponding to a new state change of the process, starting new step or accomplishment of a certain task (whatever the client wishes to be notified by using messages), messages are set in the HMI by creating a text list, linking it with a certain Tag or a PLC value, and specifying what value sent from the PLC represents as Message we want to display.

c. Process Variables: The plant during its activity generates values that can either be insignificant and do not need to be displayed or have an important role in the process control and plant behavior. The latter type is the one we are interested in; they contain details that our customer doesn't want to lose.

SIEMENS in Algeria:

Siemens is an international multi-industry company of German origin in the field of technology. Industry, Energy and Healthcare represent the main activities of the company. Siemens first activity in Algeria was in 1857 and it was not till 1962 that it opened its first representative bureau, known now as Siemens SPA.

SIEMENS projects in Algeria:

18% of the energy capacity installed in Algeria is generated by Siemens turbines

Over 300 water projects fully equipped with Siemens electrical and automation systems

First Algiers metro line by Siemens in a consortium with CAF and Vinci

Chapter 2

System Architecture and Functional Description

2.1 Introduction:

In this Chapter, we will be looking at what we used to develop our solution, with a detailed explanation of the Hardware component and characteristics. We will also discuss the architecture of the plant proposed to achieve the task at hand.

Finally, we will discuss the work performed by every component in the realization of the project, all in the hope to facilitate the comprehension of the work implemented later to create our proposed solution.

2.2 System specifications:

The required work in the plant present various utilities with a certain degree of complexity, which consists of handling simple hardware wiring and connection to programming the CPU and computer to accomplish the desired task. That is recording information before disconnection and restitute it when the connection is reestablished. The main components are presented hereafter:

2.2.1 Controllers:

We will be using two kinds of PLCs: physical and software controllers. The S7-1200 series used are plant dedicated, while the S7-1500S is solution related.

2.2.2 Software:

We will be using the STIMATIC software such as: TIA portal, S7-PLCSIM in both standard and advanced versions and Eclipse for ODK 1500S.

2.3 System Hardware description:

Our work will revolve around PLC programming and network architecture of the plant, we will start by introducing their hardware:

2.3.1 Instruments:

The solution we decided to implement required the following:

a. SIMATIC S7-1507S:

The S7-1507S is a software controller that offers the same functionalities of S7-1500 automation system only in PC-based real-time environment.

The configuration is easily done via TIA portal to select the properties we want our software controller to have and it is downloaded on an IPC (which we will see next) to be operational.
It is used in our proposed solution, due to its versatility and ability to work with high-level language program as C++. This allows us the possibility to create a function that will allow archiving the data incoming from the plant PLCs to and then being accessed from the HMI terminal.

b. IPC227E:

The IPC227E is a Nanobox embedded industry PC used for its flexibility, high resistance and versatile deployment. It can be used to interface multiple software such as: SIMATIC software controllers and/or WinCC RT advanced. It was chosen because of its characteristics and options as to allow large storage of files and a windowns 7 operating system, beneficial for our C++ program development.

c. SIMATIC S7-1217C:

The S7-1217C is a controller used at our plant to perform the automation and control of the system. There will be multiple PLCs of this type present at the plant, they will be the source of the Process Data we wish to record and archive which will be our only interest.

d. SIMATIC HMI TP900 comfort:

Our HMI used in this project is the TP900 comfort, it has a large number of options and applications. It will serve the purpose of interfacing the information of the plant during its normal use, and will access our Archive files in the S7-1507S thanks to the Windows CE 6.0 web browser. Its role in our solution is to have access to our Archiving PLC and read the stored values created by our code after it has reconnected to the network safely.

2.3.2 Networking Architecture:

The considered system architecture is proposed as shown in Fig. 2.5.

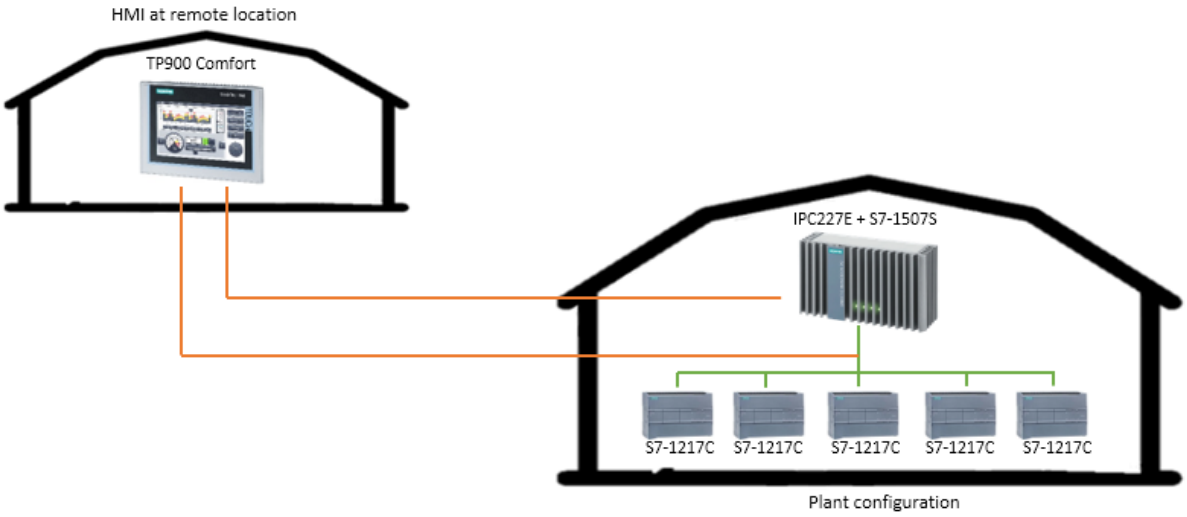


Figure 2.1: Physical aspect of the Solution proposed to our System.

As seen in this Figure, we will be adding our IPC with the Software controller installed to perform the communication and storing of data coming from the S7-1217Cs when our HMI disconnects for any reason.

Having the IPC on site will enable us to maintain a high transmission speed by using PROFINET connection through an Ethernet switch between the S7-1507S and S7-1217Cs, making sure no data exchange would be lost.

The project as worked on in the TIA portal software is represented as follows:

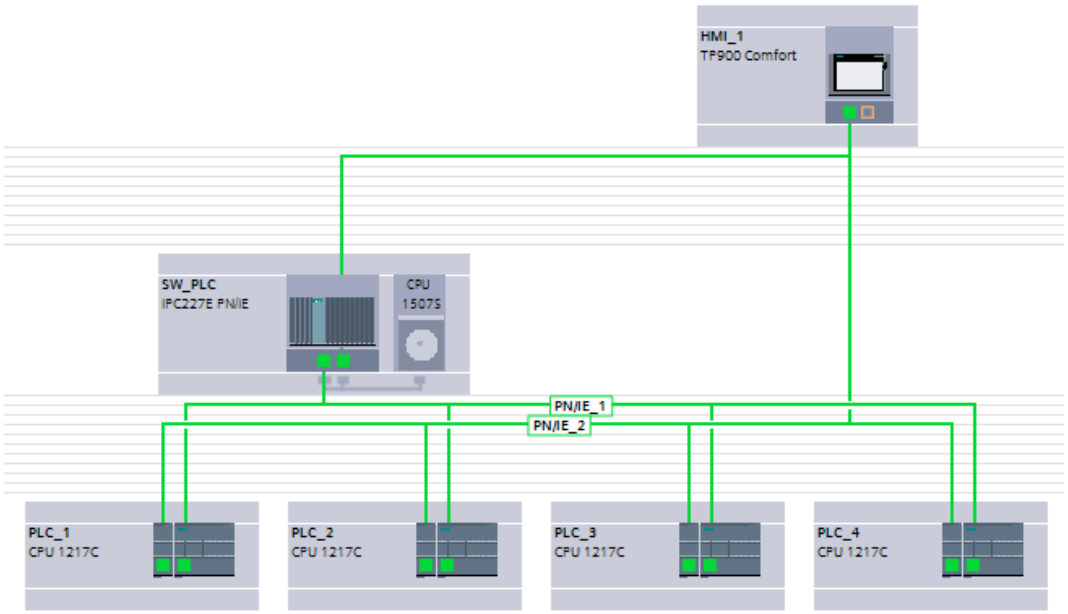


Figure 2.2: Plant connection TIA portal

2.4 Conclusion:

As we have seen, the approach undertaken to create the solution requires the addition of the industrial computer IPC227E and the software controller S7-1507S to the already existing plant PLCs and HMI. Our work will be highly dependant on the software controller as to be the center of communcation and archiving process.

Chapter 3

System Software components description:

3.1 Introduction:

In this chapter, we will be presenting the software used to create the solution, giving more details about the various functions used, both present already in the software or the ones created from our part. Its aim is to familiarize the functions to easily comprehend its use and way of work in the next chapters.

This part is considered as the main contribution that is developed in this project. While the hardware part is performed for implementing the desired system specifications.

3.2 Construction of the software solution:

This solution will be developed separately:

A) TIA PORTAL Programing:

This software is used to generate a function that enables the communication and data transfer from the S7-1217Cs to the S7-1507S. To do so, and after reading and gathering information related to data exchange between PLCs, we decided to perform this task using the TSEND_C function executed in the S7-1217Cs to send data blocks where our desired process details (Alarms, Messages and Variables) are located.

- TSEND_C:

This function will be present at the S7-1217Cs as the transferring function of the DATA BLOCK that contains our process details:

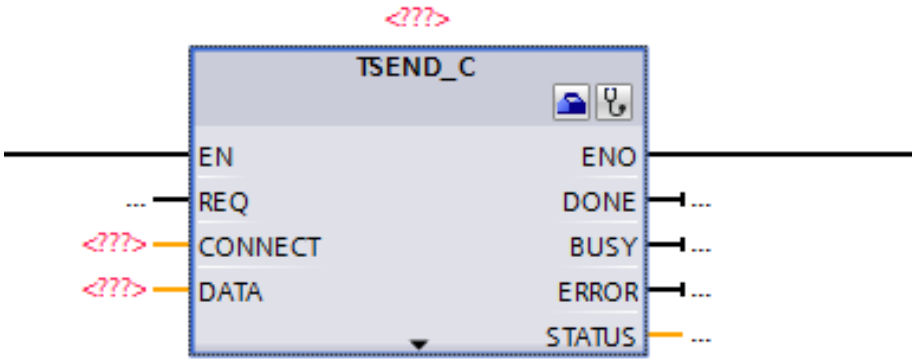


Figure 3.1: TSEND_C function block

-REQ: Rising edge activation of the function block.

-CONNECT: Structure pointing to the connection description.

-DATA: Pointer to the send area that contains the address of the data to be sent.

We will link the process details to contents of this data block that will be of the same data type as defined in the DATA_SEND block

In the S7-1507S we will be using the TRCV_C function linked to a data block that contains the recipient for our process details mentioned above.

- TRCV_C:

This function will be present in our only S7-1507S as to receive the process details transferred by the TSEND_C function in the S7-1217Cs. This function block is defined here:

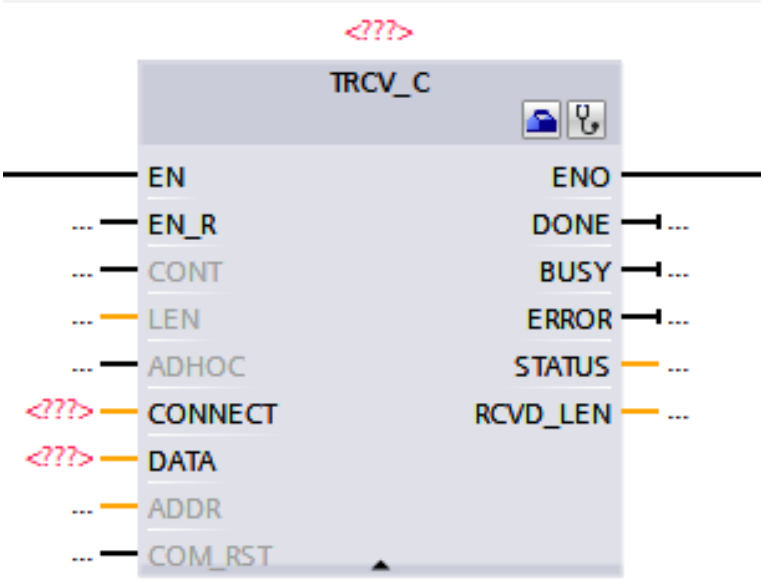


Figure 3.2: TRCV_C function block

-EN_R: Enables receive of data from the TSEND when it is equal to 1 or TRUE.

-CONT: Establishes the communication between the Sender and Receiver when it is TRUE.

-CONNECT: Structure pointing to the connection description.

-DATA: Pointer to the send area that contains the address of the data to be received.

B) Eclipse ODK1500S:

ODK is a development kit that we will be using to program out custom function that will be storing the process details we have seen in the last part in the comma separated values (CSV) file as demanded, this kit will allow us to then generate files that the STEP7 can use, those files are the Structured Control Language (SCL) file containing the functions codes to generate the function blocks for further modifications in the TIA Portal environment, Shared Object (SO) file that we can upload directly into the S7-1507S to perform the task at hand.

Eclipse for ODK 1500S is a software that contains the needed libraries, compilers and even a template to perform the programming of the functions we wish to develop. This task will be performed by coding based on the C++ template for the real-time environment, then compiled to obtain the function block we wish to import into our software controller.

Our implementation process will be as follows:

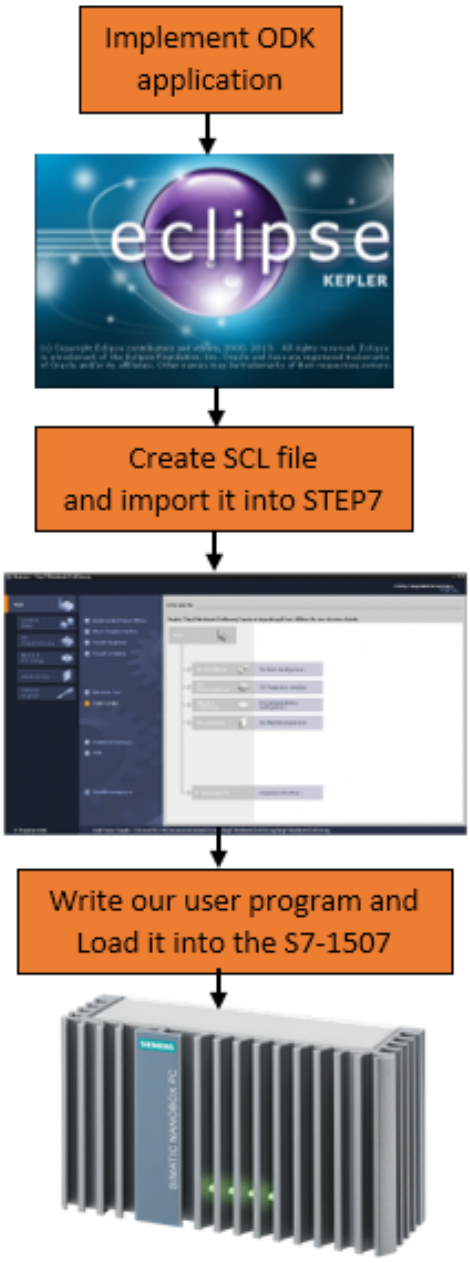


Figure 3.3: ODK application implementation process

3.3 Development of the Software Solution:

3.3.1 TIA Portal programming:

- **Communication and data transfer:**

As we have seen earlier, we will be performing the transfer of the process details by using the TIA Portal functions found in the Communication instructions folder Open User Communication:

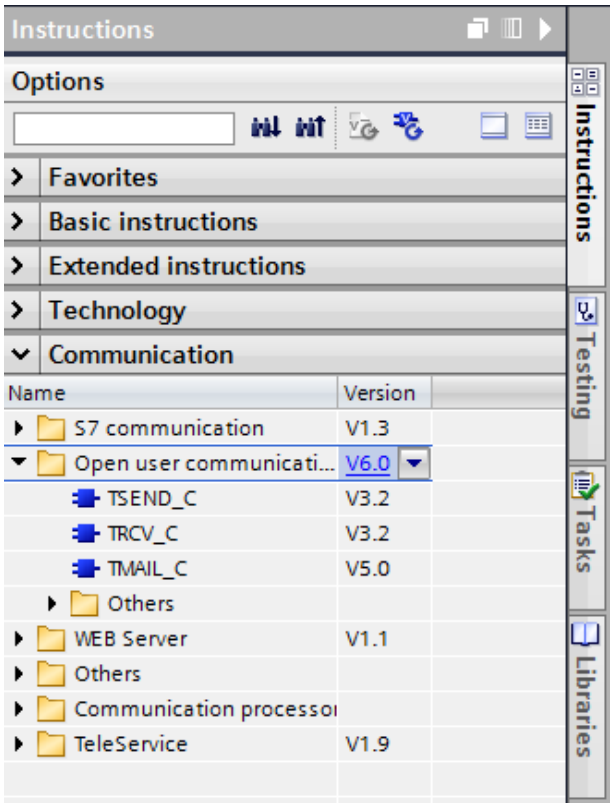


Figure 3.4: Open user communication functions

These two functions will be TSEND_C (for the 1217Cs) and TRCV_C (for the 1507S).

TSEND_C function configuration:

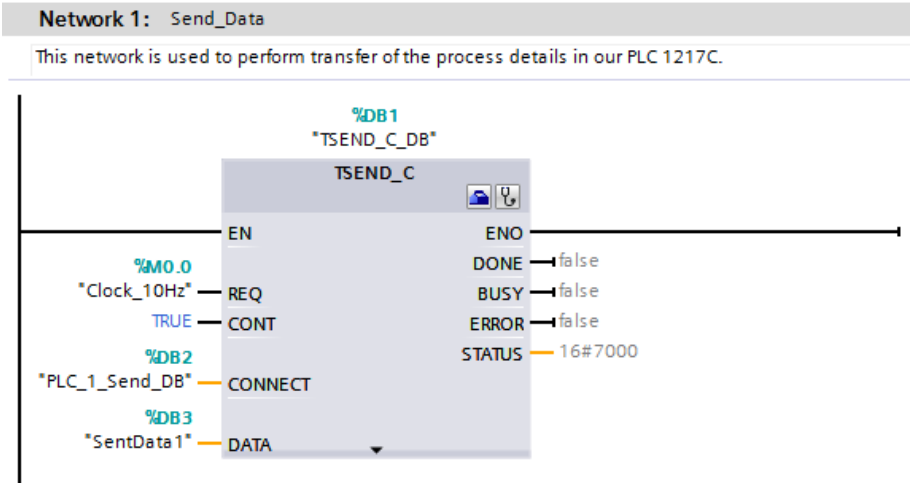


Figure 3.5: TSEND_C function block configuration

The data blocks associated to this function are:

SentData1

	Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...
1	Static						
2	Variable1	LReal	...	0.0			
3	Variable2	Int	...	0			
4	Variable3	UInt	...	0			
5	Alarm1	Bool	...	false			
6	Alarm2	Bool	...	false			
7	Alarm3	UInt	...	0			
8	Alarm4	Bool	...	false			
9	Message1	UInt	...	0			

Figure 3.6: SendData1 Data block

These will be linked to the process functions outputs to store the variables, alarms and messages generated by the PLC during its work.

PLC_1_Send_DB

	Name	Data type	Start value	Retain	Accessible f...	Writa...	Visible in ...
1	Static						
2	InterfaceId	HW_ANY	64				
3	ID	CONN_OUC	1				
4	ConnectionType	Byte	16#0B				
5	ActiveEstablished	Bool	true				
6	RemoteAddress	IP_V4					
7	ADDR	Array[1..4] of Byte					
8	ADDR[1]	Byte	192				
9	ADDR[2]	Byte	168				
10	ADDR[3]	Byte	1				
11	ADDR[4]	Byte	5				
12	RemotePort	UInt	2000				
13	LocalPort	UInt	0				

Figure 3.7: PLC_1_SEND_DB Data block

This Data Block is generated by configuring the TSEND_C function connection as follows:

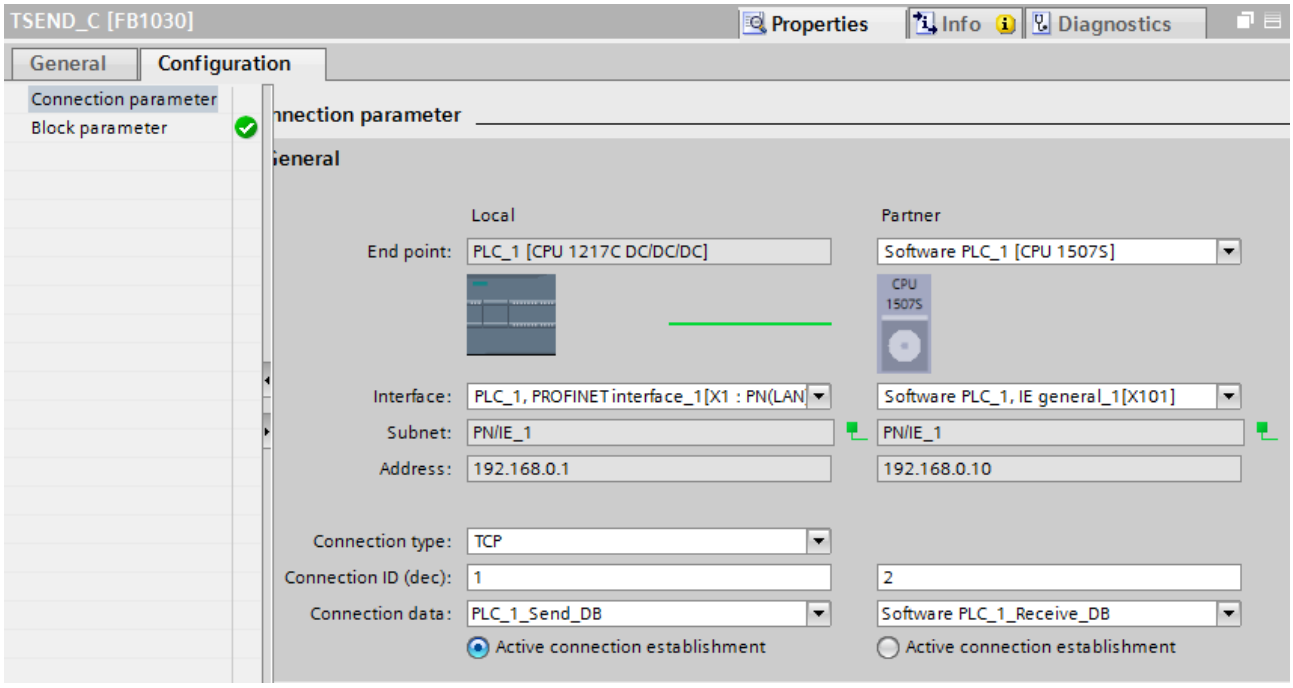


Figure 3.8: Configuration of connection parameters of TSEND_C block function

As we can see, the connection parameters are the interface of connection, and by setting the options a data block is created in the on-work PLC.
We have to clarify that the "Active connection establishment" is active for the PLC that is in charge of the data transfer (the sender).

TRCV_C function configuration

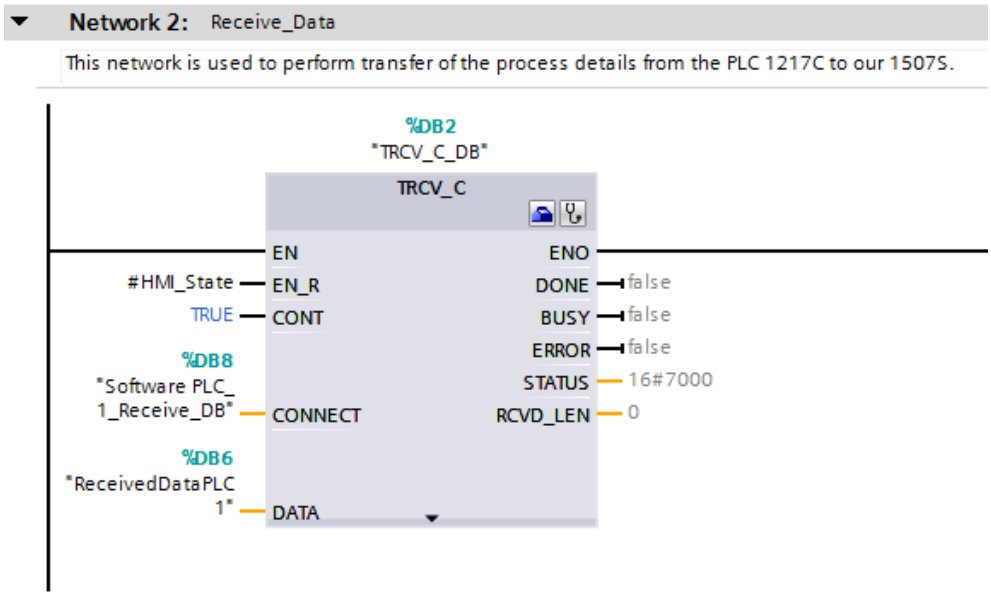


Figure 3.9: TRCV_C function block configuration

The data blocks associated to this function are:

ReceivedDataPLC1

		Name	Data type	Offset	Start value	Retain	Accessible f...	Writa...
1		Static						
2		Variable1	LReal	...	0.0			
3		Variable2	Int	...	0			
4		Variable3	UInt	...	0			
5		Alarm1	Bool	...	false			
6		Alarm2	Bool	...	false			
7		Alarm3	UInt	...	0			
8		Alarm4	Bool	...	false			
9		Message1	UInt	...	0			

Figure 3.10: ReceivedDataPLC1 Data block

These are the exact variables, alarms and messages generated by the PLC 1217C during its work, and that we will be receiving once the data transfer is activated.

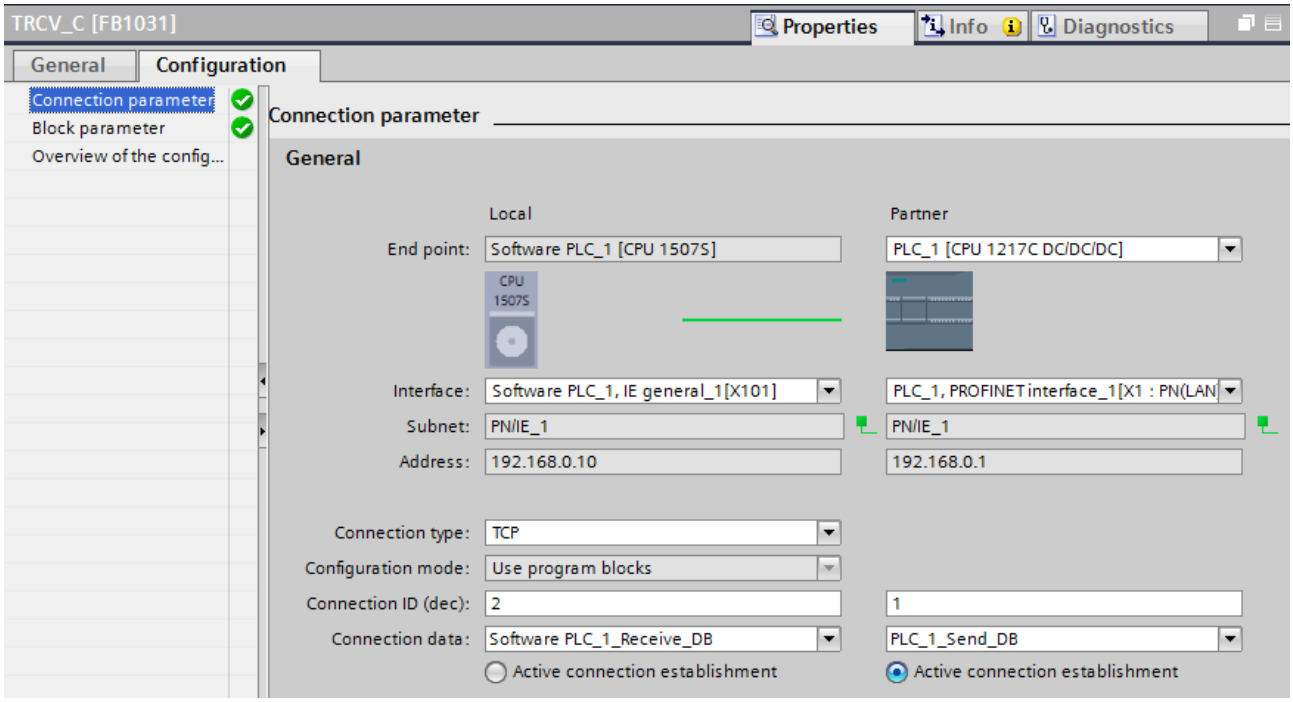


Figure 3.11: Configuration of connection parameters of TRCV_C block function

As we can see, the connection parameters are the interface of connection, and by setting the options, a data block is created in the on-work PLC.
We once again set the PLC_1 as the "Active connection establishment" for it is the sending PLC.

- **Detection of HMI disconnection:**

To detect the HMI TP900 comfort disconnection for whatever reason, we will be using the Coordination pointer of our HMI which is a word data type:

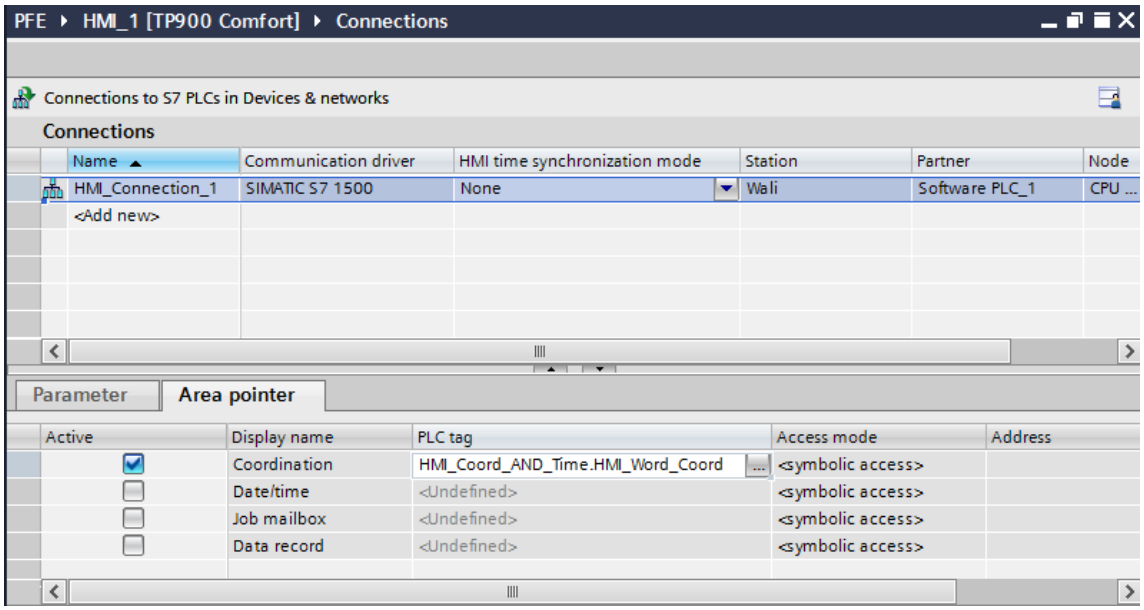


Figure 3.12: Coordination Area Pointer tagging

This will be sent into our data block named: HMI_Coord_AND_Time in the variable: HMI_Word_Coord and stored as a word, this word is devised as follows:

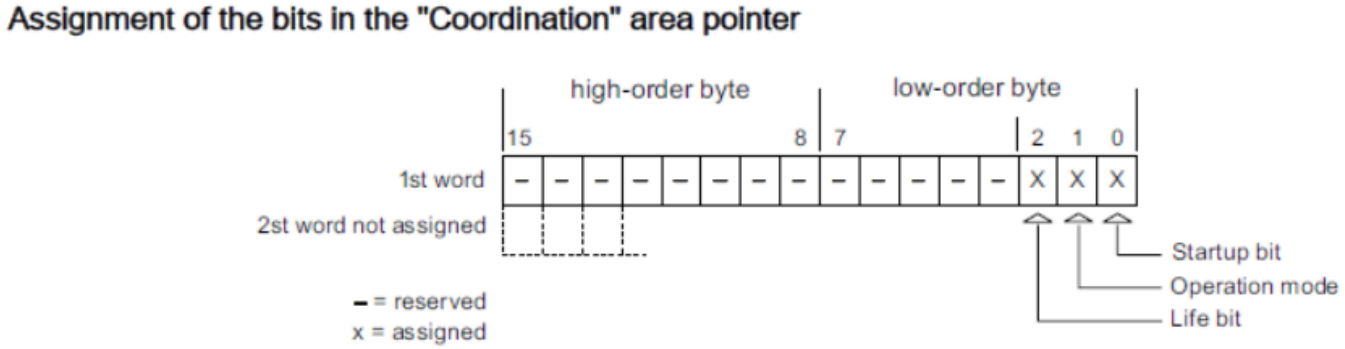


Figure 3.13: Coordination Area Pointer bits assignement

We can easily notice that the 3 least significant bits represent:

-Startup bit: is equal to 1 when it turns on and then 0.
-Operation mode: is equal to 1 when the HMI is turned off by the User and 0 during normal operation mode.
-Life Bit: this bit changes every second from 0 to 1 alternately.

We will be using the Life Bit to detect the disconnection of our HMI and use it to activate the EN_R of our TRCV_C function and launch the data transfer between the 1217Cs and the 1507S.

The Life Bit as mentioned previously will have the following form:

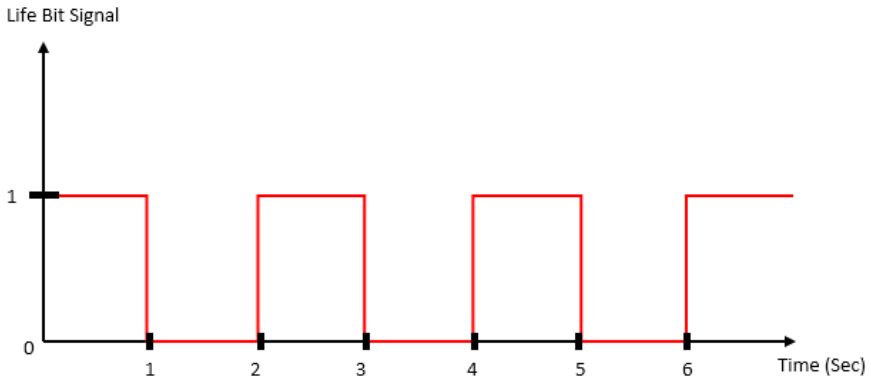


Figure 3.14: Life bit cycle representation

In the case of disconnection, this bit will retain its value at that moment, so to detect connection loss, we will create a function named HMI_LIFEE that will detect the rising edge and falling edge, with a timer of 1s to detect whether our Life Bit changed its state or not, in the case 1 second passes without detection of F/R edge, the output of the function will be set to 1, enabling communication and data transfer from the 1217Cs to the 1507S. If the connection is achieved with the HMI once again, the output of the function becomes 0 stopping unnecessary data transfer.

The function HMI_LIFEE is represented here for more details:

1.Edge detection of the LifeBit.

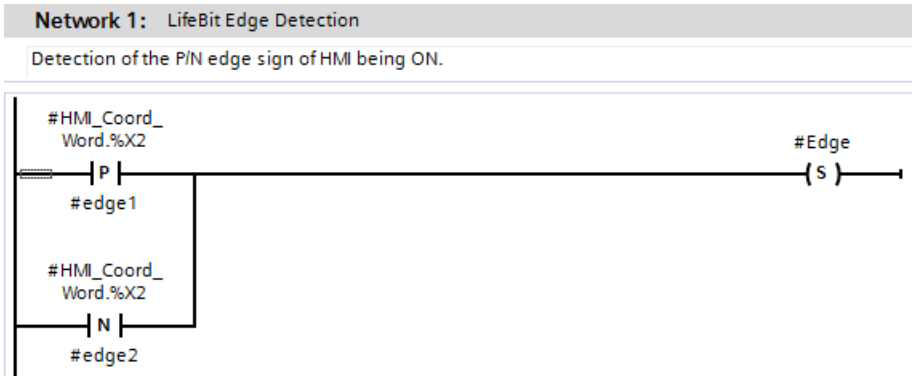


Figure 3.15: Edge detection in the Life Bit Signal

2.Resetting the Pedge and Nedge after 1second of detection:

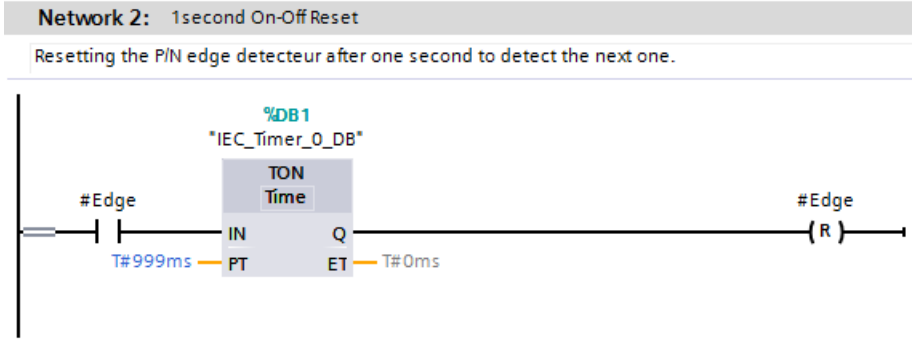


Figure 3.16: Reset of edge detector

3.Disconnection detected after more than 1 second of no edge detection:

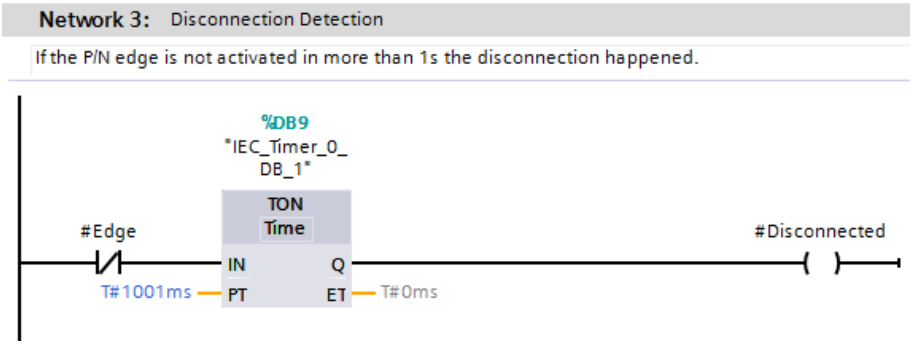


Figure 3.17: Disconnected detected and setting of the function output

- Interface in HMI:

The work would not be done if the HMI after its reconnection could not have access to the created files, that's why we will be setting the Webpage in our S7-1507S so that the HMI can access this webpage by entering the IP address of the PLC.

We will first have to setup our PLC by selecting in its properties webserver and activate web server on this module as well as the protection we want to include, for this task we will be using read access only. We can see the setting of these parameters in here:

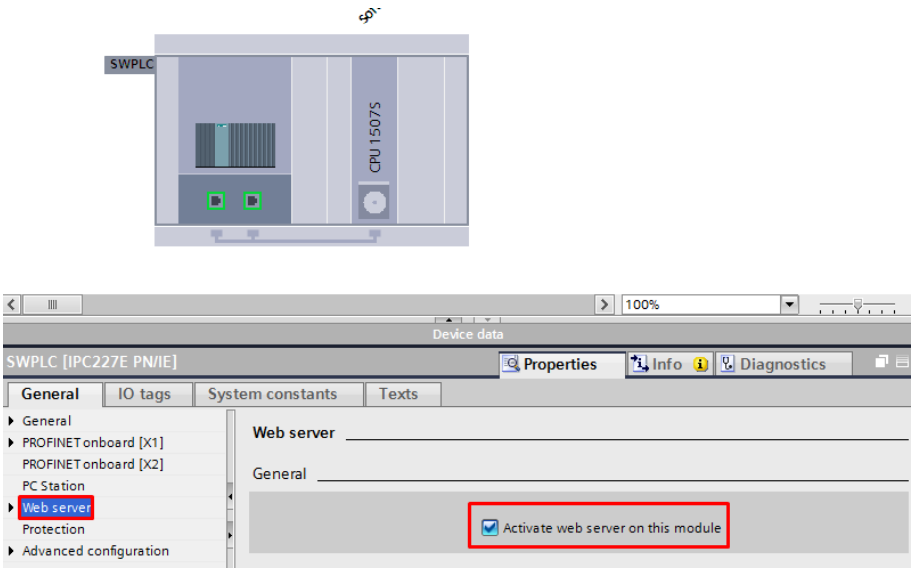


Figure 3.18: Activation of webserver on S7-1507S

This will make the PLC generate a webpage that is accessible from a web browser by entering its IP address, the resulting webpage is represented as follows:

Figure 3.19: Example of S7-1500 Webpage interface

As we can see, this webpage menu contains multiple options, the one we are mostly interested in is the "FILEBROWSER" as our HMI now can access the Files saved in the PLC and by that, it can open and read the new created "DataLog" CSV files:

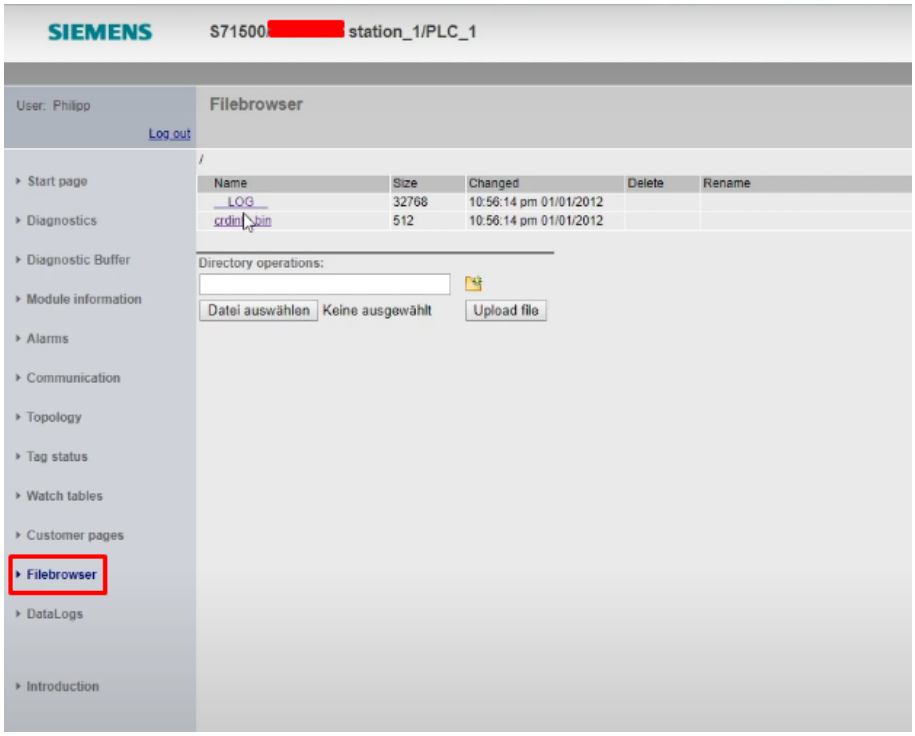


Figure 3.20: S7-1500 Webpage access to FileBrowser option

3.3.2 ODK 1500S programming:

In this part we will be discussing the creation of the function named Writer, which purpose is to archive the process details transferred from the plant PLCs in a CSV file of our choice.
The Flowchart of our program is represented here:

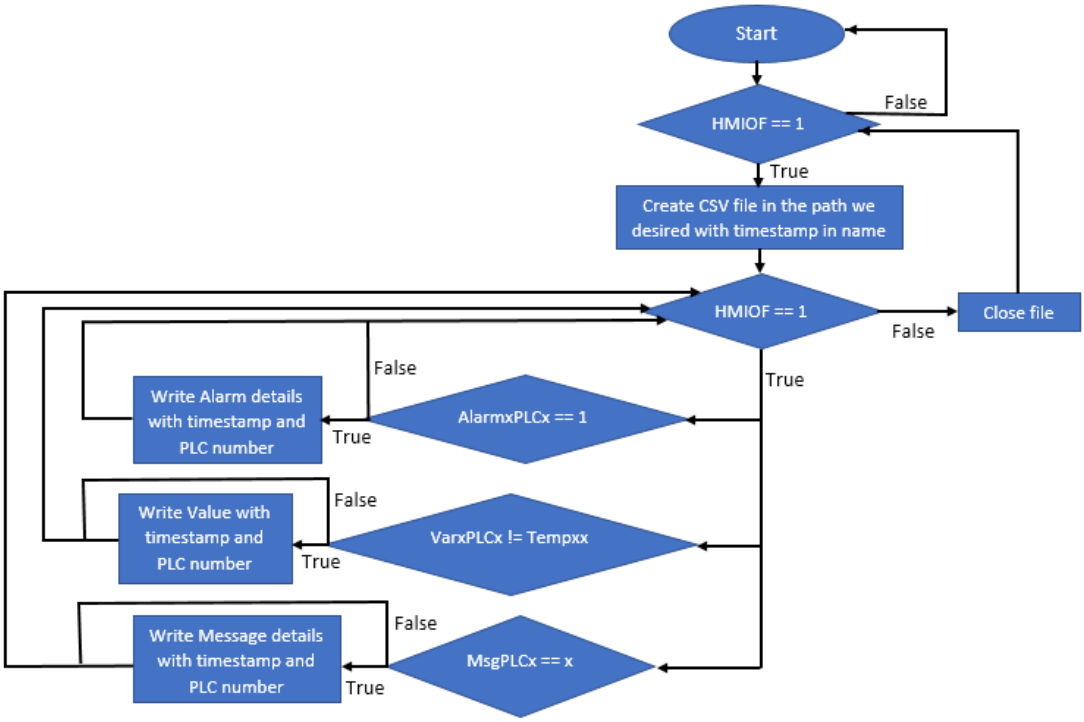


Figure 3.21: Flowchart of the ODK program

From this flowchart we can easily notice the conditional parts of the code, the result of each condition and the overall aspect of the program we are about to write.

We will first search for the libraries appropriate to perform such tasks and the corresponding functions to each of the steps.

The data types used in our ODK application can be easily introduced using the C data types:

ODK data type	SIMATIC data type	C++ data type
ODK_DOUBLE	LREAL	double
ODK_FLOAT	REAL	float
ODK_INT64	LINT	long long
ODK_INT32	DINT	long
ODK_INT16	INT	short
ODK_INT8	SINT	char
ODK_UINT64	ULINT	unsigned long long
ODK_UINT32	UDINT	unsigned long
ODK_UINT16	UINT	unsigned short
ODK_UINT8	USINT	unsigned char
ODK_LWORD	LWORD	unsigned long long
ODK_DWORD	DWORD	unsigned long
ODK_WORD	WORD	unsigned short
ODK_BYTE	BYTE	unsigned char
ODK_BOOL	BOOL	unsigned char
ODK_DTL	DTL	struct ODK_DTL
ODK_S7STRING	STRING	unsigned char

Table 3.1: Data Types between C++ and ODK

The code we used contains the following inputs:

Input	Data Type C++	Data Type ODK	Detail
HMIOF	BOOL	ODK_BOOL	HMI state ODK_TRUE == OFF ODK_FALSE == ON
MsgPLCx	UINT	ODK_UINT16	Message number.
VarxPLCx	UINT LREAL INT	ODK_UINT16 ODK_DOUBLE ODK_INT16	Process Variable value.
AlarmxPLCx	BOOL UINT	ODK_BOOL ODK_UINT16	Process alarm state or number.
Sc	USINT	ODK_UINT8	Seconds (0-59)
Mn	USINT	ODK_UINT8	Minutes (0-59)
Hr	USINT	ODK_UINT8	Hours (0-23)
Day	USINT	ODK_UINT8	Day (1-31)
Mth	USINT	ODK_UINT8	Month (1-12)
Yr	UINT	ODK_UINT16	Year (xx)

Table 3.2: Input Data types

Side note: to obtain the time, we will be using the RD_SYS_T function to generate the time structure DTL that we will store in our data block and extract the components to use them in our code. The code we will obtain is the following:

```
#include "ODK_Functions.h"
#include <stdlib.h>
#include <iostream>
#include <string>
#include <fstream>
#include <time.h>
#include <stdio.h>
#include <sstream>
using namespace std;
 * OnLoad() is invoked after the application binary was loaded.
EXPORT_API ODK_RESULT OnLoad (void)
{
    // place your code here
    return ODK_SUCCESS;
}

/*
 * OnUnload() is invoked before the application binary is unloaded or when
 *           ODK-Host terminates.
 * @return ODK_SUCCESS       notify, that no error occurs
 *         any other value   notify, that an error occurs (user defined)
 *                            - ODK application will be unloaded anyway
 */
EXPORT_API ODK_RESULT OnUnload (void)
{
    // place your code here
    return ODK_SUCCESS;
}
ODK_RESULT Writer (ODK_BOOL& HMIOF, ODK_UINT16& MsgPLC1, ODK_DOUBLE& Var1PLC1, ODK_INT16& Var2PLC1, ODK_UINT16& Var3PLC1,
        ODK_DOUBLE& Var1PLC2, ODK_INT16& Var2PLC2, ODK_UINT16& Var3PLC2, ODK_UINT16& MsgPLC2, ODK_BOOL& Alarm1PLC1,
        ODK_BOOL& Alarm2PLC1, ODK_UINT16& Alarm3PLC1, ODK_BOOL& Alarm4PLC1, ODK_BOOL& Alarm1PLC2, ODK_BOOL& Alarm2PLC2,
        ODK_BOOL& Alarm3PLC2, ODK_BOOL& Alarm4PLC2, ODK_UINT8& Sc, ODK_UINT8& Mn, ODK_UINT8& Hr,
        ODK_UINT8& Day, ODK_UINT8& Mth, ODK_UINT16& Yr)
{
    int Temp31=0, Temp11=0, Temp21=0,Temp32=0, Temp12=0, Temp22=0, TempMP1=0, TempMP2=0;
    while (true)
    {if (HMIOF == ODK_TRUE)
    {
    std::ostringstream oss ;
    oss << "C:/StoringFolder/DataLog_"<< Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<".csv";
    ofstream MyFile;
    string Timestamp = oss.str();
    MyFile.open(Timestamp.c_str());
    MyFile << "Time_Date,Content,PLC Number";
    while (HMIOF == ODK_TRUE)
    {   if (Alarm1PLC1 == ODK_TRUE)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Alarm1 has been turned ON, PLC1";}
    if (Alarm2PLC1 == ODK_TRUE)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Alarm2 has been turned ON, PLC1";}
    if (Alarm3PLC1 > 45)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Alarm3 has been turned ON, PLC1";}
    if (Alarm4PLC1 == ODK_TRUE)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Alarm4 has been turned ON, PLC1";}
    if (Alarm1PLC2 == ODK_TRUE)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Alarm1 has been turned ON, PLC2";}
```

```
    if (Alarm2PLC2 == ODK_TRUE)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Alarm2 has been turned ON, PLC2";}
    if (Alarm3PLC2 == ODK_TRUE)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Alarm3 has been turned ON, PLC2";}
    if (Alarm4PLC2 == ODK_TRUE)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Alarm4 has been turned ON, PLC2";}
    if ( MsgPLC1 != TempMP1){
    if ( MsgPLC1 == 1)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Message One, PLC1";TempMP1 = MsgPLC1;}
    if ( MsgPLC1 == 2)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Message Two, PLC1";  TempMP1 = MsgPLC1;}
    if ( MsgPLC1 == 3)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Message Three, PLC1";TempMP1 = MsgPLC1;}
    if ( MsgPLC1 == 4)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Message Four, PLC1"; TempMP1 = MsgPLC1;}}
    if ( MsgPLC2 != TempMP2){
    if ( MsgPLC2 == 1)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Message One, PLC2";TempMP2 = MsgPLC2;}
    if ( MsgPLC2 == 2)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Message Two, PLC2";TempMP2 = MsgPLC2;}
    if ( MsgPLC2 == 3)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Message Three, PLC2";TempMP2 = MsgPLC2;}
    if ( MsgPLC2 == 4)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Message Four, PLC2";TempMP2 = MsgPLC2;}}
    if ( Var1PLC1 != Temp11)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Variable 1 value is: " << Var1PLC1 <<",PLC1"; Temp11 = Var1PLC1;}
    if ( Var2PLC1 != Temp21)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Variable 2 value is: " << Var2PLC1 <<",PLC1"; Temp21 = Var2PLC1;}
    if ( Var3PLC1 != Temp31)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Variable 3 value is: " << Var3PLC1 <<",PLC1"; Temp31 = Var3PLC1;}
    if ( Var1PLC2 != Temp12)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Variable 1 value is: " << Var1PLC2 <<",PLC2"; Temp12 = Var1PLC2;}
    if ( Var2PLC2 != Temp22)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Variable 2 value is: " << Var2PLC2 <<",PLC2"; Temp22 = Var2PLC2;}
    if ( Var3PLC2 != Temp32)
    {
    MyFile << Hr << "_"<< Mn << "_"<< Sc << "-"<< Day << "_"<< Mth << "_"<< Yr <<",Variable 3 value is: " << Var3PLC2 <<",PLC2"; Temp32 = Var3PLC2;}}
    MyFile.close(); } }
        return ODK_SUCCESS;
}
```

Figure 3.22: Code of the ODK application

The ODK function demands also to define the IN,OUT,INOUT of our function as follows:

```
// how to create a function in ODK 1500S.
ODK_RESULT Writer([INOUT]   ODK_BOOL HMIOF
                 ,[INOUT]   ODK_UINT16 MsgPLC1
                 ,[INOUT]   ODK_DOUBLE Var1PLC1
                 ,[INOUT]   ODK_INT16 Var2PLC1
                 ,[INOUT]   ODK_UINT16 Var3PLC1
                 ,[INOUT]   ODK_DOUBLE Var1PLC2
                 ,[INOUT]   ODK_INT16 Var2PLC2
                 ,[INOUT]   ODK_UINT16 Var3PLC2
                 ,[INOUT]   ODK_UINT16 MsgPLC2
                 ,[INOUT]   ODK_BOOL Alarm1PLC1
                 ,[INOUT]   ODK_BOOL Alarm2PLC1
                 ,[INOUT]   ODK_UINT16 Alarm3PLC1
                 ,[INOUT]   ODK_BOOL Alarm4PLC1
                 ,[INOUT]   ODK_BOOL Alarm1PLC2
                 ,[INOUT]   ODK_BOOL Alarm2PLC2
                 ,[INOUT]   ODK_BOOL Alarm3PLC2
                 ,[INOUT]   ODK_BOOL Alarm4PLC2
                 ,[INOUT]   ODK_UINT8 Sc
                 ,[INOUT]   ODK_UINT8 Mn
                 ,[INOUT]   ODK_UINT8 Hr
                 ,[INOUT]   ODK_UINT8 Day
                 ,[INOUT]   ODK_UINT8 Mth
                 ,[INOUT]   ODK_UINT16 Yr);
```

Figure 3.23: Code of the ODK I/O ports

After the creation of our function and using the release_so compiler, we have now successfully created the SCL file we need to import into our STEP7 Tia Portal for further modifications or to simply connect it to our project, We will then open TIA Portal and select external software and upload our Read_Store.scl file, after it is completed, we generate the function blocks that can be easily noticed, for they have been added to our "Program blocks" folder:

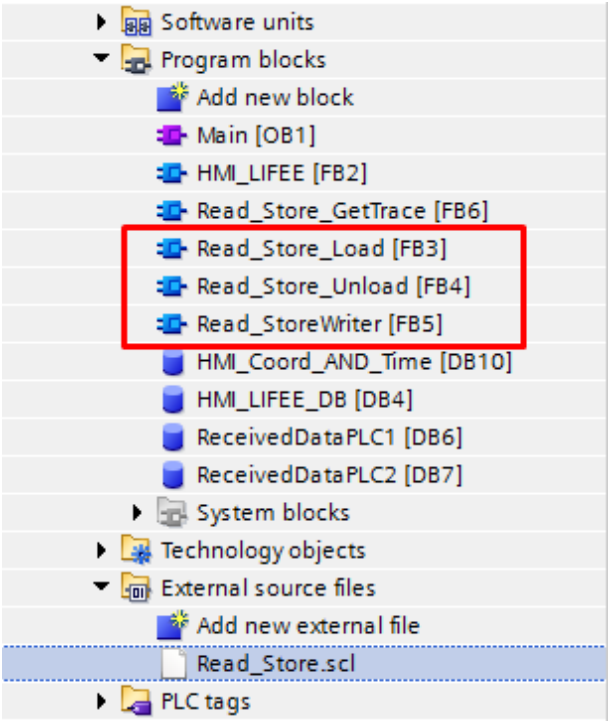


Figure 3.24: Generated function blocks from SCL file

3.4 Assembling and finalizing the Solution:

We now only have to incorporate our generated functions to the Main function of our 1507S. For the Writer function, we will first have to use the Load function as it is mandatory when working with ODK functions.

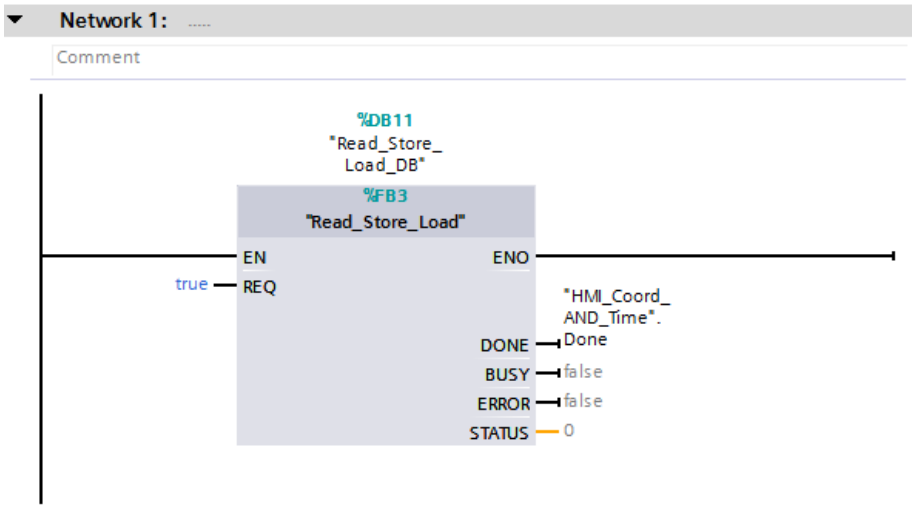


Figure 3.25: Load function of the Read_Store program

Followed by network 2 and 3 that receives the process data from PLC1 and PLC2:

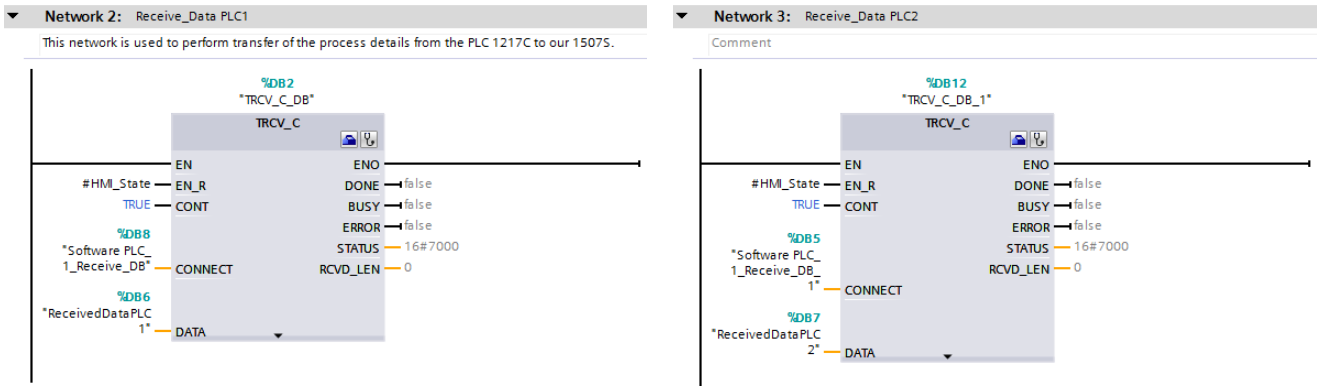


Figure 3.26: TRCV_C function blocks connected to PLC1 and PLC2

We then add in the next network the HMI_LIFEE function block we have conceived earlier:

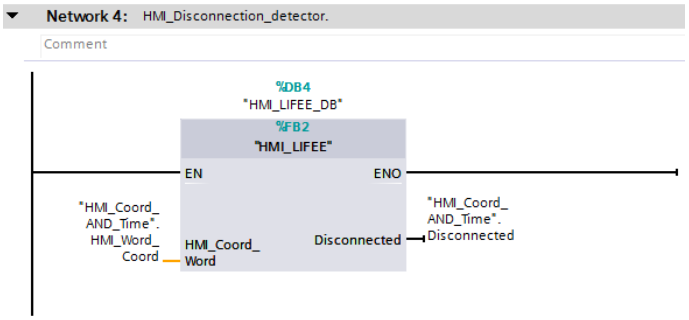


Figure 3.27: HMI_LIFEE function block connections

To activate the Writer function, we will use the Done output of the LOAD function and the Disconnected output of HMI_LIFEE function:

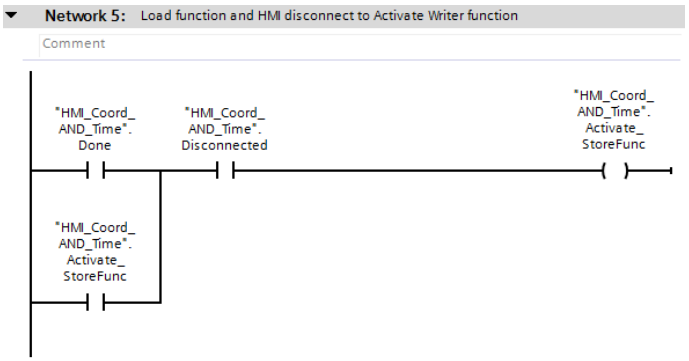
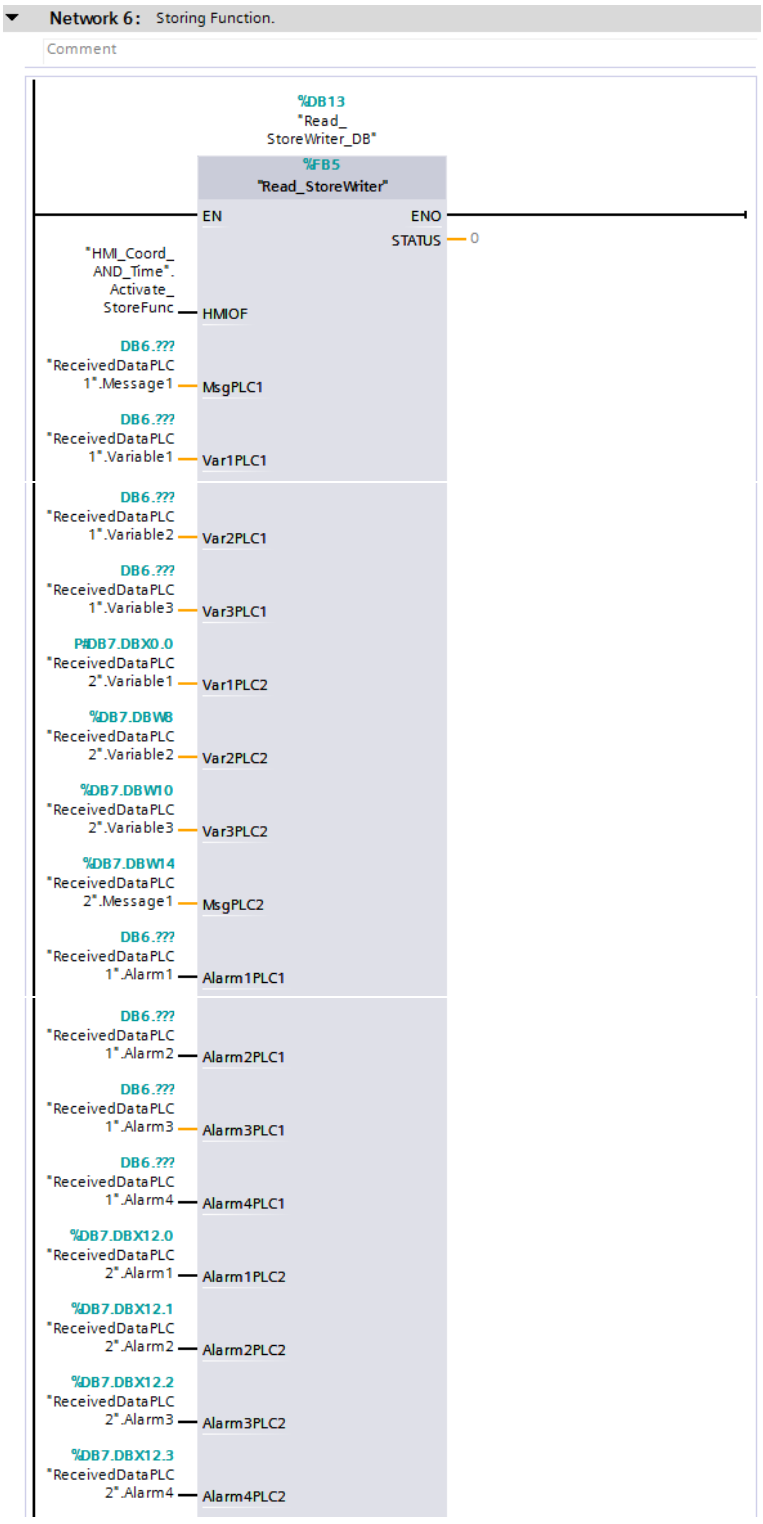


Figure 3.28: Composition of the Writer function block trigger

The next step we have to make is insert the Writer function and configure its inputs as we have mentioned earlier, the result is given here:



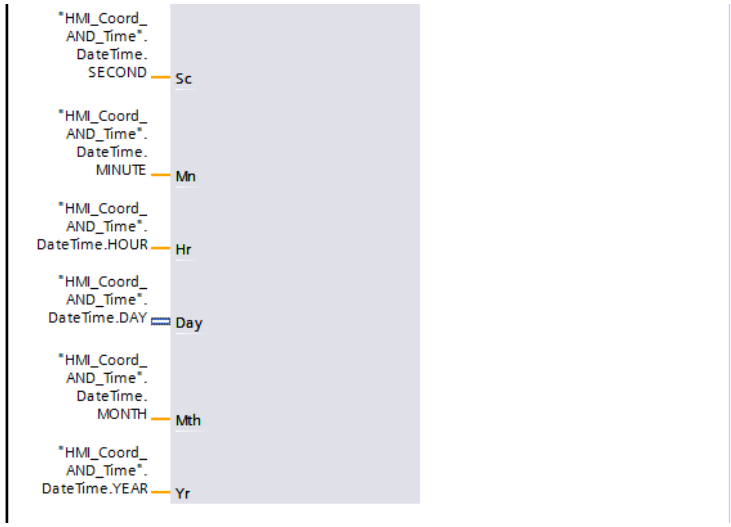


Figure 3.29: Connection of the Writer function to the Data Blocks components and the activation trigger

We also add the function RD_SYS_T that we use to obtain the Time and Date components used in our program:

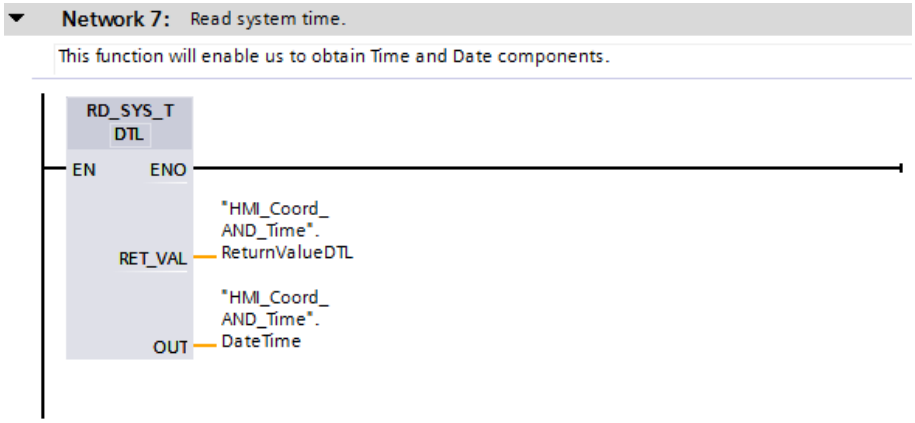


Figure 3.30: Read system time function to obtain timestamp parameters

3.5 Conclusion:

As seen, the solution is a cascade of functions working independently, simultaneously and interacting with each other to perform the complete work of communication of Process Data after detection of disconnection, its Archiving and finally, its access from the HMI terminal on re-connection.

Chapter 4

System Implementation and simulation

4.1 Introduction:

Due to the circumstances we have encountered this year, the Simulation of this project will revolve only around the Data Transfer simulation and the HMI disconnection detection, due to restricted access to the Siemens headquarter and unavailability of the device (IPC227E) and the software controller (S7-1507S), no simulation of the Archiving nor the HMI access to the generated CSV files. We have because of this, performed tests on the C++ storing file program in a third party software to confirm at least the well behaviour and actual performance of our code, as for the Webpage access, we have searched examples and visualized videos concerning this method and have gathered enough positive reviews to be reassured it will work.

The Simulation of the mentioned parts above is aimed to visualize and confirm the well behavior of our protocol concerning the work achieved, this will be performed using the S7-PLCSIM V15.1, a simulation software that will enable us to emulate the S7-1217Cs for example.

4.2 Simulation of the DATA transfer functions:

In this part, we will be testing the TSEND_C and TRCV_C functions implemented in two S7-1217Cs to perform data exchange of Data Block components we have encountered previously.

We will begin by simulating the two PLCs obtaining the following display:

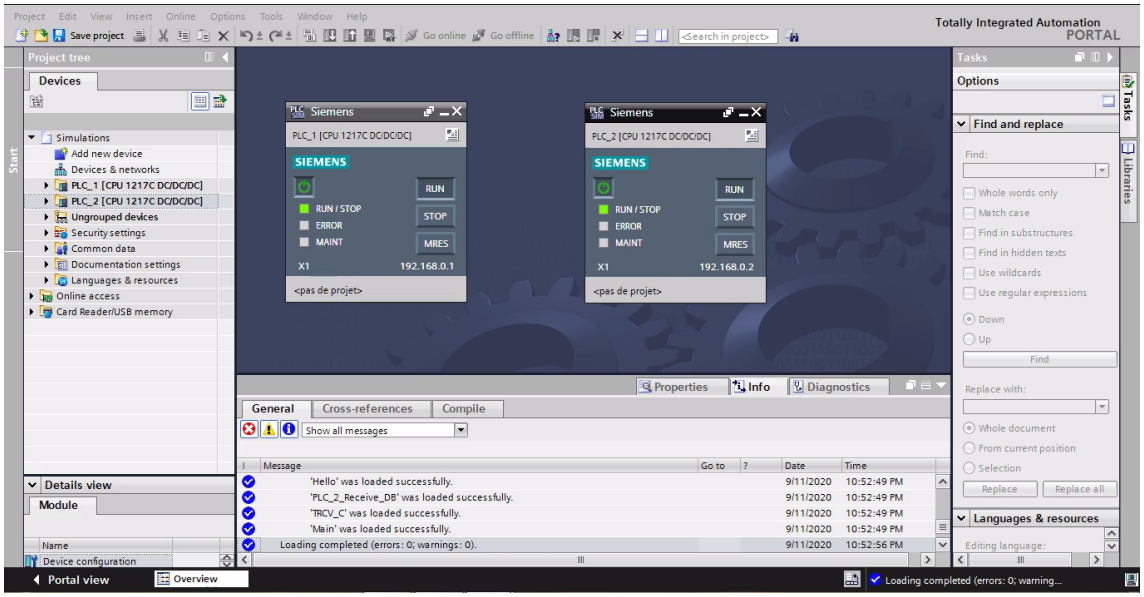


Figure 4.1: Two S7-1217Cs simulated using PLCSIM software

We then access our Data Blocks entitled:

- SendData
- HMISTATE
- ReceiveData

This to test the transfer of the components and time delay, as we can see here, the ReceiveData and SendData have different values meaning no data exchange is performed yet, but if we look closely to our HMIState data block, we can notice the element En_R is False, meaning the TRCV_C function still isn't reading the data incoming (sent) from the TSEND_C function block.

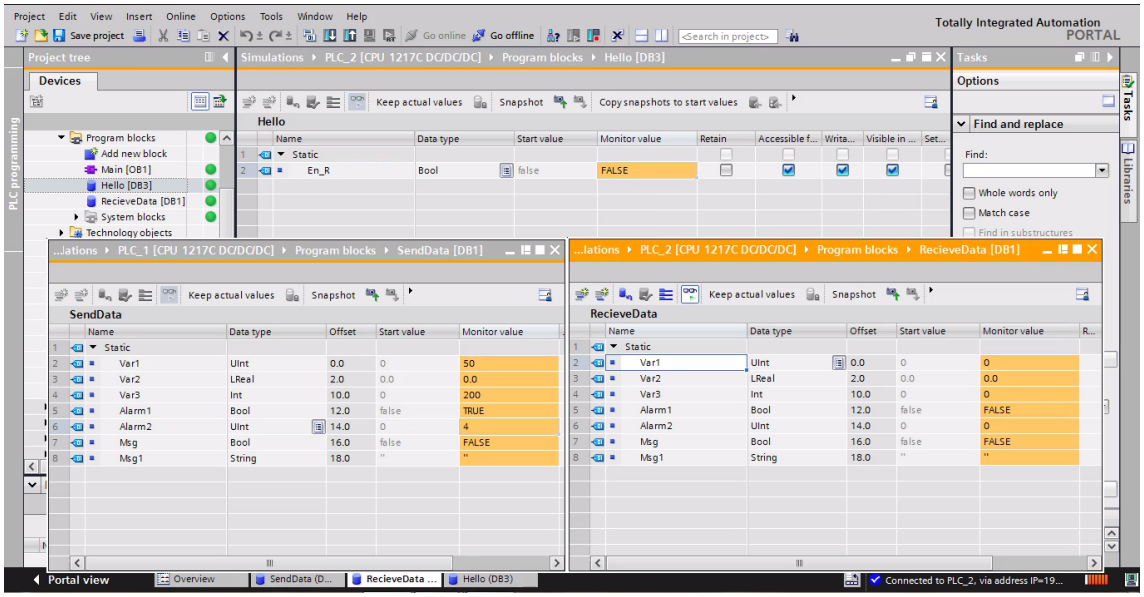


Figure 4.2: Data Block communication with EnableRead OFF

We now change the En_R in our HMISTATE data block to be True, just as the chnage occurs we can notice the update of the elements in the ReceiveData block to be exactly the same as those in the SendData block.

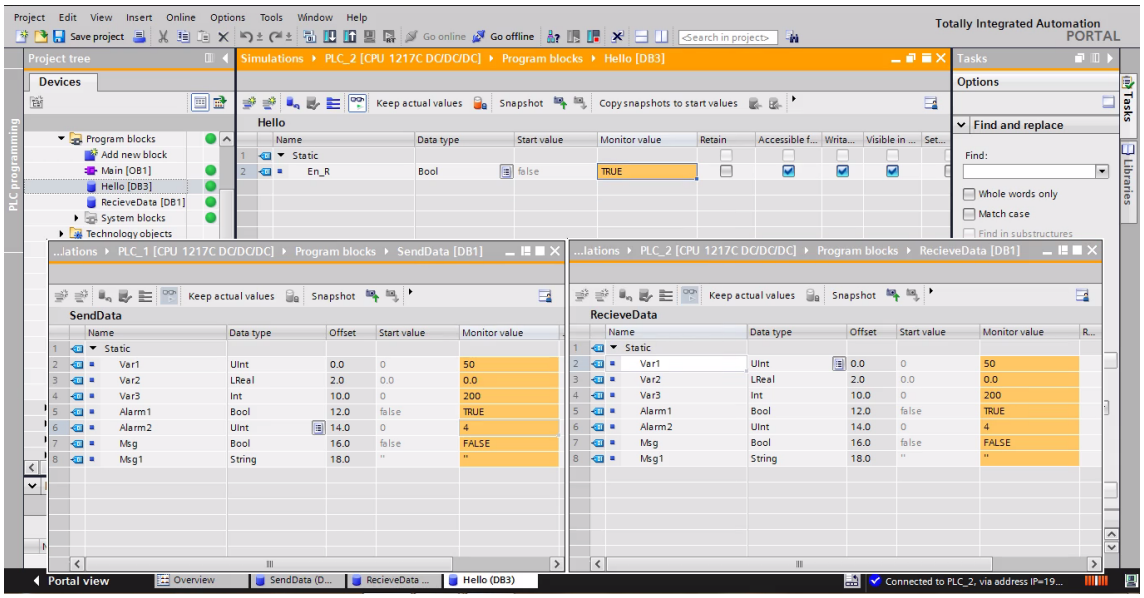


Figure 4.3: Data Block communication with EnableRead ON

We can also use the Status output of the TRCV_C function block to prove the enabling of the data receive, we will monitor the function block with En_R Boolean True and Boolean False and read the corresponding the Status code and what it denotes:

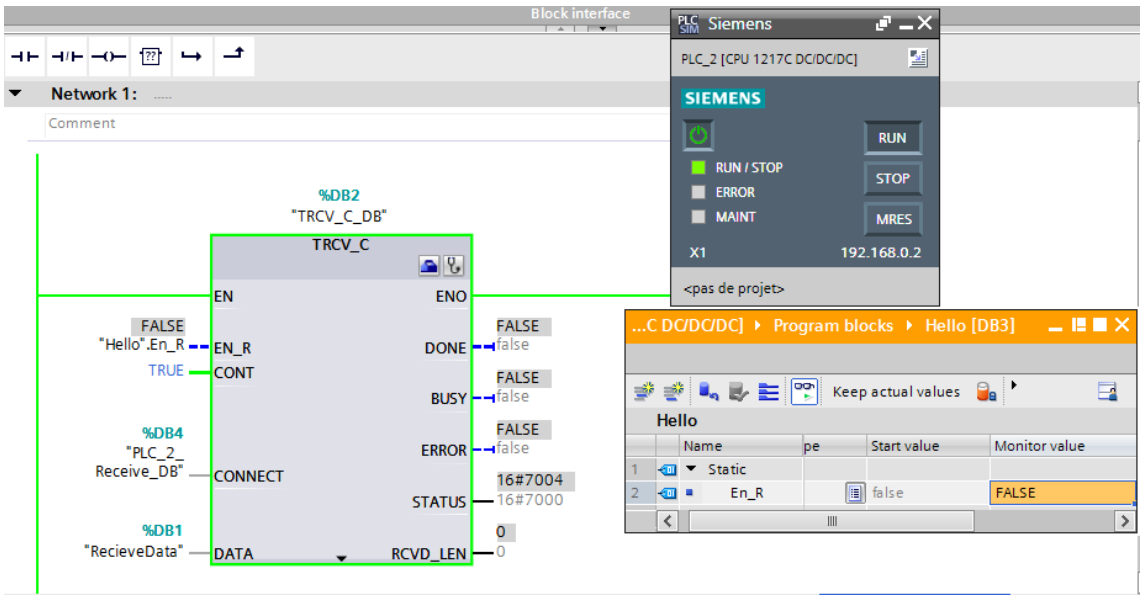


Figure 4.4: TRCV_C block function with En_R set to False

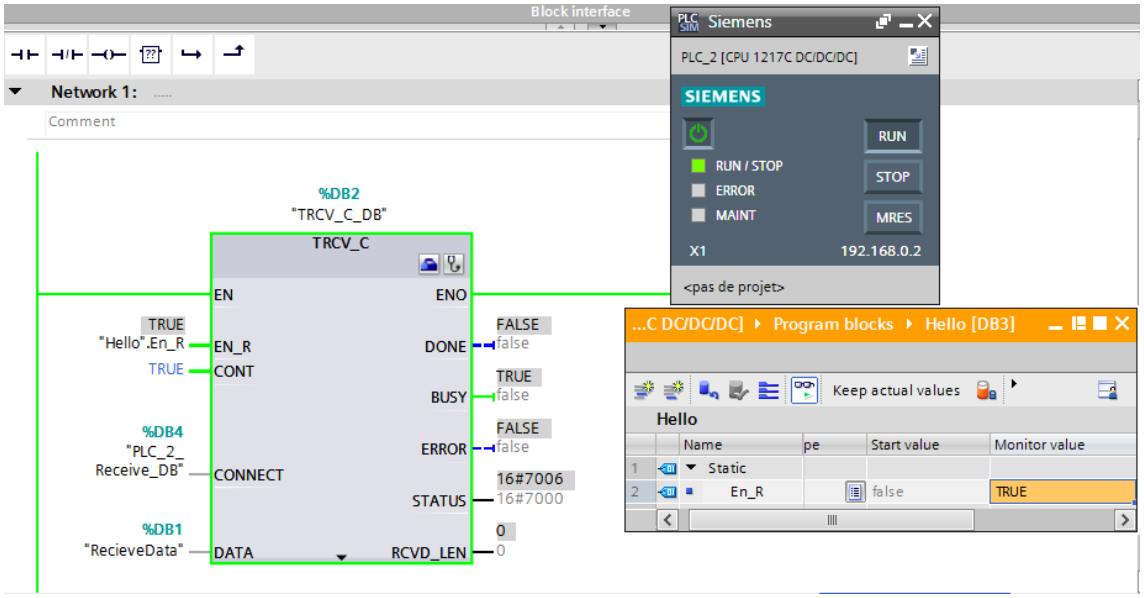


Figure 4.5: TRCV_C block function with En_R set to True

We read the Status output corresponding to the two states of the En_R input:
- En_R set to False : Status code is 16#7004
- En_R set to True : Status code is 16#7006

We access the manual of our function block to obtain the information related to these status codes we find the following:

Status Code	Description
16#7004	Connection established and monitored No job processing active
16#7006	Data is currently being received.

Table 4.1: TRCV_C Status codes and desecription

This proves the functionality and high speed rate of data exchange performed thanks to the Open User communication block functions.

We notice through multiple tests that the clock time we are using as trigger for REQ of the TSEND_C function block leaves a whole 0.1s between each data transfer, this time window even if for us might seem insignificant could hold valuable and important information.

This Data exchange from this perspective is efficient and usable as a step in our Archiving process.

4.3 HMI Disconnection Detection:

We will be testing the block function we have generated, using the Coordination Area pointer LIFE bit in addition to the functions present in the TIA Portal library, to detect the disconnection of the HMI at the remote location, we have simulated a PLC of type S7-1217C for simplicity reasons.

The simulation of this block function can be either monitored from outside as simple input/output relation, or it can be also be supervised from within the function block, having more information about its behaviour.

For simulation sake, we will be using a 1Hz clock generated by the PLC to partially simulate the HMI Life bit, this will permit the existence of an edge (will be only a rising edge but will be enough for this simulation) every one second as the Life bit signal would generate. We will also add a normally closed contact to simulate Disconnection, making the Life bit signal constant and equal its previous value before disconnection occurred.

The resulting Simulation environment be:

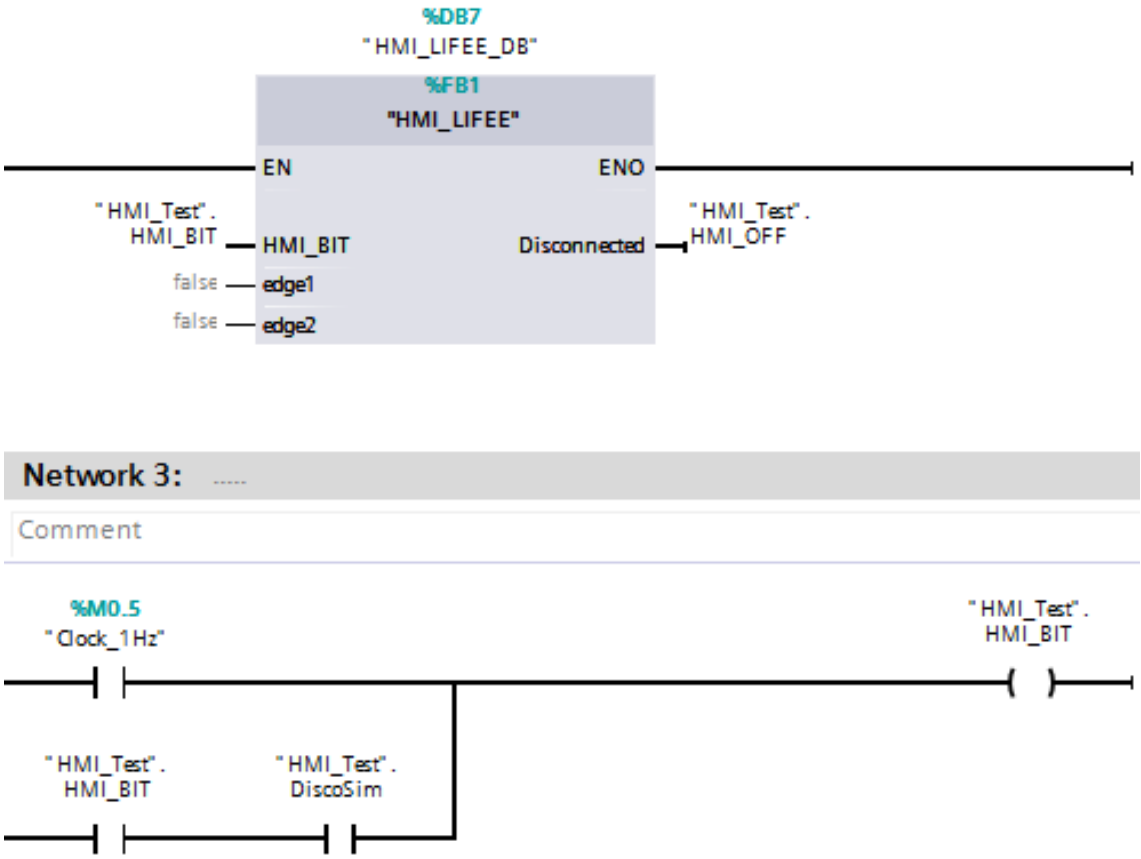


Figure 4.6: Network configuration of the HMI disconnection simulation

We start first with the simple I/O monitoring:

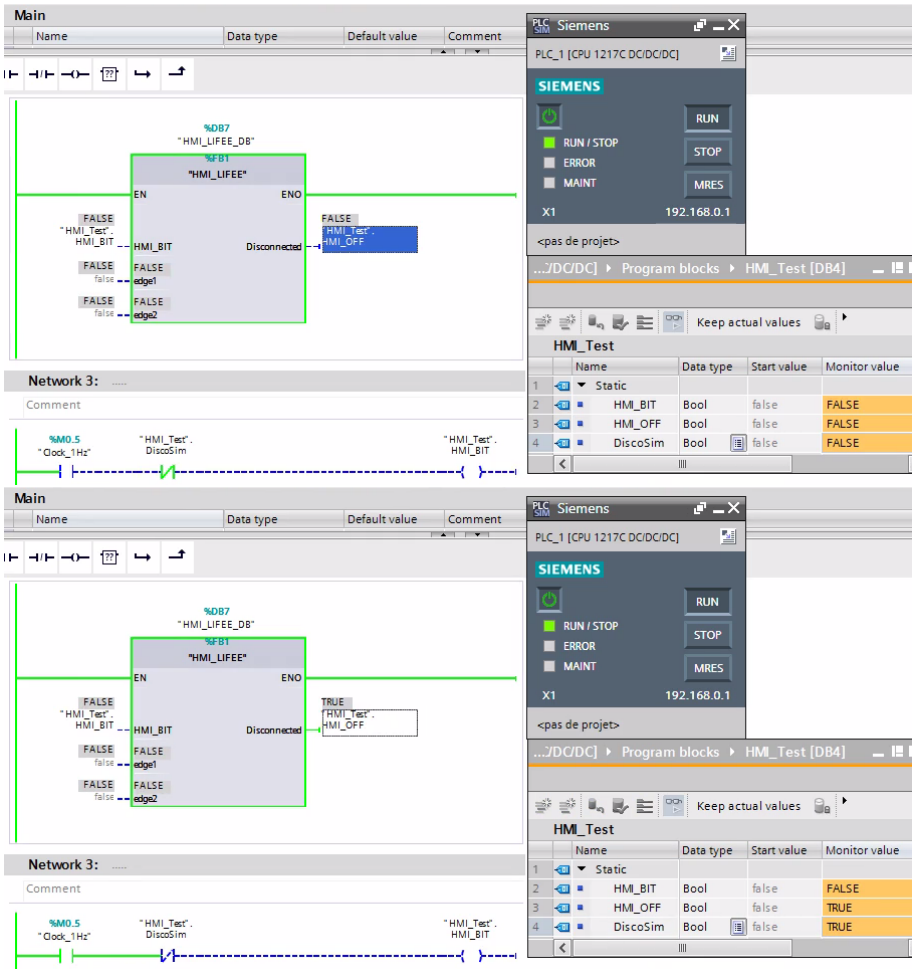
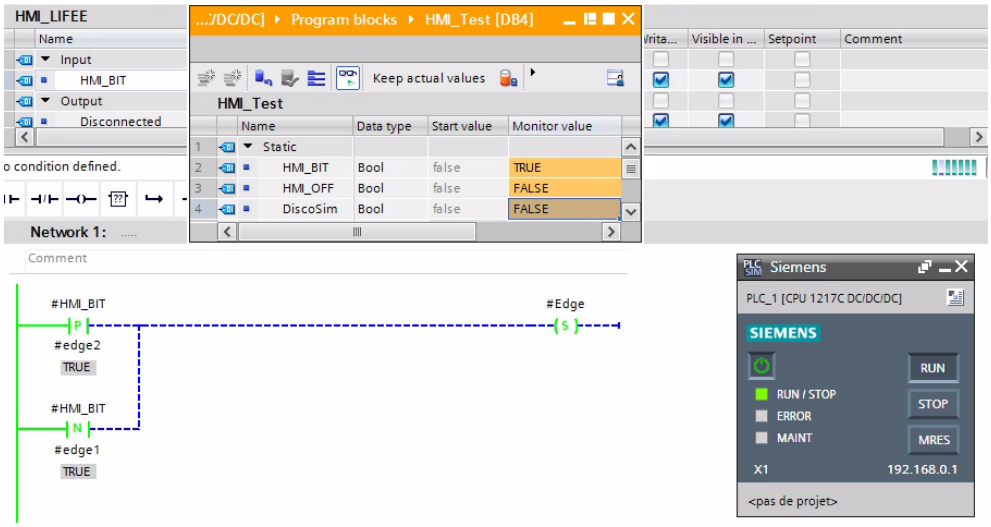


Figure 4.7: Scenario simulation of disconnection of the HMI

For the supervision from inside the function block we obtain the following supervision windows:



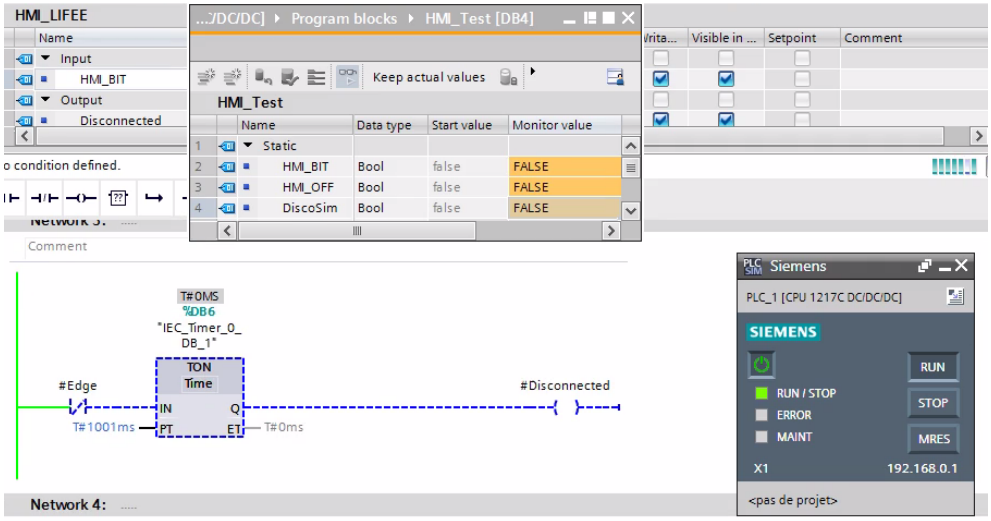
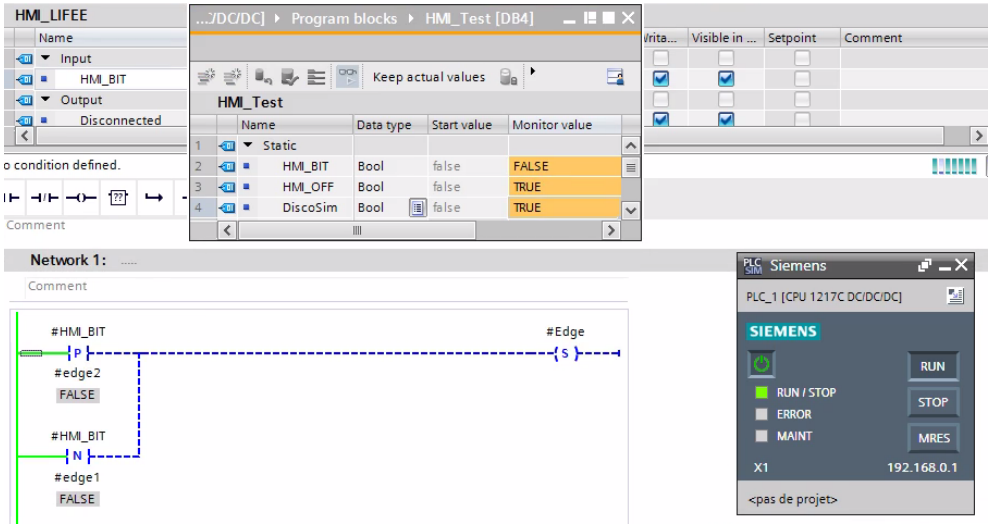


Figure 4.8: HMI connected and Life bit changing every one second



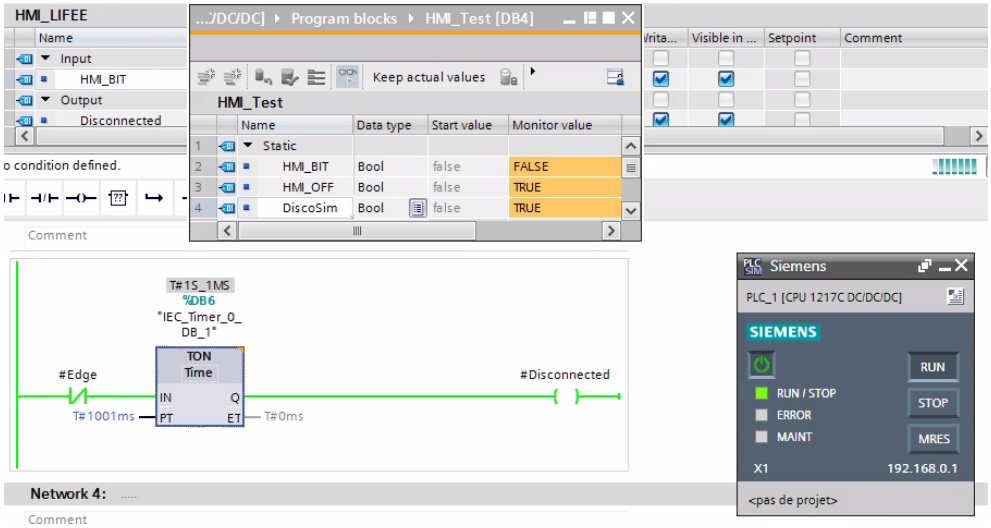


Figure 4.9: HMI Disconnection detected

We can notice that when our Disconnection simulating button is off, the 1Hz clock activates the edge detectors and confirms connection, and that it is established and maintained.

Just as we turn on our disconnection simulating button, the HMI_Bit no longer witnesses a signal change and so no edge has been detected, activating the output of our Disconnection output and setting it to Boolean True.

Through various tests and by setting our disconnection to be just after an edge change, we have a window of almost a whole second before the detection of connection loss, this means a large number of Data could be lost in the scenario this period witnesses an anomaly and then comes back to normal.

As for the rest of the solution, the simulation will be performed, we hope, as soon as the sanitary measures allow us to do so, thankfully the contact and communication with our SIEMENS supervisor has always been swift and full of support, permitting a future development and finalization of the project to reach its end goal.

Chapter 5

Conclusions

As we have seen through the past chapters, our solution is a composition of existing functions in our PLCs' libraries and the development of new ones.

And through the implementation and simulation, we can be confidant that the different functions and data blocks will work in harmony to perform the task solution constructed based on them.

The performance of our solution related to time can be measured as the addition of the time delays resulting from each part, as for example our Detection of Disconnection. We have noticed that the maximum delay that could occur would be approximately one second (in the case the disconnection occurs just after an edge is generated), leading to a 1 second delay before the disconnection is properly detected. Next comes the rate of exchange, we will have a 0.1 second between every Data transmission meaning the data generated during this 0.1 second will be lost if of course the data witnesses different values at the new cycle.

For the Archiving program, the delay is the time the program will take to make each scan of the inputs and perform the Archiving of Data and loop again the scan, leading to 174 instruction, with a maximum processing time set to 2nanoseconds. We can be sure that the delay is insignificant and leading to a high performance factor, thanks to fast execution time compared to Data receiving time.

The HMI interface of the stored CSV files is non time critical and can be done at any moment.

As a general overview of the solution, we can conclude its various advantages:

- Time efficiency:

  As for the nature of the problem at hand, time of communication and storage is of high importance as to reach the lowest percentage of Data loss, as we have seen previously, the time delays and processing time add up to a maximum of 1 second the moment of disconnection and 0.1 second after detection.

- Versatility:

  This program biggest advantage is the generality it has to perform this task for any problem of similar nature, all thanks, to its conception using high-language programming, permitting changes to be applied to the original code and creating a function based on the main block but with a flexibility as to the types to be saved, new variables, alarms or messages to be saved, and also the addition of system alarms.

- Cost:

  Thanks to the creation of the Archiving function and the use of already present hardware to develop parts of the solution, we have achieved numerous savings rather than buying a software already present to perform similar task that chains us to a single use. Since our program has been developed for and in the company, this makes it free for any future use in any other project by SIEMENS engineers for any project of similar nature, all thanks to the versatility factor.

As for expansion and enhancement in future work we would mention:

- Program robustness:

  The upgrade we can implement in our program is the verification of the creation and closure of the file after the corresponding command for these tasks, this will guarantee no problem of storage could ever originate from the Archiving code nor the malfunction of the machine, the added part will contain a conditional statement that will let the program move forward only if the creation and closure commands have been performed successfully.

- Time:

  The improvement we could make concerning time of execution and performance of our solution, would be to first create a custom clock using Pulse Width Modulation function to generate a clock with faster cycle than the 10Hz factory present in our PLC, to obtain a higher rate of Data exchange and so less Data loss in the process. We could also use the bit mode in the Coordination Area Pointer to diminish the Detection of HMI Disconnection delay.

- HMI access PLC:

  The enhancement we could perform on the part of our solution related to HMI access of the CSV file contating the Process Data saved, is by creating a customer page using HTML coding to create a webpage that enables us to directly read the CSV file without having to download them. This expansion would benefit our work by facilitating the access to the files and also, by canceling the mandatory download of the files to access them and reducing the storage problems that could occur.

Appendix A

Instruments Data Sheet

S7-1507S Software Controller:

SIMATIC Software Controller CPU 1507S

Product details | Technical data

Technical data

SIMATIC S7-1500, Software Controller CPU 1507S Single license f. 1 install., R-SW, SW and documentation on DVD, License key on USB stick, R-SW Class A, 6 languages (de,en,it,fr,es,zh), executable in Windows 7 and Windows 10; Reference HW: SIMATIC IPC2x7E, IPC4x7E, IPC4x7D, IPC6x7D, IPC8x7D

General information	
Product type designation	CPU 1507S
Software version	V20.8
Product function	
• I&M data	Yes; I&M0 to I&M3
Engineering with	
• STEP 7 TIA Portal configurable/integrated from version	V16
Configuration control	
via dataset	Yes
Memory	
SIMATIC memory card required	No; Use of the PC mass storage
Work memory	
• integrated (for program)	5 Mbyte
• integrated (for data)	20 Mbyte
• integrated (for CPU function library of CPU Runtime)	50 Mbyte
Load memory	
• integrated (on PC mass storage)	320 Mbyte
Backup	
• with UPS	Yes; all memory areas declared retentive
• with non-volatile memory	Yes; Depending on PC hardware
CPU processing times	
for bit operations, typ.	1 ns; On IPC427E, Intel Xeon processor
for word operations, typ.	2 ns; On IPC427E, Intel Xeon processor
for fixed point arithmetic, typ.	2 ns; On IPC427E, Intel Xeon processor
for floating point arithmetic, typ.	2 ns; On IPC427E, Intel Xeon processor

IPC227E Nanobox PC:

SIMATIC IPC227E (Nanobox PC)

Product details | Technical data | CAx data

Technical data

SIMATIC IPC227E (Nanobox PC); 1x display port; 2x 10/100/1000 Mbit/s Ethernet RJ45; 1 x USB3.0, 3 x USB2.0; CFast slot; 24 V DC industrial power supply

Installation type/mounting	
Mounting	DIN rail, wall mounting, portrait mounting
Design	Box PC, built-in unit
Supply voltage	
Type of supply voltage	24 V DC
Mains buffering	
• Mains/voltage failure stored energy time	20 ms
Processor	
Processor type	Intel Celeron N2807 / N2930, Intel Atom E3845
Chipset	SoC
Graphic	
Graphics controller	Integrated
Drives	
Hard disk	2.5" SATA ≥ 320 GB
SSD	Yes
Memory	
Type of memory	DDR3L SO-DIMM
Main memory	2 / 4 / 8 GB
Capacity of main memory, max.	8 Gbyte
Data areas and their retentivity	
Retentive data area (incl. timers, counters, flags), max.	512 kbyte; 128 KB can be stored in the buffer time; optional
Hardware configuration	
Slots	
• free slots	1x PCIe (x1) (optional)
• Number of PCI slots	1; Optional
• Number of compact flash slots	1; CFast

S7-1217C DC/DC/DC Controller:

Technical data

SIMATIC S7-1200, CPU 1217C, compact CPU, DC/DC/DC, 2 PROFINET ports onboard I/O: 10 DI 24 V DC; 4 DI RS422/485; 6 DO 24 V DC; 0.5A; 4 DO RS422/485; 2 AI 0-10 V DC, 2 AO 0-20 mA Power supply: DC 20.4-28.8V DC, Program/data memory 150 KB

General information	
Product type designation	CPU 1217C DC/DC/DC
Firmware version	V4.4
Engineering with	
• Programming package	STEP 7 V16 or higher
Supply voltage	
Rated value (DC)	
• 24 V DC	Yes
permissible range, lower limit (DC)	20.4 V
permissible range, upper limit (DC)	28.8 V
Reverse polarity protection	Yes
Load voltage L+	
• Rated value (DC)	24 V
• permissible range, lower limit (DC)	20.4 V
• permissible range, upper limit (DC)	28.8 V
Input current	
Current consumption (rated value)	600 mA; CPU only
Current consumption, max.	1 600 mA; CPU with all expansion modules
Inrush current, max.	12 A; at 28.8 V DC
I^2t	0.5 $A^2 \cdot s$
Output current	
for backplane bus (5 V DC), max.	1 600 mA; Max. 5 V DC for SM and CM
Encoder supply	
24 V encoder supply	
• 24 V	L+ minus 4 V DC min.
Power loss	
Power loss, typ.	12 W
Memory	
Work memory	
• integrated	150 kbyte
• expandable	No

SIMATIC HMI TP900 Comfort:

SIMATIC HMI TP900 Comfort

Product details | Technical data | CAx data

Technical data

SIMATIC HMI TP900 Comfort, Comfort Panel, Touch operation, 9" widescreen TFT display, 16 million colors, PROFINET interface, MPI/PROFIBUS DP interface, 12 MB configuration memory, Windows CE 6.0, (Microsoft Support included Security updates discontinued) configurable from WinCC Comfort V11

General information	
Product type designation	TP900 Comfort
Display	
Design of display	TFT
Screen diagonal	9 in
Display width	195 mm
Display height	117 mm
Number of colors	16 777 216
Resolution (pixels)	
• Horizontal image resolution	800 Pixel
• Vertical image resolution	480 Pixel
Backlighting	
• MTBF backlighting (at 25 °C)	80 000 h
• Backlight dimmable	Yes; 0-100 %
Control elements	
Keyboard fonts	
• Function keys	
— Number of function keys	0
— Number of function keys with LEDs	0
• Keys with LED	No
• System keys	No
• Numeric keyboard	Yes; Onscreen keyboard
• alphanumeric keyboard	Yes; Onscreen keyboard
Touch operation	
• Design as touch screen	Yes

People's Democratic Republic of Algeria
Ministry of Higher Education and Scientific Research

University M'Hamed
BOUGARA - Boumerdes

Institute of Electrical and
Electronic Engineering

Department of
Power and Control

Authorization for Final Year Project Defense

Academic year: 2019/2020

The undersigned supervisor: **OUADI Abderrahmane**
Authorizes the students:

BOULOUDEN Walid ..... Option ..... Control .....

..... Option .....

..... Option .....

to defend their final year Master program project entitled:
information Archiving of remote HMI TP900 comfort In a PLC based networked plant using ODK Development Kit and TIA Portal

during the session of: ☐ June ■ September.

Date: 14 / 09 / 2020

The Supervisor: A. Ouadi

The Department Head