CHALMERS

Virtual commissioning of production process Final report

Master thesis MPSYS - SSYX04-17

Sara Winther

Department of Signals & Systems
Chalmers University of Technology
Göteborg, Sweden 2017

Virtual commissioning of production process
Sara S. Winther

Examiner: Petter Falkman
Department of Signals and Systems
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
Telephone +46 (0)31-772 1000

Göteborg, Sweden 2017

Abstract

In modern industry, production needs to be adaptable to be able to compete. Small changes to a production cell can take time to implement, resulting in loss of profit. Issues include implementation of new PLC, which lacks accurate verification methods. Virtual commissioning is the verification of PLC using a production cell model. This enables faster ramp up times, and improves overall quality. The issues with virtual commissioning is the lack of integration. Reliable virtual commissioning methods use virtual models, which requires modeling skills, a software modeling environment, and detailed production cell information. It also requires an interface for the PLC to the virtual model, where reliable options are limited. There exists no formal method for virtual commissioning, which makes implementation for first time users awkward and troublesome.
This thesis explores the requirements for a virtual model in a virtual commissioning project, with a focus on the smart-components aside from the 3D modeling, where there is already plenty of information for other purposes. A case study virtual model and PLC are used to collect information and to verify the requirements for a virtual model, along with reviews of reports from other virtual commissioning related projects. This thesis also proposes further potential for a virtual commissioning project and how it can be realizable, for instance by placing further demands on suppliers regarding robot/component models. Demands could include simplified models, which takes little effort to generate and offer more versatile use.

Keywords: **Virtual commissioning, Process Simulate, Tecnomatrix, SIMBA, TIA Portal, PLC verification, SimulationUnit**

Contents

1
Introduction

At production companies, the procedure when new production equipment is required is a lengthy process where, in simplified steps, first hardware is specified, programmable logic controller (PLC) code is written, and then everything is assembled and tested. The testing period may end up being very long, as this is where issues with the communication interfaces in the PLC/hardware combinations are discovered. To shorten this process, virtual commissioning can be used, in which when the PLC is written, it can be immediately tested on emulated hardware. This allows issues to be discovered before assembly, and enables methods to faster and easier identify trouble components. Ultimately this results in a shorter testing period [1] and a more reliable final product. The issues with implementing virtual commissioning today lies with the lack of integration [2]. Virtual commissioning requires a virtual model which, if not designed with virtual commissioning in mind, may end up requiring considerable redesign [3, 4]. By using methods where the virtual model is designed for virtual commissioning, accurate simulations can be made, and the reliability of the final product increases significantly [2, 5]. Primarily virtual commissioning enables tests with realistic time delays and intercommunication between hardware, whereas a software only model requires data to be gathered from relevant components to model time delays, and assumes that hardware components and communication will simply work. Virtual commissioning may also be useful for optimization aspects. With a sufficiently thorough physics engine in the virtual environment, energy efficiency can be measured [6, 7], and plant layout can be more accurately represented and optimized.

1.1 Background

Virtual commissioning has existed since the early 90's [8], and has been a growing concept since. Many industries are starting to integrate it more into their production processes, as its benefits result in less development downtime, shorter ramp-up time and therefore more production time. When new production equipment is required, the process starts with mechanical and electrical specifications. Once there is an idea of what the new components will be, the PLC program is written, and the components are bought or constructed, followed by assembly. Once everything is assembled, the PLC can be verified [9]. This is where issues with the PLC program are discovered, and often these issues relate to the mechanical and electrical components. Additionally, issues with a new PLC program may also be discovered upon installation into an existing production line, resulting in production downtime during troubleshooting. With virtual commissioning, significant time loss can be avoided. Virtual commissioning enables tests on emulated hardware, meaning that once the hardware has been specified, the PLC program can be almost immediately written and tested. The benefits are considerable, and with more thorough and detailed models, virtual commissioning could be used to run more tests on a production cell, without taking down or otherwise affecting production.

1.2 Virtual Commissioning

The steps required to perform virtual commissioning include the modeling of the virtual plant. To model a plant or a production line, data needs to be gathered depending on the required detail of the model. Measurements are needed for the 3D modeling of the components, which may be simplified according to the needs and the tests [9]. In a physics based model, details on moving surfaces may be necessary to model for accurate collision detection. The exact position of sensors needs to be known to recognize when and where detection happens. Additional data such as material and/or weights may also be required to calculate loads, for modeling and optimization with regard for energy efficiency. Sequence of operations and cycle times needs to be gathered as well, if only to provide model validation data. This should yield a complete virtual model.
When the PLC is written, an I/O signals list can be defined and imported to the virtual model. Finally, using a hardware emulation platform, the PLC can be tested and validated by running simulations in the virtual model.

1.3 Report structure

The report is divided into the following chapters.

Chapter 2 defines the problem and the objective. The problem is defined as a set of research questions, and a list of constraints is provided for potential approaches to the answers.
Chapter 3 presents the approach to the problem, in the form of a literature review and a case study, which is described more in the subsequent chapter.
Chapter 4 describes the requirements for the modeling process, to provide basis for what rework may need to be done to the case study virtual model.
Chapter 5 describes how to setup all parts provided by the constraints for this virtual commissioning project, and covers some potential errors in the setup.
Chapter 6 showcases a case study virtual model, and describes some immediate issues with the model in its initial state.
Chapter 7 briefly describes the signal mapping macro which was written for the case study.
Chapter 8 covers the verification of the setup and the mapping written about in chapters 5 and 7.
Chapter 9 summarizes the required rework for the case study virtual model, and proposes future steps to expand upon virtual commissioning. This chapter also answers the research questions.
Chapter 10 contains the final discussion and conclusion, as well as a sustainability and ethical discussion, as required by the department of education.

2
Problem

For many production companies, virtual commissioning has yet to be implemented. Issues lie with complexity and lacking familiarity, and by developing a method to perform virtual commissioning it may be implemented on a wider scale.

2.1 Objectives and purpose

The objectives are as follows:

- To create a specification for a virtual model for a virtual commissioning project.
- To create and confirm a straight-forward on-desk approach to PLC validation given a virtual model, a PLC, and a hardware interface.
- To verify HMI before installation.
- To provide examples for further potential.

By being able to provide the appropriate specifications for a virtual model, a supplier could provide a virtual commissioning model ready to be plugged in and tested upon delivery. With an on-desk approach to PLC validation, companies should be able to make their own minor modifications to an existing virtual commissioning project. By being able to verify a HMI before installing it in the production cell, excessive troubleshooting times can be avoided. Additional objectives includes to gain insight into if and how virtual commissioning can be used for a more sustainable development of production lines and for further optimization of existing production lines.

2.2 Constraints

Virtual commissioning requires a software platform for its virtual environment, and a hardware or software platform for its emulated hardware. A hardware platform is used for this report, as software solutions have proven unreliable during some short, initial tests. The hardware platform is called SIMBA, and the virtual model will be simulated in Process Simulate, for which a case study model has been provided. Some segments of this report will therefore be case specific for Process Simulate.

2.2.1 Process Simulate

Process Simulate is a production line simulation environment by Siemens. A 3D plant can be modeled and simulate regular behavior, as it enables sequencing of operations, and it incorporates kinematic 3D simulation, which enables collision detection. Process Simulate may be a sufficient platform to estimate energy consumption and maintenance requirements.

2.2.2 SIMBA

SIMBA box is a hardware interface which can emulate multiple hardware platforms, which enables an accurate testing ground for PLC. Its software supports fault injection and white noise generation on analogue inputs, which enables a close to reality simulation. Given its size and relative simplicity, this is a more flexible solution than common hardware-in-the-loop (HIL) solutions, where usually entire shelves are packed with PLC related hardware which require extensive reconfiguration for different tests or upgrades.

2.3 Research questions

For this report, six primary questions have been of interest, which can be divided into two parts; practical and exploration. The practical part seeks to answer how to perform virtual commissioning with the constraints given in section 2.2. With no prior knowledge or experience with virtual commissioning, the research questions becomes the following:

1. What methods for virtual commissioning exist and how do they compare?
2. How can verification of a model be performed?
3. Are there potential errors and how can these be avoided?

The exploration part seeks to answer if there is further potential for a standard virtual commissioning project, and if this potential is implementable.

1. What types of tests should a virtual commissioning model be able to perform?
2. Can a virtual commissioning model be accurate enough to obtain plant data directly from the model?
    - Can this be performed within reasonable time limits?
3. Can virtual commissioning be expanded to perform various tests, such as maintenance or optimization?

By answering the above questions, this report should provide a sufficient guide to implement more virtual commissioning, and provide reasons why virtual commissioning should be integrated further.

3
Approach

The final objective is to propose a guideline for virtual commissioning, given the aforementioned components, as well as provide arguments for why virtual commissioning should be employed to a greater extent. To reach this objective, Process Simulate and SIMBA have been studied and tested on a finished virtual model and its corresponding PLC. Further there has been a literature review to find existing methods and to provide an idea of the capabilities and limitations of virtual commissioning.

3.1 Literature review: methods

On the development of a virtual commissioning project there are articles such as [5], which proposes a concurrent design method for a virtual commissioning project. It opens with a description of conventional design procedures of a production system, wherein the design process is divided into three primary stages, starting with process planning, then mechanical design, then electrical design. The PLC can not be tested until the final stage, given the requirement for actual hardware to run on. The article follows with a suggestion for change in the procedure with the introduction of virtual commissioning. With virtual commissioning, the mechanical and electrical design stages can run in parallel, and the PLC can be tested as soon as there exists a consensus on the kinematics and the behavior of the device.
Reports such as [9–11] cover the development of a virtual commissioning project in closer detail, wherein all reports list the same data requirements and propose a similar working order. The reports suggest first identifying the system, such as purpose, sequence of operations, the mapping of the connections between signals and PLC, resources and geometries, layout, kinematics, and material flow. The next proposed step is to design the virtual model, and to simplify resources and components where possible, followed by signal generation and defining the cell behavior. After these steps, the virtual model is ready to be tested, and then used to verify the PLC.
For faster and smoother simulations, the virtual model should be simplified, and features of interest as suggested by Hoffman et al. [12] includes the removal of irrelevant features, avoiding complex textures, and filtering out hidden or invisible parts. Further, Strahilov and Damrath [13] adds that static objects can be merged into single units.
Additional reports of interest include ones on simplified modeling strategies. With the use of discrete event systems (DEVS), a simplified virtual model can be generated. Regular atomic DEVS models are defined as a 7-tuple, however Ko and Park [4] propose a 4-tuple consisting of an input event set, an output event set, a task set, and a function set which describes succeeding tasks from any given task. A visual representation can be constructed in any given graphical environment, enabling the user to follow the event flow, but excluding physics such as motion constraints and collision detection. For PLC verification purposes, this type of model can be sufficient. For smaller companies that lack the competence required to construct 3D models, this may provide a means to validate PLC without investing in expensive licenses for software, and without spending significant time on training staff.
A model needs to be tested, and reports such as [14] suggests automatic test generation for

model validation using Petri nets. A specification model is generated as a state machine, which is mapped to an extended Petri net, from which test cases are generated. With a reliable way to automatically generate test cases, the verification of a virtual commissioning model can be sped up, as manually generating states and observing a virtual run can be time consuming.

3.2 Literature review: potential

In an article on energy efficiency and virtual commissioning [6], it is proposed that with a sufficiently realistic virtual model, energy consumption can be estimated. Subsequently, adding more details to the model may enable more information to be collected from simulations, though this might extend modeling time. Strahilov and Damrath [13] proposes a pneumatic drive model with a simplified visual component for simulation speeds, with more inclusive physics to cover some otherwise neglected forces. This yields a more accurate representation of a pneumatic component's actual energy consumption, which could be applied to a virtual commissioning model. The concept of including energy consumption modeling in system design is expanded upon in [15].
Meike [16] has through experimental models shown that energy losses in robots are mainly caused by acceleration and deceleration, and with a general velocity reduction, energy consumption is reduced. Virtual commissioning provides an environment where the motions of each robot can be optimized to minimize acceleration/deceleration, without necessarily missing deadlines.
A simplified simulation model can be applied to test maintenance strategies, such as including priority in repair orders, redistributing operator responsibilities, or distributing the workload. Gopalakrishnan et al. [17] does this by simulating production flow and comparing unit output. Using a similar model, Karlsson et al. [18] evaluates different key performance indicators (KPI) for a cell, and uses it to quantify how preventive maintenance affects the need for corrective maintenance.
For more detailed models, there are many reports on the use of virtual reality to simulate a production cell ([19–21] to cite a few), which may be useful for both production and maintenance staff, as it enables the user to enter and observe a 3D production cell without being present in the factory.

3.3 Case study: editing existing robot cell

A case study in the form of an existing robot cell has been provided by Volvo Cars Corporation, in which a virtual model has been commissioned with only one specification, to be able to run cyclic event evaluator (CEE) simulations in Process Simulate. The model has not been verified, and some conventions have been vague, leaving enough room for interpretation and thus some unexpected variations. The virtual model has a corresponding PLC which has already been in use and is already verified to be in working order. This offers a means to verify the model and signal mapping. The model is described more in section 6.

4
Virtual model

This chapter covers the requirements to construct the virtual model for a virtual commissioning project, and provides insight into what to look for in the case study model described in chapter 6.

4.1 Modeling

The virtual model needs to mimic specific behaviors of the real plant, depending on purpose. For PLC verification purposes, an automaton representation of the plant can be sufficient to confirm simple behaviors. For more complex tests such as time and path optimization, the model may need to be more complex. In a more detailed model, physics engines are commonly used to detect collision and friction. To create a realistic model in a physics based 3D environment, detailed information about the resources is required, specifically geometries and kinematics, along with range of motion. The layout of the cell, which, and how many resources, and relevant components needs to be known. The purpose of each component is required as well, in addition to a normal sequence of operations, dependencies, and material flow. A list of expected signals for each resource is also required. With this information, a model can be constructed. In a modeling environment such as Process Simulate, if the kinematics are properly defined, signals can be automatically generated to communicate with a PLC.

4.1.1 Simplified model

For larger production cells, significant processing power can be required to simulate multiple resources and their kinematics. To speed up simulation times and to make the flow more smooth, the model should be simplified. Many suppliers provide a virtual model of their products. The virtual model is usually very detailed, including some hidden components which are never exposed unless the product is opened up. Components may also include labels with product information or company names, such as the labels seen in figure 4.1.

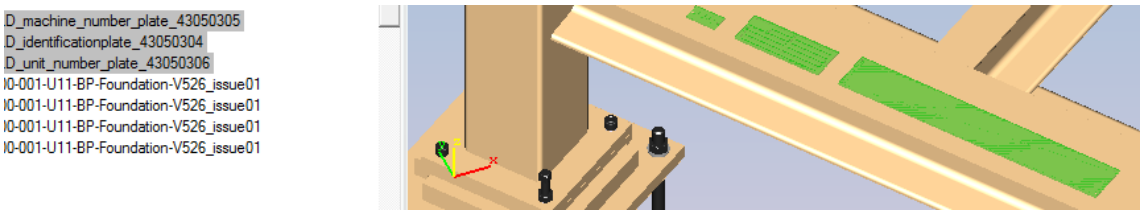

Figure 4.1: A buffer with three labels highlighted. They appear as green in the model. In a cell containing many components with unnecessary details, simulation times are negatively affected.

Any surface which is not at risk for collision, or is static, can be reduced. Static components can be merged into single units, and detailed surfaces can be stripped of complex geometries and textures.

5
Station setup

This section covers the setup of a virtual commissioning project in Process Simulate, using SIMBA box and TIA Portal as the PLC interface. A more detailed guide is provided in appendix A.

5.1 Connection to SIMBA

The connection to the SIMBA box is done via the SIMULATIONUnit program. If the project has not already been created, a new project can be created by selecting a project folder destination and importing hardware via the configuration menu. The SIMBA-box can be added by scanning the network. Once the SIMBA has been connected, the hardware needs to be configured by adding symbol- and .OMS or .dat files. The symbol file is generated by exporting the PLC tags in TIA Portal. The .dat and/or .OMS files are automatically generated by TIA Portal when the project is compiled.
Once the SIMBA is online, the next step is to close the SIMULATIONUnit program entirely. If the program is not closed, it may cause conflicts when the TIA Portal tries to access the PLC via the SIMBA box. What exactly happens during this conflict has not been explored, however during some quick checks to confirm that conflict actually occurs, the TIA Portal has slowed down dramatically, calling different errors regarding hardware connections, and overall making an otherwise smooth process tedious. To restore a smooth flow without errors, resetting the SIMBA and restarting TIA Portal has been necessary.

5.2 TIA portal

The PLC program can be observed live via TIA Portal, by opening the project and going online to the connected PLC. This can only be done when the SIMBA has been connected, or the portal will not find the PLC. SIMULATIONUnit needs to be closed after establishing the connection to avoid conflict. If the PLC is empty or contains an old PLC program, it will need to be loaded.
From the portal, by navigating to a desired PLC sequence, the process can be monitored via the ”Monitoring on/off” option. This enables the user to follow the program step-by-step during execution. This may also be useful for troubleshooting. There is also a cross-reference function, which shows where a PLC tag is used, which enables an easy way to backtrack a signal to find its origin and what might affect its state. The PLC can be set to start/stop via TIA Portal, but there have been some instances where the PLC crashes when this is done. Why has not been identified, however it is easily fixed by simply resetting the device and re-establishing the SIMBA connection.

5.3 Process Simulate

To run and observe a simulation with the PLC, Process Simulate needs to be set to simulate via PLC. In the signal viewer, a complete signals list of currently existing signals

can be seen. Their states can be monitored by selecting the signal and adding them to the simulation monitoring window.

5.3.1 Signal mapping

Process Simulate and TIA Portal provide no tools to simplify signal mapping beyond the possibility to export and import formatted excel files in which the mapping has already been performed. Given a project with hundreds of signals, this quickly becomes cumbersome manual labor. By using naming conventions, this process can be automated. One suggestion is to export a complete component list from Process Simulate, and using PLC naming conventions, convert all signal names to their corresponding PLC tags. Then the new signal list from Process Simulate can be matched with a PLC tags list from TIA portal, including addresses. The matched signal list can then be formatted so that it can be read by Process Simulate's connection mapping tool, which sets the addresses and resource connections of signals included in the excel file. A macro which performs these steps is described in section 7.

5.4 Potential issues

The PLC may be expecting signals from external PLCs. This causes no conflict, however signals representing the external PLCs can not be mapped. There are several approaches to how to manage these signals, such as manually finding all blocks containing calls to an external PLC and manually setting these signals to their expected true/false states, or deleting all calls to external PLCs entirely. In some places, the PLC is expecting a signal from an external resource, a resource not included in the specified cell. These can be treated in a similar fashion, where either the call is deleted or the signal is manually modified. While deletion is faster, manual modification may be necessary in some instances where for example the call references a product number.

6
Case study: robot cell

An existing model has been provided for the purpose of identifying what rework needs to be done for the model to be ready for a virtual commissioning project. The only specification which was provided for the model builders was that the model should be able to run CEE simulations in Process Simulate. By looking closer at the contents of the virtual model and some of the expected behavior according to the PLC, it should be possible to provide a more complete specification for future projects.

6.1 Resources

The model consists of seven robots, seven fixtures, a number of buffers, and two turntables, seen in figures 6.1, 6.2, and 6.3.

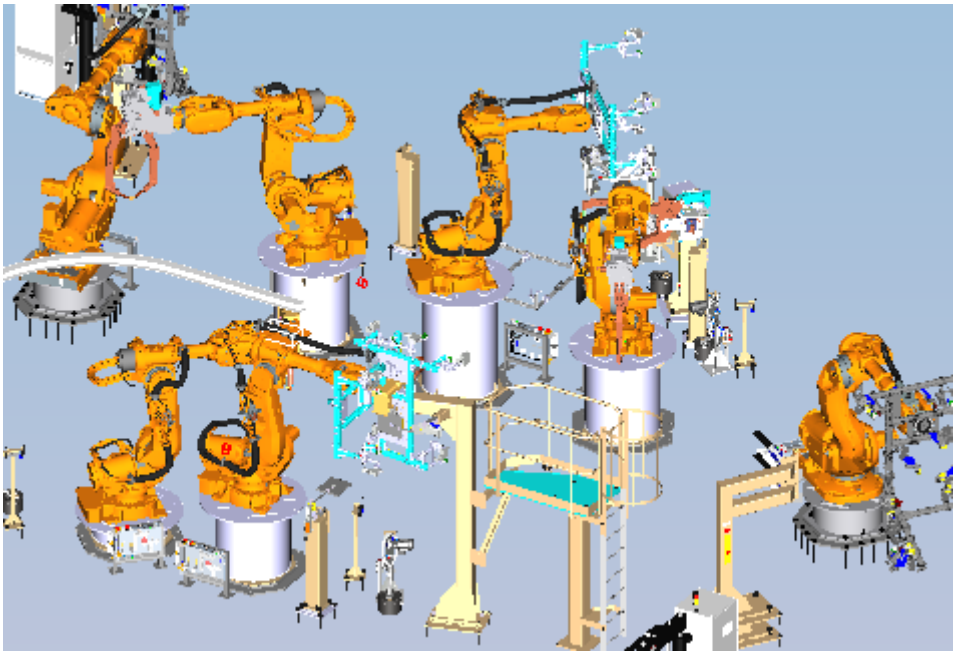

Figure 6.1: The seven robots in the provided robot cell.

Some of the robots contain defined poses and signals. The pose names contain clues about their purpose, such as the name of a material type which is handled by the cell, however which part specifically is undefined, as well as the robot's expected behavior at each position. The signals consists for most part of the signals which are automatically generated via Process Simulate functions, however some have been manually created, but do not appear to be linked to expected PLC signals in any way.

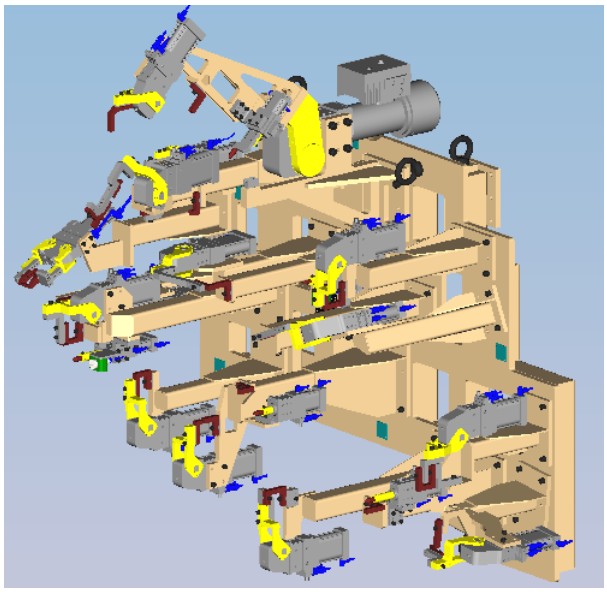

Figure 6.2: Sample fixture from the provided robot cell.

The fixtures contain a number of clamps, pins, and swings. Some components do not contain defined poses, and many components contain multiple units, such as the swings which always have a subordinated pin and/or clamp. The clamps, swings, and pins should have binary open/close operations, and the swings should have additional signals for special settings and error detection. For all components, the logic blocks need to be defined.

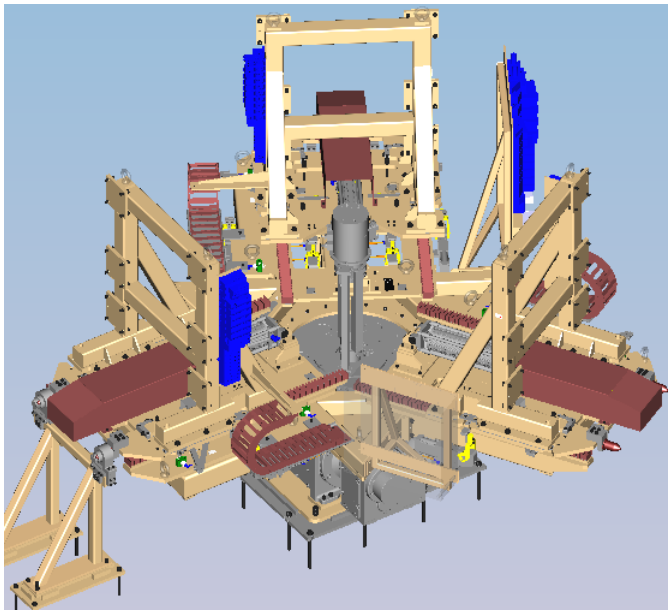

Figure 6.3: One of the two turntables from the provided robot cell.

The two turntables appear visually identical, however they have been defined very differently. They consist of a rotary table, twelve clamps, and three slides. In the PLC tags list, it appears that the turntables are expected to have a signal for a 13th clamp, however no such clamp exists in the model. Instead ghost logic will be used in its place. In Process Simulate, logic blocks are created for each component, and for the logic to apply to visual components, subordinated units have their logic in the same component.

For one of the turntables, the clamps have been subordinated to the slides, putting four clamps in each slide, resulting in the slide containing logic for itself and the clamps. For the other turntable, the clamps are subordinated to the rotary table. With twelve clamps and 19 expected signals for the rotary table, this results in a very long signal list for a single component. The logic is created manually, and with a big cluster of entries and exit signals in a single component, there is significant room for error during logic creation.

6.1.1 Resource names

The case study follows a naming convention which uses document numbers and a title. The title convention is simply ”description of the model, separated with underscores (_)”. This has been interpreted differently and resulted in a model with components having very different names yielding very little or sometimes incorrect information, such as descriptions stating that the component is a clamp, when it is a pin. With no reliable indicator in the name regarding the true contents of components in the component list, all components are renamed to follow a different naming convention. In the new naming convention, information about station number, unit type, and group number can be obtained from the component name. From this information, the signal/s and PLC tag/s for the unit/s can be generated. If all components have properly defined kinematics in the model, all signals can be automatically generated and matched externally, using the Connection Mapping tool. A macro has been created for this purpose, and is described more in section 7.

6.2 Operations

The model contains some predefined operations, however they appear to be missing some information in order to run. The operations are run on continuous time, and have fixed orders for corresponding tasks. The resources do not need to send or receive signals to operate, and any implemented signals can not be controlled externally without manually forcing new states. Operations can be used to verify offline model behavior, but serve no purpose for the verification of the PLC.

6.3 Sensors

There are no part sensors defined in the model, which needs to be created. By using descriptions given in the PLC tags list, the sensors can be matched to their corresponding parts. Many components have a visual object representing the sensor, however the object has been defined in such a way that the entire component needs to be used to act as the visual representation of the sensor. This will affect when detection happens, as detection will happen when the part is within a certain distance from the component, as opposed to when it is a certain distance from the sensor itself.

7
Automatic signal mapping

For a production cell with many components and robots, the number of required signals becomes very long. In the provided case study, the PLC tag list contains over 2000 signals. To connect the relevant signals to the Process Simulate model, a macro has been written to construct signals assuming standardized names and properties. The macro creates a signal list which follows a format expected by the Connection Mapping function in Process Simulate, allowing import of a signals list which connects each signal to a resource and logic block.

7.1 Requirements

The macro needs a component list with standardized component names. The virtual model needs components to use standardized kinematics and logic names, or the connection mapping function will not be able to map the signal.

7.2 Proposed structure

The PLC tags in the case study follows a naming convention, which for components is given by a station number, group number, and then some specific numbers/letters, depending on unit type. For other resources, such as robots or shared zones, the convention is slightly different, but with sufficient insight into the expected signals from each resource, a set of standardized signals can be reconstructed. The flow of the macro is simple, and is presented in pseudo code below. For each component in the component list, the macro extracts station number (used for the PLC tag prefix), component type (determines which signals are created), and component group (used for the PLC tag group). With this information, signals can be created and a PLC tag can be reconstructed. The reconstructed PLC tag is then used to find the signal address in the PLC tag list.

for each component in list **do**
    % Extract component information using naming convention
    % Here, the convention contains station number, component type, and PLC tag group, separated by a delimiter
    Split component by delimiter into array
    plcPrefix = component(0)
    unitType = component(1)
    unitGroup = component(2)
    if unitType = TYPE **then**
        Create signals for TYPE
        Set plctag = plcPrefix & unitGroup & TYPESPECIFIC
    else if unitType = OTHER TYPE **then**
        Create signals for OTHER TYPE
        Set plctag = plcPrefix & unitGroup & TYPESPECIFIC
    else if unitType = ... **then**
        ...

end if
Set plcAddress = find(plctag in PLC tag list)
Print component, signals, plcAddress to excel document
end for

In the actual macro, there is plenty of room for rework. Speed or memory management has not been regarded, as the macro has been constructed primarily to enable testing. The final macro can be seen in appendix B.

8
Verifying setup and mapping

Once the provided model for the case study has been modified and the signals have been mapped, the PLC is connected to perform a test run. The PLC is unfamiliar, but with some manual exploration and standardized, descriptive names of function blocks and signals, a trial run can be performed.

8.1 Trial run on PLC

Many signals from the PLC tags list have an easily identified function and are mapped without any issues. To identify the remaining signals, the sequence blocks in the PLC have been observed, and TIA Portal's cross-referencing functions have been used to find the location of the signal. Signals contain descriptive comments regarding its purpose, however often the comments provide no information about which resource it comes from, or when/why. By finding where the signal is used, it has been possible to further identify the purpose of the signal. A list has been compiled with the missing signals, which are mapped for subsequent tests.

8.2 External safety I/O

For subsequent tests, ghost logic has been created for unidentified signals, and a set of signals marked "Safety I/O" have been mapped. The safety I/O signals relate to raising errors, such as when safety zones are violated, or when the emergency stop is activated. Many signals relate to feedback on the status of the safety systems, and these need to be true in order for other systems to be able to run at all. Some of the feedback signals are connect to external PLCs and can not have their states changed internally. There are two approaches to managing these signals, either set all these signals manually with constant true/false variables, or deleting calls to external PLCs entirely. Removing blocks entirely is significantly faster, however for verification purposes it may be preferable to set signals manually.

8.3 HMI

The PLC has a set of HMI screens, which have been tested on a HMI connected to the computer. When signals have their states changed in Process Simulate, the HMI updates accordingly. The setup is straight forward, requiring no further action beyond mapping the signals and uploading the specific PLC to the HMI.

8.4 Conclusion tests

The entire range of the PLC has not been tested, as it contains more function blocks than can be tested in the allocated time. Without knowing the expected logic and relations for each signal, many signals need to be updated manually in Process Simulate. However,

the PLC and Process Simulate are able to communicate, and with a more complete model and insight into the PLC, the process should run smoothly. Both PLC and HMI can be successfully validated using Process Simulate and a SIMBA box interface.

9
Results

The final results are gathered in this section, which ends with a summary of the answers to the research questions.

9.1 Model specifications for virtual commissioning

A virtual model has a set of standard specifications for the visual aspects, including geometries, kinematics, dependencies, and layout. Ghost logic can be implemented for invisible resources, for example a resource outside the production cell which shares another resource inside the production cell. Independent resources should be modeled to enable independent logic. Logic can be implemented to dependent resources to mimic independent control, however for units with many states or many dependencies, more lines of additional logic is required. Unless the logic generation is automated, the implementation becomes prone to human error.
A PLC uses a different set of signals for different types of resources, so for every resource, there needs to be a complete list of expected signals. This includes safety signals, such as safety zone violation and alarm generation. To automate the signal mapping process, the logic block ports should use standardized names.
In Process Simulate, sensors do not use logic blocks. Sensors are defined as their own resources, and automatically generate a connected signal upon creation. For automatic mapping to the PLC, a different function needs to be used than the one used for resource mapping. This requires that the sensor signal already exists, thus sensors should use standardized names for seamless mapping.
The model needs to be ready for use before the construction or rework of the real cell begins, otherwise the purpose of virtual commissioning is defeated. This puts a time constraint on the creation of a virtual model for a virtual commissioning project.

9.2 Suggestions for faster simulations

Simplified models enable faster simulations, which enables a wider range of test cases. For production cells with multiple resources, every detail which can be removed or reduced speeds up simulation times.
One proposed solution is to request that the component designer constructs two models for the same product, one detailed and one simplified. The simplified model should be stripped of excessive details and use simplified geometry. This puts more effort on the supplier.
Another proposed solution is based on the assumption that suppliers use naming conventions for their CAD models and that these names can then be accessed in the model. By identifying how component parts are named, the model can be edited with a script to hide or remove parts with names such as ”identity plate”, ”label”, and keep parts with names such as ”base frame”, or ”detection unit”. This puts more effort on the client.

9.3 Energy consumption

Many organizations put an ever increasing demand for sustainable solutions. Energy consumption models are currently being developed and may at the time of this report not exist for all types of components in a production cell. By requesting that suppliers include energy consumption modeling in their CAD models, the research and development of accurate energy consumption models may be sped up. Virtual commissioning can then be performed with regard for energy efficiency.

9.4 Summary answers to research questions

To answer the questions posed in chapter 2, interviews have been conducted with staff at Volvo Cars Corporation, who are currently looking into integrating virtual commissioning into their work process. With additional support from a case study and literature, the answers have been summarized in this section.

9.4.1 Practical

1. What methods for virtual commissioning exist and how do they compare? There exists no formal method for virtual commissioning, however most projects follow the same work order, by starting with gathering information about the robot cell and constructing the virtual model with this information, described more in chapter 4. The PLC can be written in parallel with the modeling process, and then the model and the PLC signals can be mapped. Knowledge of the production cell's sequence of operations, and expected signals in the PLC are especially required during the verification process.

2. How can verification of a model be performed? A model can be verified offline if the simulation environment supports offline tests, by creating operations and comparing the model's behavior with the real production cell. In particular, the kinematics of each component should be observed, to see if it is able to move to expected poses, and to see if connected signals respond accordingly. Ghost logic can be implemented if a component is not behaving as expected (or is missing), however this removes the possibility to observe the behavior during simulation. Further verification can be performed when connecting to the PLC, as the model should behave the same way online as offline, thus enabling PLC verification.

3. Are there potential errors and how can these be avoided? Potential errors depend on simulation environment and human factor. Programs such as Process Simulate offers many functions which automatically generate standard resource logic, signals, operations, etc., which reduces the risk for errors. However, there are no automatic functions for generating non-standard resource logic and signals, or to enable seamless signal mapping, and this is where human factor causes the errors. It is necessary to put conventions in place and to follow these to reduce the risk for error, and to enable external automation of processes such as signal mapping.

9.4.2 Exploration

1. What types of tests should a virtual commissioning model be able to perform? Interviews have been conducted with PLC staff at Volvo Cars Corporation, who

have expressed a desire to use virtual commissioning projects as training material, for instance to observe what happens when certain alarms are triggered. There is also a desire to be able to obtain collision zones and cycle times from the model. This should be possible to obtain from a virtual commissioning project as it is, assuming it has been properly designed, and that the simulation environment supports collision detection. Further, there is an interest in being able to enter a 3D virtual model (virtual reality) to look more closely at detailed components. With a sufficiently detailed virtual model, and an interface to enable virtual reality, this is possible.

2. Can a virtual commissioning model be accurate enough to obtain plant data directly from the model? Can this be performed within reasonable time limits? The accuracy of data which can be obtained from a virtual model depends on the model and the desired data. To obtain cycle times and collision zones, the model can be nearly accurate enough without requiring any additional rework efforts, in particular during tests with the PLC, from which time data can be obtained to provide rough estimates. Further, for energy consumption, the model may need to be appended. CAD models of resources are often provided by suppliers, and by demanding that suppliers include energy consumption modeling, energy efficiency can be measured.

3. Can virtual commissioning be expanded to perform various tests, such as maintenance or optimization? A model connected to a PLC can be used to check collision zones and to verify safety signals and alarm generation. The virtual environment can also be used to minimize robot movement to reduce energy consumption, and can be included to test different maintenance strategies. Further it can be used to observe cell behavior when single units are down. During PLC verification, test cases are given different priorities, depending on how they affect real life production and safety. If the virtual commissioning project includes a simplified model, it may be used to test cases with lower priority, which may otherwise be skipped to save time.

10
Discussion and conclusion

The provided case study has been central, and a lot of time has been put into untangling a mess of information, due to the case study being developed by different teams at different times with vague specifications. Given the nature of the provided case study, the report does not contain any PLC development efforts, however the development of a PLC is relatively effortless with the use of DEVS software. The objectives can be summarized into exploring if on-desk PLC verification is possible, and providing a concrete example of how to perform it. This report provides a suggested item list to look for in a virtual model, which may me overlooked when only constructing a 3D model. This report also provides means to speed up the signal mapping process by automating signal generation and mapping, with the use of standardized names. The provided tools, Process Simulate and SIMBA box, require little previous knowledge to use with existing projects, and a simple step-by-step guide can be sufficient for model verification by inexperienced users. Regarding the exploration aspects of the report, no research has been conducted, instead a mixture of reviewing other reports and hands-on experience from working with the case study has been the basis for provided answers, supported by interviews with experienced PLC staff at Volvo Cars Corporation.

10.1 Sustainability and ethical aspects

There are multiple pros and cons with virtual commissioning, a duality which causes the points to merge and being difficult to divide. Some benefits include faster ramp up times of production cells, which reduces the strain on workers and potentially increases profits for the production company. The production company will either divide new tasks for their workers, or terminate now excessive workforce. The increase in profit and reduction of ramp up times could be used to explore more sustainable solutions for production processes, or to expand production, thus creating more employment opportunities. Virtual commissioning is another step in the automation of industry, ultimately resulting in a workload redistribution. There will be more employment opportunities for engineers with higher education, and fewer opportunities for workers in the factory. Virtual commissioning in its current state requires an initial investment which smaller companies may not be able to afford, providing an edge for already well-established companies to further out-compete and dominate the market.

Virtual commissioning comes with several sustainability properties, described more in section 3.2. If developed and integrated properly, commissioning time can be reduced by 75% [22], which ultimately results in a cheaper production. The reliability of the final product increases, meaning deadlines are more likely to be met, and reduced stress for workers as a consequence. With the use of thorough physics in the virtual model, mechanical energy consumption can be estimated, which allows design with environmental impact in mind. With collision detection, friction and wear estimates can be made, which may provide data for preemptive maintenance.

10.2 Further work

For further research, there are many possibilities. One in particular is the development of reliable software interfaces to verify PLC. One overlooked issue in this thesis regarding the chosen PLC verification platform is the required hardware and its costs. Purchasing one SIMBA box and PLC pair may be affordable to most companies, but to equip a team with all of the required hardware gets costly. Having a set of physical platforms also creates potential issues regarding availability. As mentioned in section 2.2, existing software solutions are lacking in reliability. Ideally, the PLC interface and the PLC itself could be run on a reliable software interface, and accessed on a shared network to enable access for an entire team.

Bibliography

[1] Lock-Jo Koo, Chang M. Park, Chang H. Lee, SangChul Park, and Gi-Nam Wang. Simulation framework for the verification of plc programs in automobile industries. *International Journal of Production Research*, 49(16):4925–4943, 2011.

[2] S. Seidel, U. Donath, and J. Haufe. Towards an integrated simulation and virtual commissioning environment for controls of material handling systems. pages 1–12. IEEE, 2012. ISBN 0891-7736.

[3] Henrik Carlsson. Reliable virtual commissioning, 2012.

[4] Minsuk Ko and Sang C. Park. Template-based modeling methodology of a virtual plant for virtual commissioning. *Concurrent Engineering*, 22(3):197–205, 2014.

[5] Minsuk Ko, Euikoog Ahn, and Sang C. Park. A concurrent design methodology of a production system for virtual commissioning. *Concurrent Engineering*, 21(2):129–140, 2013.

[6] Felix Damrath, Anton Strahilov, Thomas Bär, and Michael Vielhaber. Establishing energy efficiency as criterion for virtual commissioning of automated assembly systems. *Procedia CIRP*, 23:137–142, 2014.

[7] Daniel Wolff, Dennis Kulus, and Stefan Dreher. *Simulating Energy Consumption in Automotive Industries*, volume 9783642287770, chapter 4, pages 59–86. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012 edition, 2012. ISBN 9783642287763;364228776X;3642287778;9783642287770;.

[8] Patent issued for method and system for testing safety automation logic of a manufacturing cell. *Journal of Engineering*, page 7645, 2015.

[9] S. Makris, G. Michalos, and G. Chryssolouris. Virtual commissioning of an assembly cell with cooperating robots. *Advances in Decision Sciences*, 2012:1–11, 2012.

[10] Jasmin Dzinic and Charlie Yao. Simulation-based verification of plc programs, 2014.

[11] Luis V. Guerrero, Virgilio V. López, and Julián E. Mejía. Virtual commissioning with process simulation (tecnomatix). *Computer-Aided Design and Applications*, 11 (sup1):S11–S19, 2014.

[12] Peter Hoffman, Reimar Schumann, Talal M. A. Maksoud, and Giuliano C. Premier. Research on simplified modelling strategy for virtual commissioning. *Proceedings of the European Modeling and Simulation Symposium*, pages 293–302, 2012.

[13] Anton Strahilov and Felix Damrath. Simulation of the behavior of pneumatic drives for virtual commissioning of automated assembly systems. *Robotics and Computer-Integrated Manufacturing*, 36:101–108, 2015.

[14] Sebastian Sub, Stephan Magnus, Mario Thron, Holger Zipper, Ulrich Odefey, Victor Fassler, Anton Strahilov, Adam Klodowski, Thomas Bar, and Christian Diedrich. Test methodology for virtual commissioning based on behaviour simulation of production systems. pages 1–9. IEEE, 2016.

[15] Felix Damrath, Anton Strahilov, Thomas Bär, and Michael Vielhaber. Method for energy-efficient assembly system design within physics-based virtual engineering in the automotive industry. *Procedia CIRP*, 41:307–312, 2016.

[16] D. Meike. Increasing energy efficiency of robotized production systems in automobile manufacturing. *PhD Thesis. Rīga: [RTU]*, 214:1–32, 2013.

[17] Maheshwaran Gopalakrishnan, Anders Skoogh, and Christoph Laroque. Simulation-based planning of maintenance activities in the automotive industry. 2013.

[18] Nadine Karlsson, Camilla Lundgren, Maheshwaran Gopalakrishnan, and Anders Skoogh. Quantifying the effects of production maintenance decisions using discrete event simulation. 2014.

[19] LP Berg and JM Vance. Industry use of virtual reality in product design and manufacturing: a survey. *VIRTUAL REALITY*, 21(1):1–17, 2017;2016;.

[20] Hwa J. Yap, Zahari Taha, Siti ZawiahDawal, and Siow-Wee Chang. Virtual reality based support system for layout planning and programming of an industrial robotic work cell: e109692. *PLoS One*, 9(10), 2014.

[21] Christopher J. Turner, Windo Hutabarat, John Oyekan, and Ashutosh Tiwari. Discrete event simulation and virtual reality use in industry: New opportunities and future trends. *IEEE Transactions on Human-Machine Systems*, 46(6):882–894, 2016.

[22] Chi G. Lee and Sang C. Park. Survey on the virtual commissioning of manufacturing systems. *Journal of Computational Design and Engineering*, 1(3):213–222, 2014.

A
Step-by-step guide

While Process Simulate and TIA Portal cover many needs and offer multiple functions, reliability is slightly less guaranteed. Throughout testing and repeating all steps in the exact same order every time, once in a while something will cease to work, something might crash, an unexpected failure will occur, and with a simple restart, everything runs smoothly again. For all of the above steps, if everything is done correctly, it should work. If for some reason it does not work, repeating the steps again and/or restarting something might make things work again. The best solution to unnecessarily losing work is to save frequently and to remain patient.

A.1 SIMBA box

The SIMBA is connected to the PLC via port 1, and to the PC via port CTRL. The SIMBA has a reset button which can be used to reset the SIMBA if there is a glitch.

A.2 Simulation unit

To create a new project, Project -> New and choose a folder destination.

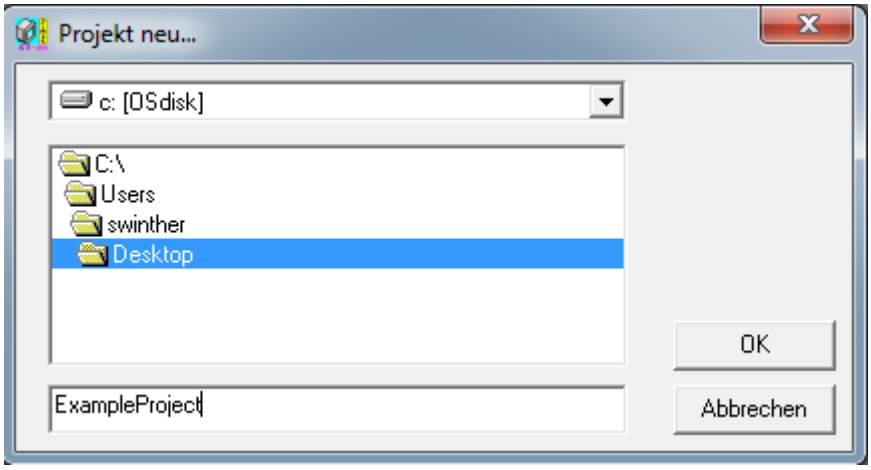

Figure A.1: Create new project

To connect the SIMBA, go to Configuration -> Import hardware.
From the import hardware window, scan the network for SIMBAS. Verify that the connection has been established by using the blink button. Once the SIMBA has been selected, add a symbol file, and an OMS file or system data block file. The symbol file can be exported from TIA Portal, described more in section A.3. The OMS and/or system data block files can be found in a folder called TIAExports when the PLC project has been compiled.

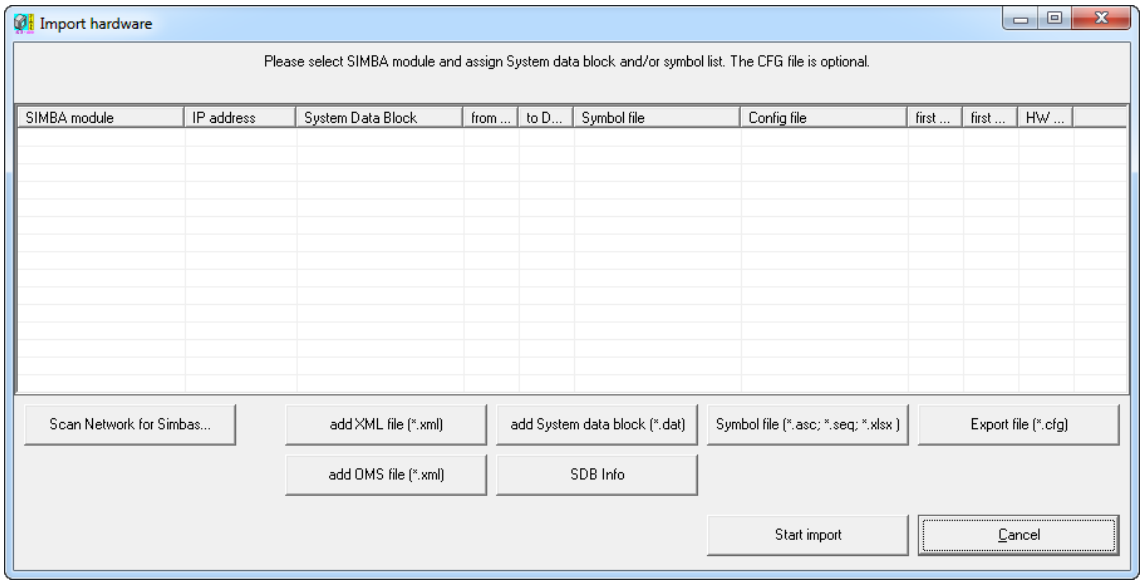

Figure A.2: Import hardware window

A.3 TIA Portal

Open the desired project. To compile the project, select Edit -> Compile. To load the project to the PLC, select Online -> Download to device. In the Download to device window, select "Show accessible devices" in the Select target device drop down menu, and Start search. Find your PLC, and verify the connection with the Flash LED button.

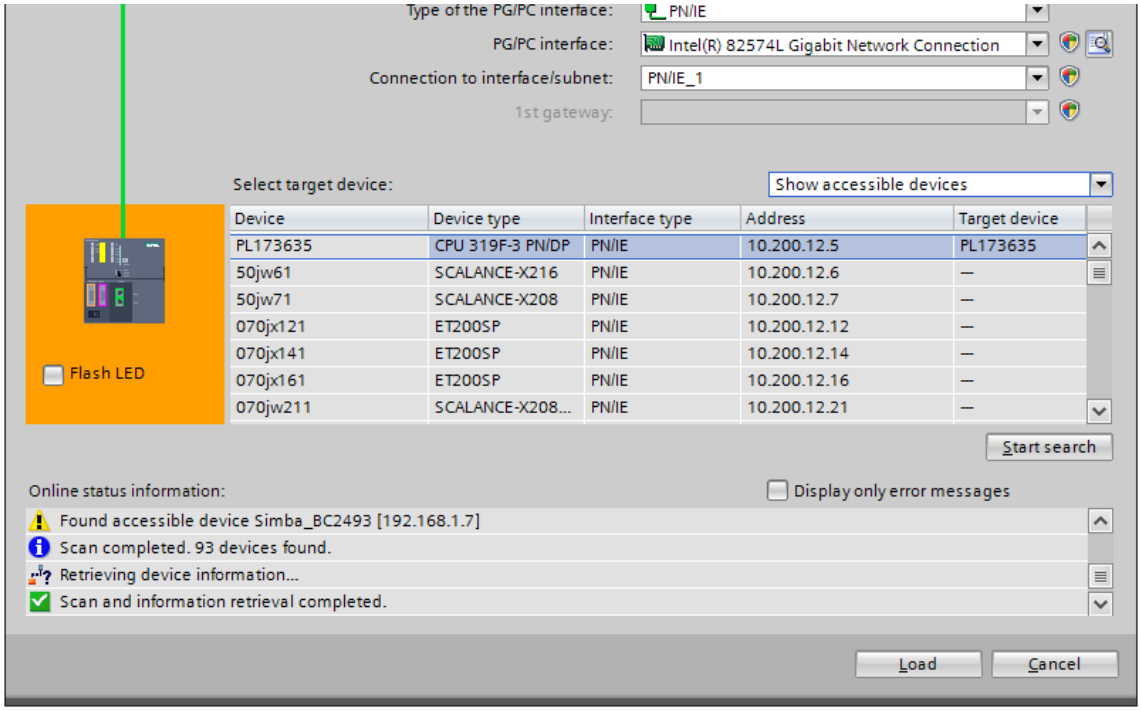

Figure A.3: Download to PLC

Once the project has been downloaded to the PLC, select Online -> Go Online. To monitor a sequence or a block, select Monitoring on/off. When monitoring is enabled,

manual or automatic mode can be selected. Automatic mode will execute the sequences automatically, manual mode requires that the user manually selects the next block in a sequence. To go to a specific block, manual mode is required. To go to a specific block, right-click it and choose Activate step.

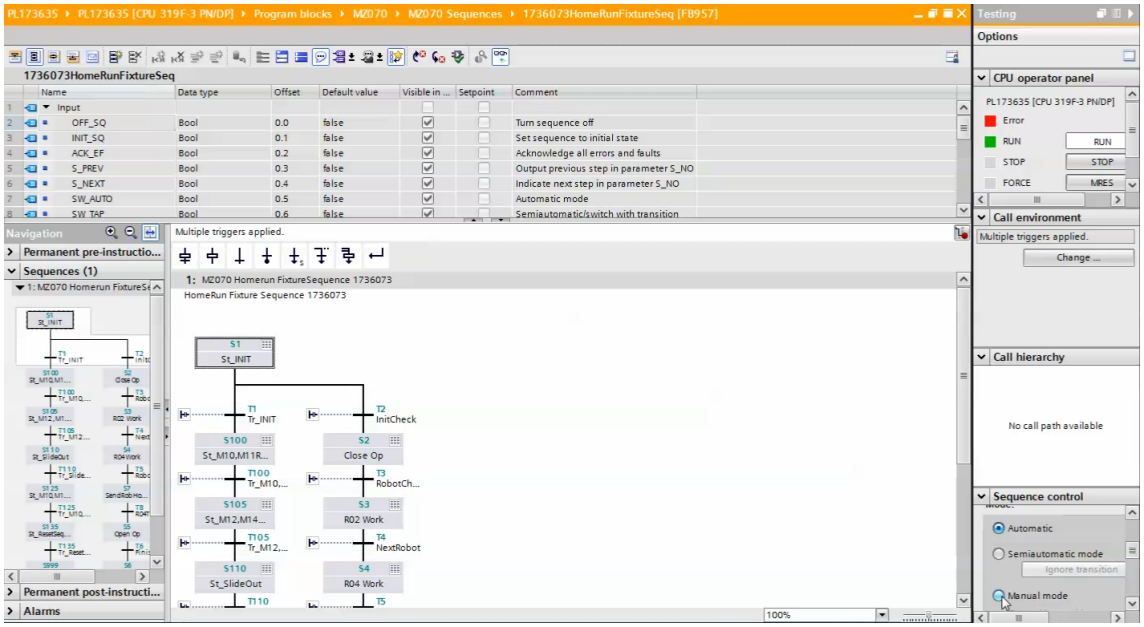

Figure A.4: Monitoring PLC execution in TIA Portal

If you are uncertain about the purpose of a signal, you can use cross-references to obtain more information. In the cross-references window there is a column "Access", which shows how the signal is used. The links can be used to open the block where the signal is used.

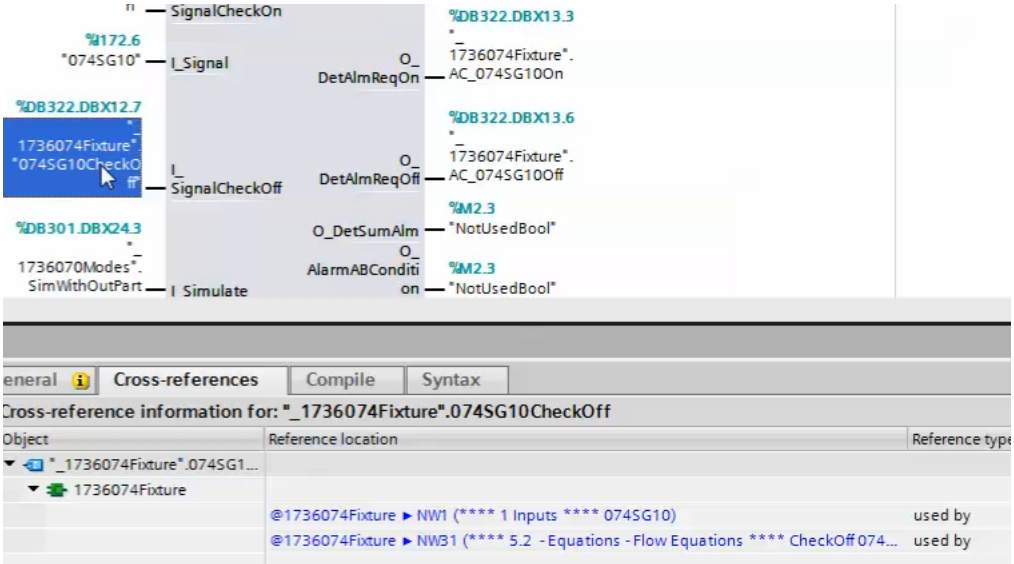

Figure A.5: Identifying signals with cross-references

Function blocks can be opened to reveal the input/output connections.
To use a HMI, download to the HMI by performing the same steps as above (Extended download to device), however choose the HMI among the accessible devices instead.

A.4 Process Simulate

To connect to the PLC via the SIMBA, go to Program Settings and select the path to the SIMBA project. To connect to the PLC, go to Options, find the PLC tab, and select PLC External Connection. Verify the connection by going to Connection Settings and selecting Validate.
Signal mapping can be done externally in excel. The information must be entered according to specifications to work with their corresponding functions.
To import signals mapped to resources, use Connection Mapping. This function creates signals and connects them to resources containing logic blocks.

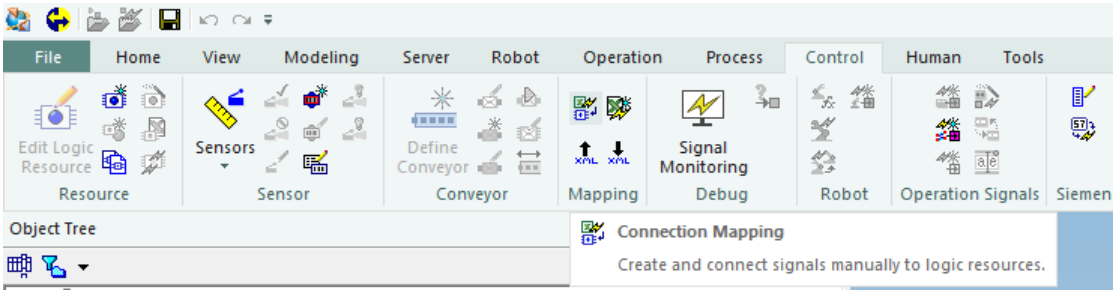

Figure A.6: Connection mapping

To map sensor signals, use Signal Mapping. Sensors do not use logic blocks and can not be mapped via Connection Mapping.

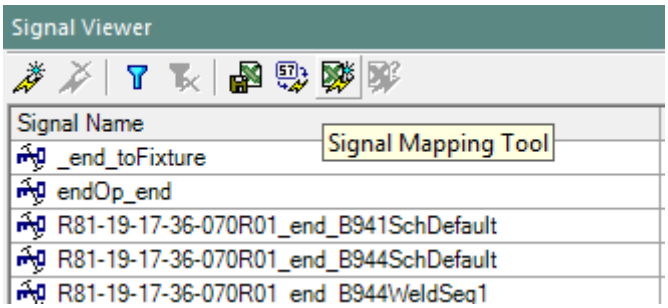

Figure A.7: Signal mapping tool

B
Mapping Macro

```
Private Sub CommandButton1_Click()
' variable definitions, if any variable is missing, add it here

' COLUMN INFORMATION
Dim tagColumn As String
Dim pinNameColumn As String
Dim sigNameColumn As String
Dim sigTypeColumn As String
Dim addrColumn As String
Dim extConColumn As String
Dim resourceColumn As String
Dim srcColumn As String

' OTHER VARIABLES
Dim iLastRow As Integer
Dim clampCtr As Integer
Dim pinCtr As Integer
Dim clampAppend As String
Dim pinAppend As String
Dim temp As String
Dim clampNbr As String
Dim nbrClamps As Integer
Dim k As Integer
Dim pinNbr As String
Dim nbrPins As Integer


' UNIT AND COMPONENT INFORMATION
Dim cName As String
Dim component As String
Dim units() As String
Dim unitGroup As String
Dim unitType As String
Dim plcPrefix As String

' SIGNAL ARRAYS
Dim signInfo As Variant
Dim swingStates As Variant
Dim swingSign As Variant
Dim swingTags As Variant
Dim swingPorts As Variant
Dim clampStates As Variant
Dim clampSign As Variant
Dim pinStates As Variant
Dim pinSign As Variant
Dim clampTags As Variant
Dim pinTags As Variant
Dim turntabStates As Variant
Dim turntabSign(23) As String
Dim turntabTags As Variant
Dim turntabPorts As Variant

' WORKSHEET INFORMATION
Dim srcSheet As String
Dim compList As Worksheet
Dim targetList As Worksheet
Dim compPath As String


On Error GoTo ErrHandler
' column allocation according to PS specification
```

```
resourceColumn = "A"
' B is currently free space
pinNameColumn = "C"
sigNameColumn = "D"
sigTypeColumn = "E"
addrColumn = "F"
' G is for comments. here it will be used to store the PLC tag
extConColumn = "H"


' prompt the user to select the component list file
With Application.FileDialog(msoFileDialogOpen)
    .Title = "Select␣the␣component␣list"
    .AllowMultiSelect = False
    ' set file types that are displayed in the browse window. can skip this and
        instead use error management if
    ' user selects an incompatible file
    .Filters.Clear
    .Filters.Add "", "*.xlsm"
    .Filters.Add "", "*.csv"
    .Filters.Add "", "*.xlsx"
    .Show
    ' if cancel, exit
    If .SelectedItems.Count = 0 Then
        Exit Sub
    ElseIf .SelectedItems.Count = 1 Then
        compPath = .SelectedItems(1)
    End If
End With


srcSheet = "Standard"
srcColumn = Application.InputBox(prompt:="Enter␣the␣column␣in␣which␣the␣components
    ␣are␣found␣(A␣by␣default).␣Cancel␣to␣select␣default.", Type:=1 + 2)
If srcColumn = vbNullString Then
    srcColumn = "A" ' component source column
End If
tagColumn = "G" ' plc tag column destination

' define target location
Set targetList = ActiveWorkbook.Worksheets(1)

' create new instance of excel, and make it invisible
Set xlApp = New Excel.Application
xlApp.Visible = False
' open your workbook in this new instance
On Error Resume Next
Set compList = xlApp.Workbooks.Open(compPath, ReadOnly:=True).Worksheets(srcSheet)
If Err.Number = 9 Then
    Err.Raise 9, "ReadWorksheet", _
        "The␣sheet␣name␣did␣not␣meet␣expectation.␣The␣component␣list␣should␣have␣a
            ␣sheet␣named␣Standard."
    compList.Parent.Close
    xlApp.Quit
    GoTo ErrHandler
End If


xtrRows = 1
iLastRow = compList.Range(srcColumn & Rows.Count).End(xlUp).Row ' get the last row
     in the component list
If iLastRow < 2 Then
    Err.Raise vbObjectError + 514, "ReadWorksheet" _
        , "There␣appears␣to␣be␣empty␣cells␣in␣column␣" & srcColumn & "␣in␣the␣
            component␣list."
    compList.Parent.Close
    xlApp.Quit
    GoTo ErrHandler
End If
```

```
' create header
targetList.Cells(1, resourceColumn).Value = "ResourceName"
targetList.Cells(1, "B").Value = "CategoryNameProvider"
targetList.Cells(1, pinNameColumn).Value = "PinName"
targetList.Cells(1, sigNameColumn).Value = "SignalWireName"
targetList.Cells(1, sigTypeColumn).Value = "SignalType"
targetList.Cells(1, addrColumn).Value = "Address"
targetList.Cells(1, "G").Value = "Comment"
targetList.Cells(1, extConColumn).Value = "External Connection"


' start at k=2 to avoid the header. there are probably better looking ways to do
    that, but k=2 is very short and concise
For k = 2 To iLastRow
    cName = compList.Range(srcColumn & k).Value
    If cName = "" Then
        Err.Raise vbObjectError + 514, "ReadWorksheet" _
            , "There appears to be empty cells in column " & srcColumn & " in the
                component list. Issue with row number " & CStr(k)
        compList.Parent.Close
        xlApp.Quit
        GoTo ErrHandler
    End If


    '''' EXTRACT UNIT INFORMATION FROM cNAME HERE
    units = Split(cName, "-")

    ' extract PLC and component information before adjusting unit array
    plcPrefix = units(LBound(units))
    plcPrefix = Right(plcPrefix, Len(plcPrefix) - 3)
    For i = 1 To UBound(units)
        units(i - 1) = units(i)
    Next i
    ReDim Preserve units(UBound(units) - 1)



    ' INFO:
    ' we'd rather adapt this macro to PS than the other way around, to enable as
        many automatic functions in PS as possible.
    ' this check is to see if there are multiple compontents in the same unit. for
         singular components, default names are used.
    ' for multi-components, names are appended with information such as c1, pin1,
        c2, c3...
    clampCtr = 0
    pinCtr = 0

    If InStr(plcPrefix, "F") <> 0 Then
        component = "fixture"
    ElseIf InStr(plcPrefix, "CTT") <> 0 Then
        component = "turntable"
    Else
        Err.Raise vbObjectError + 513, "ReadWorksheet" _
            , "Could not recognize any machine types. Types recognized by this
                macro: F (fixture), CTT (turntable)." _
            & " Issue with component in row " & CStr(k)
        compList.Parent.Close
        xlApp.Quit
        GoTo ErrHandler
    End If
    plcPrefix = Left(plcPrefix, 3)

    ' signal generation for fixture pins/clamps and turntable pins/clamps is
        different (MIGHT NOT STILL BE THE CASE)
    ' increment by 2 with each step
    For N = LBound(units) To UBound(units) Step 2 ' why are all n:s automatically
        capitalized?

        unitType = units(N)
```

```
unitGroup = units(N + 1)

If component = "fixture" Then
    ' if clamp, create at_OPEN/CLOSE, rmtp_OPEN/CLOSE
    If InStr(unitType, "CLAMP") <> 0 Then

        clampNbr = Right(unitType, 2)
        If IsNumeric(nbrClamps) Then
            nbrClamps = CInt(clampNbr)
        Else
            Err.Raise vbObjectError + 515, "ReadWorksheet", _
                "Component␣name␣missing␣number,␣expected␣format␣TYPExx,␣
                    such␣as␣CLAMP02.␣Issue␣with␣component␣in␣row␣" & CStr(
                    k)
            compList.Parent.Close
            xlApp.Quit
            GoTo ErrHandler
        End If

        temp = unitGroup
        For h = 1 To nbrClamps

            unitGroup = Left(temp, 3)
            temp = Right(temp, Len(temp) - 3)

            xtrRows = xtrRows + 4

            targetList.Range(resourceColumn & xtrRows - 3 & ":" &
                resourceColumn & xtrRows).Value = cName

            ' see INFO
            If UBound(units) > 2 Or nbrClamps > 1 Then
                clampCtr = clampCtr + 1
                clampAppend = "c" & clampCtr & "_"
            Else
                clampAppend = ""
            End If

            clampStates = Array(clampAppend & "at_OPEN", clampAppend & "
                at_CLOSE", clampAppend & "rmtp_CLOSE", _
                        clampAppend & "rmtp_OPEN")
            clampSign = Array(cName & "_" & clampAppend & "at_OPEN", cName
                 & "_" & clampAppend & "at_CLOSE", _
                        cName & "_" & clampAppend & "rmtp_CLOSE", cName &
                            "_" & clampAppend & "rmtp_OPEN")
            clampTags = Array(plcPrefix & unitGroup & "SG", plcPrefix &
                unitGroup & "SG", plcPrefix & unitGroup & "YE1", _
                        plcPrefix & unitGroup & "YE2")


            targetList.Range(pinNameColumn & xtrRows - 3 & ":" &
                pinNameColumn & xtrRows).Value = _
                Application.Transpose(clampStates) ' set entry/exit name

            targetList.Range(sigNameColumn & xtrRows - 3 & ":" &
                sigNameColumn & xtrRows).Value = _
                Application.Transpose(clampSign) ' set signal name

            targetList.Range(sigTypeColumn & xtrRows - 3 & ":" &
                sigTypeColumn & xtrRows).Value = _
                Application.Transpose(Array("I", "I", "Q", "Q")) ' set I/O

            targetList.Range(tagColumn & xtrRows - 3 & ":" & tagColumn &
                xtrRows).Value = _
                Application.Transpose(clampTags) ' set plc tag

            targetList.Range(extConColumn & xtrRows - 3 & ":" &
                extConColumn & xtrRows).Value = "SIMBA" ' set external
                connection unit
```

```
        Next h

    ' if pin, create at_SET/RESET, rmtp_SET/RESET
    ElseIf InStr(unitType, "PIN") <> 0 Then

        pinNbr = Right(unitType, 2)
        If IsNumeric(nbrClamps) Then
            nbrPins = CInt(pinNbr)
        Else
            Err.Raise vbObjectError + 515, "ReadWorksheet", _
                "Component␣name␣missing␣number,␣expected␣format␣TYPExx,␣
                    such␣as␣CLAMP02.␣Issue␣with␣component␣in␣row␣" & CStr(
                    k)
            compList.Parent.Close
            xlApp.Quit
            GoTo ErrHandler
        End If

        temp = unitGroup
        For h = 1 To nbrPins

            unitGroup = Left(temp, 3)
            temp = Right(temp, Len(temp) - 3)

            xtrRows = xtrRows + 4

            targetList.Range(resourceColumn & xtrRows - 3 & ":" &
                resourceColumn & xtrRows).Value = cName

            ' see INFO
            If UBound(units) > 2 Or nbrPins > 1 Then
                pinCtr = pinCtr + 1
                pinAppend = "pin" & pinCtr & "_"
            Else
                pinAppend = ""
            End If

            pinStates = Array(pinAppend & "at_RESET", pinAppend & "at_SET"
                , pinAppend & "rmtp_SET", pinAppend & "rmtp_RESET")
            pinSign = Array(cName & "_" & pinAppend & "at_RESET", cName &
                "_" & pinAppend & "at_SET", _
                    cName & "_" & pinAppend & "rmtp_SET", cName & "_" &
                        pinAppend & "rmtp_RESET")
            pinTags = Array(plcPrefix & unitGroup & "SG", plcPrefix &
                unitGroup & "SG", plcPrefix & unitGroup & "YE1", _
                    plcPrefix & unitGroup & "YE2")

            targetList.Range(pinNameColumn & xtrRows - 3 & ":" &
                pinNameColumn & xtrRows).Value = _
                Application.Transpose(pinStates) ' set entry/exit name

            targetList.Range(sigNameColumn & xtrRows - 3 & ":" &
                sigNameColumn & xtrRows).Value = _
                Application.Transpose(pinSign) ' set signal name

            targetList.Range(sigTypeColumn & xtrRows - 3 & ":" &
                sigTypeColumn & xtrRows).Value = _
                Application.Transpose(Array("I", "I", "Q", "Q")) ' set I/O

            targetList.Range(tagColumn & xtrRows - 3 & ":" & tagColumn &
                xtrRows).Value = _
                Application.Transpose(pinTags) ' set plc tag

            targetList.Range(extConColumn & xtrRows - 3 & ":" &
                extConColumn & xtrRows).Value = "SIMBA" ' set external
                connection unit

        Next h

    ' if swing, create at_OPEN/CLOSE/RDY/SPEED/nSPEED
```

```
        ElseIf unitType = "SWING" Then
            xtrRows = xtrRows + 8

            swingStates = Array("at_OPEN", "at_CLOSE", "at_SPEED", "at_nSPEED"
                , "at_RDY", "rmtp_CLOSE", "rmtp_OPEN", "SPEED")
            swingSign = Array(cName & "_at_OPEN", cName & "_at_CLOSE", cName &
                 "_at_SPEED", cName & "_at_nSPEED", _
                        cName & "_at_RDY", cName & "_rmtp_CLOSE", cName & "
                            _rmtp_OPEN", cName & "_SPEED")
            swingTags = Array(plcPrefix & unitGroup & "SG", plcPrefix &
                unitGroup & "SG", plcPrefix & unitGroup & "SG", _
                    plcPrefix & unitGroup & "SG", plcPrefix & unitGroup & "
                        UE1_RDY", plcPrefix & unitGroup & "MUE1_L", _
                    plcPrefix & unitGroup & "MUE1_R", plcPrefix & unitGroup &
                        "MUE1_S")
            swingPorts = Array("I", "I", "I", "I", "I", "Q", "Q", "Q")

            targetList.Range(resourceColumn & xtrRows - 7 & ":" &
                resourceColumn & xtrRows).Value = cName ' set resource

            targetList.Range(pinNameColumn & xtrRows - 7 & ":" & pinNameColumn
                 & xtrRows).Value = _
                Application.Transpose(swingStates) ' set entry/exit name

            targetList.Range(sigNameColumn & xtrRows - 7 & ":" & sigNameColumn
                 & xtrRows).Value = _
                Application.Transpose(swingSign) ' set signal name

            targetList.Range(sigTypeColumn & xtrRows - 7 & ":" & sigTypeColumn
                 & xtrRows).Value = _
                Application.Transpose(swingPorts) ' set I/O

            targetList.Range(tagColumn & xtrRows - 7 & ":" & tagColumn &
                xtrRows).Value = _
                Application.Transpose(swingTags) ' set plc tag

            targetList.Range(extConColumn & xtrRows - 7 & ":" & extConColumn &
                 xtrRows).Value = "SIMBA" ' set external connection unit

        Else
            Err.Raise vbObjectError + 516, "ReadWorksheet", _
                "Could␣not␣recognize␣any␣component␣types.␣Types␣recognized␣by␣
                    this␣macro:␣CLAMP,␣PIN,␣SWING" _
                & "␣(must␣be␣entirely␣upper␣case).␣Issue␣with␣component␣in␣row
                    ␣" & CStr(k)
            compList.Parent.Close
            xlApp.Quit
            GoTo ErrHandler
        End If
        ' fixture type end
    ElseIf component = "turntable" Then

        If unitType = "TURNTABLE" Then
            xtrRows = xtrRows + 23

            turntabStates = Array("at_HOME", "rmtp_HOME", "at_LA", "at_LAEND",
                 "at_FORWARD", "at_BACKWARD", _
                        "at_OVERFORWARD", "at_OVERBACKWARD", "SYSERR", "
                            MOTORPROT", "mov_FORWARD", "mov_BACKWARD", _
                        "BIT1", "BIT2", "RLS_SET", "RLS_RESET", "at_POS1", "
                            at_POS2", "at_POS3", "ghost_to_OPEN", _
                        "ghost_to_CLOSE", "ghost_at_OPEN", "ghost_at_CLOSE")
            For m = LBound(turntabStates) To UBound(turntabStates)
                turntabSign(m) = cName & "_" & turntabStates(m)
            Next m

            turntabTags = Array(plcPrefix & unitGroup & "BG1_A3", plcPrefix &
                unitGroup & "BG1_E3", _
                    plcPrefix & unitGroup & "BG1_A10", plcPrefix & unitGroup &
                         "BG1_A7", plcPrefix & unitGroup & "BG1_A8", _
```

```
                plcPrefix & unitGroup & "BG1_A9", plcPrefix & unitGroup &
                    "BG1_A4", plcPrefix & unitGroup & "BG1_A5", _
                plcPrefix & unitGroup & "BG1_A6", plcPrefix & unitGroup &
                    "QM1", plcPrefix & unitGroup & "BG1_E1", _
                plcPrefix & unitGroup & "BG1_E2", plcPrefix & unitGroup &
                    "BG1_E5", plcPrefix & unitGroup & "BG1_E6", _
                plcPrefix & unitGroup & "QC1", plcPrefix & unitGroup & "
                    QC2", plcPrefix & unitGroup & "SG1", _
                plcPrefix & unitGroup & "SG2", plcPrefix & unitGroup & "
                    SG3", plcPrefix & "M13YE2", _
                plcPrefix & "M13YE1", plcPrefix & "M13SG1", plcPrefix & "
                    M13SG2")

            turntabPorts = Array("I", "Q", "I", "I", "I", "I", "I", "I", "I",
                "I", "Q", "Q", "Q", "Q", "Q", "Q", "I", "I", _
                        "I", "Q", "Q", "I", "I")

            targetList.Range(resourceColumn & xtrRows - 22 & ":" &
                resourceColumn & xtrRows).Value = cName ' set resource

            targetList.Range(pinNameColumn & xtrRows - 22 & ":" &
                pinNameColumn & xtrRows).Value = _
                Application.Transpose(turntabStates) ' set entry/exit name

            targetList.Range(sigNameColumn & xtrRows - 22 & ":" &
                sigNameColumn & xtrRows).Value = _
                Application.Transpose(turntabSign) ' set signal name

            targetList.Range(sigTypeColumn & xtrRows - 22 & ":" &
                sigTypeColumn & xtrRows).Value = _
                Application.Transpose(turntabPorts) ' set I/O

            targetList.Range(tagColumn & xtrRows - 22 & ":" & tagColumn &
                xtrRows).Value = _
                Application.Transpose(turntabTags) ' set plc tag

            targetList.Range(extConColumn & xtrRows - 22 & ":" & extConColumn
                & xtrRows).Value = "SIMBA" ' set external connection unit

        ElseIf InStr(unitType, "PIN") <> 0 Then
            ' for now, i'm presuming that the "pins" are never grouped
            xtrRows = xtrRows + 4

            clampStates = Array("at_OPEN", "at_CLOSE", "rmtp_CLOSE", "
                rmtp_OPEN")
            clampSign = Array(cName & "_" & "at_OPEN", cName & "_" & "at_CLOSE
                ", _
                        cName & "_" & "rmtp_CLOSE", cName & "_" & "rmtp_OPEN")
            clampTags = Array(plcPrefix & unitGroup & "SG", plcPrefix &
                unitGroup & "SG", plcPrefix & unitGroup & "YE1", _
                        plcPrefix & unitGroup & "YE2")

            targetList.Range(resourceColumn & xtrRows - 3 & ":" &
                resourceColumn & xtrRows).Value = cName ' set resource name

            targetList.Range(pinNameColumn & xtrRows - 3 & ":" & pinNameColumn
                 & xtrRows).Value = _
                Application.Transpose(clampStates) ' set entry/exit name

            targetList.Range(sigNameColumn & xtrRows - 3 & ":" & sigNameColumn
                 & xtrRows).Value = _
                Application.Transpose(clampSign) ' set signal name

            targetList.Range(sigTypeColumn & xtrRows - 3 & ":" & sigTypeColumn
                 & xtrRows).Value = _
                Application.Transpose(Array("I", "I", "Q", "Q")) ' set I/O

            targetList.Range(tagColumn & xtrRows - 3 & ":" & tagColumn &
                xtrRows).Value = _
                Application.Transpose(clampTags) ' set plc tag
```

```
        targetList.Range(extConColumn & xtrRows - 3 & ":" & extConColumn &
            xtrRows).Value = "SIMBA" ' set external connection unit

    ElseIf InStr(unitType, "CLAMP") <> 0 Then

        ' with only two turntable samples, I'm very uncertain regarding
            what forms they'll appear in. for now i'll assume
        ' that the clamps will either all be in one place, or divided with
            reasonable limits, and that the group numbers
        ' will always range from M14 to M19, and the clamps will always be
            grouped in pairs
        clampNbr = Right(unitType, 2)
        If IsNumeric(nbrClamps) Then
            nbrClamps = CInt(clampNbr)
        Else
            Err.Raise vbObjectError + 515, "ReadWorksheet", _
                "Component␣name␣missing␣number,␣expected␣format␣TYPExx,␣
                    such␣as␣CLAMP02.␣Issue␣with␣component␣in␣row␣" & CStr(
                    k)
            compList.Parent.Close
            xlApp.Quit
            GoTo ErrHandler
        End If

        If nbrClamps = 12 Then
            unitGroup = "M14M14M15M15M16M16M17M17M18M18M19M19"
        End If
        temp = unitGroup
        For h = 1 To nbrClamps

            unitGroup = Left(temp, 3)
            temp = Right(temp, Len(temp) - 3)

            xtrRows = xtrRows + 4

            targetList.Range(resourceColumn & xtrRows - 3 & ":" &
                resourceColumn & xtrRows).Value = cName

            clampCtr = clampCtr + 1
            clampAppend = "c" & clampCtr & "_"

            clampStates = Array(clampAppend & "at_OPEN", clampAppend & "
                at_CLOSE", clampAppend & "rmtp_CLOSE", _
                        clampAppend & "rmtp_OPEN")
            clampSign = Array(cName & "_" & clampAppend & "at_OPEN", cName
                 & "_" & clampAppend & "at_CLOSE", _
                        cName & "_" & clampAppend & "rmtp_CLOSE", cName &
                            "_" & clampAppend & "rmtp_OPEN")
            clampTags = Array(plcPrefix & unitGroup & "SG", plcPrefix &
                unitGroup & "SG", plcPrefix & unitGroup & "YE1", _
                        plcPrefix & unitGroup & "YE2")

            targetList.Range(pinNameColumn & xtrRows - 3 & ":" &
                pinNameColumn & xtrRows).Value = _
                Application.Transpose(clampStates) ' set entry/exit name

            targetList.Range(sigNameColumn & xtrRows - 3 & ":" &
                sigNameColumn & xtrRows).Value = _
                Application.Transpose(clampSign) ' set signal name

            targetList.Range(sigTypeColumn & xtrRows - 3 & ":" &
                sigTypeColumn & xtrRows).Value = _
                Application.Transpose(Array("I", "I", "Q", "Q")) ' set I/O

            targetList.Range(tagColumn & xtrRows - 3 & ":" & tagColumn &
                xtrRows).Value = _
                Application.Transpose(clampTags) ' set plc tag

            targetList.Range(extConColumn & xtrRows - 3 & ":" &
```

```
                        extConColumn & xtrRows).Value = "SIMBA" ' set external
                        connection unit

                Next h

            Else
                Err.Raise vbObjectError + 517, "ReadWorksheet", _
                    "Could not recognize any component types. Types recognized by
                        this macro: CLAMP, PIN, TURNTABLE" _
                    & " (must be entirely upper case). Issue with component in row
                        " & CStr(k)
                compList.Parent.Close
                xlApp.Quit
                GoTo ErrHandler

            End If
            ' turntable type end
        End If

    Next N


Next k

Dim indx As Integer
Dim noxt As Range
Dim thisCell As String

' all signals ending with G need to be appended with a number. the number is based
     on how many signals with the same
' prefix and group number there are
iLastRow = targetList.Cells(Rows.Count, "a").End(xlUp).Row ' make sure this sets
    iLastRow for wot sheet


For k = 2 To iLastRow

    ' for some reason, the "while not noxt is nothing" check does not work, and
       instead of looking up why, i've implemented
    ' a workaround by starting the counter at 2 and setting number 1 manually. if
       you can get a "noxt is nothing/empty/null"
    ' check to work, counter can be set to 1, the initial search should start at 1
        (currently starts at k = 2), the hard "set
    ' to 1" assignment can be removed, and the firstAddress assignment should be
       moved to appear in the "if not noxt is..."
    ' check
    ctr = 2
    thisCell = targetList.Cells(k, "G").Value
    firstAddress = targetList.Range("G" & k).Address

    ' if this cell ends with G
    If Right(thisCell, 1) = "G" Then
        ' this returns the second occurrence of the signal. by starting at index
           G1, the first occurrence can be found.
        Set noxt = targetList.Range("G" & k & ":G" & iLastRow).Find(thisCell, _
                    LookIn:=xlValues, _
                    LookAt:=xlWhole, _
                    SearchOrder:=xlByRows, _
                    SearchDirection:=xlNext)
                    ', _ after:=noxt)

        ' find all signals with identical name, append name with a digit
        If Not noxt Is Nothing Then
            'firstAddress = noxt.Address
            Do
                targetList.Cells(noxt.Row, noxt.Column).Value = targetList.Cells(
                    noxt.Row, noxt.Column).Value & ctr
                Set noxt = targetList.Range("G2:G" & iLastRow).Find(thisCell, _
                    LookIn:=xlValues, _
                    LookAt:=xlWhole, _
```

```
                    SearchOrder:=xlByRows, _
                    SearchDirection:=xlNext, _
                    after:=noxt)
                ctr = ctr + 1
            Loop While Not noxt Is Nothing And firstAddress <> noxt.Address
            ' set the first occurrence to 1
            targetList.Cells(k, "G").Value = targetList.Cells(k, "G").Value & "1"
        End If
    End If

Next k

compList.Parent.Close
xlApp.Quit

Exit Sub

' error management
ErrHandler:
    'Select Case Err.Number

    'Case 9
        'MsgBox Err.Source & ": The following error occured:  " & Err.Description
    'Case ERROR_INVALID_DATA
    MsgBox "The following error occured:  " & Err.Description
    'End Select

End Sub


Private Sub CommandButton2_Click()

' SET PLC ADDRESS
' TODO: cleaning
' TODO: error management
' TODO: bunch of trials

' LOTS OF THINGS HERE FROM TROUBLESHOOTING, THEY'RE KEPT FOR NOW FOR FURTHER
    TROUBLESHOOTING, BUT CLEANUP IN THE FUTURE
Dim thisCell As String
Dim ctr As Integer
Dim noxt As Range
Dim wut As Range
Dim plcTags As String
Dim plcRow As Integer
Dim plcList As Worksheet
Dim targetList As Worksheet
Dim missingPLCList As Worksheet
Dim plcPath As String
Dim matchedTag As Range
Dim matchedAddr As Integer
Dim addrColumn As String
Dim srcColumn As String
Dim tagColumn As String
Dim plcAddrColumn As String
Dim srcSheet As String
Dim addr As String
Dim iLastRow2 As Integer
Dim iLastRow As Integer
Dim countr As Integer


' prompt the user to select the component list file
With Application.FileDialog(msoFileDialogOpen)
    .Title = "Select the PLC Tags list"
    .AllowMultiSelect = False
    .Show
    If .SelectedItems.Count = 1 Then
        plcPath = .SelectedItems(1)
```

```
    End If
End With

' potentially introduce a prompt that asks the user to enter or select the
    corresponding sheets/columns
srcSheet = "PLC Tags"

srcColumn = Application.InputBox(prompt:="Enter the column in which the components
    are found (A by default). Cancel to select default.", Type:=1 + 2)
If srcColumn = vbNullString Or srcColumn = "False" Then
    srcColumn = "A" ' component source column
End If

addrColumn = "F"
plcAddrColumn = "D"
tagColumn = "G" ' desired plc tag column destination

' define target location
Set targetList = ActiveWorkbook.Worksheets(1)
targetList.Name = "Matched tags"


' if the list is sufficiently large, there are other more appropriate methods
' for now this is used:
' create new instance of excel, and make it invisible
Set xlApp = New Excel.Application
xlApp.Visible = False
' open your workbook in this new instance
Set plcList = xlApp.Workbooks.Open(plcPath, ReadOnly:=True).Worksheets(1)


' find a match for the signal and plc tag list
iLastRow = targetList.Cells(Rows.Count, tagColumn).End(xlUp).Row ' make sure this
    sets iLastRow for wot sheet
iLastRow2 = plcList.Cells(Rows.Count, srcColumn).End(xlUp).Row ' make sure this
    sets iLastRow for wot sheet

targetList.Range("F2:F" & iLastRow).NumberFormat = "@"

For k = 2 To iLastRow
    thisCell = targetList.Cells(k, tagColumn).Value
    If thisCell <> "" Then
        On Error Resume Next
        matchedTag = plcList.Range(srcColumn & "2:" & srcColumn & iLastRow2).Find(
            thisCell)
        matchedAddr = plcList.Range(srcColumn & "2:" & srcColumn & iLastRow2).Find
            (thisCell).Row
        If matchedAddr = 0 Then
            addr = "WWNO MATCH"
        Else
            matchedAddr = matchedTag.Row
            addr = plcList.Cells(matchedAddr, plcAddrColumn).Value
        End If
        targetList.Cells(k, addrColumn).Value = Right(addr, Len(addr) - 2)
        matchedAddr = 0
    End If
Next k


Set missingPLCList = ActiveWorkbook.Worksheets.Add
missingPLCList.Name = "Missing tags"
countr = 2
missingPLCList.Cells(1, "A").Value = "PLC tag"
missingPLCList.Cells(1, "B").Value = "Type"
missingPLCList.Cells(1, "C").Value = "Address"
missingPLCList.Cells(1, "D").Value = "Comment"
'' generate a list with unmatched PLC tags
'' take note of their safety tag
For k = 2 To iLastRow2
    thisCell = plcList.Cells(k, srcColumn).Value
```

```
    If thisCell <> "" Then
        On Error Resume Next
        matchedTag = targetList.Range("G2:G" & iLastRow).Find(thisCell)
        matchedAddr = targetList.Range("G2:G" & iLastRow).Find(thisCell).Row
        If matchedAddr = 0 Then
            missingPLCList.Cells(countr, "A").Value = thisCell
            missingPLCList.Cells(countr, "B").Value = plcList.Cells(k, "B").Value
            missingPLCList.Cells(countr, "C").Value = plcList.Cells(k, "D").Value
            missingPLCList.Cells(countr, "D").Value = plcList.Cells(k, "E").Value
            countr = countr + 1
        End If
        matchedAddr = 0
    End If
Next k

plcList.Parent.Close
xlApp.Quit

End Sub
```