INDUSTRIAL AUTOMATION PRACTICES:

SPECIFICATION AND PROGRAMMING OF LOGIC CONTROL APPLICATIONS IN THE "ITS PLC" TRAINING ENVIRONMENT

Copyright © 2012

ISBN 978-989-96460-1-8

D.L. 313843/10

No part of this book may be reproduced, translated into a machine language, stored in a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording or otherwise) without the prior written permission from the publisher.

Copyright 2012 by

REAL GAMES LDA.
Rua Elísio de Melo nº 39, Piso 3
4000-196 Porto, PORTUGAL

Email: info@realgames.pt
Internet: http://www.realgames.pt

Translated from the Portuguese edition by António Pessoa de Magalhães

Revised by Carlos Duarte

Cover designed by Bruno Vigário and Nuno Silva, REAL GAMES LDA.

INDUSTRIAL AUTOMATION PRACTICES:

SPECIFICATION AND PROGRAMMING OF LOGIC CONTROL APPLICATIONS IN THE "ITS PLC" TRAINING ENVIRONMENT

António Pessoa de Magalhães

1st edition, 2012

DISCLAIMER

The purpose of the problems and respective solutions presented in this book is solely didactical – specifically, Programmable Logic Controllers (PLCs) programming training using the "ITS PLC" platform. Hence, despite both the author's and editor's belief that the information presented throughout this book is correct, under no circumstances will they be held responsible for the use of the exact or any derived program presented in this book in any application whatsoever which may result in any damage to property or people.

To
João Rodrigo and José Diogo
who do so love playing around "crazy machines"...

Índice

PREFACE

I have been using virtual systems in my Programmable Logic Controllers (PLCs) programming classes for a number of years now. Without meaning to demean the role and importance of real target systems, virtual training plants are a low cost, risk and hazard free solution for both trainers and trainees. Moreover, virtual systems can easily be installed and multiplied, whilst assuring that trainees are kept in contact with the right control equipment throughout the entire process, as if they were training in real plants.

However, when a further motivational element is added to the above-mentioned benefits, namely an extremely realistic and interactive system capable of transmitting the environment and motivation of modern day computer games to the classroom, involving trainers and trainees alike, then some extremely positive, almost unique conditions, have been found to create a simplified, fascinating, natural and efficient learning environment.

I thus strongly believe and defend that the training software ITS PLC is a privileged means of creating an extremely motivating learning environment, capable of motivating students in such a manner as to reveal their natural desire to achieve, think critically and persist as they so often do when playing computer games. Consequently, I eagerly accepted the proposal presented by Real Games Lda. inviting me to come up with an "exercise manual" which would accompany their ITS PLC product.

This has been a complex, stimulating and extremely interesting and enriching task, which has made me feel as if I were playing a computer game on a number of occasions. Thus, I have chosen to follow a typical computer game methodology. I decided on creating several game scenarios in which each mission begins with a number of small challenges, fundamental but easily attainable, moving on to a sequence of graded tasks in which each training exercise adds something to the previous one, and thereby allowing the trainee to acquire ever-increasing knowledge and confidence. However, each game is nothing more than a work proposal that each trainer can and should adapt to his own objectives and target trainees.

The presentation of the solutions in Structured Text (ST) may come somewhat as a surprise to some, seeing that it is not the most common PLC programming language. Two fundamental reasons lie behind this choice: on the one hand, ST is an advanced textual programming language that significantly facilitates the presentation and explanation of PLC programs to a heterogeneous public; on the other hand, the selection of this language contributes very naturally but strongly to the dissemination and spreading of the IEC 61131-3 standard. PLC programmers who don´t usually use ST programming language nor have adhered to this standard yet, will surely have experienced little and reflected even less on their virtues. Consequently, they are the precise type of reader that this initiative is aimed at.

Considering that Structured Text is not the most common programming language of many programmers, it is highly likely that the solutions hereby presented will come up translated in a variety and diversity of PLCs programming languages and dialects. The interest and pertinence

of such translations is understandable, as is their subsequent public release. However, it is clear that the potential of the ITS PLC is not limited to such translations and much less to the proposals enclosed in this book. There will always be room for both the pleasure and desire of elaborating new problems and presenting novel solutions which represent new games and forms of playing ITS PLC. One can only hope that trainers and trainees will both come to share such pleasures!

Porto, Portugal, September 2011

António J. Pessoa de Magalhães

PART 1
PRESENTATION

This book presents a number of exercises for the ITS PLC educational software produced by Real Games Lda. Its prime objective is to maximise the usage of this training environment by presenting work plans capable of promoting a progressive, coherent and solid learning experience, leading to the specification and development of logical control techniques and the corresponding programming Programmable Logic Controllers – PLCs.

Despite the fact that the proposals set forth have been inspired in virtual environments, the ultimate aim of this training is that the results and applications hereby produced transcend this virtual realm as far as possible. Thus, if on a comprehensive picture, the ITS PLC plants represent relatively inclusive and advanced logical problems and concerns in the Industrial Automation field, on a more practical view they are a pretext to introduce and discuss elementary issues related to the most common errors, doubts and difficulties presented by those initiating or deepening their knowledge on PLC programming. The definition, presentation, treatment and logical ordering of these problems has been based on the author's vast experience in the field of teaching PLC programming which has included both real and virtual systems.

The prime objective of the exercises presented throughout this book is thus to program a PLC to correctly and efficiently control the ITS PLC plants. Yet, in order to attain superior results, this should represent a starting point for the development of further applications. For example, applications which combine human-machine interface consoles supervisory systems, distributed control or even information management. Additionally, it is extremely useful to seek robust solutions capable of dealing with situations of failure and uncertainty. Work proposals related to both of these cases are presented throughout this book. They are aimed at an advanced level of education and training and, as such, trainers should adapt these orientations to their specific course objectives, the equipment available to them and to the level of their trainees.

The exercises presented throughout this book are based on the PLC control of the five ITS PLC plants and comprise a total of sixty exercises: twelve per plant. For each problem proposed, an adequate solution is set forth in the form of a PLC program which is duly commentated. In the case of the simpler programs, these are preceded by a relatively simple and informal commentary. Alternatively, more complex problems are accompanied by an explanation provided in the *GRAFCET*[1] language in accordance with the second edition of the IEC 60484 standard. Despite not being the sole possible solution, the specifications and solutions presented are aimed at being modular and generic, in addition to leading to a process of thought and reflection for trainers and trainees alike.

It is increasingly common to begin the project of a sequential system with its *GRAFCET*

[1] Following the trend of specialised literature, the text uses the term "*GRAFCET*" to denote the French specification language "*GRAphe Fonctionnel de Commande Étape Transition*" and the term "*grafcet*" to denote a graphical schema developed according to the *GRAFCET* language.

specification. The fact that *GRAFCET*, which is often wrongly referred to as "*Sequential Function Chart*", is a standardised graphic methodology, which is far more succinct, objective and encompassing than state or timing diagrams, has largely contributed to it being a widely used and well-known tool by the majority of PLCs programmers. This is further enhanced by the fact that the translation of a *grafcet* in a generic PLC program is a relatively simple process, a topic which will be further explored in this book. Due to the above-mentioned reasons, the selection of *GRAFCET* as the descriptive language of the behaviour of a system throughout this book has been very natural.

On the other hand, the choice of a programming language to present the solutions found, was subjected to great ponderation and thought terminating in the adoption of the "Structured Text" in accordance with the second edition of the IEC 61131-3 standard. Apart from serving the objectives of this study quite well, this decision promotes an increasingly important language and standard in the PLCs domain thereby broadening the didactic interest of this book.

Seeing that the Structured Text programming language and the IEC 61131-3 standard may not be well mastered by those initiating the programming of PLCS, information relating to both has been included in this book. This information is presented prior to solutions. Additionally, the IEC 61499 standard has been promoted in a number of exercises. Although this is not directly related to the prime objectives of this book, trainers who are more familiarised with this topic will surely be able to identify the correct means to achieve this end.

Thus, one believes that there are a number of reasons that justify the interest of this book, which has been organised in the following manner:

Part 1 – Presentation – presents all the necessary issues for a correct and complete introduction of the reader to the learning environment awaiting him. More specifically, it begins by contextualising the sequence, objectives and public targets of the challenges presented throughout this text. Next, a number of considerations relating to the manner the reader should view the challenges he is faced with, as well as his expected learning progress in light of his previous knowledge, are presented. Lastly, a list of the resources required for the resolution of the exercises which have been proposed is put forth.

Part 2 – The Problems – presents the five main challenges or missions in the form of a computer game that the reader will have to progressively master. Each challenge corresponds to the control of a virtual ITS PLC plant which has been translated into twelve programming exercises. These have been sequenced in a number of small tasks, in order to promote a simple, natural and efficient learning process. Each mission is preceded by an explanation relating to its environment in order to familiarise the reader with the typical control problems introduced by the plant and the practical interest of the proposed exercises. At the end of each mission, work proposals aimed at a more advanced level are presented. These proposals have been organised in accordance with two perspectives: On the one hand, the integration of the applications with supervisory systems and human-machine interface consoles; on the other hand, the improvement of the performance of the programs developed, through the inclusion of procedures to support failures in sensors and actuators and other fault based scenarios.

Part 3 – The Solutions – This section begins with various considerations relating to programming in Structured Text and the IEC 61131-3 standard. This section is aimed at readers who are not very familiar with these topics. As such, appropriate literature, websites and software is suggested. Next, the solutions to the challenges presented in Part two are put forth. Each solution is presented along with the respective justification of the programming procedures, which are often based on a *grafcet*, followed by the duly commentated corresponding program.

Part 4 – Epilogue – Provides a summary of the main conclusions presented throughout the text.

The Game

Welcome to the game ITS PLC! If you didn't happen to come upon this book by chance, then you will know that ITS PLC is a software package developed by Real Games Lda., which emulates five industrial plants to be controlled by an external PLC. For more information on this product please consult www.realgames.pt.

Before continuing, it is important that the reader become familiar enough with the software in question by understanding its goals and features, as well as the properties of the sensors and actuators included in the virtual plants. A non-licensed version of the ITS PLC software can be used for this purpose. This, in addition to the respective user guide, is to be found on the Real Games Lda. site.

The prime objective of the current text is to aid the reader solve each of the five scenarios that comprise the ITS PLC. To do so, the reader, who is also the player, will have to wire up a PLC to the computer where the ITS PLC has been installed and program it correctly.

Upon familiarising himself with the application, the reader will soon come to realise that this game presents him with a total of five missions, namely:

- Automating a sorting plant;
- Automating a batching plant;
- Automating a palletising machine;
- Automating a pick and place station;
- Automating an automatic warehouse.

The practical interest and the exact objectives of each mission will soon be revealed. For the moment it is suffice to say that each mission is comprised of a total of twelve tasks which are to be carried out in a set order so that the player will progressively familiarise himself with the system that is to be automated, in addition to acquiring the necessary skills to effectively complete the assigned mission. Upon completion of each task, the player will move on to the next level. Completing the last level implies terminating the mission. Thus, completing all five missions directly implies finishing the game and becoming an "expert PLC programmer"!

The justification and demonstration of the practical interest of each mission are aspects that have received due consideration. Therefore, each mission begins with an explanation of the physical and functional aspects of the target plant, thereby allowing one to perceive the interest, objectives and difficulties of the particular mission in a real identical installation context.

Each task is accompanied by a simple yet precise text which includes the mission's scenario containing the objective to be attained and the I/O signals to consider. A number of tasks are aimed at the automated control of just a part of the equipment available, thus implying that the player is to manually control the remaining parts. In such cases, the text provides a complete explanation as to the how and why in question. This is also the case when the verification of the programs developed requires the simulation of failures or the forcing of a functional state to one or more agents.

The player is given one or two "tips" to complete each mission. Although it makes sense for the player to use these clues, it is far more desirable for him to solve the mission on his own.

Each task is accompanied by the respective solution which the player should consult upon completing the task. Should the player not be able to solve the task on his own, this solution should be consulted in a careful manner so as to guarantee that he is able to move onto the following task. It is important that the player not give up and that he meditate on the commentaries and justifications that have been provided for each task.

Upon completing the mission, which implies automating a particular system, there is still room for further development. Thus, the player is challenged to complete more ambitious tasks in the form of two sets of supplementary exercises:

- ITS SUPER – Supervisory Environments and Systems;
- ITS DEEP – Dependable Environments Programming.

The former aims to develop distributed and flexible solutions by the integration of technological resources – such as PLCs, HMI consoles and supervisory systems amongst others. The latter involves a set of challenges aimed at improving the reliability of the solutions found, through the inclusion of detection techniques aiming to support faulty situations. More than being specific exercises with specific and rigid tasks, the proposals contained in these exercises act as guiding lines for broader tasks requiring more time and effort. Thus, the trainers are to adapt them to the equipment they have access to and to their particular trainees. One possibility is that they be viewed as topics for small individual projects at a more advanced level to be done individually or in groups.

The Players

The ITS PLC training software is a didactical platform which can and should be used by beginner PLCs programming trainees for developing simple programs. However, programming a PLC to control and endow the five systems included in the ITS PLC package with relatively elaborate functions, is a task that requires an Intermediate level of knowledge and skills, namely:

- Logical systems: binary variables and codes; binary, octal and hexadecimal numbering systems, logical operators and Boolean algebra; memory elements, *set* and *reset* operations, load and store operations; arithmetic, shift and rotation operations;

- Specification: functional description using timing diagrams, state diagrams and *GRAFCET*;

- PLCs: the functional model of a PLC; allocation and wiring of inputs and outputs; data storage and internal memory organisation, timers and counters, minimal programming experience of PLCs and the usage of development tools and associated testing; ease in reading and understanding a PLC manual.

Should all of the above be familiar, then you may consider yourself to be apt to begin and end the game. If however, you are not aware of or do not minimally master many of the above-mentioned topics, then you should begin by developing your knowledge prior to accepting the challenges put forth in this book. A fair amount of literature is available on the theme. Your trainer may direct you in this respect.

For those preparing to initiate the game, the most obvious piece of advice is to avoid cheating as one would in computer games. Having difficulties in finding a solution and knowing that it is just at the turn of a page may prove hazardous to one's perseverance. Whenever you feel this temptation, remember that it will imply missing out on the opportunity to discover the solution for yourself – which may prove to be the most interesting quest of all and the most valuable part of the entire learning process.

Always seek to find your own solutions. This will allow you to broaden your knowledge in a solid and natural manner. If you fall into the temptation of cheating your way to the solution you will verify that it will just be a matter of time before you are no longer able to remember the solution in question. Perhaps you will not even have become aware of the fact that the programming of PLCs is a matter of "logic" and not of cheap and easy "tricks"!

The Equipment

You will surely be aware of the equipment that is required to begin the game: a computer with a duly licensed ITS PLC installation, and a PLC with a minimal number of inputs and outputs. Verify the type of inputs and outputs of your PLC and make sure that you have wired them correctly to the I/O board that accompanies the software. Consult the ITS PLC user guide in relation to this matter. Keep this manual at hand for you will surely need to frequently consult the input and output map for the virtual plants.

You should also keep the manual of your PLC and of the associated development software at hand. However, this may not always prove to be sufficient. Thus, you should also make sure you have some literature about logical systems and *GRAFCET* specification (preferably the second edition of the IEC 60484 standard) for easy and rapid consultation.

Even a modest PLC will be capable of controlling the installations presented throughout this book.

However, should you have a more sophisticated PLC, you should not forget to explore its potential by inventing novel exercises and alternative solutions.

If you are able to connect two monitors to your computer you should do so. One should be used to visualise the ITS PLC environment and the other to analyze all the information associated with the execution of the PLC program online with the aid of a debugging tool. You will rapidly become aware of the advantages of using such a strategy.

Lastly, if you are the type of person who likes computer games and are easily excited by logical problems, then you will need a large pot of freshly brewed coffee or of your favourite tea for the day or night will surely be long.

Now that you know the essential about the ITS PLC game, the time has come to play the game itself!

PART 2
THE PROBLEMS

You are probably already in front of your computer, properly equipped and willing to know your first challenge. Just a few short notes before you begin:

As stated before, problems are suggested in five self-contained modules, agreeing with the order by which virtual plants are introduced in the ITS PLC training software: *sorting*, *batching*, *palletiser*, *pick and place* and *automatic warehouse*. Missions do not have to be necessarily accomplished in this order, but you are advised to follow it. This is because, for the sake of the writing, and to avoid the duplication of detailed descriptions, in some cases, the resolution of a task points to comprehensive matters and detailed justifications provided in the resolution of a task of a previous mission, which are assumed to have been properly mastered at the time they were presented. But really important here is that, within each mission, you will try solve the tasks by the order they are presented in. This is because a task typically develops, in one way or the other, from the previous one and will be further developed in the next one. So, when you feel it is being hard to progress in a mission, you may leave it temporarily open, without consulting the solution, and proceed to the next mission. Perhaps this will provide the lessons and the inspiration that will allow you to later resume the mission left open.

Also important is to make sure that you have perfectly understood the statement of a task before starting to think about its resolution. Checking if the I/O listed in the task statement seems consistent with the objectives of the task is a good practice to ensure this. Always start by defining roughly, but objectively, the required procedures to solve the task. A timing diagram, state diagram or *grafcet* is a good approach for modelling such procedures, especially in the more complex cases. Then start developing your program. If the statement of a problem does not seem quite clear, do not use this argument as an excuse to peek at the solution. Instead, try to bypass the problem by taking the interpretation that seems most plausible to you, and not simpler!

Finally, two pieces of advice:

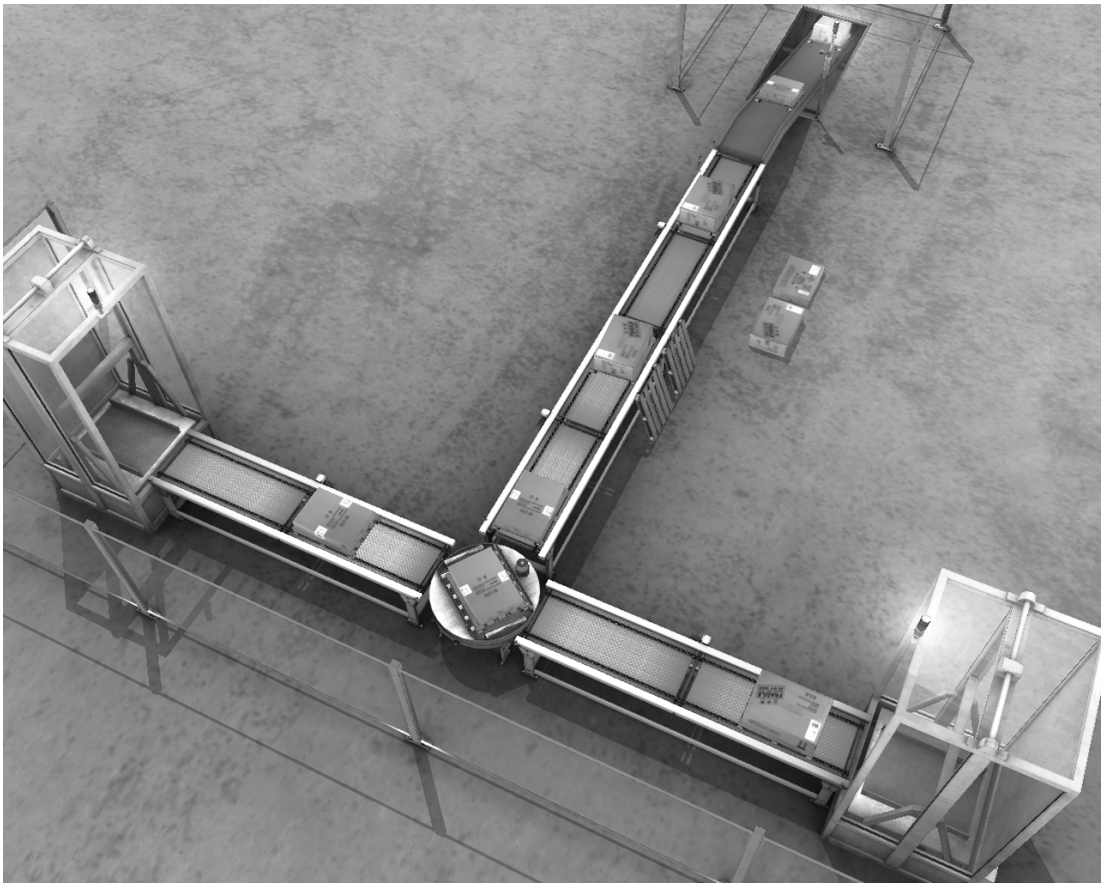
During your experiences, you will surely often feel the need to restart your system; i.e., the virtual plant and your PLC. The virtual plant is reset by pressing the clear button existent in the utility panel. As for restarting your PLC, we suggest this: include in your program a procedure for restarting its internal variables when a non-used input, for instance a sensor or a button, is forced or pressed; this way, you can easily restart your PLC status any time, especially after pressing the clear button, while keeping it in the running mode. Alternatively, you can use the input signal from the manual/auto switch (Input 11) to reset your PLC. However, in this case, the PLC will restart every time the installation is launched in automatic mode, which is not always an interesting situation.

It is very important that you do not let you PLC go into the running mode before launching the virtual plant as, in this case, some of the internal PLC variables may change to values which are inconsistent with the initial state of the target plant.

Having made these considerations, the time to meet the challenges ahead has finally come. Good luck now!

MISSION 1: AUTOMATED CONVEYING AND SORTING OF CASES ON PALLETS

OBJECTIVE: To move cases from the entry bay to the exit elevators, sorting them by height



About this Mission

The movement of materials and parts on automated transportation systems, such as conveyers or transfer units, is very common in industrial plants. From a functional point of view, a unidirectional conveyer is the simplest transportation device, as its present state, handles or does not handle materials, can be represented by just a binary variable. Yet, most practical devices are far more flexible and complex than this, having a considerable number of internal states. That is the case of the sequential transporters, like the turntable included in this application which, performing both transferring and sorting jobs typically have several entry and exit interfaces.

At the entry (or tail) of a conveyer there is something that provides the materials to be moved; e.g., another conveyer, an operator or an automatic feeder. At the end (or head) of the conveyer, another system retrieves the moved materials; e.g., a sorting device, another conveyer, an operator or a packing station. The mission of a conveying system is thus to move materials from a location to another in an efficient way. Efficiency typically means that materials should be correctly conveyed and routed from a pre-defined origin to a pre-defined destination in minimum time and with the least power consumption possible. This is synonymous of saying that:

- A conveyer should not be running if it is not handling any material;
- A conveyer handling a material should not be stopped, unless this is absolutely necessary;
- The sorting devices should properly route all the materials.

The main goal of this mission is to show that, even in a complex plant, the exact characterization of the devices existent at the entry and end of a conveyer are not relevant for its effective control; the central issue in this matter is the synchronisation of each conveyer with the systems that feed and retrieve the moved materials. Mastering this notion is the key element for designing a centralised or distributed modular solution perfectly adaptable to the large conveying systems commonly found in real industrial plants.

Particularly important in any flexible conveying and sorting system is being able to manage all the information required for the proper routing of the materials. For such, the information related to the materials in transit – no matter if they originate from more or less sophisticated identification systems such as bar code or RFIDs readers, or just from ordinary proximity sensors, as in the case of the present application –, usually has to be acquired, routed and tracked in a very similar way to that of the conveyed materials.

Also important in any conveying system is the openness of the local controllers to the interchange of information with human-machine interface consoles and supervisory systems. Last, but not least, one must also consider the effective detection and proper handling of those situations that may lead to the damage of the conveyed materials or the physical degradation of the conveying system itself.

This mission covers all these issues, starting with the basic ones.

TASK 1: Automated movement of single pallets on the entry conveyer	
Scenario:	The entry conveyer starts running when it senses a pallet at its entry, moves it to the turntable and then stops.
Objective:	To automate the control of the entry conveyer for moving pallets, one at a time.
Starting Conditions:	No pallets on the entry conveyer.
I/O Signals:	Inputs: sensors 0 and 3.
	Outputs: actuator 1.
Manual Procedures:	Manually control the feeding conveyer (by forcing actuator 0 on and off) to let pallets reach the entry conveyer, one at a time, and just when the entry conveyer is idle.
	Keep the rollers of the turntable running (by forcing actuator 2 on) enabling this way the discharge of pallets coming from the entry conveyer, throwing them onto the ground.
	Remove cases and pallets that accumulate behind the turntable, keeping it clear.
Tips:	Consider a Boolean variable (a memory bit) denoting if the entry conveyer should be running or not, and find its dependency from other variables…
	The entry and exit of pallets on the conveyer introduce logical transitions in the corresponding sensors…

TASK 2: Automated feeding and movement of single pallets on the entry conveyer	
Scenario:	The entry conveyer behaves as in Task 1. The feeding conveyer is now automated and lets pallets reach the entry conveyer, one at a time, and just when it is idle.
Objective:	To automate the feeding of pallets onto the entry conveyer, thus eliminating the manual procedure included in Task 1.
Starting Conditions:	No pallets on the entry conveyer.
I/O Signals:	Inputs: sensors 0 and 3 – or others you may find more suitable…
	Outputs: actuators 0 and 1.
Manual Procedures:	Keep the rollers of the turntable running (by forcing actuator 2 on) enabling this way the discharge of pallets coming from the entry conveyer, throwing them onto the ground.
	Remove cases and pallets that accumulate behind the turntable, keeping it clear.
	Introduce failures in the feeding conveyer (by forcing actuator 0 off) in order to disable the arrival of pallets to the entry conveyer, to make sure this conveyer does stop when it is not handling any pallet.
Tips:	The transfer of pallets between the feeding and the entry conveyers requires that both are running simultaneously…
	The feeding conveyer should not necessarily be stopped when the entry conveyer handles a pallet. Define a Boolean variable, “Busy_1”, to be TRUE when the entry conveyer cannot receive a pallet…

TASK 12: Storing and retrieving items randomly

Scenario:	The restart button light indicator turns on when the warehouse is launched in automatic mode. By pressing this button, the following cyclical running mode starts: Boxes are sequentially stored in randomly selected (empty) compartments, until 35 boxes exist in the warehouse. When this is reached, boxes are sequentially retrieved from randomly selected storing compartments until just 15 boxes exist in the warehouse. When this is reached, the storage phase restarts, so that random box storage and random box retrieval phases alternate indefinitely. As in previous tasks, the start button light indicator blinks when the transelevator is moving.
Objective:	To implement a storage and retrieval demonstration mode based on random selections.
Starting Conditions:	Transelevator anywhere, but unloaded. Warehouse empty.
I/O Signals:	Inputs: sensors 1, 3, 5, 6 and 7 and restart and emergency buttons.
	Outputs: actuators 0 to 7 and start and restart button light indicators.
Manual Procedures:	Press the restart button to start the application. Press the emergency button whenever you want to introduce an alarm condition.
Tips:	Inventory management can be based on Boolean variables again…
	You can generate random integer numbers from the current value of a timer that is periodically reset. Whenever a random value so generated cannot be used, consider the immediately above...

And now that the fifth mission has been accomplished…

So, here we are, at the end of the fifth and final ITS PLC mission! Congratulations for such an extraordinary job. This was definitely a very interesting mission! Had you ever wondered on how to generate random numbers in a PLC? Was the reflection on ASCII code and packed BCD helpful? Do you know better the available instructions and features of your PLC for code conversion? In short, we can surely state that this mission, apart from being very appealing visually, allowed us to survey in an organised fashion a set of challenging but central issues for any PLC programmer.

This mission is therefore almost finished and so is the ITS PLC game itself. Still remaining are the last TS SUPER and ITS DEEP proposals. And in these, and in the many others that one can imagine, there is still much more room for discovering more exciting challenges around the ITS PLC automatic warehouse.

ITS SUPER – Integration with supervisory systems and human-machine interface consoles

Most of the tasks proposed in this mission can indeed benefit a lot from the integration of an HMI console. For instance, with such an equipment you are able to:

- Issue the moving, storage and retrieval orders from the HMI console instead of just from the virtual command buttons;
- Get more detailed error codes than just the short blinking of a light indicator;
- Set and read the values for the configuration variables such as Place, Box_Id and Class_Id;
- Analyze and initialise the inventory tables at your will;
- Know the stored boxes according to their classes and codes.

Yet, if a SCADA system is available, then more interesting and advanced scenarios can be considered. For instance:

- Changing dynamically the areas allocated to product classes;
- Knowing for how long each item is stored in the warehouse;
- Periodic reallocation of the stored items, allowing a better management of the available space;
- Defining and selecting different criteria for storage and retrieval of items;
- Allowing that certain classes of items can only be requested by credentialed operators;
- Creating and managing queues for storage and retrieval requests;

- Logging availability and unavailability periods of the warehouse;
- Logging emergency conditions and corresponding duration;
- Detailed logging of all the storage and retrieval activities;
- Logging of any of the detected abnormal conditions proposed in the ITS DEEP package that follows.

ITS DEEP – Detection and handling of error conditions

As in Mission 4, some of the available sensors were ignored in the solutions provided for the current mission, as they are not fundamental for insuring the proper functioning of the automatic warehouse. This is true for sensors 0, 2 and 4. Yet, these are very useful for detecting error conditions. For instance:

- Devise an alarm procedure for the case of sensor 0 being active and the transelevator is not expected at position 51, and another one for the case of sensor 0 not being active and transelevator is expected to be at position 51;
- Devise an alarm procedure for the case of sensors 2 or 4 not being activated during a transfer operation, and another one for the case of sensors 2 or 4 being active while the transferring platform is expected to be retracted.

Additional abnormal events for which devising an effective detection and handling procedures is a worthwhile job, are the following ones:

- Moving orders while the platform is not completely retracted;
- Failure in transferring a box from the input transfer chariot to the transelevator;
- Failure in transferring a box from the transelevator to the output transfer chariot;
- Loss of a box while it is being carried on the transelevator;
- Lengthy delay of the transelevator in reaching the intended position;
- Lengthy unavailability of the output transfer chariot;
- Lengthy unavailability of the input transfer chariot;
- Unexpected departure of the output transfer chariot;
- Unexpected departure of the input transfer chariot;
- Failure of a sensor and corresponding identification – including if stuck at 0 or 1;

- Failure of an actuator and corresponding identification – including if stuck at 0 or 1.

From here, you are challenged to try to develop some effective procedures for detecting and dealing with the stated abnormal conditions, which you can force or simulate by using the valuable ITS PLC interaction features. Then, conduct some tests to properly assess the efficiency of your error detection techniques and draw your own conclusions.

PART 3

THE SOLUTIONS

Most likely, your secret desire that the solutions for the proposed problems were provided in the programming language of your PLC will not come true. But, having so many PLC brands and models, the probability of this happening was, in fact, very low!

So, given the impossibility to present the solutions in the preferred language of each reader, it was decided to do it in a standard language of growing acceptance, which is also very efficient, highly expressive, quite understandable – even to those who have never used it – and easily translatable into any other PLC programming language: Structured Text.

The Structured Text and the IEC 61131-3 Standard

Structured Text (or just ST for short) is a modern textual PLC programming language considered in the IEC 61131-3 standard. Its simplicity and expressiveness is bringing it an ever increasing number of enthusiasts among PLC programmers and manufacturers.

If this is the first time you are dealing with this language, then consider this as an additional point of interest in this workbook. Surely, you will soon be very enthusiastic about structured text. And if you happen to know some general high-level computer programming languages, such as BASIC, Pascal or C, then you'll notice that ST is, after all, a very easy language to understand and use, and to some extent, already known to you.

For many authors, ST is the programming language that allows the fastest and the most efficient development and use of modern PLC based control systems. Whereas this is obviously arguable, the fact is that the benefits of this programming language are particularly conspicuous for complex applications.

Yet, another feature has contributed greatly in the last years to the efficient design, implementation and re-engineering of controlling solutions based on PLCs: the IEC 61131 standard. If you do not know this standard yet, then the time has come for you to know it. If are working or studying at a University, you will surely find it at the library of your institution. If this is not the case, then consulting some of the numerous books, articles and websites that dedicate considerable attention to this standard, and in particular to its Part 3 (IEC 61131-3), will surely help you. Hence, presenting the solutions of suggested exercises in ST language also works as a foundation to stimulate you to know (or to know better) the IEC 61131 standard, although many of its features are not reflected in the proposed resolutions.

Yet, if it is your intention to continue to adopt a programming language that "has nothing to do with ST" – as ladder diagrams, functional block diagrams or instruction list – then, the IEC 61131-3 standard also fits your interests and needs. This is because these traditional programming languages are also included in the standard. Any of these languages could therefore have been chosen for

presenting the solutions for the proposed exercises, while keeping the purposes of compliance with, and dissemination of, the IEC 61131-3 standard. The stated languages were not adopted in here because it is our conviction that ST is the clearest, simplest and most efficient language for introducing and explaining a PLC program, especially to an extremely heterogeneous audience.

The IEC 61131-3 standard also covers the SFC (Sequential Function Chart) graphical language in order to structure, sequence and develop a controlling program. Hence, SFC is not really an alternative to ST or to any other programming language covered in the standard. Yet, the SFC language is one of the most interesting and important aspects of IEC 61131-3. It is often confused with the *GRAFCET* modelling language, as defined in the first version of the IEC 60848 standard, as the result of their graphical and semantic similarities; but SFC and *GRAFCET* have very different scopes. Nevertheless, *GRAFCET* is a major gateway to the understanding of some major features of the IEC 61131-3 standard, particularly in code organisation. This is another reason for the widespread use of *GRAFCET* in this textbook over other possible specification methodologies and languages.

If the PLC you are using is minimally compliant with the IEC 61131-3 standard but it only supports the more classical languages, then you can add another point of interest to the suggested exercises: the translation of the presented solutions to "your" programming language, while adopting the semantics and the representative symbols provided in the IEC 61131-3 standard. There are, after all, a lot of interesting subjects in this workbook!

Finally, if you are already an ST enthusiast and usually develop your PLC programs in accordance with the IEC 61131-3 standard, then you will certainly find the presented programs somehow inelegant, inefficient, and even sometimes non-conforming to the spirit of the standard. While agreeing completely that ST and the IEC 61131-3 standard would allow for coding the solutions in some ways that are far more interesting than those adopted – for example, by using enumerated data types to deal with state machines changes, or by encapsulating and reusing user code in functions and function blocks – it must be emphasised that such approaches would certainly introduce some confusion and misinterpretation into readers unfamiliar to both, the standard and the adopted programming language. And the presented solutions are expected to have two merits: firstly, and the most important, to be understandable to any PLC programmer, even if only minimally qualified; secondly, to provide to every reader the additional exercise of translating and optimising them to the programming language(s) of interest. Consequently, if you are an enthusiast of ST, take the presented solutions as pieces of code to optimise according to the IEC 61131-3 standard – or according to the IEC 61499 standard, if you happen to know this one too.

Literature and Supporting Materials

If you wish to initiate yourself in the Structured Text programming language and in the IEC 61131 standard, you have many departing options. Finding the IEC 61131 standard in a library should not be difficult for you. Another possibility is to acquire it from the "International Electrotechnical Commission" – http://www.iec.ch.

The most interesting part of the standard is Part 3, dedicated entirely to programming languages. It is commonly called the IEC 61131-3 standard. This is presented, discussed and exemplified in several texts. For instance, in the following books:

- Robert W. Lewis – "*Programming Industrial Control Systems Using IEC 1131-3*" (IEE Control Engineering Series), 1998 - ISBN: 0-85296-950-3;

- Karl-Heinz John – "*IEC 61131-3: Programming Industrial Automation Systems: Concepts and Programming Languages, Requirements for Programming Systems, Aids to Decision-Making Tools*". 1995, Springer Verlag. ISBN 3-540-67752-6. Part of this book is available on the Internet from the author's personal web page: http://www.fen-net.de/karlheinz.john/.

A must-visit site on PLCs and PLC standards is "PLCopen" web site – www.plcopen.org. Plentiful and useful information can be found there.

Submitting a "structured text programming" search to Google, or to any other search engine, leads to an endless list of sites and interesting articles. Among them, the book "Automating Manufacturing Systems with PLCs" by Hugh Jack, available at the time of this publication from the site claymore.engineer.gvsu.edu/~jackh/books/plcs. This is absolutely recommended reading.

If you do not happen to have a PLC compliant with the IEC 61131-3 standard but you are willing to try start programming in ST, then we suggest a tour to the world of the "soft PLCs". Most of these software packages are fully IEC 61131-3 compliant. They are expensive, but their manufacturers offer free demo versions that you can use for training. Most of these demo versions are well documented and include very interesting exemplifying applications. Thus, whilst somehow limited, free soft PLCs are excellent resources and points of departure for programming according to IEC 61131-3 standard languages, including ST. We suggest you try the following products:

- ISaGRAF – http://www.isagraf.com;

- MULTIPROG – http://www.kw-software.com;

- CoDeSys – http://www.3s-software.com;

- TwinCAT PLC – http://www.beckhoff.com.

So, if you do not know "soft PLCs" yet, then grasp this excellent opportunity to finally start dealing with such a remarkable resource that is currently controlling many of the modern industrial facilities. Another benefit from having read this book!

About the Solutions

After having made some considerations about ST and the IEC 61131-3 standard, the time has finally come to reveal the solutions to the proposed problems.

So, each resolution starts by a more or less brief and justified description of the procedures used

for solving the current task. Then, the corresponding PLC program, properly commented and coded in ST language, is provided. Since, in most cases, tasks complement themselves as a mission progresses, the resolution of a task is usually the starting point to solve the next. Consequently, much of the code developed for solving a task tends to be reused in subsequent tasks. Hence, we emphasise here again the importance of fully understanding the resolution of a task before trying to solve the next.

In the case of not so trivial tasks or when the procedures to follow are likely prone to ambiguity, program development is based on a prior and duly explained *GRAFCET* model. This specification is developed in accordance with the second edition of the IEC 60848 standard. Since this version of the standard is relatively new, it is likely that some readers still do not know much about it. Namely, the changes introduced relatively to the first edition. So, those readers may find some odd aspects in a few *grafcets*; for example, the allocation of actions to transitions.

Having in mind the readers less familiar with the current *GRAFCET* standard, the presented solutions make notice of some of the advances introduced in the second edition of the IEC 60848 standard. Yet, such enlightenments do not intend to replace the reading of the standard. In fact, the promotion of the second edition of the IEC 60848 standard – which differs much more from the SFC language than the first edition – is another aim of this book.

The declaration and initialization of variables is an important issue in any programming task; but this is particularly relevant when a job requires programming according to the IEC 61131-3 standard. Consequently, the declaration of variables has its own space in the presented solutions, and to which special attention should also be given by you. This is mostly because variable declaration also includes the corresponding initialization, implicit or explicit. And many of the declared variables do require a precise initial value. For example, the allocation of the initial TRUE value to a state variable reveals or confirms that it represents an initial step or state. Hence, the declaration of a variable should not be seen as a mere formalism of minor interest to the presented solutions, but rather as essential information.

For the sake of the organisation of the solutions, variable declarations are grouped in six sets: one for those variables that are common to all missions, and five for those that appear within each mission. The first group includes only the I/O variables and is presented in the next section. Later on, when the solutions for the twelve tasks of each mission are provided, then the corresponding variables and the standard function block instances will be declared.

As a final note, it is worth emphasising, once again, that the presented solutions are just “possible solutions”; not the *only* or the *best* solutions in any way. You are thus invited to find your own solutions and to compare them to those herein provided, while trying to find comparative advantages and disadvantages. The writing of a notebook, reflecting the personal conclusions about the outcome of each task and mission, is the ultimate purpose of this book, and surely one of the most important achievements for trainers and trainees alike.

Declaring I/O variables

Variables denoting the input and output parameters of a computer program are generally referred to as "I/O variables". I/O variables are central to any PLC program, since they are the means of information interchange between the software PLC application and the hardware peripherals such as sensors, actuators and human-machine interface devices.

The ITS PLC plants are no exception to this rule, requiring thus the correct information interchange between the virtual devices and the external controlling PLC. For this, it is required that the PLC continuously receive the information about the current state of the plant, in the form of input variables of the PLC, and that plant change its state in accordance with the appropriate PLC controls, reflected in the output variables of the PLC. This exchange of information requires a physical connection between the PLC and the USB interface card accompanying the ITS PLC software. It is this physical connection that defines the mapping of the I/O variables of the controlling PLC into the I/O variables of the ITS PLC virtual systems.

The declaration of I/O variables has three main purposes: to provide them with names that can be used during the programming, to define their types, and to assign them to physical PLC addresses. As such, when declaring an I/O variable, each programmer names it at his own will and maps it according to the physical requirements of his PLC. Common to all declarations is just the statement that each I/O variable is a Boolean variable. Consequently, the declaration of I/O variables provided in here is necessarily very general.

The IEC 61131-3 standard considers the possibility of assigning a physical PLC space (i.e., a memory, input or output address) to a variable when this is declared. This feature, which is particularly useful for declaring I/O variables, makes use of the "AT" attribute. Thus, for a PLC with 8-bit input and output modules, I/O variables could be declared like this:

```
(*************************************************
       Declaration of Input and Output Variables
*************************************************)

VAR_INPUT
    In_0 AT %IX0.0 : BOOL; (* Sensor 0 *)
    In_1 AT %IX0.1 : BOOL; (* Sensor 1 *)
    In_2 AT %IX0.2 : BOOL; (* Sensor 2 *)
    In_3 AT %IX0.3 : BOOL; (* Sensor 3 *)
    In_4 AT %IX0.4 : BOOL; (* Sensor 4 *)
    In_5 AT %IX0.5 : BOOL; (* Sensor 5 *)
    In_6 AT %IX0.6 : BOOL; (* Sensor 6 *)
    In_7 AT %IX0.7 : BOOL; (* Sensor 7 *)
    In_8 AT %IX1.0 : BOOL; (* Sensor 8 *)
    In_9 AT %IX1.1 : BOOL; (* Sensor 9 *)
    In_10 AT %IX1.2 : BOOL; (* Sensor 10 *)
```

```
    In_11 AT %IX1.3 : BOOL; (* Manual/Automatic mode Switch *)
    In_12 AT %IX1.4 : BOOL; (* Start Button *)
    In_13 AT %IX1.5 : BOOL; (* Stop Button *)
    In_14 AT %IX1.6 : BOOL; (* Restart Button *)
    In_15 AT %IX1.7 : BOOL; (* Emergency Button *)
END_VAR

VAR_OUPUT
    Out_0 AT %QX0.0 : BOOL; (* Actuator 0 *)
    Out_1 AT %QX0.1 : BOOL; (* Actuator 1 *)
    Out_2 AT %QX0.2 : BOOL; (* Actuator 2 *)
    Out_3 AT %QX0.3 : BOOL; (* Actuator 3 *)
    Out_4 AT %QX0.4 : BOOL; (* Actuator 4 *)
    Out_5 AT %QX0.5 : BOOL; (* Actuator 5 *)
    Out_6 AT %QX0.6 : BOOL; (* Actuator 6 *)
    Out_7 AT %QX0.7 : BOOL; (* Actuator 7 *)
    Out_8 AT %QX1.0 : BOOL; (* Start button light indicator *)
    Out_9 AT %QX1.1 : BOOL; (* Restart button light indicator *)
END_VAR

(************************************************
               End of Declaration
************************************************)
```

Most PLC programming environments adopt procedures more or less similar to this for declaring variables. But, admitting that this statement is confusing for those readers who are less familiar with the IEC 61131-3 standard, the central aspects to retain from the variables declaration presented above are:

- For all the tasks of all the missions, Boolean input variables In_0, In_1, …, In_10 map, in this order, the logic levels of virtual sensors 0, 1, …, 10;
- For all the tasks of all the missions, Boolean input variables In_11, In_12, …, In_15 map, in this order, the logic levels of the Mode Switch and of the Start, Stop, Restart and Emergency buttons;
- For all the tasks of all the missions, Boolean output variables Out_0, Out_1, …, Out_7 map, in this order, the logic levels of virtual actuators 0, 1, …, 7;
- For all the tasks of all the missions, Boolean output variables Out_8 and Out_9, map the logic levels of the Light Indicators of the Start and Restart buttons, respectively.

Note that the IEC 61131-3 standard makes it possible to initialise a variable during the corresponding declaration. Yet, this possibility is not very interesting for I/O variables – in fact, it does not even make sense to initialise an input variable. This is why I/O variables were not initialised in the above declaration. Yet, in future declarations, variables initialization will be a major issue.

MISSION 1: AUTOMATED CONVEYING AND SORTING OF CASES ON PALLETS

Variables and function blocks instances within Mission 1

According to the IEC-61131 standard, the code of a PLC program must be headed by the declaration of variables and function blocks instances used within such program. Yet, presenting the solutions for the twelve tasks that make up each mission according to this rule would lead, in most cases, to precede the PLC code by a long list of variables and function blocks already introduced and explained in several previous tasks.

Hence, to avoid tedious and subsequent declarations, which would contribute little or nothing to the understanding of the presented programs, while simplifying simultaneously the organisation and the reading of the text, it was decided to join in a single table the declaration of all the variables and function blocks instances within the twelve tasks of each mission. This joint statement, which opens the presentation of the solutions to each mission, is presented in accordance with the IEC 61131-3 standard, and must be taken into account in assessing the PLC code corresponding to the solution of each task. In particular, it is especially important to observe the initial value assigned to each variable.

It is important to state that, according to the IEC 61131-3 standard, the declaration of a variable allows one to assign it an initial value distinct from the default one, which depends on the variable type – for instance. Booleans, integers and reals are all initiated at 0, by default. As such, it may seem odd that the declaration of some variables includes the corresponding initialization to its default value. For instance, the reader will notice that many Boolean variables are explicitly initialised to FALSE. Yet, such "redundancy" is meant to explicitly reveal the initial value of some particularly important variable. This is the case, for example, of the initial value of a state variable, which typically reflects an initial condition provided in the scenario where the task takes place. Therefore, it must be initialised to TRUE or FALSE accordingly, hopefully in an explicit manner.

According to this principle, it was decided, in variables declarations, to explicitly initialise all the variables whose initial value is relevant for the application, and not just those whose default initial value does match the value of interest. Experience shows this is a worthy approach as most newbie trainees tend to incorrectly understand the absence of an explicit initial value for a variable as a "don´t care" initial value, instead of recognising the value of interest in the default initial value. Consistently, it was decided to not explicitly initialise all the variables whose initial value is irrelevant.

Comments included in the corresponding declarations reveal the set of tasks in which each variable and function block instance appears. Yet, one should note that some variables are expected to take distinct initial values for distinct tasks. The decision of declaring all the variables within a mission in a single table makes it impossible to support such diversity of initial values in compliance with the IEC 61131-3 standard. To minimise this problem, comments were included in variable declarations

for which multiple initial values exist, revealing the corresponding initial value for each task in which it appears. Comments also highlight those purely illustrative initial values, such as configuration parameters, recipe tables and others, which trainers and trainees are free to change at their will. Variables are grouped according to their context and alphabetically ordered, making it easier to find the declaration of a particular variable within the declaration table. Variables declaration precedes the declaration of function block instances facilitating the reading of the declaration table.

These explanations having been provided, they are followed by the declaration of the variables used in the twelve tasks that make up Mission 1, which corresponds to the following list:

```
(*****************************************************
        Declaration of variables and IEC 61131-3
      function blocks instances used within Mission 1
*****************************************************)

VAR
    (* Declaration and initialization of variables *)

    (* Variables assigned to entry conveyer control *)

    Count : WORD := 16#8000;                              (* Tasks 3, 5-12 *)
    (* Ring counter, starting at zero -> 8000H *)
    Entry_busy : BOOL := FALSE;                           (* Tasks 2-3, 5-12 *)
    (* Acceptance of pallets: 0 -> Accepts *)
    Mem_release_timeout : BOOL;                           (* Tasks 9-12 *)
    (* Holding Memory for Release_Timeout *)
    Pallet_on_entry : BOOL := FALSE;                      (* Tasks 1-3, 5-12 *)
    (* Carrying state: 0 -> No pallets *)
    Queue : WORD;                                         (* Tasks 8-12 *)
    (* Heights of the pallets *)

    (* Variables assigned to turntable control *)

    Charging : BOOL := FALSE;                             (* Tasks 4-12 *)
    (* Possible state (grafcet step) *)
    Discharge_direction : BOOL := FALSE;                  (* Tasks 6-12 *)
    (* If 0, discharges onto the right exit conveyer *)
    Discharging : BOOL := FALSE;                          (* Tasks 4-12 *)
    (* Possible state (grafcet step) *)
    Idle : BOOL := TRUE;                                  (* Tasks 4-12 *)
    (* Initial state (grafcet step) *)
    Mem_timeout_turntable : BOOL := FALSE;                (* Tasks 11-12 *)
    (* Holding Memory for Timeout_turntable *)
    Turns_charged : BOOL := FALSE;                        (* Tasks 4-12 *)
    (* Possible state (grafcet step) *)
    Turns_discharged : BOOL := FALSE;                     (* Tasks 4-12 *)
```

```
(* Possible state (grafcet step) *)
Turntable_busy : BOOL := FALSE;                          (* Tasks 5-12 *)
(* Acceptance of pallets: 0 -> Accepts *)

(* Variables assigned to right exit conveyer control *)

Mem_timeout_right : BOOL := FALSE;                       (* Tasks 11-12 *)
(* Holding Memory for Timeout_right *)
Pallet_on_right : BOOL := FALSE;                         (* Tasks 5-12 *)
(* Carrying state: 0 -> No pallets *)
Right_busy : BOOL := FALSE;                              (* Tasks 7-12 *)
(* Acceptance of pallets: 0 -> Accepts *)

(* Variables assigned to left exit conveyer control *)

Left_busy : BOOL := FALSE;                               (* Tasks 7-12 *)
(* Acceptance of pallets: 0 -> Accepts *)
Mem_timeout_left : BOOL := FALSE;                        (* Tasks 11-12 *)
(* Holding Memory for Timeout_left *)
Pallet_on_left  : BOOL := FALSE;                         (* Tasks 6-12 *)
(* Carrying state: 0 -> No pallets *)

(* Variables assigned to plant mode control *)

Automatic : BOOL := FALSE;                               (* Tasks 10-12 *)
(* Possible mode (grafcet step) - initial for Task 10 *)
Cleaning : BOOL := FALSE;                                (* Tasks 11-12 *)
(* Possible mode (grafcet step) *)
Plant_idle : BOOL;                                       (* Tasks 10-12 *)
(* Plant is Idle *)
Ready : BOOL := FALSE;                                   (* Tasks 10-12 *)
(* Possible mode (grafcet step) *)
Shutdown : BOOL := FALSE;                                (* Tasks 10-12 *)
(* Possible mode (grafcet step) *)
Standby : BOOL := TRUE;                                  (* Tasks 11-12 *)
(* Possible mode (grafcet step) - initial for Tasks 11-12 *)

(* Functional Blocks Instances *)

(* Falling edge (TRUE to FALSE) events detection *)

F_In_0 : F_TRIG;                                         (* Tasks 2-3, 5-12 *)
(* In_0 falling transition *)
F_In_3 : F_TRIG;                                         (* Tasks 1-12 *)
(* In_3 falling transition *)
F_In_7 : F_TRIG;                                         (* Tasks 4-12 *)
(* In_7 falling transition *)
```

```
    F_In_8 : F_TRIG;                                    (* Tasks 6-12 *)
    (* In_8 falling transition *)
    F_In_9 : F_TRIG;                                    (* Tasks 5-12 *)
    (* In_9 falling transition *)
    F_In_10 : F_TRIG;                                   (* Tasks 6-12 *)
    (* In_10 falling transition *)

    (* Rising edge (FALSE to TRUE) events detection *)

    R_Cleaning : R_TRIG;                                (* Tasks 11-12 *)
    (* Cleaning rising transition *)
    R_In_0 : R_TRIG;                                    (* Task 1 *)
    (* In_0 rising transition *)
    R_Plant_idle : R_TRIG;                              (* Tasks 11-12 *)
    (* Plant_idle rising transition – Plant has stopped *)

    (* Timers (on-delay) *)

    Auto_shutdown_blinker_1 : TON;                      (* Task 12 *)
    (* Timer 1 for Auto_shutdown_blinker *)
    Auto_shutdown_blinker_2 : TON;                      (* Task 12 *)
    (* Timer 2 for Auto_shutdown_blinker *)
    Cleaning_blinker_1 : TON;                           (* Tasks 11-12 *)
    (* Timer 1 for Cleaning_blinker *)
    Cleaning_blinker_2 : TON;                           (* Tasks 11-12 *)
    (* Timer 2 for Cleaning_blinker *)
    Release_Timeout : TON;                              (* Tasks 9-12 *)
    (* Timeout assigned to entry conveyer control *)
    Shutdown_blinker_1 : TON;                           (* Tasks 10-12 *)
    (* Timer 1 for Shutdown_blinker *)
    Shutdown_blinker_2 : TON;                           (* Tasks 10-12 *)
    (* Timer 2 for Shutdown_blinker *)
    Timeout_entry : TON;                                (* Tasks 11-12 *)
    (* Timeout in entry conveyer control *)
    Timeout_left : TON;                                 (* Tasks 11-12 *)
    (* Timeout in left exit conveyer control *)
    Timeout_right : TON;                                (* Tasks 11-12 *)
    (* Timeout in right exit conveyer control *)
    Timeout_turntable : TON;                            (* Tasks 11-12 *)
    (* Timeout in turntable control *)

    (* Counters (down) *)

    Batch_counter : CTD;                                (* Task 12 *)
    (* Pallet counter in batch conveying *)

END_VAR
```

```
(*******************************************************
                  End of declaration
*******************************************************)
```

Resolution of Task 1

According to the task statement, the entry conveyer must be running if there is a pallet on it, and stopped otherwise. Thus, if one assumes that these complementing states are given by Pallet_on_entry = TRUE and Pallet_on_entry = FALSE, respectively, being Pallet_on_entry an internal (or memory) Boolean variable, then Out_1 is trivially computed as:

```
Out_1 := Pallet_on_entry
```

Yet, since no sensor provides the information whether the entry conveyer handles or does not handle a pallet, the logic value of Pallet_on_entry has to be computed from the movement of pallets. This is easy: the arrival of a pallet onto the entry conveyer sets Pallet_on_entry, and the departure of the pallet resets Pallet_on_entry. Note that Pallet_on_entry starts FALSE, as the entry conveyer initially has no pallets.

The entry and departure of pallets onto and from to the entry conveyer are "events", since they are both recognised by logic transitions: the first, by the FALSE to TRUE transition of In_0; the second by the TRUE to FALSE transition of In_3. Thus, the setting and resetting of Pallet_on_entry are for ↑In_0 and ↓In_3, respectively.

The code implementing the solution for the present task is the following:

```
(*************************************
          Mission 1 - Task 1
*************************************)

(**** DETECTION OF RELEVANT EVENTS ****)

R_In_0(CLK := In_0);
(* R_In_0.Q is TRUE when In_0 transits from FALSE to TRUE, meaning that a pallet has
arrived onto the entry conveyer *)

F_In_3(CLK := In_3);
(* F_In_3.Q is TRUE when In_3 transits from TRUE to FALSE, meaning that a pallet has
departed from the entry conveyer *)

(**** ENTRY CONVEYER CONTROL ****)

(* Updating the state of the entry conveyer – given by Pallet_on_entry - according to
```

```
the events detected above *)

IF R_In_0.Q THEN (* When a pallet arrives *)
    Pallet_on_entry := TRUE; (* Set Pallet_on_entry *)
END_IF;

IF F_In_3.Q THEN (* When a pallet departs *)
    Pallet_on_entry := FALSE; (* Reset Pallet_on_entry *)
END_IF;

(* The current state of the conveyer defines Out_1 *)
Out_1 := Pallet_on_entry;

(*****************************************
              End of Program
*****************************************)
```

Resolution of Task 2

The current problem is quite easy, but it deals with a set of matters that are central to the solving of most tasks of Mission 1 – in fact, to the solving of most conveying control problems of the real world.

One must start by noting that the conveying of a pallet has three distinct phases. In the case of the entry conveyer, these are:

- Charging – requires the entry and the feeder conveyer to be both running. The charging phase starts when a pallet is detected at sensor 0, and ends when this sensor stops detecting the pallet. Thus, during charging, the logical value of In_0 is TRUE;
- Carriage – during carriage just the entry conveyer is required to run, moving a pallet that is not detected by any sensor. Thus, during carriage, both In_0 and In_3 are FALSE;
- Discharging – requires the entry conveyer and the rollers of the turntable to be both running. Discharging starts when the pallet is detected by sensor 3, and ends when this sensor stops detecting the pallet. Thus, during discharging, the logical value of In_3 is TRUE.

Since charging a pallet onto the entry conveyer is synonymous of discharging it from the feeding conveyer – as discharging a pallet from the entry conveyers is equivalent to charging it onto the turntable – one concludes that when the entry conveyer is unavailable to receive a new pallet, because it is still moving the previous one, then the feeding conveyer cannot go into its discharge phase, i.e., it cannot move a pallet beyond sensor 0. In this situation the feeding conveyer must stop and wait for the entry conveyer not to be handling any pallet, becoming thus available for charging.

A common and simple solution for implementing this requirement is to assume that a conveyer A

issues the signal A_busy = TRUE when, for some reason, it is not ready for charging a pallet. By reading this signal, a conveyer B, which feeds A, gets to know that it must not run to discharge a pallet onto conveyer A. This will only be possible from the moment in which conveyer A issues the signal A_busy = FALSE.

Adapting this technique to the current problem is trivial: the entry conveyer sets Entry_busy just after it has been charged – i.e., when it reads a TRUE to FALSE transition in In_0 – and resets it just after discharging – i.e., when it reads a TRUE to FALSE transition in In_3. In turn, the controller of the feeding conveyer reads the Entry_busy signal when it detects a pallet at sensor 0, suspending the feeding process if the entry conveyer is busy.

Another important practical issue is to guarantee that the entry conveyer does not have a transient stop after discharging a pallet if there is already a pallet waiting to be charged at the end of the feeding conveyer. For this, the set/reset conditions for Pallet_on_entry have to change relatively to those of Task 1. One possibility is to issue the reset order when In_3 transits from TRUE to FALSE only if In_0 is FALSE at that moment. Alternatively, the setting of Pallet_on_entry can go for In_0 = TRUE and made dominant over the resetting. The solution presented below follows the second approach.

```
(*************************************
           Mission 1 - Task 2
*************************************)

(**** DETECTION OF RELEVANT EVENTS ****)

F_In_0(CLK := In_0);
(* A pallet was just charged onto the entry conveyer - i.e., discharged from the
 feeding conveyer *)
F_In_3(CLK := In_3);

(**** ENTRY CONVEYER CONTROL ****)

(* Updating Pallet_on_entry and Entry_busy *)

IF F_In_3.Q THEN (* A pallet abandons the conveyer*)
    Pallet_on_entry := FALSE; (* Reset Pallet_on_entry *)
    Entry_busy := FALSE; (* Reset Entry_busy *)
END_IF;

IF F_In_0.Q THEN (* When a pallet is charged *)
    Entry_busy := TRUE; (* Set Entry_busy *)
END_IF;

IF In_0 THEN (* If there is a pallet at the end of the feeder *)
    Pallet_on_entry := TRUE; (* Set Pallet_on_entry *)
```

```
END_IF;

(** IMPORTANT NOTE:
Note that the setting of Pallet_on_entry is now for In_0 and dominant over the reset
ting – since the setting instruction comes after the resetting. Thus, when F_In_3.Q
and In_0 are both TRUE, Pallet_on_entry stays TRUE and so does Out_1. This means that
the entry conveyer is kept running, without a transient FALSE, if a pallet exists at
the entry of the conveyer when the previous one is discharged **)

(* Computing Out_1: as in Task 1 *)
Out_1 := Pallet_on_entry;

(**** FEEDING CONVEYER CONTROL ****)

(* The feeding conveyer is not running if the entry conveyer is busy and there is a
pallet at the end of the feeder *)
Out_0 := NOT Entry_busy OR NOT In_0;
(* Or: Out_0 := NOT (Entry_busy AND In_0) *)

(***************************************
            End of Program
***************************************)
```

Resolution of Task 3

In this task the feeding conveyer is controlled as in Task 2, but the controller of the entry conveyer must have two changes:

- Pallet_on_entry goes FALSE when the entry conveyer discharges a pallet and goes empty;
- Entry_busy goes TRUE when there are two pallets on the entry conveyer.

It is thus important to have a counter to know how many pallets are on the entry conveyer. Most PLCs include several up and down counting functions – usually named CTUD counters – that may be used for signalling the two reference counting values for the present task: 0 (conveyer empty) and 2 (conveyer full).

Another possibility is to implement a counting function particularly well fitted to the general goals of Mission 1. In this case – and this will be justified in a later task – it makes sense to implement a very peculiar counter: a “ring counter”. By definition, incrementing a ring counter is synonymous to rotating its content one position left; decrementing it means to rotate its content one position right.

In the present case, a ring counter named “Count” is implemented using a 16-bit register, which is the typical length of most PLC registers. The 0 counting value is coded as the word having the TRUE value for the leftmost bit and FALSE for all the others. Thus, the Count register changes

as shown in Figure M1T3, although in this task, it just goes from 0 to 2, to which correspond the hexadecimal codes 16#8000 and 16#0002, respectively. The left rotation (increment) is provided by the instruction Count := ROL(Count, 1), and the right rotation (decrement) by Count := ROR(Count, 1).

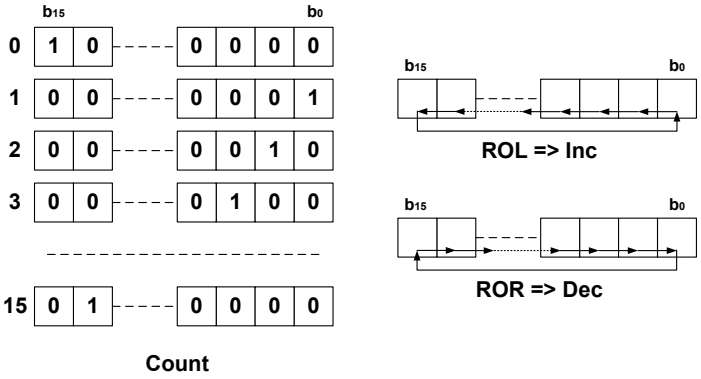


Figure M1T3 – *Counting evolution for a ring counter*

The still mysterious virtues of this counter will be revealed within the solution of a forthcoming task; namely, when it will be required to know the heights of the pallets existing on the entry conveyer and it is allowed to handle as many pallets as possible. Yet, it may already be stated that such virtues come from the fact that, for counting values greater than zero, the counting code closely matches the "queue" of pallets on the entry conveyer, where the "1" indicates the leading pallet and the "0's" to its right the pallets moving after.

The major differences between Task 2 and this one reflect the counting procedures and the updating of the variables Pallet_on_entry and Entry_busy:

- The Count register starts at 16#8000 coding the "0" value – check this in the declaration of variables within Mission 1;
- Every time a pallet is charged onto the entry conveyer the register is rotated left and if it reaches 2, then Entry_busy goes TRUE;
- Every time a pallet is discharged from the entry conveyer the Count register is rotated right and if it reaches 0, then Pallet_on_entry goes FALSE.

The complete coding for this task is as follows:

```
(*****************************************
          Mission 1 - Task 3
*****************************************)
```

```
(**** DETECTION OF RELEVANT EVENTS ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);

(**** ENTRY CONVEYER CONTROL ****)

(* If it carries at least one pallet: Pallet_on_entry = TRUE
If it carries two pallets: Entry_busy = TRUE *)

IF F_In_0.Q THEN (* When a pallet is charged *)
    Count := ROL(Count, 1); (* Increments counter *)

    IF (Count = WORD#16#2) THEN (* If Counter equals 2 *)
        Entry_busy := TRUE; (* Set Entry_busy *)
    END_IF;
END_IF;

IF F_In_3.Q THEN (* When a pallet is discharged *)
    Entry_busy := FALSE; (* Reset Entry_busy *)
    Count := ROR(Count, 1); (* Decrement counter *)

    IF (Count = WORD#16#8000) THEN (* If counter equals 0 *)
        Pallet_on_entry := FALSE; (* Reset Pallet_on_entry *)
    END_IF;
END_IF;

IF In_0 THEN (* If there is a pallet at the end of the feeder *)
    Pallet_on_entry := TRUE; (* Set Pallet_on_entry *)
END_IF;

Out_1 := Pallet_on_entry;

(**** FEEDING CONVEYER CONTROL ****)
(* As in Task 2 *)

Out_0 := NOT Entry_busy OR NOT In_0;

(****************************************
            End of Program
****************************************)
```

Resolution of Task 4

The *grafect* presented in Figure M1T4 reveals the five modes (or states) to be considered in the control of the turntable. These evolve as follows:

- Initially the turntable is "Idle", and thus ready to receive a pallet;
- When "Idle" and a pallet arrives at the end of the entry conveyer (In_3 = TRUE) the turntable goes into "Charging" mode;
- When the pallet is charged (In_6 = TRUE), the turntable enters the "Turns Charged" mode;
- From "Turns Charged" it goes into "Discharging" mode when the discharging position is reached (In_5 = TRUE);
- From "Discharging" it goes into "Turns Discharged" mode when the pallet abandons the turntable (↓In_7 = TRUE);
- From "Turns Discharged" the turntable returns to "Idle" mode when the charging reference is reached (In_4 = TRUE).

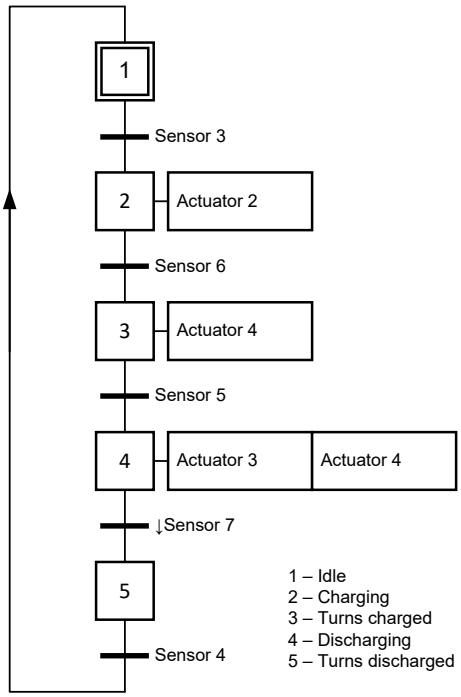


Figure M1T4 – *Grafcet for the control of the turntable in Task 4*

Since the turntable always discharges onto the right exit conveyer, control outputs (deduced from *grafcet* actions) must be provided as:

- Charging –> Out_2 = TRUE;
- Turns Charged –> Out_4 = TRUE;
- Discharging -> Out_3 = Out_4 = TRUE.

Thus:

- Out_2 = TRUE for Charging;
- Out_3 = TRUE for Discharging;
- Out_4 = TRUE for Turns Charged OR Discharging.

Developing a program for controlling the turntable according to the *grafcet* of Figure M1T4 is quite easy, as it just requires the proper coding of the above-stated procedures.

There are two main approaches for translating a *grafcet* into a PLC program: the synchronous and the asynchronous ones. The synchronous approach guarantees very naturally most GRAFCET rules, as it does not allow the *grafect* so coded to transit two or more consecutive steps in a single PLC scan, even for unstable situations. Yet, it results in longer programs and requires more time to program than the asynchronous approach. This second approach, although resulting in less code, requires special attention to be paid to certain "delicate" situations; for instance, it may lead to the firing of several consecutive transitions in a single PLC scan, a scenario which is only acceptable if no important actions are prevented from being performed.

Details about this interesting subject, not much referred to in scientific literature, go beyond the scope of this text. Yet, it is worth stating that all the solutions provided in the text resulting from the translation of a *grafect* into PLC code adhere to the synchronous methodology. When a particular translation requires special attention to be paid to a particular aspect, this is properly noted.

The translation of the *grafect* presented in Figure M1T4 into PLC code is simple to perform and quite easy to understand. It closely reflects the general procedures for this kind of translation and corresponds to the PLC code presented bellow.

```
(*************************************
           Mission 1 - Task 4
*************************************)

(**** DETECTION OF RELEVANT EVENTS ****)

F_In_7(CLK := In_7);
(* A pallet abandons the turntable, being charged onto the right exit conveyer *)
```

```
(**** TURNTABLE CONTROL ****)

(* Computing the evolution of the internal state of the turntable according to the
triggering of the transitions stated in the grafcet specification *)

IF Idle AND In_3 THEN (* Idle -> Charging *)
    Idle := FALSE;
    Charging := TRUE;
END_IF;

IF Charging AND In_6 THEN (* Charging -> Turns charged *)
    Charging := FALSE;
    Turns_charged := TRUE;
END_IF;

IF Turns_charged AND In_5 THEN (* Turns charged -> Discharging *)
    Turns_charged := FALSE;
    Discharging := TRUE;
END_IF;

IF Discharging AND F_In_7.Q THEN (* Discharging -> Turns discharged *)
    Discharging := FALSE;
    Turns_Discharged := TRUE;
END_IF;

IF Turns_Discharged AND In_4 THEN (* Turns discharged -> Idle *)
    Turns_Discharged := FALSE;
    Idle := TRUE;
END_IF;

(** IMPORTANT NOTE:
The state variable "Idle" is initially TRUE, since it maps the initial step of the
grafcet. All the other state variables are initially FALSE. Check this in the decla
ration of variables within the first mission *)

(* Dependency of the outputs (actions) from the state variables (steps) *)

Out_2 := Charging;
Out_3 := Discharging;
Out_4 := Turns_charged OR Discharging;

(***************************************
            End of Program
***************************************)
```

Resolution of Task 5

The exit conveyer requires less time to transfer a pallet than the turntable. Thus, in normal conditions, the exit conveyer is always ready to receive a pallet from the turntable and never handles more than one pallet simultaneously. Hence, the control developed in Task 1 for the entry conveyer can be reused in this task for the exit conveyer.

In turn, the synchronization between the entry conveyer and the turntable for pallet transferring can be developed by closely following the approach developed in Task 2 to synchronise the feeding and the entry conveyers. Thus:

- The turntable must issue Turntable_busy = TRUE when it is not ready to receive a pallet from the entry conveyer – i.e., from the moment In_3 transits from TRUE to FALSE until the turntable becomes Idle;
- The entry conveyer must stop when a pallet reaches sensor 3 and the turntable is busy.

The control of the feeding conveyer is kept unchanged.

The present task thus has a globally simple solution, which corresponds to the PLC code presented below.

```
(****************************************
           Mission 1 - Task 5
****************************************)

(**** DETECTION OF RELEVANT EVENTS ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_9(CLK := In_9); (* A pallet abandons the right exit conveyer *)

(**** RIGHT EXIT CONVEYER CONTROL ****)

(* Pallet_on_right represents the state of the right exit conveyer:
If it carries a pallet, then Pallet_on_right = TRUE *)

IF F_In_9.Q THEN (* When a pallet abandons the conveyer *)
    Pallet_on_right := FALSE; (* Reset Pallet_on_right *)
END_IF;

IF In_7 THEN (* When there is a pallet at the entry of the conveyer *)
    Pallet_on_right := TRUE; (* Set Pallet_on_right *)
END_IF;
```

```
(* Right exit conveyer runs for Pallet_on_right = TRUE *)
Out_6 := Pallet_on_right;

(**** TURNTABLE CONTROL ****)

IF Idle AND In_3 THEN
    Idle := FALSE;
    Charging := TRUE;
END_IF;

IF Charging AND In_6 THEN
    Charging := FALSE;
    Turns_charged := TRUE
END_IF;

IF Turns_charged AND In_5 THEN
    Turns_charged := FALSE;
    Discharging := TRUE;
END_IF;

IF Discharging AND F_In_7.Q THEN
    Discharging := FALSE;
    Turns_Discharged := TRUE;
END_IF;

IF Turns_Discharged AND In_4 THEN
    Turns_Discharged := FALSE;
    Idle := TRUE;
END_IF;

(* Defining Turntable_busy *)
IF F_In_3.Q THEN (* When a pallet is charged onto the turntable *)
    Turntable_busy := TRUE; (* Set Turntable_busy *)
END_IF;

IF Idle THEN (* When the turntable is Idle *)
    Turntable_busy := FALSE; (* Reset Turntable_busy *)
END_IF;

(* Definition of the output values *)
Out_2 := Charging;
Out_3 := Discharging;
Out_4 := Turns_charged OR Discharging;

(**** ENTRY CONVEYER CONTROL ****)
```

```
IF F_In_0.Q THEN
    Count := ROL(Count, 1);

    IF (Count = WORD#16#2) THEN
        Entry_busy := TRUE;
    END_IF;
END_IF;

IF F_In_3.Q THEN
    Entry_busy := FALSE;
    Count := ROR(Count, 1);

    IF (Count = WORD#16#8000) THEN
        Pallet_on_entry := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Pallet_on_entry := TRUE;
END_IF;

(* Redefining Out_1: the entry conveyer runs when it carries at least one pallet and
is allowed to move it *)
Out_1 := Pallet_on_entry AND (NOT Turntable_busy OR NOT In_3);

(**** FEEDING CONVEYER CONTROL ****)
(* As in Task 2 *)

Out_0 := NOT Entry_busy OR NOT In_0;

(***************************************
             End of Program
***************************************)
```

Resolution of Task 6

The solutions for Tasks 1 and 5 provide the basis for programming the control of the left exit conveyer.

In turn, allowing pallets to be discharged onto the right and left exit conveyers requires some changes to be made in the control of the turntable. The *grafcet* represented in Figure M1T6, which specifies the controlling algorithm to be implemented in the present task, shows such changes:

- The condition for the transition 4/5 is the disjunction of two events;
- Logical expression for Out_2 and Out_3 must include somehow the direction of the discharge.

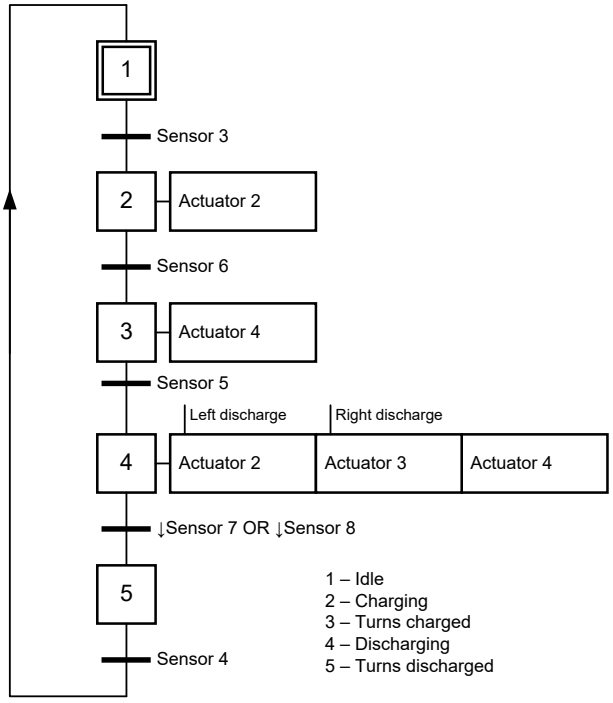


Figure M1T6 – *Turntable control with configurable discharging direction*

The first change is trivial to programming. For the second one, attention must be paid to the following:

Since there are two conveyers onto which the turntable can discharge, the direction of the discharge can be defined by a Boolean variable. Let's define such a variable as "Discharge_direction" and assume that Discharge_direction = FALSE means that the discharge should be performed onto the right exit conveyer. On this assumption, the output variables for the control of the turntable are formulated as:

- Out_2 = TRUE when Charging OR Discharging AND Discharge_direction = TRUE;
- Out_3 = TRUE when Discharging AND Discharge_direction = FALSE;
- Out_4 = TRUE when Turns_charged OR Discharging.

To alternate the direction of the discharge, it is required to complement "Discharge_direction" every time a pallet passes a given location before being discharged. In the future, the direction of the discharge will depend on the height of a case, which is recognised by the controller of the entry conveyer and passed to the controller of the turntable when a pallet is transferred from the first to the second. Hence, it makes sense to use the TRUE to FALSE transition of In_3 for complementing the

variable “Discharge_direction”, and include such a procedure in the control of the entry conveyer.

It is worth noting that, in this scenario, the variable “Discharge_direction” is complemented before a pallet starts being discharged onto an exit conveyer. Thus, if Discharge_direction is initialised as TRUE, then the first pallet is discharged onto the left exit conveyer.

```
(*************************************
           Mission 1 - Task 6
*************************************)

(**** DETECTION OF RELEVANT EVENTS ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_8(CLK := In_8); (* A pallet abandons the turntable, being charged onto the left
exit conveyer *)
F_In_9(CLK := In_9);
F_In_10(CLK := In_10); (* A pallet abandons the left exit conveyer *)

(**** LEFT EXIT CONVEYER CONTROL ****)

(* Pallet_on_left represents the state of the left exit conveyer:
If it carries a pallet, then Pallet_on_left = TRUE *)

IF F_In_10.Q THEN (* When a pallet abandons the conveyer *)
    Pallet_on_left := FALSE; (* Reset Pallet_on_left *)
END_IF;

IF In_8 THEN (* When there is a pallet at the entry of the conveyer *)
    Pallet_on_left := TRUE; (* Set Pallet_on_left *)
END_IF;

(* Left exit conveyer runs for Pallet_on_left = TRUE *)
Out_5 := Pallet_on_left;

(**** RIGHT EXIT CONVEYER CONTROL ****)
(* As in Task 5 *)

IF F_In_9.Q THEN
    Pallet_on_right := FALSE;
END_IF;

IF In_7 THEN
    Pallet_on_right := TRUE;
END_IF;
```

```
Out_6 := Pallet_on_right;

(**** TURNTABLE CONTROL ****)

IF Idle AND In_3 THEN
    Idle := FALSE;
    Charging := TRUE;
END_IF;

IF Charging AND In_6 THEN
    Charging := FALSE;
    Turns_charged := TRUE;
END_IF;

IF Turns_charged AND In_5 THEN
    Turns_charged := FALSE;
    Discharging := TRUE;
END_IF;

(* The turntable now discharges onto both conveyers. Thus: *)

IF Discharging AND (F_In_7.Q OR F_In_8.Q) THEN
    Discharging := FALSE;
    Turns_Discharged := TRUE;
END_IF;

IF Turns_Discharged AND In_4 THEN
    Turns_Discharged := FALSE;
    Idle := TRUE;
END_IF;

IF F_In_3.Q THEN
    Turntable_busy := TRUE;
END_IF;

IF Idle THEN
    Turntable_busy := FALSE;
END_IF;

(* Computing control outputs according to the current state of the turntable and the
direction of the discharge *)

Out_2 := Charging OR Discharging AND Discharge_direction;
(* Out_2 is TRUE if the discharge is onto the left conveyer *)
Out_3 := Discharging AND NOT Discharge_direction;
(* Out_3 is TRUE if the discharge is onto the right conveyer *)
```

```
Out_4 := Turns_charged OR Discharging;

(**** ENTRY CONVEYER CONTROL ****)

IF F_In_0.Q THEN
    Count := ROL(Count, 1);

    IF (Count = WORD#16#2) THEN
        Entry_busy := TRUE;
    END_IF;
END_IF;

IF F_In_3.Q THEN
    Discharge_direction := NOT Discharge_direction;
    (** Defining the direction of the discharge:
    The direction of the discharge changes every time a pallet is discharged from the
    entry conveyer onto the turntable. If Discharge_direction = FALSE, then the
    discharge is onto the right conveyer **)

    Entry_busy := FALSE;
    Count := ROR(Count, 1);

    IF (Count = WORD#16#8000) THEN
        Pallet_on_entry := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Pallet_on_entry := TRUE;
END_IF;

Out_1 := Pallet_on_entry AND (NOT Turntable_busy OR NOT In_3);

(**** FEEDING CONVEYER CONTROL ****)
(* As in Task 2 *)

Out_0 := NOT Entry_busy OR NOT In_0;

(***************************************
            End of Program
***************************************)
```

Resolution of Task 7

The unavailability of the exit elevators during alarm conditions (i.e., for In_15 = FALSE, since the emergency button has a normally closed contact) impacts the control of the exit conveyers in a very similar way to that of a "Busy" signal, as it defines that pallets are not allowed to go beyond the exit sensor of the exit conveyers. In turn, imposing that an exit conveyer never handles more than one pallet, requires that its controller provide a "Busy" signal to the turntable controller.

Thus, the control code developed in Task 6 for the exit conveyers must now be updated to include the support of alarm situations and the issuing of Busy signals. As a direct consequence, the turntable control also has to be modified in order to handle the busy signal provided from both exit conveyers.

In the present scenario, the turntable is only allowed to discharge onto an exit elevator if it is ready to receive a pallet. As such, one must introduce a slight change in the *grafcet* of Figure M1T6: actions "Actuator 2" and "Actuator 3" in step 4 which were only conditioned by "Discharge direction" are now also conditioned by the present state of the exit conveyers – Figure M1T7.

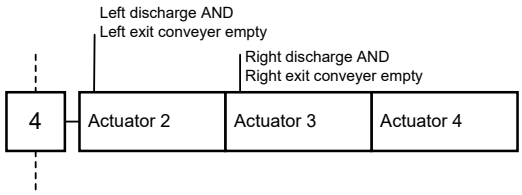


Figure M1T7 – *Changes in turntable control comparatively to Task 6*

The control code corresponding to Task 7 is listed bellow.

```
(*****************************************
          Mission 1 - Task 7
*****************************************)

(**** DETECTION OF RELEVANT EVENTS ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_8(CLK := In_8);
F_In_9(CLK := In_9);
F_In_10(CLK := In_10);

(**** LEFT EXIT CONVEYER CONTROL ****)

IF F_In_10.Q THEN (* When a pallet abandons the conveyer *)
```

```
    Pallet_on_left := FALSE; (* Reset Pallet_on_left *)
    Left_busy := FALSE; (* and is ready for charging *)
END_IF;

IF F_In_8.Q THEN (* When a pallet is charged *)
    Left_busy := TRUE; (* no more pallets are accepted *)
END_IF;

IF In_8 THEN (* When there is a pallet to charge *)
    Pallet_on_left := TRUE; (* Set Pallet_on_left *)
END_IF;

(* Left exit conveyer runs when it carries a pallet and there is no alarm condition
and a pallet at the end of the conveyer *)

Out_5 := Pallet_on_left AND (NOT In_10 OR In_15);
(* Note that alarm is given by In_15 = FALSE *)

(**** RIGHT EXIT CONVEYER CONTROL ****)

IF F_In_9.Q THEN (* When a pallet abandons the conveyer *)
    Pallet_on_right := FALSE; (* Reset Pallet_on_right *)
    Right_busy := FALSE; (* and is ready for charging *)
END_IF;

IF F_In_7.Q THEN (* When a pallet is charged *)
    Right_busy := TRUE; (* no more pallets are accepted *)
END_IF;

IF In_7 THEN (*When there is a pallet to charge *)
    Pallet_on_right := TRUE; (* Set Pallet_on_right *)
END_IF;

(* Right exit conveyer runs when it carries a pallet and there is no alarm and a pal
let at the end of the conveyer *)

Out_6 := Pallet_on_right AND (NOT In_9 OR In_15);
(* Note that alarm is given by In_15 = FALSE *)

(**** TURNTABLE CONTROL ****)

IF Idle AND In_3 THEN
    Idle := FALSE;
    Charging := TRUE;
END_IF;

IF Charging AND In_6 THEN
```

```
    Charging := FALSE;
    Turns_charged := TRUE;
END_IF;

IF Turns_charged AND In_5 THEN
    Turns_charged := FALSE;
    Discharging := TRUE;
END_IF;

IF Discharging AND (F_In_7.Q OR F_In_8.Q) THEN
    Discharging := FALSE;
    Turns_Discharged := TRUE;
END_IF;

IF Turns_Discharged AND In_4 THEN
    Turns_Discharged := FALSE;
    Idle := TRUE;
END_IF;

IF F_In_3.Q THEN
    Turntable_busy := TRUE;
END_IF;

IF Idle THEN
    Turntable_busy := FALSE;
END_IF;

(* Defining the outputs according to the current state of the turntable, the direction
of the discharge and the state of the exit conveyers *)

Out_2 := Charging OR Discharging AND Discharge_direction AND NOT Left_busy;
(* Left discharge requires left exit conveyer not busy *)
Out_3 := Discharging AND NOT Discharge_direction AND NOT Right_busy;
(* Right discharge requires right exit conveyer not busy *)
Out_4 := Turns_charged OR Discharging;

(**** ENTRY CONVEYER CONTROL ****)
(* As in Task 6 *)

IF F_In_0.Q THEN
    Count := ROL(Count, 1);

    IF (Count = WORD#16#2) THEN
        Entry_busy := TRUE;
    END_IF;
END_IF;
```

```
IF F_In_3.Q THEN
    Discharge_direction := NOT Discharge_direction;
    Entry_busy := FALSE;
    Count := ROR(Count, 1);

    IF (Count = WORD#16#8000) THEN
        Pallet_on_entry := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Pallet_on_entry := TRUE;
END_IF;

Out_1 := Pallet_on_entry AND (NOT Turntable_busy OR NOT In_3);

(**** FEEDING CONVEYER CONTROL ****)
(* As in Task 2 *)

Out_0 := NOT Entry_busy OR NOT In_0;

(***************************************
            End of Program
***************************************)
```

Resolution of Task 8

Comparatively to Task 7, the single difference for the present one is to manage to update Discharging_direction according to the height of the case that goes onto the turntable, when In_3 transits from TRUE to FALSE. Everything else remains unchanged.

The identification of the height of a case is provided by the reading of sensor 2 (In_2) at the moment its transporting pallet is charged onto the entry conveyer, that is to say, when In_0 transits from TRUE to FALSE. Yet, this being Boolean information acquired when a pallet goes onto the entry conveyer but only used to update "Discharging_direction" when such pallet goes onto the entry conveyer, one has that this information has to be stored somewhere while the pallet moves on the entry conveyer. Moreover, since the entry conveyer may handle several pallets simultaneously, there has to be room to store the heights of several pallets simultaneously and rules to properly manage both the stored information and the storage space. This is the problem to solve in this task.

A very simple solution for tracing and updating the information about the heights of the cases existing on the entry conveyer is to store it in a register, named "Queue", and proceed as follows:

Every time a pallet goes onto the entry conveyer, the Queue register is shifted left one position, and the rightmost bit gets the current value of In_2: TRUE if the arriving case is high, FALSE if it

is low. To implement this procedure, one must use the instruction SHL(Queue, 1), forcing then the rightmost bit of Queue on if In_2 is TRUE (i.e., the arriving case is high).

The usage of a ring counter to count the pallets on the entry conveyer – also moved left by one bit every time a pallet goes onto the entry conveyer – starts making sense now. If true, for counting values greater than zero, the position of the single "1" bit in the Count register is also the position where, in the Queue register, the information about the height of the leading pallet on the entry conveyer is stored. Thus, the result of the logical operation "Count AND Queue" computed when a pallet is discharged onto the turntable (but before decrementing Count) gives the height of the case existing on the pallet. Note this: since "Count" has its *kth* bit at "1" and all the others at "0", then Count AND Queue results in a word having an "0" in all its bits other than the *kth*, which gets the logic value of the *kth* bit of the Queue register – i.e., the height of the leading pallet. Thus, Count AND Queue results in a value different from zero if and only if the case in the leading pallet is high. The usage of a ring counter for counting the pallets on the entry conveyer is now justified: the content of the Count register acts thus as a mask revealing the *kth* bit of Queue every time the logic operation Count AND Queue is performed.

Figure M1T8 illustrates these concepts for two scenarios. In the first one there is a single pallet on the entry conveyer, which carries a high case. In the second one, there are two pallets on the conveyer; the leading pallet carries a low case and the following a high case. Note that this strategy remains valid if more than two pallets exist on the entry conveyer – in fact, it remains valid for *n*-1 pallets, *n* being the extension (number of bits) of Count and Queue registers. Thus, it will remain effective in future tasks where the entry conveyer is allowed to handle more than two pallets simultaneously.

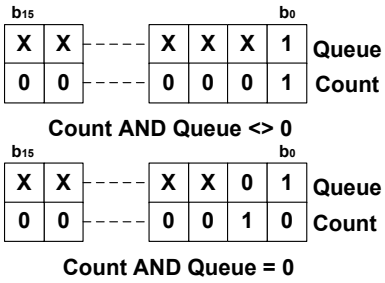


Figure M1T8 – *Getting the height of a case*

The program that solves Task 8 is the following:

```
(****************************************
          Mission 1 - Task 8
****************************************)

(**** DETECTION OF RELEVANT EVENTS ****)
```

```
F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_8(CLK := In_8);
F_In_9(CLK := In_9);
F_In_10(CLK := In_10);

(**** LEFT EXIT CONVEYER CONTROL ****)
(* As in Task 7 *)

IF F_In_10.Q THEN
    Pallet_on_left := FALSE;
    Left_busy := FALSE;
END_IF;

IF F_In_8.Q THEN
    Left_busy := TRUE;
END_IF;

IF In_8 THEN
    Pallet_on_left := TRUE;
END_IF;

Out_5 := Pallet_on_left AND (NOT In_10 OR In_15);

(**** RIGHT EXIT CONVEYER CONTROL ****)
(* As in Task 7 *)

IF F_In_9.Q THEN
    Pallet_on_right := FALSE;
    Right_busy := FALSE;
END_IF;

IF F_In_7.Q THEN
    Right_busy := TRUE;
END_IF;

IF In_7 THEN
    Pallet_on_right := TRUE;
END_IF;

Out_6 := Pallet_on_right AND (NOT In_9 OR In_15);

(**** TURNTABLE CONTROL ****)
(* As in Task 7 *)
```

```
IF Idle AND In_3 THEN
    Idle := FALSE;
    Charging := TRUE;
END_IF;

IF Charging AND In_6 THEN
    Charging := FALSE;
    Turns_charged := TRUE;
END_IF;

IF Turns_charged AND In_5 THEN
    Turns_charged := FALSE;
    Discharging := TRUE;
END_IF;

IF Discharging AND (F_In_7.Q OR F_In_8.Q) THEN
    Discharging := FALSE;
    Turns_Discharged := TRUE;
END_IF;

IF Turns_Discharged AND In_4 THEN
    Turns_Discharged := FALSE;
    Idle := TRUE;
END_IF;

IF F_In_3.Q THEN
    Turntable_busy := TRUE;
END_IF;

IF Idle THEN
    Turntable_busy := FALSE;
END_IF;

Out_2 := Charging OR Discharging AND Discharge_direction AND NOT Left_busy;
Out_3 := Discharging AND NOT Discharge_direction AND NOT Right_busy;
Out_4 := Turns_charged OR Discharging;

(**** ENTRY CONVEYER CONTROL ****)

IF F_In_0.Q THEN (* When a pallet is charged *)
    Count := ROL(Count, 1); (* Increments the counter *)

    IF (Count = WORD#16#2) THEN
        Entry_busy := TRUE;
    END_IF;

    Queue := SHL(Queue, 1); (* And updates the "Queue" as follows: *)
```

```
    (* Shifts left the "queue" register one bit, introducing the 0 value in its
    rightmost position *)

    IF In_2 THEN
        Queue := Queue OR WORD#1;
        (* But if the charged pallet is high, the value of the rightmost bit is
        changed to 1 *)
    END_IF;
END_IF;

IF F_In_3.Q THEN (* When a pallet abandons the entry conveyer, the corresponding
height will be checked *)
    Entry_busy := FALSE;

    IF (Count AND Queue) <> WORD#0 THEN
        (* If it is a high pallet... *)
        Discharge_direction := TRUE;
        (* ...it will be discharged onto the left conveyer *)
    ELSE
        Discharge_direction := FALSE;
        (* else, onto the right conveyer *)
    END_IF;

    Count := ROR(Count, 1); (* The counter decrements *)

    IF (Count = WORD#16#8000) THEN
        Pallet_on_entry := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Pallet_on_entry := TRUE;
END_IF;

Out_1 := Pallet_on_entry AND (NOT Turntable_busy OR NOT In_3);

(**** FEEDING CONVEYER CONTROL ****)
(* As in Task 2 *)
Out_0 := NOT Entry_busy OR NOT In_0;

(***************************************
            End of Program
***************************************)
```

Resolution of Task 9

To allow the entry conveyer to handle as many pallets as possible means to remove the dependency of the Entry_busy signal from the value of the Count register. Yet, the Entry_busy signal cannot be simply removed from the application, as the entry conveyer is not always ready to receive a pallet from the feeding conveyer. To be precise, it cannot charge a pallet when it handles a pallet that has just reached the end of the conveyer but cannot be charged onto the turntable as it is currently busy. Thus, Entry_busy is now formulated as:

```
Entry_busy = Turntable_busy AND In_3;
```

This small change seems suffice for achieving the goals of the task, increasing significantly the performance of the plant. Yet, if you try this, you will notice that pallets tend to get close to each other as they are transferred from the feeding to the entry conveyer, being later sensed as a single pallet by sensor 3, and from where serious problems result in the turntable control.

The problem happens when the entry conveyer has to stop, waiting for the turntable to become ready, and a pallet on the conveyer stays shortly beyond sensor 0, but not detected by this sensor. Since, in this case, the feeding conveyer is allowed to move a pallet to the position of sensor 0, before stopping, the two pallets become very close to each other. Thus, when the entry and the feeding conveyer start running simultaneously, the pallets end up together.

The problem is partially solved using a very simple strategy: if sensor 1, which is placed at the entry of the entry convey, is detecting a pallet when Turntable_busy goes from TRUE to FALSE, then the TRUE to FALSE transition of Entry_busy will be postponed until sensor 1 no longer detects any pallet. Tor implement this, let the set of Entry_busy be for Turntable_busy AND In_3 and its reset for NOT Turntable_busy AND NOT In_1. There is finally a role for sensor 1!

However, the conveying still does not perform correctly using this approach. In fact, if the entry conveyer gets blocked during the charging of a pallet, when such a pallet is already detected by sensor 1 but not “sufficiently charged” to be moved by the single running of the entry conveyer, then the pallet is not removed from its position when the entry conveyer restarts running and sensor 1 never stops detecting it. As a consequence, Entry_busy never goes FALSE and the feeding conveyer never restarts running.

The problem is finally solved with the inclusion of a “timeout” signal that forces Entry_busy to go FALSE if sensor 1 does not go FALSE a few moments after the rerunning of the entry conveyer, releasing the trapped pallet. The code presented below reveals the implementation of this technique. A 5-second timeout proved to be appropriate for the application.

```
(*************************************
          Mission 1 - Task 9
*************************************)
```

```
(**** DETECTION OF RELEVANT EVENTS ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_8(CLK := In_8);
F_In_9(CLK := In_9);
F_In_10(CLK := In_10);

(**** LEFT EXIT CONVEYER CONTROL ****)
(* As in Task 7 *)

IF F_In_10.Q THEN
    Pallet_on_left := FALSE;
    Left_busy := FALSE;
END_IF;

IF F_In_8.Q THEN
    Left_busy := TRUE;
END_IF;

IF In_8 THEN
    Pallet_on_left := TRUE;
END_IF;

Out_5 := Pallet_on_left AND (NOT In_10 OR In_15);

(**** RIGHT EXIT CONVEYER CONTROL ****)
(* As in Task 7 *)

IF F_In_9.Q THEN
    Pallet_on_right := FALSE;
    Right_busy := FALSE;
END_IF;

IF F_In_7.Q THEN
    Right_busy := TRUE;
END_IF;

IF In_7 THEN
    Pallet_on_right := TRUE;
END_IF;

Out_6 := Pallet_on_right AND (NOT In_9 OR In_15);

(**** TURNTABLE CONTROL ****)
```

```
(* As in Task 7 *)

IF Idle AND In_3 THEN
    Idle := FALSE;
    Charging := TRUE;
END_IF;

IF Charging AND In_6 THEN
    Charging := FALSE;
    Turns_charged := TRUE;
END_IF;

IF Turns_charged AND In_5 THEN
    Turns_charged := FALSE;
    Discharging := TRUE;
END_IF;

IF Discharging AND (F_In_7.Q OR F_In_8.Q) THEN
    Discharging := FALSE;
    Turns_Discharged := TRUE;
END_IF;

IF Turns_Discharged AND In_4 THEN
    Turns_Discharged := FALSE;
    Idle := TRUE;
END_IF;

IF F_In_3.Q THEN
    Turntable_busy := TRUE;
END_IF;

IF Idle THEN
    Turntable_busy := FALSE;
END_IF;

Out_2 := Charging OR Discharging AND Discharge_direction AND NOT Left_busy;
Out_3 := Discharging AND NOT Discharge_direction AND NOT Right_busy;
Out_4 := Turns_charged OR Discharging;

(**** ENTRY CONVEYER CONTROL ****)

IF F_In_0.Q THEN
    Count := ROL(Count, 1);
    Queue := SHL(Queue, 1);

    IF In_2 THEN
        Queue := Queue OR WORD#1;
```

```
    END_IF;
END_IF;

IF F_In_3.Q THEN
    IF (Count AND Queue) <> WORD#0 THEN
        Discharge_direction := TRUE;
    ELSE
        Discharge_direction := FALSE;
    END_IF;

    Count := ROR(Count, 1);

    IF (Count = WORD#16#8000) THEN
        Pallet_on_entry := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Pallet_on_entry := TRUE;
END_IF;

(* Defining Entry_busy *)
IF Turntable_busy AND In_3 THEN (* If the entry conveyer cannot move pallets, then it
    cannot receive a pallet from the feeding conveyer *)
    Entry_busy := TRUE; (* Set Entry_busy *)
END_IF;

(* If the turntable can receive pallets and the entry conveyer does not have a pallet
at sensor 1 or Release_timeout has expired, then it can receive pallets again from
the feeding conveyer *)

IF NOT Turntable_busy AND NOT In_1 OR Release_timeout.Q THEN
    Entry_busy := FALSE; (* Reset Entry_busy *)
    Mem_release_timeout := FALSE; (* Turns the timer off *)
END_IF;

(* When the turntable becomes ready to receive pallets and the entry conveyer is busy
and has a pallet at its entry, the reset of Entry_busy must be delayed relatively to
that of Turntable_busy. Otherwise the pallets will end up too close to each order on
the entry conveyer *)

IF NOT Turntable_busy AND Entry_busy THEN
    Mem_release_timeout := TRUE;
END_IF;

(* In these circumstances, Release_timeout guarantees a 5-second delay between the
rerunning of the entry conveyer and the restart of the feeding *)
```

```
Release_timeout(IN := Mem_release_timeout, PT := T#5s);

IF In_0 THEN
    Pallet_on_entry := TRUE;
END_IF;

Out_1 := Pallet_on_entry AND (NOT Turntable_busy OR NOT In_3);

(**** FEEDING CONVEYER CONTROL ****)
(* As in Task 2 *)
Out_0 := NOT Entry_busy OR NOT In_0;

(****************************************
            End of Program
****************************************)
```

Resolution of Task 10

The plant now has three functional modes which change as illustrated in the *grafcet* of Figure M1T10a. Such modes are:

- Automatic – The plant works continuously, as in Task 9;
- Shutdown – The feeding conveyer does not feed the plant, and the other devices are conveying the pallets still in transit;
- Ready – the plant is idle due to pallet starvation.

The interesting fact resulting from this new scenario is that, except for the feeding conveyer, all the devices are controlled as defined in Task 9 in any of the three different modes! It is just due to "pallet starvation" that the plant transits from Automatic to Shutdown mode, and later from Shutdown to Ready mode. "Ready" means precisely that the plant will return to automatic mode if the feeding conveyer starts feeding the plant again. Thus, comparatively to Task 9, the feeding conveyer controlling software – which has been kept unchanged since Task 2 – is the only device control code needing to be changed.

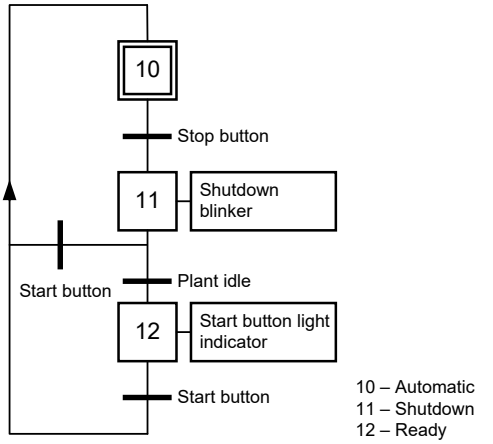


Figure M1T10a – *Mode changes and mode signalling for the conveying plant*

Notice that, if a pallet is being detected by sensor 0 at the moment the shutdown order is issued, then such a pallet must still be fed onto the entry conveyer, then one has that the control code for the feeding conveyer is now given by:

```
Out_0 = (NOT In_0 OR NOT Entry_busy) AND (In_0 OR NOT Shutdown) AND NOT Ready;
```

Or, equivalently:

```
Out_0 = (NOT In_0 OR NOT Entry_busy) AND (In_0 OR Automatic);
```

"Automatic", "Shutdown" and "Ready" are state variables representing the three possible modes of the plant and take the TRUE value when the plant is in the corresponding mode. This coding allows the easy programming of the *grafcet* presented in Figure M1T10a. To recognise that the plant is idle, which is the required condition to go from step 11 to step 12, it is suffice to recognise that all the conveyers and turntable are idle:

```
Plant_idle = NOT Pallet_on_entry AND Idle AND NOT Pallet_on_left AND NOT
Pallet_on_right;
```

The proper signalling of the shutdown mode requires the implementation of a blinker providing a rectangular waveform when Shutdown is TRUE. The easiest way of doing this is to use two on-delay timers programmed as represented in Figure M1T10b.

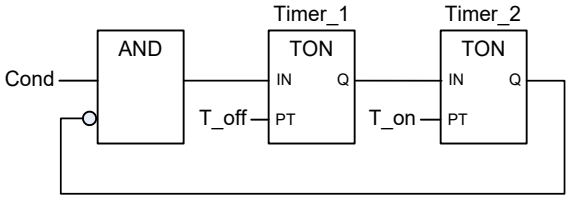


Figure M1T10b – *A blinker implemented from two "on-delay" timers*

A close look at the timing diagram of Figure M1T10c helps to understand the way the blinker works:

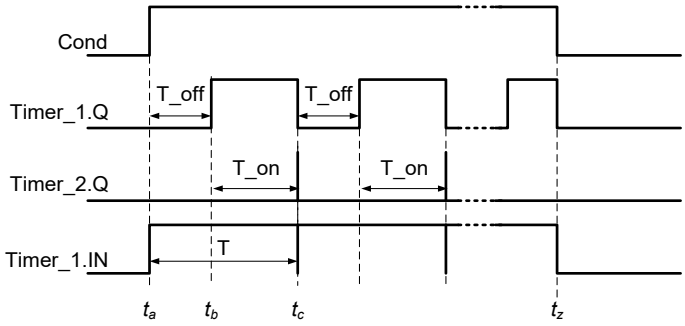


Figure M1T10c – *Timing diagram for the blinker of Figure M1T10b*

Assume that the blinker must blink when a condition "Cond" is TRUE. Also assume that "Cond" is initially FALSE. In this case the inputs and outputs of both timers stay FALSE.

The first timer, Timer_1, starts running at the time *ta* i.e., when Cond – which will be replaced by Shutdown in the case of the present task – goes TRUE. If Cond stays TRUE, then the output of Timer_1 goes TRUE at the time *tb* = *ta* + T_off, making Timer_2 to start running. If Cond remains TRUE, then the output of Timer_2 goes TRUE at the time, *tc* = *tb* + T_on, i.e., T_on time units after the Timer_1 output has gone TRUE, making it stop. This makes Timer_2 also stop, which lets Timer_1 restart. Thus *tc* is the time at which the conditions observed at *ta* are met again, and which will repeat periodicity with T = T_on + T_off while Cond stays TRUE. Thus, the output of Timer_1 blinks with a period T = T_off + T_on and duty cycle = T_on/T until Cond goes FALSE at time *tz*.

Assuming that the blinker uses two on-delay timers named "Shutdown_bliker_1" and "Shutdown_ bliker_2", then the signalling of shutdown mode using the start button light indicator is provided by the following code line:

```
Out_8 = Plant_idle OR Shutdown_blinker_1.Q;
```

The complete code for Task 10 is the following:

```
(*****************************************
           Mission 1 - Task 10
*****************************************)

(**** DETECTION OF RELEVANT EVENTS ****)

F_In_0(CLK := In_0);
F_In_3(CLK := In_3);
F_In_7(CLK := In_7);
F_In_8(CLK := In_8);
F_In_9(CLK := In_9);
F_In_10(CLK := In_10);

(**** MODE CHANGES CONTROL AND SIGNALLING ****)
(* State variables management *)

IF Automatic AND NOT In_13 THEN (* Automatic -> Shutdown *)
    Shutdown := TRUE;
    Automatic := FALSE;
END_IF;

IF Shutdown AND Plant_idle THEN (* Shutdown -> Ready *)
    Ready := TRUE;
    Shutdown := FALSE;
END_IF;

IF In_12 THEN (* When the start button is pressed *)
    Automatic := TRUE; (* Shutdown or Ready -> Automatic *)
    Shutdown := FALSE;
    Ready := FALSE;
END_IF;

(* Detecting the plant is idle *)
Plant_idle := NOT Pallet_on_entry AND Idle AND NOT Pallet_on_left AND NOT
Pallet_on_right;

(* Implementing the shutdown blinker *)
Shutdown_blinker_1(IN := Shutdown AND NOT Shutdown_blinker_2.Q, PT := T#1s);
Shutdown_blinker_2(IN := Shutdown_blinker_1.Q, PT := T#1s);

(* Mode Signalling *)
Out_8 := Ready OR Shutdown_blinker_1.Q;

(**** LEFT EXIT CONVEYER CONTROL ****)
```

```
(* As in Task 7 *)

IF F_In_10.Q THEN
    Pallet_on_left := FALSE;
    Left_busy := FALSE;
END_IF;

IF F_In_8.Q THEN
    Left_busy := TRUE;
END_IF;

IF In_8 THEN
    Pallet_on_left := TRUE;
END_IF;

Out_5 := Pallet_on_left AND (NOT In_10 OR In_15);

(**** RIGHT EXIT CONVEYER CONTROL ****)
(* As in Task 7 *)

IF F_In_9.Q THEN
    Pallet_on_right := FALSE;
    Right_busy := FALSE;
END_IF;

IF F_In_7.Q THEN
    Right_busy := TRUE;
END_IF;

IF In_7 THEN
    Pallet_on_right := TRUE;
END_IF;

Out_6 := Pallet_on_right AND (NOT In_9 OR In_15);

(**** TURNTABLE CONTROL ****)
(* As in Task 7 *)

IF Idle AND In_3 THEN
    Idle := FALSE;
    Charging := TRUE;
END_IF;

IF Charging AND In_6 THEN
    Charging := FALSE;
    Turns_charged := TRUE;
END_IF;
```

```
IF Turns_charged AND In_5 THEN
    Turns_charged := FALSE;
    Discharging := TRUE;
END_IF;

IF Discharging AND (F_In_7.Q OR F_In_8.Q) THEN
    Discharging := FALSE;
    Turns_Discharged := TRUE;
END_IF;

IF Turns_Discharged AND In_4 THEN
    Turns_Discharged := FALSE;
    Idle := TRUE;
END_IF;

IF F_In_3.Q THEN
    Turntable_busy := TRUE;
END_IF;

IF Idle THEN
    Turntable_busy := FALSE;
END_IF;

Out_2 := Charging OR Discharging AND Discharge_direction AND NOT Left_busy;
Out_3 := Discharging AND NOT Discharge_direction AND NOT Right_busy;
Out_4 := Turns_charged OR Discharging;

(**** ENTRY CONVEYER CONTROL ****)
(* As in Task 9 *)

IF F_In_0.Q THEN
    Count := ROL(Count, 1);
    Queue := SHL(Queue, 1);

    IF In_2 THEN
        Queue := Queue OR WORD#1;
    END_IF;
END_IF;

IF F_In_3.Q THEN
    IF (Count AND Queue) <> WORD#0 THEN
        Discharge_direction := TRUE;
    ELSE
        Discharge_direction := FALSE;
    END_IF;
```

```
    Count := ROR(Count, 1);

    IF (Count = WORD#16#8000) THEN
        Pallet_on_entry := FALSE;
    END_IF;
END_IF;

IF In_0 THEN
    Pallet_on_entry := TRUE;
END_IF;

IF Turntable_busy AND In_3 THEN
    Entry_busy := TRUE;
END_IF;

IF NOT Turntable_busy AND NOT In_1 OR Release_timeout.Q THEN
    Entry_busy := FALSE;
    Mem_release_timeout := FALSE;
END_IF;

IF NOT Turntable_busy AND Entry_busy THEN
    Mem_release_timeout := TRUE;
END_IF;

Release_timeout(IN := Mem_release_timeout, PT := T#5s);

IF In_0 THEN
    Pallet_on_entry := TRUE;
END_IF;

Out_1 := Pallet_on_entry AND (NOT Turntable_busy OR NOT In_3);

(**** FEEDING CONVEYER CONTROL ****)

Out_0 := (NOT Entry_busy OR NOT In_0) AND (Automatic OR In_0);
(* No feeding is provided in shutdown or ready modes. Yet, if there is a pallet at
sensor 0 at the moment the shutdown order is issued, such pallet must still go into
the plant *)

(***************************************
            End of Program
***************************************)
```