

Desenvolvimento de aplicações de teste de um *soft*PLC orientado ao ensino prático do controlo lógico

Adriana Manuela Pereira e Silva

Dissertação de Mestrado

Orientador: Prof. António José Pessoa de Magalhães



Mestrado Integrado em Engenharia Mecânica
Ramo de Automação

Julho 2021

Aos meus pais.

Resumo

Nos últimos anos, a sociedade mundial tem-se tornado mais exigente relativamente aos produtos que consome, fazendo com que a indústria tenha de se adaptar para conseguir sobreviver. Nesse sentido, a automação industrial é essencial para que as empresas consigam dar resposta às crescentes exigências da sociedade.

No entanto, a automação industrial requer elementos de controlo robustos face às condições hostis típicas dos ambientes fabris e que permitam flexibilizar as linhas de produção. É deste modo que os controladores lógicos programáveis (PLCs) surgem como peça central dos sistemas de automação. Mas, à semelhança do que acontece com qualquer equipamento industrial, são necessários profissionais qualificados para programar estes controladores.

Dada a natureza da função dos PLCs, ou seja, o controlo de sistemas de automação, o processo de qualificação destes profissionais deve incluir uma aprendizagem prática de qualidade em que os estudantes possam implementar os programas que desenvolvem nas aulas em sistemas reais, ou, na impossibilidade, em cenários tão próximos destes quanto possível. No entanto, esta vertente prática acarreta custos e riscos consideráveis. Isto porque o processo de aprendizagem não existe sem que ocorram erros, sendo que, no caso da programação de sistemas de automação, esses erros podem levar a funcionamentos adversos dos equipamentos controlados e causar acidentes.

Para dar resposta a este problema existe *software* de simulação. Este tipo de programas permite simular sistemas de automação num ambiente virtual, ou seja, os estudantes podem testar os seus códigos sem qualquer risco de acidentes. Uma solução existente no mercado é o *Factory I/O*, desenvolvido pela Real Games. Este permite a ligação com PLCs físicos de várias marcas e inclui ainda um controlador proprietário, o *Control I/O*. No entanto, este apresenta várias limitações, facto que levou a Real Games a decidir desenvolver um novo *softPLC* cuja linguagem de programação central é o Texto Estruturado e que será, espera-se, futuramente incorporado no *software* de simulação, o *Factory I/O*.

É neste contexto que a presente dissertação se enquadra. Esta tem como objetivo produzir material de teste para o *softPLC* em desenvolvimento que deverá ainda servir propósitos didáticos. Para isso, começa-se pela introdução dos elementos fundamentais da programação em Texto Estruturado, sendo apresentados exercícios simples que representam os blocos funcionais, de modo a facilitar a explicação dos mesmo. De seguida, introduz-se a programação de máquinas de estados, também fazendo uso de exercícios adequados e apresentando instruções utilizadas na programação em Texto Estruturado. Finalmente, apresentam-se sistemas baseados em cenários que incorporam a biblioteca do *Factory I/O* através dos quais se procura demonstrar a aplicabilidade dos conceitos introduzidos anteriormente. Ao longo de dois capítulos apresentam-se dois graus de dificuldade com o intuito de mostrar a versatilidade da programação em Texto Estruturado, bem como as diferentes possibilidades que advêm da mesma.

Com este projeto, proporcionou-se, à Real Games, testes para o seu novo *softPLC*, bem como resultados dos mesmos, e, a quem tiver interesse, material didático para a introdução à programação de controladores em Texto Estruturado.

Palavras-chave: Autómatos Programáveis, *Factory I/O*, Programação em Texto Estruturado, Controladores Lógicos Programáveis, *Software* de apoio ao ensino.

Development of test applications of a *soft*PLC oriented to the practical teaching of logic control

Abstract

In the last years, global society has become more demanding regarding the products it consumes, forcing the industry to adapt in order to thrive. As such, industrial automation is essential for companies to be able to respond to the ever-growing demands of society.

However, industrial automation requires control elements that are robust to the hostile conditions typical of the factory environments and allow for the flexibilization of the production lines. Therefore, the programmable logic controllers (PLCs) have become a central piece of the automation systems. Yet, similarly to what happens with other industrial equipment, it is also required to have qualified professionals to program these controllers.

Given the nature of the role the PLCs play, that is, the control of automation systems, the qualification process of these professionals must include a quality practical apprenticeship where the students can implement the programs developed during classes in real systems, or, if impossible to do so, in systems as close to these as possible. However, this practical approach carries costs and risks. That's because the learning process does not exist without the occurrence of errors, which, in the case of programming automation systems, can lead to adverse behaviors of the controlled equipment and lead to accidents.

To answer this problem there is simulation software. These kinds of programs allow to simulate automation systems in a virtual environment, which means that the students can test their code without any risk of accidents occurring. One solution that exists in the market is Factory I/O, developed by Real Games. This application allows communication with PLCs from a variety of different brands and includes their own controller, Control I/O. However, this last one has several limitations which has led Real Games to decide to create a new *soft*PLC which has Structured Text as its central programming language and is expected to be, in the future, incorporated in the simulation software, Factory I/O.

It is in this context that the present dissertation fits. The objective is to produce testing material for the *soft*PLC currently in development which should also be able to double as didactic material. To accomplish that, we start by introducing the fundamental elements of Structured Text programming, where simple exercises that represent the different functional blocks are presented so that the explanation is more clear and easier to understand. As a follow up, the programming of state machine is introduced, accompanied by adequate exercises and the presentation of programming instructions used in Structured Text. Finally, systems based on scenes from the Factory I/O library are presented and used to demonstrate the applicability of the concepts previously introduced. Throughout two chapters there are two degrees of difficulty presented with the objective of showing the versatility of Structured Text programming, as well as the different possibilities that come with it.

With this project was given, to Real Games, tests for their new *soft*PLC, as well as their results, and, for those interested, didactic material for the introduction to controller programming using Structured Text.

Keywords: Factory I/O, Programmable Automats, Programmable Logic Controllers, Structured Text Programming, Educational Software.

Agradecimentos

Em primeiro lugar gostaria de agradecer ao meu orientador, Professor António Pessoa de Magalhães, pela disponibilidade que sempre apresentou e pela orientação prestada. Deixo ainda uma palavra de apreço pelo acolhimento que me ofereceu quando o abordei, no verão de 2019, com vontade de aprender mais sobre os autómatos que me deu a conhecer enquanto docente da Unidade Curricular de Automação e Segurança Industrial. Esse momento foi, sem dúvida, a chave para eu descobrir a minha paixão.

Um agradecimento especial aos meus pais, por sempre me incentivarem a fazer o que me faz feliz, por me permitirem esta oportunidade de seguir uma paixão que se mostrou mais tarde do que o ideal e pelo seu amor e apoio incondicionais.

Aos meus irmãos, Alexandra e João, por estarem sempre prontos a animar-me e por poder contar com eles, aconteça o que acontecer.

Quero ainda agradecer ao meu namorado, André Reis, pela paciência demonstrada ao longo dos meses de desenvolvimento deste trabalho.

E, finalmente, a toda a minha família, de sangue, ou não, por todo o amor e apoio que me dão todos os dias.

Índice de Conteúdos

1	Introdução	1
1.1	Objetivo da dissertação.....	2
1.2	Organização da dissertação.....	2
2	Contexto e motivação da dissertação	5
2.1	O ensino da programação dos PLCs	5
2.2	Da aplicação <i>Factory I/O</i> ao <i>softPLC ST I/O</i>	8
2.3	Definição do trabalho a desenvolver	11
2.4	Síntese	12
3	Aspetos fundamentais da programação em ST	15
3.1	Funções lógicas elementares.....	16
3.2	Máquinas de estados	30
3.3	Síntese	52
4	Controlo de cenários de demonstração <i>Factory I/O</i> relativamente simples	55
4.1	Primeiro cenário	56
4.2	Segundo cenário	64
4.3	Terceiro cenário	68
4.4	Quarto cenário	74
4.5	Teste no novo PLC	78
4.6	Síntese	79
5	Controlo de estações de produção em <i>Factory I/O</i>	81
5.1	Primeiro cenário	81
5.2	Segundo cenário	91
5.3	Teste no novo PLC	109
5.4	Síntese	109
6	Conclusões e trabalhos futuros.....	111
6.1	Trabalhos futuros	112
	Referências	113

Índice de Figuras

Figura 2.1 – Programa em Texto Estruturado desenvolvido no <i>Visual Studio Code</i> .	10
Figura 2.2 – Visualização de avisos e erros no <i>softPLC ST I/O</i> .	10
Figura 2.3 – Visualização do estado das variáveis no <i>softPLC ST I/O</i> .	11
Figura 3.1 – Quadro elétrico utilizado para testar os programas apresentados ao longo da primeira secção.	16
Figura 3.2 – Diagrama temporal a considerar no primeiro problema.	17
Figura 3.3 – Diagrama de funcionamento de um bloco TON [37].	18
Figura 3.4 – Diagrama temporal a considerar no segundo problema.	19
Figura 3.5 – Diagrama de funcionamento de um bloco TOF [38].	20
Figura 3.6 – Diagrama temporal a considerar no terceiro problema.	21
Figura 3.7 – Diagrama de funcionamento de um TP [39].	21
Figura 3.8 – Diagrama temporal a considerar no quarto problema.	22
Figura 3.9 – Diagrama temporal do quinto problema.	24
Figura 3.10 – Diagrama temporal do sexto problema.	27
Figura 3.11 – Diagrama de estados do primeiro problema.	32
Figura 3.12 – Diagrama de estados do segundo problema.	35
Figura 3.13 – Diagrama de estados do terceiro problema.	38
Figura 3.14 – Grafcet a considerar para o primeiro problema.	40
Figura 3.15 – Programa em <i>ladder</i> das transições do grafcet G2 do primeiro problema.	41
Figura 3.16 – Programa em <i>ladder</i> das etapas do grafcet G2 do primeiro problema.	42
Figura 3.17 – Programa em <i>ladder</i> da escrita das saídas do grafcet G2 do primeiro problema.	42
Figura 3.18 – Grafcet a considerar no segundo problema.	46
Figura 3.19 – Resultado da divisão do grafcet original.	46
Figura 3.20 – Grafcet do terceiro problema.	49
Figura 4.1 – Primeiro cenário.	56
Figura 4.2 – Diagrama de estados do primeiro cenário.	58
Figura 4.3 – Alternativa de modelação do primeiro problema.	59
Figura 4.4 – Alternativa do primeiro cenário.	61
Figura 4.5 – Diagramas de estados da alternativa do primeiro cenário.	62
Figura 4.6 – Cenário do segundo problema.	65
Figura 4.7 – Ilustração do funcionamento do segundo cenário.	65
Figura 4.8 – Diagrama de estados do segundo cenário.	66
Figura 4.9 – Alternativa de modelação do segundo cenário.	67
Figura 4.10 – Terceiro cenário.	69
Figura 4.11 – Grafcet do terceiro cenário.	71
Figura 4.12 – Quarto cenário.	74
Figura 4.13 – Grafcet do quarto cenário.	76
Figura 5.1 – Primeiro cenário.	82
Figura 5.2 – Ilustração do funcionamento do primeiro sistema.	83
Figura 5.3 – Grafcet do primeiro problema.	85
Figura 5.4 – Grafcet G0 do primeiro cenário.	86
Figura 5.5 – Comunicação entre o grafcet G0 e os grafkets dos subsistemas de controlo das caixas e das paletes, G1 e G2.	87
Figura 5.6 – Comunicação entre os subsistemas de controlo das caixas e das paletes e o subsistema principal do robô.	88
Figura 5.7 – Comunicação entre os subsistemas principal e o Get Box do robô.	89
Figura 5.8 – Comunicação entre o subsistema principal do robô e o subsistema de controlo das paletes.	89
Figura 5.9 – Segundo cenário.	92

Figura 5.10 – Ilustração do funcionamento do segundo cenário.	93
Figura 5.11 – Grafcet do segundo cenário, sem interface homem-máquina.....	95
Figura 5.12 – Grafcet G0 do segundo cenário.	96
Figura 5.13 – Detalhe da comunicação entre G0 e os subsistemas de controlo das tampas e das bases.	97
Figura 5.14 – Comunicação entre o subsistema de controlo das tampas e o ciclo principal de funcionamento do robô.	98
Figura 5.15 – Comunicação entre os dois ciclos de funcionamento do robô.....	99
Figura 5.16 – Comunicação entre o ciclo de carga e descarga das tampas e o subsistema de controlo das tampas.....	99
Figura 5.17 – Comunicação entre o ciclo de carga e descarga das tampas e o subsistema das bases.	100
Figura 5.18 – Grafcet do segundo cenário, com interface homem-máquina.	104
Figura 5.19 – Grafcet da interface do segundo cenário.	105
Figura 5.20 – Comunicação entre a interface e os subsistemas de controlo das tampas e das bases.	106

Índice de Tabelas

Tabela 3.1 – Resumo das variáveis, e respectivas funções, do bloco funcional TON.....	18
Tabela 3.2 – Resumo das variáveis, e respectivas funções, do bloco funcional TOF.....	20
Tabela 3.3 – Resumo das variáveis, e respectivas funções, do bloco funcional TP.....	21
Tabela 3.4 – Resumo das variáveis, e respectivas funções, do bloco funcional F_Tríg.....	23
Tabela 3.5 – Resumo das variáveis, e respectivas funções, do bloco funcional SR.....	25
Tabela 3.6 – Resumo das variáveis, e respectivas funções, do bloco funcional RS.....	26
Tabela 3.7 – Resumo das variáveis, e respectivas funções, do bloco funcional CTU.....	28
Tabela 3.8 – Resumo das variáveis, e respectivas funções, do bloco funcional CTD.....	29
Tabela 3.9 – Resumo das variáveis, e respectivas funções, do bloco funcional CTUD.....	30
Tabela 3.10 – Definição dos diferentes estados do primeiro problema.....	32
Tabela 3.11 – Transições do diagrama do primeiro problema.....	33
Tabela 3.12 – Codificação dos estados do sistema do segundo problema.....	36
Tabela 3.13 – Transições do diagrama de estados do segundo problema.....	37
Tabela 3.14 – Codificação dos estados do sistema do terceiro problema.....	38
Tabela 3.15 – Transições do diagrama de estados do terceiro problema.....	39
Tabela 4.1 – Requisitos de funcionamento do primeiro cenário.....	57
Tabela 4.2 – Resumo da modelação do primeiro problema.....	58
Tabela 4.3 – Requisitos de funcionamento da alternativa do primeiro cenário.....	61
Tabela 4.4 – Resumo da modelação da alternativa do primeiro cenário.....	63
Tabela 4.5 – Requisitos de funcionamento do segundo cenário.....	66
Tabela 4.6 – Resumo da modelação do sistema do segundo cenário.....	67
Tabela 4.7 – Requisitos de funcionamento do terceiro cenário.....	69
Tabela 4.8 – Resumo da modelação do terceiro problema.....	72
Tabela 4.9 – Requisitos de funcionamento do quarto cenário.....	75
Tabela 4.10 – Resumo da modelação do quarto problema.....	77
Tabela 5.1 – Requisitos de funcionamento do primeiro cenário.....	84
Tabela 5.2 – Legenda do grafcet do primeiro cenário.....	86
Tabela 5.3 – Resumo dos requisitos do segundo cenário, sem interface.....	94
Tabela 5.4 – Legenda do grafcet do segundo cenário, sem interface.....	96
Tabela 5.5 – Resumo dos requisitos do segundo cenário, com interface.....	102
Tabela 5.6 – Legenda do grafcet do segundo cenário, com interface.....	105

Siglas e Acrónimos

FB – *Function Block*

FBD – *Function Block Diagram*

GRACET – *GRAphe Fonctionnel de Commande Etape Transition*

HMI – *Human Machine Interface*

IEC – *International Electrotechnical Commission*

IL – *Instruction List*

LD – *Ladder Diagram*

OPC UA – *OLE for Process Control Unified Architecture*

PLC – *Programmable Logic Controller*

SFC – *Sequential Function Chart*

ST – *Structured Text*

1 Introdução

Nas últimas décadas a indústria mundial tem sido alvo de mudanças constantes e de uma sociedade mais exigente em termos económicos, de qualidade e de consistência. As empresas são obrigadas a acompanhar essas exigências para se manterem no mercado. No entanto, estes aspetos não são passíveis de ser conseguidos sem empregar mecanismos de automação, nomeadamente em processo de produção. O que faz que com seja possível, atualmente, encontrar automação em qualquer lugar, desde o fundo do oceano até ao espaço [1].

A implementação de sistemas de automação requer elementos de controlo dos processos de produção que sejam robustos, versáteis e flexíveis, isto é, que permitam adaptar o processo que controlam a novas necessidades. Assim, os controladores lógicos programáveis – PLCs – surgem como unidade central de quase todos os sistemas de controlo industriais. Estes equipamentos permitem a aquisição de informação obtida por sensores, por meio de entradas analógicas ou digitais, e possibilitam ainda a atuação de equipamentos, por meio de saída que também podem ser analógicas ou digitais. Os sinais associados a estas entradas e saídas são habitualmente referidos como sinais IO – *Input* e *Output*. Isto permite customizar o PLC de modo que este se adeque a diferentes aplicações e sistemas [2]. Estes equipamentos podem ser programados numa variedade de linguagens que são regradas pela norma IEC 61131-3:2013 [3]. Essas linguagens são a Lista de Instruções – IL –, Diagrama de Contactos – LD ou *ladder* –, Diagramas de Blocos Funcionais – FBD – e Texto Estruturado – ST.

No entanto, a utilização e programação destes equipamentos requer profissionais treinados. Para conseguir formar indivíduos são necessários meios de aprendizagem eficazes, seguros e que permitam utilização fora de laboratório. Tendo em conta que o objetivo final da programação de PLCs é a implementação do código desenvolvido de modo a controlar um sistema de produção, é de todo o interesse ter a possibilidade de visualizar esse processo, usando maquetas de sistemas industriais. No entanto, num contexto de ensino existe um risco associado a essa prática. O processo de aprendizagem não existe sem a ocorrência de erros, erros esses que podem colocar em perigo tanto as pessoas como os equipamentos, o que se traduz em investimentos de risco. Surgem então os simuladores virtuais, que permitem, como o nome indica, simular ambientes industriais passíveis de ser controlados por PLCs, possibilitando, assim, a implementação e visualização do resultado da programação, sem os riscos de segurança anteriormente mencionados.

Um *software* que permite essa simulação é o *Factory I/O*, da Real Games [4]. Esta aplicação permite o controlo dos seus cenários por PLC, sendo que, não só inclui suporte para vários PLCs comerciais, como ainda incorpora um *softPLC* proprietário, o *Control I/O*. No entanto, este controlador apresenta algumas limitações, nomeadamente, não tem uma execução determinística sendo, por isso, incompatível com a norma IEC 61131-3. Assim, com o objetivo de fornecer uma ferramenta de ensino e aprendizagem mais completa a Real Games decidiu criar um novo controlador, partindo do zero.

Este é, pois, um projeto a médio prazo que se encontra, à data de escrita desta dissertação, num desenvolvimento ainda embrionário. O *soft*PLC em que a empresa está a investir tem, portanto, como objetivo apresentar uma programação próxima da que é definida pela norma e ter, como referido, características vincadamente didáticas, procurando sempre ser simples de utilizar, comparativamente com os PLCs comerciais. Ainda na vertente de ensino e aprendizagem, pretende-se que este controlador seja acompanhado de material didático e de aplicações desenvolvidas em conjunto com o *Factory I/O* de modo a tornar-se uma ferramenta única no género – o que tem sido solicitado com frequência. Embora incluindo também *ladder* e Blocos Funcionais, este PLC será concebido com o Texto Estruturado como linguagem de programação principal.

Atualmente existe um protótipo deste *soft*PLC que executa programas em Texto Estruturado e que importa testar exaustivamente. É neste âmbito que a dissertação foi proposta e se desenrola.

1.1 Objetivo da dissertação

O projeto apresentado neste documento tem como objetivo o desenvolvimento de material de teste para o *soft*PLC *ST I/O*, em desenvolvimento pela Real Games, que deverá ainda servir propósitos didáticos. A seu devido tempo, mais concretamente, a quando da disponibilização do PLC, também o material aqui desenvolvido poderá ser facultado como acompanhamento do referido *software*.

Como foi mencionado, a empresa responsável pelo desenvolvimento do PLC que serve de objeto de estudo desta dissertação é a Real Games. Esta é uma empresa com mais de 10 anos de desenvolvimento de *software* didático que tem como base a simulação em 3D. Os seus principais clientes são empresas industriais e tecnológicas, bem como centros de investigação e instituições de ensino superior. Uma das aplicações que a empresa disponibiliza é o *Factory I/O*, um simulador 3D que permite a conceção de cenários industriais fazendo uso de uma vasta biblioteca de elementos, incorporada no *software*. Esta aplicação permite ainda controlar os cenários desenvolvidos usando um PLC.

Embora os conteúdos que integrarão o referido acompanhamento ainda sejam uma questão em aberto, o público-alvo destes materiais, e do *soft*PLC, serão iniciantes na programação de autómatos em ST, mais concretamente, indivíduos que já possuam algum conhecimento de programação em *ladder*, mas não tenham tido ainda qualquer contacto com a linguagem de Texto Estruturado.

O objetivo do material produzido e apresentado ao longo desta dissertação é, portanto, proporcionar bases teóricas e práticas sobre a programação de PLCs em ST utilizando para isso exemplos de aplicação diversificados que permitam, assim, uma transição mais fácil para PLCs comerciais de referência, como *Codesys* e *Siemens*.

1.2 Organização da dissertação

Tendo em conta os objetivos definidos para a presente dissertação decidiu-se estruturar a mesma da seguinte forma:

O segundo capítulo procura expor diferentes abordagens ao ensino de autómatos programáveis, analisando cursos universitários, técnicos e profissionais, bem como livros

técnicos. De seguida, introduz-se, mais detalhadamente, a motivação e o contexto que levaram a Real Games à decisão de desenvolver um novo *soft*PLC que integrará a sua aplicação já existente no mercado, o *Factory I/O*. Para concluir o capítulo apresenta-se e justifica-se a abordagem definida para cumprir os objetivos desta dissertação.

Estando, então, apresentado e contextualizado o objetivo do presente documento, segue-se o capítulo 3. Este tem como objetivo apresentar as questões e os elementos fundamentais da programação em Texto Estruturado de acordo com a norma IEC 61131-3. Assim, inicia-se este capítulo pela contextualização dos referidos exercícios. De seguida, apresenta-se cada exercício, começando pelo respetivo enunciado, seguido de uma breve análise e apresentando, finalmente, a respetiva solução em Texto Estruturado. A partir desta procura-se apresentar e explicar, o tão detalhadamente quanto possível, os blocos funcionais mais simples que constituem as bases da programação em ST, tentando sempre ser o mais claro possível.

Ainda no terceiro capítulo procura-se expor casos particulares que demonstrem peculiaridades de programação e explicar o desenvolvimento de programas a partir de máquinas de estados, apresentando estas sob a forma de diagramas de estados e grafets. Estes são, por sua vez, alvo de uma breve análise seguida dos programas desenvolvidos a partir dos mesmos.

Os resultados da testagem, no novo PLC, dos códigos apresentados são expostos no final de cada secção.

Tendo as bases da programação em Texto Estruturado apresentadas e explicadas segue-se a aplicação das mesmas. Assim, no capítulo 4 são apresentados diferentes cenários desenvolvidos no *Factory I/O* com o objetivo de os controlar. Para isso, além do referido cenário, é apresentada a modelação dos mesmos, em diagramas de estados ou em GRAFCET, e de acordo com a norma IEC 60848.2013 [5], e o programa desenvolvido para controlar os sistemas em questão. O capítulo tem como objetivo demonstrar a aplicabilidade e utilidades dos conceitos introduzidos no capítulo 3 e ainda realçar algumas estratégias de modelação, em diagramas de estados e grafets, que permitem obter soluções simples e de fácil compreensão que se traduzem numa programação mais imediata. No final do capítulo são apresentados os resultados dos testes feitos no novo PLC com os programas desenvolvidos ao longo do mesmo.

No capítulo 5 procura-se continuar a exposição de aplicações dos conceitos apresentados no capítulo 3, desta vez com sistemas mais complexos do que os que se encontram no capítulo anterior. Para isso são apresentados dois cenários da biblioteca da aplicação *Factory I/O*, expondo os requisitos de funcionamento dos mesmos. A partir desta informação apresenta-se, então, a modelação resultante, seguida dos programas desenvolvidos a partir da mesma. Além de demonstrar as potencialidades dos elementos fundamentais da programação em Texto Estruturado, o presente capítulo tem ainda como objetivo apresentar estratégias de modelação de sistemas complexos e as comunicações que devem ser estabelecidas entres os diferentes elementos do mesmo. Depois, a partir dessa modelação, demonstra-se como se programam essas questões. Finalmente, tal como aconteceu com os capítulos que o antecedem, também neste são apresentados os resultados do desempenho do novo PLC na execução dos programas desenvolvidos no âmbito deste capítulo.

O último capítulo expõe as conclusões retirada do desenvolvimento desta dissertação e propõe trabalhos futuros que completem o trabalho desenvolvido ao longo deste documento.

2 Contexto e motivação da dissertação

Serve o presente capítulo para estabelecer o contexto desta dissertação e justificar a mesma. Para tal começa-se por analisar o estado atual do ensino de programação de controladores lógicos programáveis e ainda algumas necessidades do mesmo. Segue-se a apresentação do *soft*PLC *ST I/O*, que se encontra em desenvolvimento pela Real Games, expondo a motivação da sua criação e quais as características e funcionalidades que se deseja implementar no mesmo. Finalmente, termina-se apresentando e justificando a abordagem definida para esta dissertação.

2.1 O ensino da programação dos PLCs

Num mundo onde a indústria é obrigada a dar respostas rápidas às exigências da sociedade, a automação ganha cada vez mais importância. No entanto, para que se consiga que essa automação possa, de facto, dar resposta às necessidades da sociedade são necessários profissionais qualificados. Surge, então, a questão de como esses profissionais são treinados para desempenharem as suas funções.

A abordagem dada ao ensino de PLCs varia de curso para curso, e com a finalidade do mesmo, pelo que se pesquisou uma variedade de cursos que abordam, de alguma forma, este tópico, desde cursos universitários e técnicos até cursos profissionais. Pesquisou-se ainda livros técnicos sobre autómatos programáveis de modo a conhecer melhor as diferentes abordagens. Graças a esta pesquisa verificou-se que existe, portanto, uma estrutura comum que começa por abordar os autómatos enquanto dispositivo, estudando aspetos como a sua arquitetura e funcionalidades, passando de seguida para a lógica associada aos mesmos, e, finalmente, a sua programação de um modo mais concreto. Nos próximos parágrafos procura-se expor os resultados desta pesquisa.

Como ponto de partida desta pesquisa procurou-se conhecer como é que as universidades portuguesas abordam o ensino de PLCs. Para isso consultaram-se as páginas de unidades curriculares das seguintes instituições: Universidade do Porto [6-9], Universidade do Minho [10, 11], Universidade de Aveiro [12-15], Universidade de Coimbra [16-18], Universidade Beira Interior [19] e da Universidade Nova de Lisboa [20].

De um modo geral, as unidades curriculares que efetivamente abordam este tópico começam, maioritariamente, pela álgebra de Boole e funções lógicas como AND, OR, XOR e NOT, sejam estas implementadas na prática com exercícios de cablagem ou de escrita de programas. O tópico seguinte mais abordado são as máquinas de estados e os grafets, onde a lógica previamente ensinada é aplicada. No entanto, muitos cursos não lecionam o primeiro, havendo situações em que o ensino é iniciado diretamente pelo segundo. Já outras metodologias de ensino não referem nenhum dos dois, abordando os PLCs de uma forma muito ligeira e simples. Terminados estes tópicos passa-se à introdução do conceito de PLC, seguindo-se então a programação propriamente dita.

Relativamente a esta última a linguagem mais abordada é o *ladder*, seguida pelos Diagramas de Blocos Funcionais e finalmente, em alguns casos, o Texto Estruturado.

Relativamente ao que não é ensinado, foi possível apurar, que a norma de programação de autómatos programáveis, a IEC 61131-3, as variáveis analógicas, as funções matemáticas, sensores e atuadores, métodos de comunicação, interfaces homem-máquina e a possibilidade de ligação a atuadores, como motores, estão relativamente ausentes dos conteúdos lecionados, embora haja exceções.

Além dos cursos universitários temos ainda cursos profissionais, técnicos e formações disponibilizadas por fabricantes como meio de ensino de autómatos programáveis.

Analisando primeiramente formações de fabricantes como a Siemens [21-23] e a Omron [24], verifica-se que estas valorizam consideravelmente a arquitetura dos seus autómatos, detalhando não só os constituintes físicos, mas também questões de instalação, comunicação, diagnóstico e reparação. No entanto, estas formações também ensinam programação, ainda que menos detalhadamente.

Em contraponto, os cursos técnicos e profissionais, como os que são disponibilizados pela Escola Tecnológica Vale de Cambra [25], pela UDEMY [26], pela Dual: Qualificação Profissional [27, 28], pela Cenertec [29] e pela Master D [30], valorizam, comparativamente mais a programação no seu ensino. Estes abordam temas como lógica, set e reset, temporizadores e contadores, sendo as linguagens de programação mais utilizadas os blocos de funções e os diagramas de contactos. Tudo isto assenta numa base teórica sobre a constituição do PLC.

No entanto, é de salientar que estes cursos e formações são mais focados na instalação e manutenção dos autómatos, sendo que, de uma forma geral, a programação destes não é o seu foco. Assim, são vários os temas excluídos do plano de ensino, como as funções matemáticas, vetores, a álgebra de Boole e, nomeadamente, a norma IEC 61131-3.

Contrariamente ao que se verifica para os cursos mencionados, os livros técnicos apresentam abordagens mais disparees pelo que serão mais aprofundados.

O livro "Autómatos Programáveis" [31], de António Francisco, aborda a programação de PLCs com profundidade, sendo um livro muito completo sobre o tema. Este inicia-se com a introdução dos autómatos, referindo e falando brevemente sobre os seus componentes. De seguida refere o seu funcionamento, nomeadamente a leitura de variáveis de entrada, a execução do programa e a atualização das referidas variáveis. Feita esta introdução o autor passa, então, para o tópico que é de maior interesse para a presente dissertação: a programação de autómatos. Esta secção é iniciada pela análise de duas linguagens de programação: a Lista de Instruções e *ladder*. Nos capítulos seguintes o autor aborda diferentes autómatos, referindo aspetos mais arquitetónicos dos mesmos, mas sem ignorar a vertente da programação, apresentando exemplos de programação utilizando diferentes funções, como temporizadores, contadores, set e reset, instruções de comparação e relés, incluindo ainda, em alguns casos, a consola.

Concluído o estudo dos autómatos segue-se o GRAFCET, sendo que o autor procura expor a teoria e as regras que definem esta modelação. É apresentada, ainda, a vertente prática, onde são apresentados diferentes problemas e as respetivas soluções, incluindo ainda o desenvolvimento de programas para PLC a partir de graficets. Sendo que os programas apresentados são desenvolvidos em Lista de Instruções e *ladder*.

O capítulo final foca-se em apresentar problemas de aplicações reais como o controlo de um motor ou a seleção de caixas, onde o autor detalha o processo e analisa como a questão deve ser abordada e resolvida, passando pelo desenvolvimento do graficet e a sua tradução num programa para o autómato. É de notar que, nos anexos incluídos no livro, estão presentes informações não só sobre a cablagem dos PLCs, mas também sobre sensores, passivos e ativos, e atuadores, como motores ou cilindros.

Outro livro de grande interesse é "Técnicas de Automação" [32], de J. R. Caldas Pinto. Este livro começa por introduzir elementos lógicos de seguida refere sistemas pneumáticos e hidráulicos, uma parte essencial dos sistemas automatizados. No entanto, é no 3º capítulo que surgem os autómatos programáveis. Nesta secção o autor expõe a arquitetura, interfaces e aplicações dos mesmos, referindo as suas vantagens. Sendo que aborda também a modelação em GRAFCET.

O "Curso de Automação Industrial" [33], de Paulo Oliveira, é um livro escrito com o objetivo de servir não só professores e alunos, mas também profissionais e empresas, abordando, por isso, temas que são, no presente contexto, menos relevantes, mas mencionando, ainda que levemente, temas de interesse para esta dissertação. Assim, a publicação começa por abordar a automação de um modo genérico, definindo o conceito e introduzindo os autómatos programáveis como uma solução para a automação da produção, abordando ainda a sua arquitetura e constituição. Terminado esse assunto, o autor passa então para a programação dos autómatos e outros equipamentos. Como foi referido, este livro é direcionado igualmente ao ensino como aos profissionais, pelo que a secção seguinte aborda questões de instalação e manutenção de PLCs, desviando-se significativamente da programação e daquele que é o tema deste trabalho.

Na coletânea de exercícios de António Pessoa de Magalhães, "Práticas de Automação Industrial" [34], é possível encontrar uma outra abordagem ao ensino da programação de autómatos. Neste livro são apresentados vários problemas industriais cuja automatização deverá ser feita com recurso a um PLC, neste caso, um *soft*PLC. Os problemas, denominados de missões, são introduzidos com uma descrição da situação em estudo e algum contexto sobre a sua aplicabilidade no mundo real. De seguida, o problema é dividido em várias tarefas, sendo o objetivo desta divisão a criação de exercícios cuja dificuldade vai aumentando gradualmente, permitindo ao aluno a incorporação de diferentes aspetos e detalhes da programação, sem que haja saltos no ensino, que poderiam dar origem a lacunas no conhecimento. A tarefa final é, usando o que foi aprendido nas tarefas anteriores, resolver o problema que é apresentado no início de cada missão. As próprias missões vão escalando em dificuldade ao longo do livro.

Um livro que surge como bibliografia recomendada de várias unidades curriculares de cursos universitários é o livro "Programmable Logic Controllers" [35] da autoria de William Bolton. Este documento inicia a sua abordagem pela análise e descrição da arquitetura e constituintes dos autómatos programáveis, complementando com dispositivos que possam ser utilizados como entradas ou saídas do PLC. Terminado o estudo do autómato enquanto equipamento, o autor procede para os códigos binário, octal e hexadecimal, algo pouco abordado noutros livros ou unidades curriculares. De seguida introduz os sistemas combinacionais e os sequenciais, outro tópico relativamente pouco abordado, embora menos ignorado do que os códigos mencionados anteriormente. Segue-se o estudo do processamento de sinais de entrada e saída e a introdução de linguagens de programação, mais concretamente, IL, Gráfico de Funções Sequenciais – SFC – e Texto Estruturado.

De um modo geral, verifica-se que os cursos universitários se focam em introduzir o conceito de PLC, definindo o que é, quais os seus componentes gerais, como entradas, saídas e portas de comunicação e utilidade no meio industrial desde controlo e automação de processos, aquisição de dados até à interação homem-máquina, por meio de interfaces homem-máquina – HMIs. Em contraste, os cursos técnicos e profissionais focam-se mais na arquitetura e cablagem do PLC, com o objetivo de formar profissionais que instalem e façam a manutenção dos PLCs, dando, por isso, menos valor à componente de programação, muitas vezes evitando a mesma por completo. A prioridade que é atribuída ao GRAFCET e aos PLCs é outro aspeto que varia entre os cursos. Alguns começam por abordar o primeiro antes de introduzirem o PLC, sendo que esta ordem se mantém nos exercícios estudados nas disciplinas, no entanto, existem outros que fazem o inverso, estruturando primeiro programas de PLC e depois, então, passam ao desenho dos respetivos graficets. No entanto, é de referir que existem ainda cursos que são mais

focados no PLC em si, não fazendo qualquer menção aos graficets. Conclui-se, então, que a relevância dada à modelação GRAFCET e à programação de PLC não é uniforme no ensino.

Já os livros têm uma abordagem mais variada. Alguns focam-se mais no autómato e na sua arquitetura e cablagem enquanto outros tentam ser o mais abrangentes possível. Isto pode ser justificado pelo facto de os livros terem como alvo não só estudantes, mas também profissionais.

Relativamente à abordagem aos PLCs, do que foi possível averiguar, estes são vistos e ensinados como uma solução para a automação de diferentes sistemas de produção, sendo apreciada a sua flexibilidade, a possibilidade de os ligar a saídas e entradas digitais e/ou analógicas, as suas propriedades físicas, que os tornam indicados para ambientes industriais face a outros controladores e computadores, bem como a capacidade de aquisição de dados e interface com os operadores, embora estes dois tópicos sejam menos abordados nas disciplinas e cursos analisados.

Já os exercícios utilizados para o ensino de PLCs parecem ser, na sua maioria, algo encadeados. Isto é, o mais comum é começar por apresentar exercícios mais simples que permitam introduzir questões mais simples como o uso de temporizadores ou de contadores e escalar a dificuldade com os exercícios que se seguem, incorporando os tópicos abordados nos problemas anteriores.

2.2 Da aplicação *Factory I/O* ao *soft*PLC *ST I/O*

Como se pode observar, são várias as abordagens ao ensino de PLCs, no entanto, verifica-se um défice na componente prática. Isto verifica-se porque existem obstáculos no que toca a material de ensino. Estes obstáculos passam principalmente pelo custo de aquisição de sistemas que permitam aos estudantes ter uma experiência de aprendizagem que seja a mais próxima da realidade possível. Isto é, alguém que esteja a aprender a programar um autómato irá, certamente, cometer erros, visto que estes fazem parte do processo de aprendizagem. No entanto, esses erros podem levar a situações de elevado risco para a integridade humana e/ou do sistema em questão, o que faz com que os elevados investimentos que esses equipamentos requerem sejam demasiado arriscados e, por consequência, injustificáveis.

Surge assim, graças à evolução da computação gráfica, uma solução para esta questão: os simuladores gráficos. Este tipo de *software* permite construir ambientes industriais operacionais que podem ser controlados por autómatos, constituindo, por isso, uma ferramenta de ensino e aprendizagem de elevado interesse, visto que oferecem uma experiência mais realista sem os riscos anteriormente mencionados.

Uma das empresas que têm investido nesta área é a Real Games, criadora da aplicação *Factory I/O*, um simulador de ambiente industrial que apresenta uma vasta gama de elementos que podem ser utilizados para a construção de sistemas industriais. Além da possibilidade de criar cenários esta aplicação apresenta ainda a capacidade de estabelecer ligação com PLCs de várias marcas como *Allen-Bradley*, *Siemens*, *Shneider* e ainda ao *soft*PLC *Codesys*, permitindo, por isso, o teste de programas desenvolvidos nestes autómatos, sem que haja, como mencionado, perigo para a integridade humana e/ou do sistema controlado.

Os cenários desenvolvidos em *Factory I/O* podem ser controlados por qualquer PLC de qualquer marca recorrendo a diversas soluções. Porém, e pensando num público-alvo que não tem um PLC à sua disposição, O *Factory I/O* inclui também um *soft*PLC proprietário: o *Control I/O*. Trata-se de uma solução simples e relativamente informal, programável em blocos lógicos e que embora permita aprender e praticar questões lógicas, tem bastantes limitações. No entanto é excelente para demonstração de conceitos exemplificáveis no *Factory I/O* já que a troca de

variáveis entre as duas aplicações é realizada transparentemente e modificável sem necessidade de interromper a execução do programa. Esta simplicidade de utilização e integração é reconhecidamente uma mais-valia do *Control I/O*.

Porém, são também reconhecidos alguns aspetos menos interessantes do *Control I/O*. Nomeadamente, a ausência de uma linguagem de programação mais universal – como o *ladder* – ou mais moderna - como o Texto Estruturado. Se a isto se acrescentar a preferência por PLCs compatíveis com a norma IEC 61131-3, tirando com isso partido de modelos de execução de *software* muito mais eficientes e interessantes do que os disponíveis em PLC clássicos, é compreensível a ambição da Real Games de substituir o *Control I/O* por um *soft*PLC muito mais atual e útil aos utilizadores do *Factory I/O*.

A presente dissertação foi proposta a um tempo em que os requisitos do PLC a desenvolver não estavam ainda completamente fechados. E muito menos havia algo decidido em definitivo sobre a forma como o mesmo iria ser desenhado e construído. No entanto, as exigências eram elevadas e, no essencial, resumiam-se a construir um PLC desde logo compatível com a norma IEC 61131-3 e especialmente vocacionado para iniciantes em programação lógica que teriam no *Factory I/O* o seu terreno de experimentação.

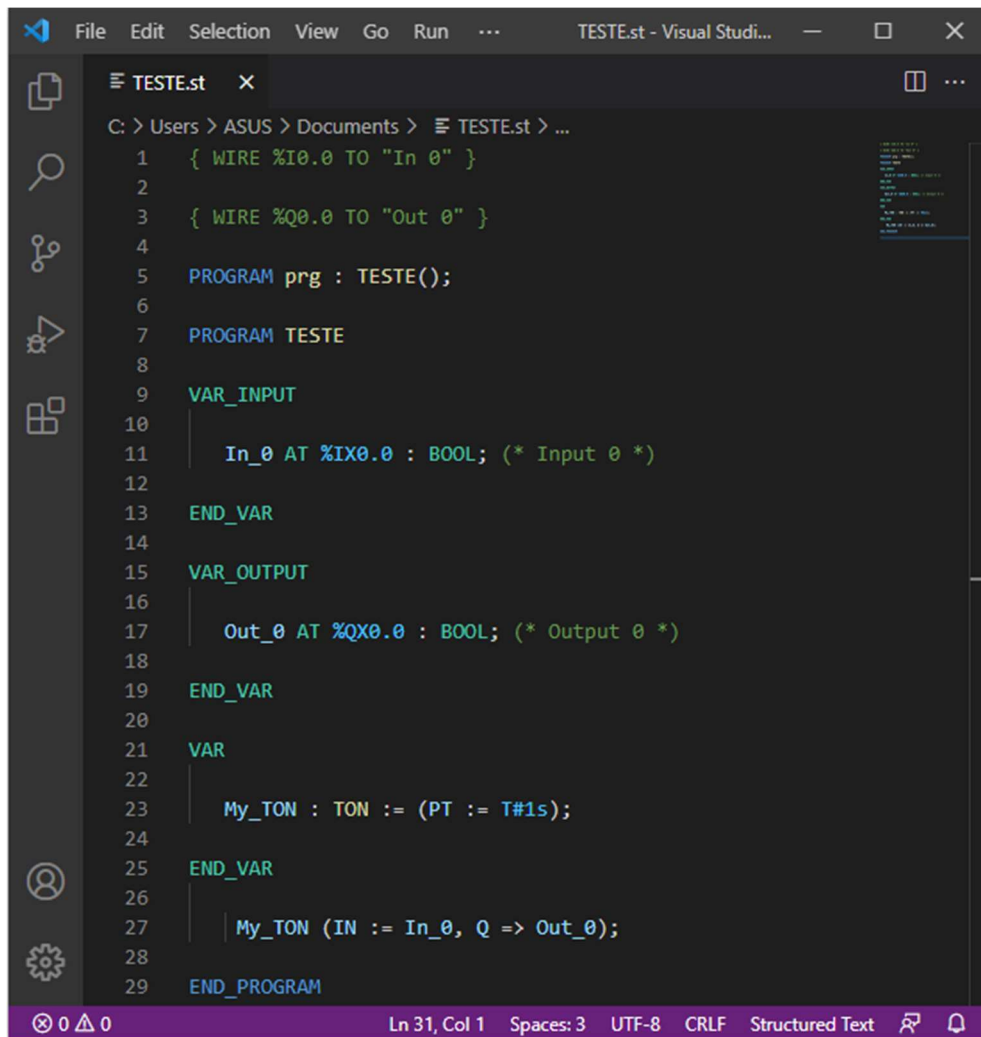
Ao tempo de início desta dissertação, havia já opções tomadas quanto ao método de desenvolvimento do novo PLC. Nomeadamente, decidiu a Real Games que a linguagem central do novo PLC seria o Texto Estruturado. Assim, programas escritos em ST seriam compilados e executados como acontece noutras linguagens, mas programas nas linguagens gráficas LD e FBD seriam transparentemente convertidos para ST e compilados a partir daí. Ou seja, em termos de desenvolvimento, o ST seria a linguagem capital do PLC. Como tal a correção do PLC na execução de programas nesta linguagem seria crítica.

Por questões várias, nomeadamente ligadas à pandemia COVID19, houve algum atraso no desenvolvimento do protótipo deste *soft*PLC tendo só em maio ficado este disponível. Uma vez que o aspeto central era a programação em ST, foi-lhe dado do nome provisório de *ST I/O*.

Ao tempo de escrita desta dissertação o *ST I/O* é uma aplicação ainda não muito alinhavada, mas que mostra já, como se documenta ao longo deste documento, um excelente desempenho e um grande potencial.

Não incluindo ainda um editor próprio, o *soft*PLC é uma aplicação Windows capaz de abrir um ficheiro de texto, interpretar o seu conteúdo na linguagem ST, assinalar eventuais erros e executar o programa em causa.

Veja-se, na Figura 2.1, um exemplo muito simples de um programa em Texto Estruturado desenvolvido na aplicação *Visual Studio Code*, tendo esta sido utilizada como editor temporário, devido à ausência de um próprio do *ST I/O*.



```

1  { WIRE %I0.0 TO "In 0" }
2
3  { WIRE %Q0.0 TO "Out 0" }
4
5  PROGRAM prg : TESTE();
6
7  PROGRAM TESTE
8
9  VAR_INPUT
10 |
11 |   In_0 AT %IX0.0 : BOOL; (* Input 0 *)
12 |
13 | END_VAR
14
15 VAR_OUTPUT
16 |
17 |   Out_0 AT %QX0.0 : BOOL; (* Output 0 *)
18 |
19 | END_VAR
20
21 VAR
22 |
23 |   My_TON : TON := (PT := T#1s);
24 |
25 | END_VAR
26
27 |   My_TON (IN := In_0, Q => Out_0);
28
29 END_PROGRAM
    
```

Figura 2.1 – Programa em Texto Estruturado desenvolvido no *Visual Studio Code*.

Como foi referido, o *ST I/O* assinala erros e variáveis que possam ter sido mal declaradas e, por isso, não sejam passíveis de ser ligadas aos seus correspondentes no *Factory I/O*, indicando ainda a linha de código onde o erro foi detetado, como se pode ver na Figura 2.2.

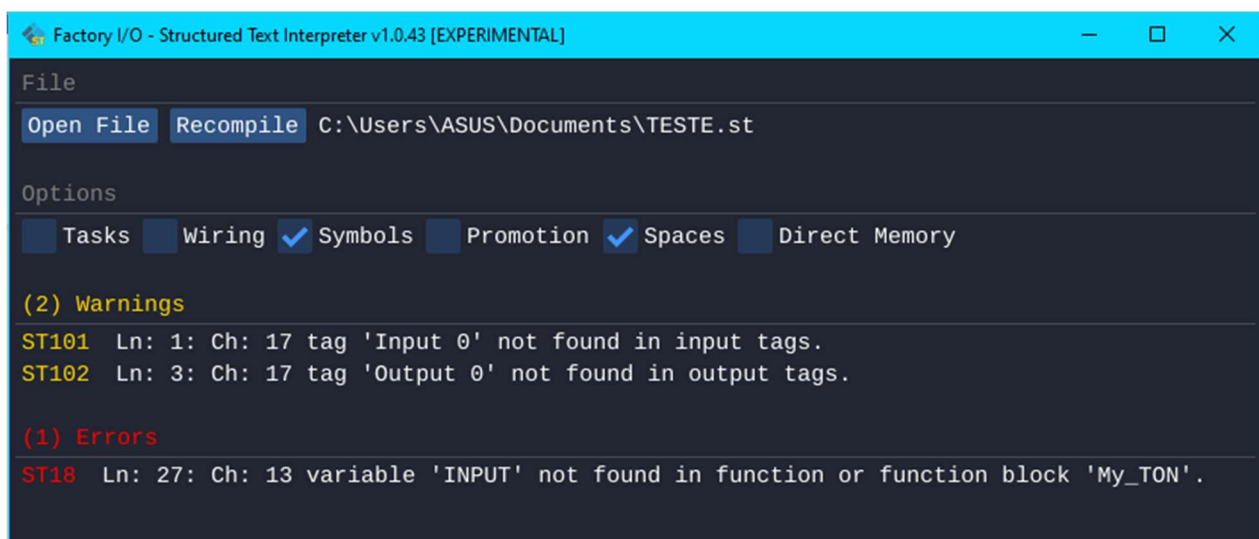


Figura 2.2 – Visualização de avisos e erros no *soft*PLC *ST I/O*.

Além disso, o *soft*PLC *ST I/O* permite ainda acompanhar o estado das variáveis que constam no programa. A Figura 2.3 ilustra essa questão.

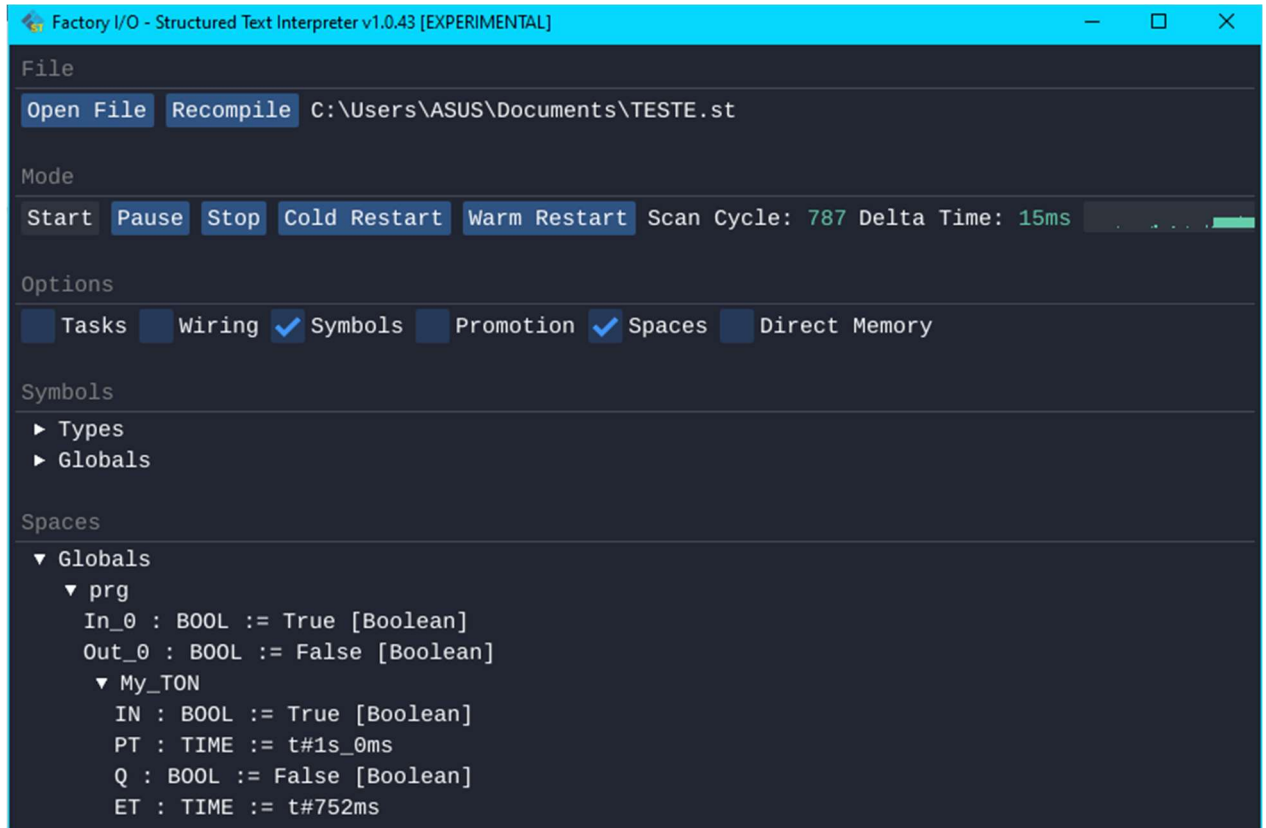


Figura 2.3 – Visualização do estado das variáveis no *soft*PLC *ST I/O*.

2.3 Definição do trabalho a desenvolver

O objetivo da presente dissertação é, como referido anteriormente, o desenvolvimento de material de teste do *soft*PLC que a Real Games está a desenvolver, sendo que este deve, ainda, servir como material didático que, poderá, futuramente, acompanhar o *Factory I/O*.

Inicialmente, definiu-se que a base da abordagem para este trabalho, teria uma componente de análise de outros PLCs, com o objetivo de perceber qual a oferta existente no mercado e que características é que estes apresentavam, procurando assim, recolher os aspetos positivos que fossem de interesse para integrar no novo PLC e aspetos negativos que fossem desejavelmente evitados, sempre com o objetivo de desenvolver o melhor *soft*PLC possível. Assim, tentaram-se vários equipamentos, desde *hard*PLCs a *soft*PLCs. Um desses foi o Siemens S7-1200, no entanto, a programação em Texto Estruturado que este apresenta afasta-se demasiado da norma, sendo por isso pouco interessante, visto que o novo PLC se baseia na norma e procura aplicar a mesma o mais fielmente possível. Tentaram-se ainda alguns *soft*PLCs, no entanto, muitos confirmaram-se inutilizáveis. Por exemplo, procurou-se fazer uso do Crimson, disponibilizado pela *Red Lion Controls*, no entanto, este implicava a necessidade de ter um PLC físico, algo que não foi possível de adquirir. Outros encontram-se descontinuados, tendo sido impossível obtê-los.

Assim, as opções ficaram reduzidas ao *soft*PLC *Codesys*. Um *software* de desenvolvimento gratuito que aplica a norma de um modo muito fiel e com a possibilidade de

ligar ao *Factory I/O*, permitindo a simulação dos sistemas programados. Tendo-se, por isso, desenvolvido todos os programas nesta aplicação para, posteriormente, serem usados como teste do novo PLC.

Assim sendo, tendo em conta o objetivo desta dissertação, decidiu-se que o material a desenvolver deveria ser o mais abrangente possível de modo a testar de forma intensiva o PLC e cobrindo assim diferentes matérias de programação do mesmo em Texto Estruturado, visto ser esta a sua linguagem de programação principal.

Começa-se, então, por introduzir os elementos fundamentais da programação em ST. Para isso, apresentam-se, primeiramente, enunciados simples, constituído por diagramas temporais que representam blocos funcionais, como temporizadores, contadores, *latches* e eventos e expõe-se as respetivas soluções acompanhadas de uma breve explicação dos respetivos blocos. De seguida, aborda-se a programação de máquinas de estado, apresentando diagrama de estados e grafets que devem ser utilizados para desenvolver programas que empregam algumas das instruções fundamentais da programação em ST, nomeadamente, as instruções `IF` e `CASE OF`.

De seguida procura-se mostrar algumas aplicações desses conceitos, fazendo uso de cenário desenvolvidos em *Factory I/O*. Para isso, começa-se por apresentar os mesmos, referindo os respetivos requisitos de funcionamento, seguidos da modelação dos mesmos e, finalmente, a apresentação do código desenvolvido para controlar os mesmos. Com estes casos, procura-se realçar algumas questões referentes à modelação dos problemas chamando à atenção para diferentes estratégias que se podem empregar, como isso pode impactar a programação do sistema e ainda como programar algumas dessas questões.

2.4 Síntese

Ao longo deste capítulo procurou-se fazer um ponto de situação relativamente ao ensino de autómatos programáveis em Portugal, analisando os conteúdos lecionados em disciplinas universitárias, cursos técnicos e profissionais e ainda que abordagem é feita a estes equipamentos enquanto tema alvo de livros técnicos.

Conclui-se que em termos de teoria existe muito material e pouca dificuldade em arranjar meios para transmitir tal conhecimento. No entanto, quando se chega à vertente prática do ensino, o mesmo não se verifica. Isto deve-se, em grande parte, a questões económicas relacionadas com a aquisição de equipamentos. Equipamentos esses que estarão expostos a grandes riscos, dado serem utilizados num contexto de aprendizagem onde erros são cometidos com frequência, o que leva à ocorrência de situações de perigo para as pessoas envolvidas e para os equipamentos, refletindo-se em gastos injustificáveis.

No entanto, a prática é tão, ou mais, importante que a teoria, pelo que surge a necessidade de contornar o problema mencionado. Surgem então os *software* de simulação. No entanto, para funcionar, estes implicam, por vezes, o uso de *hard*PLC, algo que nem sempre é acessível. Assim, a Real Games propõe um *soft*PLC, *ST I/O*, que acompanhará o seu programa de simulação de ambientes industriais, o *Factory I/O*, e que permitirá criar uma ferramenta de ensino e aprendizagem mais completa e com menos limitações.

Finalmente apresentou-se a abordagem definida para a presente dissertação, onde se procurará desenvolver material didático que servirá, ao mesmo tempo, de material de testagem do referido *soft*PLC.

Assim, no capítulo 3 abordar-se-ão as questões fundamentais da programação em Texto Estruturado, de acordo com a norma IEC 61131-3, e ainda a interpretação de diagramas de estados e grafets de modo a desenvolver, a partir dos mesmos, programas que imponham os comportamentos modelados.

3 Aspetos fundamentais da programação em ST

O presente capítulo dedica-se a apresentar e explicar conceitos fundamentais para o desenvolvimento de programas em texto estruturado (ST), programas esses que serviram de elementos de teste do *soft*PLC *ST I/O*. O objetivo é facilitar a transição da utilização de linguagens de baixo nível, como *ladder*, para uma linguagem de alto nível como é o texto estruturado.

O capítulo está dividido em duas grandes secções. A primeira expõe os blocos funcionais que servem de base à programação em ST, enquanto a segunda apresenta diferentes abordagens à programação de máquinas de estados. Tudo feito de acordo com a norma IEC 61131-3.

É extensa a lista de exercícios realizados com o propósito enunciado. Por razões de tempo e espaço apenas alguns deles são aqui apresentados e discutidos. Os exercícios em causa foram retirados do caderno “Exercícios de Programação de PLCs”, proposto como atividade de revisão aos alunos de Computação Industrial [36].

Os problemas da primeira secção têm como objetivo a leitura e interpretação de diagramas temporais, a dedução dos componentes lógicos requeridos para a sua programação e, por fim, o desenvolvimento e teste de um programa em texto estruturado, que imponha o funcionamento apresentado pelo diagrama temporal.

A segunda secção, por sua vez, apresenta problemas que pretendem cimentar a capacidade de leitura e interpretação de especificações lógicas na forma de diagramas de estados e grafets, a compreensão das diferentes formas de programar máquinas de estados, e das vantagens e desvantagens relativas, bem como, à semelhança dos problemas da secção anterior, a programação do comportamento imposto pelos diagramas em questão e verificação dos programas desenvolvidos.

Por razões já apontadas no capítulo anterior, os programas a apresentar foram desenvolvidos em *Codesys 3.5 V16 SP2*. Uma vez que a programação em ST neste ambiente não obedece totalmente ao exposto na norma IEC 61131-3, necessidades houve de fazer pequenas alterações ao código desenvolvido por forma a adaptá-lo ao *soft*PLC *ST I/O*. Estas referidas a devido tempo.

Aproveitando o facto de todos os exercícios contemplarem exclusivamente variáveis de IO booleanas – e apenas uma ou duas de cada tipo, no máximo –, os programas desenvolvidos foram testados num cenário *Factory I/O* constituído por um simples quadro elétrico com botões e luzes, um utilitário natural que facilitou grandemente tanto a materialização das variáveis de entrada e saída como a visualização do resultado da execução do programa. A Figura 3.1 procura ilustrar o referido quadro elétrico. Verificações mais detalhadas foram conseguidas através das facilidades de *debug* do *Codesys*. A ligação entre o *Codesys* e o *Factory I/O* foi feita por OPC UA.

Utilizou-se ainda o *ST I/O* com o objetivo de verificar se este executa corretamente os programas apresentados, de modo a garantir que este permite, nesta fase do seu desenvolvimento, a programação de códigos simples, mas fundamentais. Garantindo que este tem uma base de funcionamento sólida para continuar o seu desenvolvimento.

No final de cada secção apresentam-se, então, algumas observações relativamente ao desempenho do programa de cada problema analisado no *soft*PLC em estudo.



Figura 3.1 – Quadro elétrico utilizado para testar os programas apresentados ao longo da primeira secção.

3.1 Funções lógicas elementares

Nesta primeira secção apresenta-se o primeiro grupo de exercícios. Este tem como objetivo relembrar e consolidar conhecimentos básicos de lógica programada. Nomeadamente, a leitura e interpretação de especificações lógicas na forma de diagramas temporais, a identificação dos componentes necessários à concretização da funcionalidade pretendida e respetiva interação e, finalmente, a programação da solução em texto estruturado segundo a norma IEC 61131-3 e respetiva verificação funcional.

Dos vários exercícios resolvidos apresentam-se neste ponto alguns dos considerados mais representativos dos conceitos didáticos a reter e do modo de os transmitir a quem se inicia na programação em texto estruturado. Concretamente, correspondem às alíneas a), c), d), f), l) e r) do exercício 1.1 do citado caderno. Neles são observadas funcionalidades lógicas essenciais – como *latches*, detetores de transições, temporizadores e contadores, todas bem conhecidas dos programadores em *ladder* – que interessa rever agora no contexto da programação em texto estruturado. Todos os blocos funcionais utilizados nestes problemas podem ser encontrados na biblioteca de qualquer *software* de programação de um PLC compatível com a norma. As variáveis de entrada e saída destes problemas conservam, respetivamente, as denominações *In_n* e *Out_n*, (sendo *n* um identificador numérico superior ou igual a 0), que lhes são atribuídas no enunciado dos exercícios. Há, em alguns casos, a referência a uma variável de entrada denominada *Fact_Run* que corresponde a um booleano com origem na aplicação *Factory I/O*

que toma o valor `TRUE` quando a mesma se encontra em modo “Run”, permitindo que o PLC reconheça e responda à situação de arranque ou paragem de execução do cenário virtual, caso tal seja desejado.

Face ao exposto, deve pois considerar-se que as variáveis de IO dos programas citados neste ponto têm as seguintes declarações:

```
VAR_INPUT:

    In_0, In_1, Factory_Running : BOOL;

END_VAR

VAR_OUTPUT:

    Out_0 : BOOL.

END_VAR
```

Como já foi referido, os programas apresentados ao longo desta secção foram desenvolvidos usando o *software Codesys*. No entanto, este difere na norma IEC 61131-3 no que toca à nomenclatura usada para algumas variáveis dos blocos funcionais. Tendo em conta que os diferentes códigos apresentados foram todos desenvolvidos no referido programa, optou-se por manter a notação empregue por este. Será, no entanto, feita uma chamada de atenção sempre que houver alguma diferença entre a norma e a notação aplicada.

3.1.1 Primeiro Problema

Para este primeiro problema é pedido que se programe o comportamento dado pelo diagrama temporal da Figura 3.2.

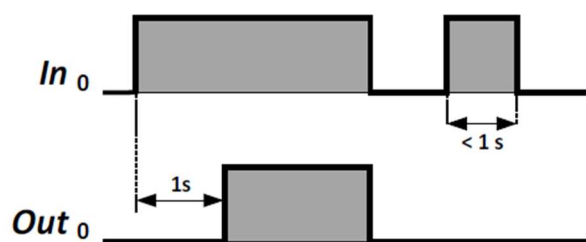


Figura 3.2 – Diagrama temporal a considerar no primeiro problema.

A funcionalidade representada na Figura 3.2 denota um atraso ao trabalho da saída em relação à entrada. Para responder a esta solicitação, a norma contempla um bloco funcional (FB) específico denominado `TON` – temporizador *on-delay*. Este FB está descrito no ponto 6.6.3.5.5 da norma.

À semelhança do que acontece com qualquer variável, a inclusão de um FB num programa IEC 61131-3 só é possível depois de o mesmo ter sido declarado como variável interna e, consequentemente, criado em memória. Concretamente – seja em texto estruturado, seja em qualquer outra linguagem IEC 61131-3 – para criar um temporizador *on-delay*, denominado, a título de exemplo, `MY_TON`, é necessário declará-lo da seguinte forma:

```
VAR
```

```

My_TON : TON;

END_VAR

```

Qualquer instância de um FB do tipo `TON` possui duas variáveis de entrada – `IN`, do tipo `BOOL`, e `PT`, do tipo `TIME` – e duas variáveis de saída – `Q`, do tipo `BOOL`, e `ET` do tipo `TIME`.

A variável `IN` (ou, `My_TON.IN` se se incluir o respetivo *namespace*) funciona como habilitação da temporização; isto é, a transição ascendente desta variável inicia a temporização e a transição descendente cancela-a. A variável `My_TON.PT` quantifica a temporização. Nas saídas, `Q` representa a saída do temporizador, ou seja, indica se a temporização terminou, ou não, e `ET` contabiliza o tempo decorrido desde o início da temporização. Estes aspetos do FB estão resumidos na Tabela 3.1.

Tabela 3.1 – Resumo das variáveis, e respetivas funções, do bloco funcional `TON`.

Variável	Nome	Tipo	Comentário
Entrada	IN	BOOL	Habilitação da temporização (quando <code>TRUE</code>).
	PT	TIME	Valor de referência da temporização.
Saída	Q	BOOL	Saída do temporizador (<code>TRUE</code> quando temporização finalizada).
	ET	TIME	Andamento da temporização.

A Figura 3.3 ilustra o funcionamento deste bloco funcional.

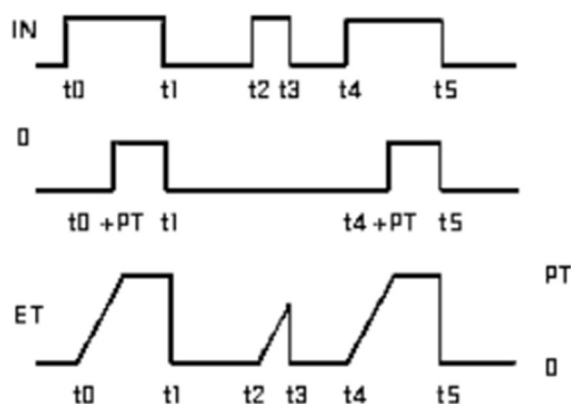


Figura 3.3 – Diagrama de funcionamento de um bloco `TON` [37].

Como solução do presente problema optou-se pelo seguinte código:

```

My_TON (IN := In_0, PT := T#1S, Q => Out_0);

```

Note-se que a variável `PT` é quantificada na forma `T#` seguida do tempo desejado, sendo assumido milissegundos por omissão de unidades de tempo. Para definir unidades deve-se acrescentar, por exemplo, “s” para segundos, como se pode observar no código acima apresentado, “m” para minutos ou “h” para horas.

Existem alternativas à programação. Por exemplo, sendo o valor de `PT` constante é razoável defini-lo logo aquando da declaração do bloco, na forma:

```

VAR

```

```

My_TON : TON := (PT := T#1S);

END_VAR

```

Esta forma dispensa a explicitação da quantificação da temporização no programa, tornando-o mais simples:

```

My_TON (IN := In_0, Q => Out_0);

```

Outra hipótese de programação seria copiar a saída do temporizador para a variável de saída do programa só após a execução do FB, na forma:

```

My_TON (IN := In_0, PT := T#1S);

Out_0 := My_TON.Q;

```

Esta possibilidade, que no presente caso não seria vantajosa, alerta para o facto de as variáveis de entrada e saída dos FBs serem estáticas e terem de ser obrigatoriamente precedidas dos respectivos *namespaces* quando usadas fora das chamadas de execução. Assim, é possível definir o valor da variável *IN* do bloco *My_TON* antes da chamada de execução fazendo-se:

```

My_TON.IN := In_0;

My_TON ();

```

Questões desta ordem aplicam-se aos diversos temporizadores - e FBs em geral - contemplados na norma como se mostra nos próximos exemplos.

3.1.2 Segundo Problema

Para o segundo problema o diagrama a programar é o ilustrado na Figura 3.4.

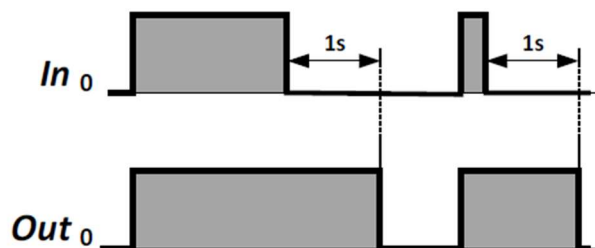


Figura 3.4 – Diagrama temporal a considerar no segundo problema.

O diagrama temporal apresentado ilustra um atraso ao repouso de 1 segundo da saída em relação à entrada. Constata-se, portanto, a necessidade de um temporizador *off-delay*. Este, tal como o bloco analisado no problema anterior, *TON*, está presente na norma, no mesmo ponto, como um FB denominado *TOF*.

À semelhança do bloco TON, o TOF tem duas variáveis de entrada – IN, do tipo BOOL, e PT, do tipo TIME – e duas variáveis de saída – Q, do tipo BOOL, e ET, do tipo TIME.

As variáveis deste FB têm as mesmas funcionalidades que os seus equivalentes no bloco TON. A diferença entre estes blocos reside na lógica da variável IN. Esta continua a servir de gatilho para a temporização, no entanto, no bloco TOF, a temporização inicia-se com a transição descendente de IN, e é cancelada pela sua transição ascendente, contrariamente ao que se verificava no bloco TON. A Tabela 3.2 resume as características das variáveis deste FB.

Tabela 3.2 – Resumo das variáveis, e respetivas funções, do bloco funcional TOF.

Variável	Nome	Tipo	Comentário
Entrada	IN	BOOL	Habilitação da temporização (quando FALSE).
	PT	TIME	Valor de referência da temporização.
Saída	Q	BOOL	Saída do temporizador (FALSE quando temporização finalizada)
	ET	TIME	Andamento da temporização.

O funcionamento deste FB está ilustrado na Figura 3.5.

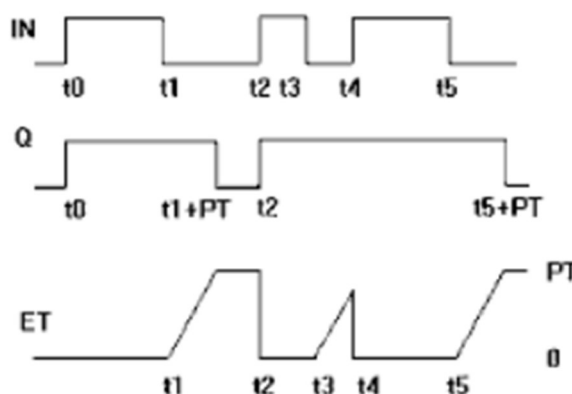


Figura 3.5 – Diagrama de funcionamento de um bloco TOF [38].

A declaração e programação deste bloco segue o mesmo formato que o FB do caso anterior, TON:

VAR

My_TOF : TOF;

END_VAR

My_TOF (IN := In_0, PT := T#1S, Q => Out_0);

3.1.3 Terceiro Problema

O diagrama temporal da Figura 3.6 representa o comportamento a programar.

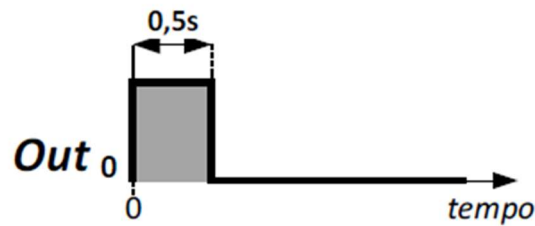
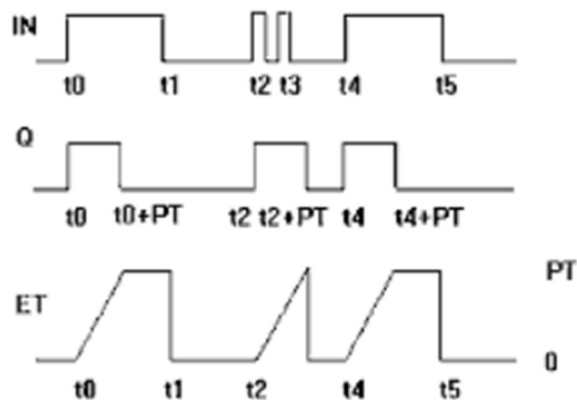


Figura 3.6 – Diagrama temporal a considerar no terceiro problema.

Como se pode observar no diagrama apresentado este problema trata, mais uma vez, de um temporizador, mas agora é um temporizador impulsional. À semelhança dos temporizadores referidos nos problemas anteriores, também este é contemplado na norma, no ponto referido anteriormente, 6.6.3.5.5. Neste caso, o bloco funcional associado tem a denominação *TP*.

Relativamente às variáveis, este FB tem as mesmas que os blocos apresentados anteriormente, no entanto, à semelhança do que se verifica no caso anterior, o comportamento de *IN* é diferente. Enquanto nos blocos *TON* e *TOF* esta variável pode iniciar e cancelar a temporização, no *TP* a primeira transição ascendente inicia a temporização e, até que esta termine, *IN* não tem qualquer efeito no funcionamento do bloco. A Figura 3.7 ilustra o funcionamento deste bloco.

Figura 3.7 – Diagrama de funcionamento de um *TP* [39].

As variáveis de IO deste FB encontram-se resumidas na Tabela 3.3.

Tabela 3.3 – Resumo das variáveis, e respetivas funções, do bloco funcional *TP*.

Variável	Nome	Tipo	Comentário
Entrada	IN	BOOL	Gatilho da temporização;
	PT	TIME	Duração do impulso.
Saída	Q	BOOL	Saída (impulso gerado).
	ET	TIME	Andamento da temporização; Quando atinge o valor de PT mantém-se constante.

A declaração do bloco *TP* é feita da seguinte forma:

```
VAR
    My_TP : TP;
END_VAR
```

Outro aspeto que distingue este problema dos restantes é o facto de este não apresentar uma variável de entrada. O objetivo aqui é que um eventual sinalizador luminoso associado à variável de saída acenda durante 0,5 segundos sempre que se inicia o PLC. No entanto, o *software* de programação utilizado não inclui nenhuma variável que informe se o atual ciclo de execução do PLC é o primeiro ou não. Para resolver esta questão, criou-se uma variável auxiliar, denominada *First_Scan*, que é inicializada a *TRUE*, ou seja, assume este valor quando se inicia o PLC. É esta variável que é usada como entrada do temporizado, resultando o seguinte programa:

```
VAR

    My_TP : TP;

    First_Scan : BOOL := TRUE;

END_VAR

My_TP (IN := First_Scan, PT := T#0.5S, Q => Out_0);
```

Há, no entanto, uma outra hipótese de ver a questão apresentada no contexto da utilização do *Factory I/O* a que se fará alusão em futuros capítulos. A aplicação *Factory I/O* tem, ao contrário do *Codesys*, uma variável que indica o estado do cenário. Isto é, esta variável, mais concretamente denominada de *Fact_Run*, permite saber se a aplicação está em modo “Run” ou não. Assim pode-se utilizar esta variável como condição para iniciar a temporização. Neste caso, o código ficaria:

```
VAR

    My_TP : TP;

    Fact_Run : BOOL;

END_VAR

My_TP (IN := Fact_Run, PT := T#0.5S, Q => Out_0);
```

3.1.4 Quarto Problema

O diagrama a programar neste problema está representado na Figura 3.8.

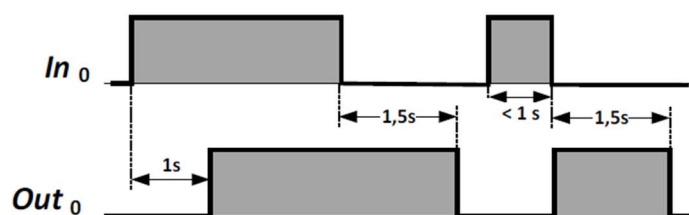


Figura 3.8 – Diagrama temporal a considerar no quarto problema.

Numa primeira análise do diagrama em questão deduz-se facilmente a necessidade de um bloco TON e outro TOF, visto que o diagrama apresenta atrasos quer ao trabalho, quer ao repouso. No entanto, o problema exige um outro bloco funcional: um FB de deteção de transições descendentes. Esta necessidade surge do facto de, na situação em que o TON não dispara, ou seja, a sua temporização não termina, o TOF tem de ser armado. Acontece, no entanto, que utilizar a variável de entrada como habilitação do TOF não é uma condição aceitável, pois, quando esta for a TRUE, a saída do temporizador *off-delay* também será colocada a TRUE, o que vai contra o funcionamento desejado. É preciso, então, detetar a transição descendente da entrada e utilizá-la como condição de habilitação. O FB que permite esta deteção está definido no ponto 6.6.3.5.3 da norma e chama-se F_Trig.

Como já foi referido, é necessário declarar os blocos funcionais para que se possa utilizar os mesmos no programa. Neste problema foi utilizado um bloco F_Trig, ao qual se atribuiu a denominação My_F_Trig. A sua declaração foi feita da seguinte forma:

```
VAR
    My_F_In_0 : F_TRIG;
END_VAR
```

O bloco tem uma variável de entrada - CLK, do tipo BOOL - e uma variável de saída - Q, também do tipo BOOL.

A variável CLK é responsável pela monitorização do sinal cujas transições se quer detetar. Q, por sua vez, devolve o valor TRUE quando é detetada uma transição correspondente à do bloco em uso, neste caso, uma transição descendente. Esta informação pode ser consultada na Tabela 3.4.

Tabela 3.4 – Resumo das variáveis, e respetivas funções, do bloco funcional F_Trig.

Variável	Nome	Tipo	Comentário
Entrada	CLK	BOOL	Variável booleana a monitorizar.
Saída	Q	BOOL	TRUE quando há uma transição descendente entre duas execuções consecutivas.

Este bloco tem a particularidade de ser necessário definir, preferencialmente, no início do programa, qual a variável booleana a monitorizar. Neste caso quer-se avaliar as transições da variável de entrada In_0. Isso é feito da seguinte maneira:

```
My_F_In_0 (CLK := In_0);
```

A solução escolhida é dada pelo seguinte código:

```
My_F_In_0 (CLK := In_0);
My_TON (IN := In_0, PT := T#1S);
My_TOF (IN := My_TON.Q OR FTrig_In_0.Q, PT := T#1.5S, Q => Out_0);
```

Na prática, além de ser possível detetar transições descendentes, como apresentado, pode-se ainda detetar transições ascendentes. Para tal, é definido, no mesmo ponto da norma o bloco *R_Trig*. Este FB é em tudo semelhante ao *F_Trig* já explicado. Por exemplo, a sua declaração, é feita da mesma maneira:

```
VAR

    My_R_In_0 : R_TRIG;

END_VAR
```

As variáveis são as mesmas – *CLK* e *Q*, e o seu funcionamento é o mesmo. A diferença entre estes dois blocos é o que faz a saída assumir o valor *TRUE*. No caso do *F_Trig*, como se pode verificar na Tabela 3.4, é a transição descendente, e no *R_Trig* é a transição ascendente.

3.1.5 Quinto Problema

A Figura 3.9 ilustra o digrama temporal a considerar neste caso.

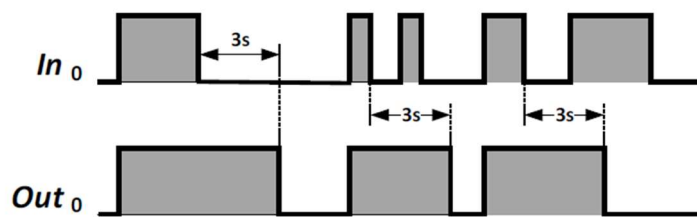


Figura 3.9 – Diagrama temporal do quinto problema.

Observando a Figura 3.9 percebe-se imediatamente a presença de um atraso na propagação do sinal de entrada para a saída. Ou seja, é necessário um bloco *TOF*. No entanto, analisando em detalhe o diagrama, verifica-se que esse bloco, por si só, não é suficiente para a resolução deste problema. Isto acontece porque não é desejável que uma segunda atuação da entrada tenha efeito no temporizador, nomeadamente a reinicialização deste. Assim sendo, será preciso um elemento de memória que retenha a informação sobre a atuação da variável de entrada. Este elemento permitirá então que inibir e desinibir o efeito da entrada na reinicialização do *TOF* de acordo com o que é pedido. Para isso, existe na norma, mais especificamente no ponto 6.6.3.5.2, um grupo de blocos funcionais biestáveis, também chamados *latches*.

À semelhança dos outros FB, estes também têm de ser declarados. Na resolução deste problema utilizou-se um bloco *SR* – cuja condição de set é dominante sobre a de reset - tendo sido feita a sua declaração da seguinte forma:

```
VAR

    My_SR_Latch : SR;

END_VAR
```

Segundo a norma, este bloco tem duas variáveis de entrada - S1 e R, ambas do tipo `BOOL` - e uma variável de saída - Q1, também do tipo `BOOL`. No entanto, como referido anteriormente, há diferenças entre a notação utilizada no *Codesys*, o *software* utilizado para desenvolver os programas apresentados, e a notação definida na norma. Para este caso, a notação utilizada pelo *software* é `SET1` e `RESET`, para as variáveis S1 e R, respetivamente.

A variável `SET1` coloca a saída da *latch* a `TRUE`, quando se verifica a condição associada e esta, enquanto a variável `RESET` faz a saída ser `FALSE`. No entanto, para o bloco `SR` em particular, na situação de ambas as entradas terem o mesmo valor, é a variável `SET1` que determina o valor de Q1. A Tabela 3.5 apresenta esta informação de modo resumido.

Tabela 3.5 – Resumo das variáveis, e respetivas funções, do bloco funcional `SR`.

Variável	Nome	Tipo	Comentário
Entrada	SET1	BOOL	Coloca Q1 a <code>TRUE</code> - dominante.
	RESET	BOOL	Coloca Q1 a <code>FALSE</code> .
Saída	Q1	BOOL	Saída.

Tendo o bloco declarado pode-se agora escrever o programa deste exercício:

```
VAR

    RTrig_In_0 : R_Trig;

    FTrig_TP : F_Trig;

    My_TP : TP;

    MY_SR_Latch : SR;

END_VAR

RTrig_In_0 (CLK := In_0);

FTrig_TP (CLK := My_TP.Q);

My_SR_Latch (SET1 := RTrig_In_0.Q, RESET := FTrig_TP.Q, Q1 => Out_0);

My_TP(IN := NOT In_0 AND My_SR_Latch.Q1, PT := T#3S);
```

Tal como referido no ponto 3.1.1 existem outras possibilidades de programação. Pode-se, por exemplo, utilizar uma variável auxiliar, denominada, a título de exemplo, `My_Bool`, para copiar a saída da *latch*:

```
VAR

    RTrig_In_0 : R_Trig;

    FTrig_TP : F_Trig;

    My_TP : TP;

    My_Bool : BOOL;

    My_SR_Latch : SR;
```

```

END_VAR

RTrig_In_0 (CLK := In_0);

FTrig_TP (CLK := My_TP.Q);

My_SR_Latch (SET1 := RTrig_In_0.Q, RESET := FTrig_TP.Q, Q1 =>
My_Bool);

My_TP(IN := NOT In_0 AND My_SR_Latch.Q1, PT := T#3S);

Out_0 := My_Bool;

```

No entanto, como se pode observar, neste caso, a utilização desta abordagem não acarreta nenhuma vantagem.

Existem dois blocos funcionais biestáveis – SR e RS. O primeiro já foi apresentado. O último é em tudo semelhante ao SR, sendo que a diferença de funcionamento reside na dominância das variáveis de entrada. Ao contrário do que se verifica no FB já apresentado, no RS é a variável de reset que prevalece na situação em que SET e RESET1 têm o mesmo valor.

No que toca às variáveis deste FB são as mesmas que as do SR, havendo apenas uma diferença na sua ortografia. Enquanto o SR tem como variáveis de entrada S1 e R, de acordo com a norma, o bloco RS tem como entradas S e R1. No entanto, tal como no caso anterior, a notação empregue pelo *Codesys* é diferente da definida pela norma. O *Codesys* refere-se a estas variáveis como SET e RESET1, respetivamente. As variáveis de IO deste FB podem ser consultadas na Tabela 3.6.

Tabela 3.6 – Resumo das variáveis, e respetivas funções, do bloco funcional RS.

Variável	Nome	Tipo	Comentário
Entrada	SET	BOOL	Coloca Q1 a TRUE.
	RESET1	BOOL	Coloca Q1 a FALSE - dominante.
Saída	Q1	BOOL	Saída.

Caso fosse mais interessante, ou até mesmo, necessária a utilização de uma *latch* deste tipo - com reset dominante - então a resolução teria a seguinte apresentação:

```

VAR

    RTrig_In_0 : R_Trig;

    FTrig_TP : F_Trig;

    My_TP : TP;

    My_RS_Latch : RS;

END_VAR

RTrig_In_0 (CLK := In_0);

```

```

FTrig_TP (CLK := My_TP.Q);

My_RS_Latch (SET := RTrig_In_0.Q, RESET1 := FTrig_TP.Q, Q1 => Out_0);

My_TP(IN := NOT In_0 AND My_RS_Latch.Q1, PT := T#3S);

```

Neste exercício poderia ser utilizado qualquer um dos dois blocos biestáveis apresentados, visto que o funcionamento do programa teria o mesmo resultado em ambas as situações. Isto pode ser explicado pelo facto de as variáveis de entrada nunca terem o mesmo valor, isto é, as condições de set e reset são eventos de curta duração o que faz com que nunca sejam *TRUE* ao mesmo tempo. Assim, nunca surge a necessidade de verificar qual a dominância em causa.

3.1.6 Sexto Problema

Para este último problema é pedido ao aluno que desenvolva um programa baseado no diagrama temporal da Figura 3.10.

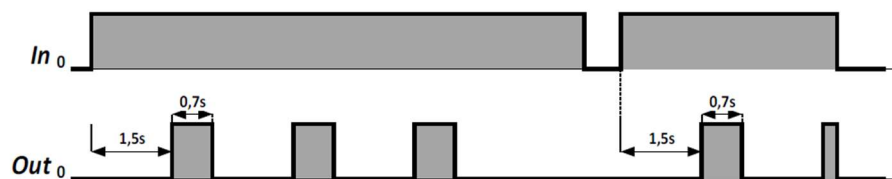


Figura 3.10 – Diagrama temporal do sexto problema.

Analisando o diagrama apresentado identifica-se a necessidade de dois temporizadores, um *on-delay* e um *off-delay*, formando um astável. No entanto, é possível observar no mesmo diagrama que é preciso limitar o número de vezes que a variável de saída é colocada a *TRUE*. Para dar resposta a esta situação a norma contempla blocos funcionais na forma de contadores. Estes estão definidos no ponto 6.6.3.5.4, onde é referido que dentro deste grupo existem três FB diferentes. Estes blocos podem ser crescentes, CTU, decrescentes, CTD, ou crescentes/decrescentes, CTUD. Esta classificação está relacionada com o funcionamento dos contadores, tal como se explica nos parágrafos seguintes.

Para se poder utilizar este FB no programa é necessário, tal como referido anteriormente, declarar o bloco a utilizar. Neste caso optou-se pela utilização de um CTU:

```

VAR

    My_Counter : CTU;

END_VAR

```

No entanto, como se perceberá brevemente, poder-se-ia ter feito uso de qualquer um dos três contadores disponíveis, sendo apenas necessário fazer a respetiva adaptação.

De acordo com a norma, um contador crescente tem três variáveis de entrada – CU e R, ambas do tipo *BOOL* e PV, do tipo *INT* – e duas variáveis de saída – Q, do tipo *BOOL*, e CV do tipo

INT. No entanto, como referido, o *Codesys* apresenta uma notação diferente. A variável *R* surge como *RESET* e as variáveis *PV* e *CV* são apresentadas como sendo do tipo *WORD*. Pelas razões já apresentadas aplicou-se, novamente, a notação do *Codesys*.

Relativamente às variáveis de entrada, *CU* funciona como condição de incremento enquanto a variável *RESET*, quando *TRUE*, força *CV* a assumir o valor de zero. A variável de entrada *PV* permite definir o limite máximo do contador, isto é, o valor até ao qual o bloco incrementa. Quanto às variáveis de saída, *Q* é a saída propriamente dita, ou seja, é *TRUE* quando *CV* é maior, ou igual, ao valor definido em *PV*. Caso contrário esta variável devolve o valor *FALSE*. Finalmente, a variável *CV* indica o valor atual do contador. Esta informação encontra-se resumida na Tabela 3.7.

Tabela 3.7 – Resumo das variáveis, e respetivas funções, do bloco funcional *CTU*.

Variável	Nome	Tipo	Comentário
Entrada	CU	BOOL	Incrementa o contador (transição ascendente).
	Reset	BOOL	Faz reset de CV (quando <i>TRUE</i>).
	PV	WORD	Valor de referência.
Saída	Q	BOOL	É <i>TRUE</i> se CV for maior ou igual a PV.
	CV	WORD	Valor atual do contador.

Utilizando então um *CTU* a solução do problema vem:

```
My_F_In_0 (CLK := In_0);

My_TOF(IN := My_TON.Q, PT := T#0.7S);

My_TON(IN := NOT My_TOF.Q AND In_0, PT := T#1.5S);

My_Counter (CU := My_TON.Q, RESET := My_F_In_0.Q, PV := 4);

Out_0 := My_TOF.Q AND In_0 AND NOT My_Counter.Q;
```

Como foi dito, podia ter-se decidido utilizar um contador decrescente, *CTD*, em vez de um crescente. Se se tivesse seguido essa abordagem era necessário apenas ter em conta que, agora, estaríamos a começar de um valor diferente de zero, e a diminuir esse valor até se chegar a zero, ao contrário do que acontecia com o *CTU*.

Segundo a norma, um bloco *CTD* tem também três variáveis de entrada – *CD* e *LD*, do tipo *BOOL*, e *PV* do tipo *INT* – e duas de saída – *Q*, do tipo *BOOL*, e *CV*, do tipo *INT*. A notação empregue pelo *Codesys* apresenta *PV* e *CV* como *WORD* e a variável *LD* surge como *LOAD*.

As variáveis *CV* e *Q* têm um comportamento semelhante ao que foi apresentado para o *CTU*. A diferença entre estes dois blocos reside pois nas restantes variáveis. Relativamente às entradas do *CTD*, a variável *CD* funciona como gatilho para o decremento, sendo este, à semelhança do incremento anteriormente referido, unitário. *LOAD*, quando está a *TRUE*, força *CV* a um valor igual ao definido em *PV*. Por fim, esta última define o valor inicial do contador, ou seja, o valor a partir do qual se decrementa. A Tabela 3.8 apresenta esta informação de uma forma resumida.

Tabela 3.8 – Resumo das variáveis, e respectivas funções, do bloco funcional CTD.

Variável	Nome	Tipo	Comentário
Entrada	CD	BOOL	Decrementa o contador (transição ascendente).
	LOAD	BOOL	Inicializa o contador no valor PV (quando TRUE).
	PV	WORD	Valor de referência.
Saída	Q	BOOL	É TRUE se CV for igual ou menor que zero.
	CV	WORD	Valor atual do contador.

Caso se decidisse utilizar um CTD a sua declaração e o código para a resolução deste problema teriam a seguinte forma:

```

VAR

    My_Counter : CTD;

END_VAR

My_F_In_0(CLK := In_0);

My_TOF(IN := My_TON.Q, PT := T#0.7S);

My_TON(IN := NOT My_TOF.Q AND In_0, PT := T#1.5S);

My_Counter (CD := My_TON.Q, LOAD := My_F_In_0.Q, PV := 4);

Out_0 := My_TOF.Q AND In_0 AND NOT My_Counter.Q;

```

Neste caso observa-se que além das variáveis do contador não existem outras diferenças em termos de código. Isto porque a condição utilizada para incrementar o CTU pode ser usada como condição para decrementar o CTD. Além disso, as suas saídas são colocadas a TRUE quando os contadores incrementam, ou decrementam, o mesmo número de vezes. Isto faz com que o resultado seja o mesmo, independentemente do FB usado.

Como foi referido, além dos contadores CTU e CTD existe também o CTUD. Este tem a capacidade de operar como qualquer um dos anteriores.

Em termos de variáveis, e de acordo com a norma, um CTUD, tem cinco de entrada – CU, CD, R e LD, todas do tipo BOOL, e PV, do tipo INT – e três de saída – QU e QD, do tipo BOOL, e CV, do tipo INT. O que se verificou para os contadores anteriormente apresentados também se constata neste. No *Codesys*, R e LD são denominadas de RESET e LOAD, respetivamente, e PV e CV são do tipo WORD.

As variáveis deste FB têm todas as mesmas funções que desempenhavam nos blocos anteriormente abordados, CTU e CTD. A única diferença, relativamente a esses FB, reside na denominação das saídas, mais concretamente, QU e QD. Estas representam, respetivamente, a saída associada à função incremental e decremental deste contador, como se pode observar na Tabela 3.9.

Tabela 3.9 – Resumo das variáveis, e respetivas funções, do bloco funcional CTUD.

Variável	Nome	Tipo	Comentário
Entrada	CU	BOOL	Incrementa o contador (transição ascendente).
	CD	BOOL	Decrementa o contador (transição ascendente).
	Reset	BOOL	Faz reset de CV (quando TRUE).
	LOAD	BOOL	Inicializa o contador no valor PV (quando TRUE).
	PV	WORD	Valor de referência.
Saída	QU	BOOL	É TRUE se CV for maior ou igual a PV.
	QD	BOOL	É TRUE se CV for igual ou menor que zero.
	CV	WORD	Valor atual do contador.

Os programas apresentados como solução para os problemas apresentados foram todos testados no *soft*PLC *ST I/O* sendo que foram todos executados sem problemas. No entanto, foi necessário fazer algumas alterações a alguns dos programas, nomeadamente, no caso das *latches* e contadores. As variáveis SET1, SET, RESET1, RESET e LOAD tiveram e ser alteradas para S1, S, R1, R e LD, respetivamente. Estas alterações prendem-se com facto de, como já mencionado, o *Codesys* utilizar uma notação diferente da definida pela norma, sendo que o *ST I/O* aplica a notação da norma.

3.2 Máquinas de estados

As máquinas de estados são modelos que permitem mapear o comportamento de sistemas sequenciais. Estas são habitualmente constituídas por estados que contêm a informação sobre a saída que o sistema deve produzir em cada um deles. A mudança de estados é estabelecida por transições. Estas indicam a condição, ou condições, a verificar para que o sistema transite de um estado para outro.

Estes modelos são importantes porque permitem obter uma representação gráfica fácil de ler e intuitiva do comportamento do sistema. Além disso, é possível modelar este tipo de sistemas desde os mais simples, constituídos por apenas dois estados, até aos mais complexos, com dezenas de estados.

Dois tipos muito comuns de máquinas de estados são os diagramas de estados e os grafsets. No entanto, a sua aplicabilidade não é necessariamente a mesma. Isto porque, enquanto os grafsets permitem modelar, com facilidade, tanto sistemas simples como complexos, os diagramas de estados convencionais devem ser utilizados apenas em casos com poucos estados. Isto porque tornam-se difíceis de ler com o aumento do número de estados.

Esta segunda secção apresenta um segundo grupo de exercícios que tem como objetivo relembrar e cimentar conceitos relacionados com a leitura e interpretação de especificações lógicas sequenciais mais complexas e genéricas. Estas podem surgir sob a forma dos referidos diagramas de estados ou grafsets. Procura-se então, com este grupo, consolidar a compreensão das diferentes formas de programar máquinas de estados em texto estruturado, de acordo com a norma IEC 61131-3, bem como as respetivas vantagens e desvantagens e, semelhantemente ao grupo anterior, a verificação funcional da solução obtida e ainda eventuais alternativas.

Tem-se como objetivo, tal como foi dito, expor diferentes maneiras de programar máquinas de estados. Isto porque não há uma fórmula genérica para traduzir eficientemente estes modelos. É preciso, pois, analisar cada caso para determinar qual a melhor estratégia a empregar de modo que o sistema funcione como desejado.

Como referido, as especificações lógicas podem ser apresentadas sob a forma de diagramas de estado ou grafsets. Para melhor expor as possibilidades de programação destes,

foram resolvidos alguns exercícios cujo enunciado continha estes elementos. Nesta secção apresentam-se os considerados mais relevantes. Para os diagramas de estados apresentam-se todos visto tratar-se de apenas três alíneas. Já para os grafets foram seleccionadas as alíneas a), b) e c) do exercício 3.5 do caderno já mencionado, por serem as que facilitam a apresentação de detalhes de interesse. Foi escolhido este conjunto de exercícios pois permitem expor uma diversidade de situações.

Relativamente às variáveis de IO envolvidas nos exercícios deste grupo, estas são identificadas como I_n e O_n (onde o índice n é um identificador numérico superior ou igual a 0), de acordo com os enunciados dos exercícios.

3.2.1 Leitura, interpretação e programação de diagramas de estado

Os diagramas de estados são ideais para a representação de sistemas simples que se resumem a poucos estados. Na programação destes diagramas, em texto estruturado, ou em qualquer outra linguagem de programação contemplada pela norma, a chave é saber que estados compõem o sistema e o que é que faz set e reset de cada um, ou seja, o que é que faz com que se ative ou desative cada estado.

Para estes problemas, à semelhança do que acontece nos da secção anterior, é necessário declarar as variáveis de IO.

```
VAR_INPUT
    In_1 : BOOL;
    In_2 : BOOL;
    In_3 : BOOL;
    In_4 : BOOL;
END_VAR

VAR_OUTPUT
    Out_0 : BOOL;
    Out_1 : BOOL;
    Out_2 : BOOL;
END_VAR
```

3.2.1.1 Primeiro Problema

Para este problema de diagramas de estados é pedido que se programe o diagrama da Figura 3.11.

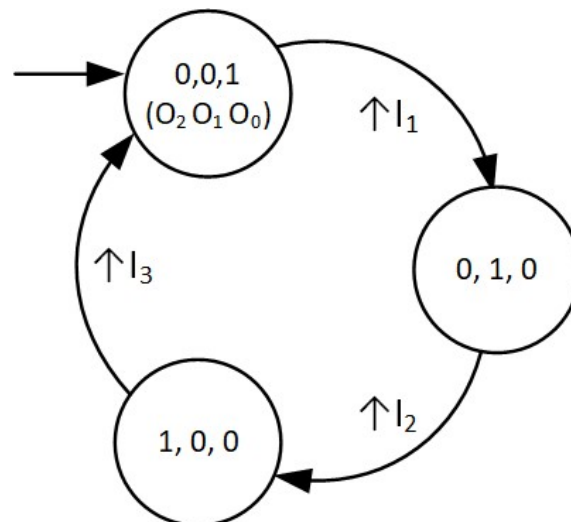


Figura 3.11 – Diagrama de estados do primeiro problema.

Como se pode observar no diagrama apresentado, este é composto por três estados e três transições, todas diferentes entre si. Para facilitar a compreensão da explicação aqui apresentada define-se o estado 0 como sendo o estado inicial, ou seja, o estado em que as variáveis de saída têm os valores (0,0,1), o estado 1 como sendo o imediatamente a seguir, isto é, o estado caracterizado por (0,1,0) e, finalmente, o estado 2 será o estado dado por (1,0,0). Esta informação pode ser facilmente consultada na Tabela 3.10.

Para a programação deste sistema pode-se recorrer a duas *latches* para codificar, através da sua combinação, os estados do sistema. Assim, utilizaram-se dois blocos deste tipo, de reset dominante, cuja denominação utilizada é *RS_State_1* e *RS_State_2*. Utilizaram-se FB de reset dominante, no entanto, podia-se ter utilizado uma *latch* de set dominante, visto que neste caso não tem impacto no funcionamento do sistema.

Tabela 3.10 – Definição dos diferentes estados do primeiro problema.

Estado	RS_State_1.Q	RS_State_2.Q	O ₀	O ₁	O ₂
0	FALSE	FALSE	TRUE	FALSE	FALSE
1	TRUE	FALSE	FALSE	TRUE	FALSE
2	TRUE	TRUE	FALSE	FALSE	TRUE

Tendo em conta o diagrama apresentado no enunciado, e como se pode verificar na Tabela 3.10, definiu-se o estado inicial como sendo caracterizado pela situação em que a saída das duas *latches* estão a FALSE. O segundo estado corresponde, assim, ao caso em que a *latch* *RS_State_1* está a TRUE e a *RS_State_2* é FALSE. Por sua vez, o estado 2 ocorre quando ambas as *latches* estão a TRUE.

Como foi referido, é necessário conhecer as condições de set e reset dos estados. Analisando o diagrama da Figura 3.11, verifica-se que a condição de ativação do estado 1, ou seja, da *latch* *RS_State_1*, é a variável de entrada *I*₁ ser colocada a TRUE quando o sistema está no seu estado inicial. Por sua vez, a condição de set do estado 2, isto é, da *latch* *RS_State_2*, é o sistema estar no estado 1 e haver atuação da variável *I*₂. O reset, no entanto, é igual para as duas *latches*, sendo feito quando a variável *I*₃ vai a TRUE quando o sistema está no estado 2. Esta informação encontra-se resumida na Tabela 3.11.

Tabela 3.11 – Transições do diagrama do primeiro problema.

Estado de partida	Estado atingido		
	R_In_1.Q = TRUE	R_In_2.Q = TRUE	R_In_3.Q = TRUE
0	1	X	X
1	X	2	X
2	X	X	0

Quanto às variáveis de saída, como se pode observar na Tabela 3.10, a variável O_0 só está a TRUE quando o sistema está no seu estado inicial. Já a variável O_1 vai a TRUE quando o sistema está no estado 1, ou seja, RS_State_1 é TRUE, mas RS_State_2 é FALSE. Esta segunda condição é necessária para que O_1 vá a FALSE quando o sistema transita para o estado 2. A variável O_2 é colocada a TRUE quando o estado 2 está ativo.

De seguida apresenta-se o código para este problema.

VAR

R_In_1 : R_Trig;

R_In_2 : R_Trig;

R_In_3 : R_Trig;

RS_State_1 : RS;

RS_State_2 : RS;

END_VAR

R_In_1 (CLK := In_1);

R_In_2 (CLK := In_2);

R_In_3 (CLK := In_3);

RS_State_1 (SET := R_In_1.Q, RESET1 := R_In_3.Q AND RS_State_2.Q1);

RS_State_2 (SET := R_In_2.Q AND RS_State_1.Q1, RESET1 := R_In_3.Q AND RS_State_2.Q1);

Out_0 := NOT RS_State_1.Q1;

Out_1 := RS_State_1.Q1 AND NOT RS_State_2.Q1;

Out_2 := RS_State_2.Q1;

Neste caso, visto que as transições são diferentes entre si, é possível omitir o estado do sistema das condições de set e reset, definindo estas usando apenas as variáveis de entrada, ao contrário do que acontece no problema que se segue, como se atentará.

Outra hipótese de programação deste diagrama é utilizando a instrução IF, e substituindo as *latches* por variáveis binárias com uma denominação do tipo Mem_n (onde n é, à semelhança do que se definiu para as variáveis de IO, um identificador numérico igual ou

superior a 0), que funcionam como memórias. Ou seja, estas substituem as *latches* como retentoras do estado do sistema. A programação correspondente a esta hipótese apresenta-se de seguida.

```
VAR

    R_In_1 : R_Trig;

    R_In_2 : R_Trig;

    R_In_3 : R_Trig;

    Mem_0 : BOOL;

    Mem_1 : BOOL;

END_VAR


R_In_1 (CLK := In_1);
R_In_2 (CLK := In_2);
R_In_3 (CLK := In_3);

IF NOT Mem_0 AND NOT Mem_1 AND R_In_1.Q THEN

    Mem_0 := TRUE;

ELSIF Mem_0 AND NOT Mem_1 AND R_In_2.Q THEN

    Mem_1 := TRUE;

ELSIF Mem_0 AND Mem_1 AND R_In_3.Q THEN

    Mem_0 := FALSE;

    Mem_1 := FALSE;

END_IF


Out_0 := NOT Mem_0 AND NOT Mem_1;
Out_1 := Mem_0 AND NOT Mem_1;
Out_2 := Mem_1;
```

3.2.1.2 Segundo Problema

A Figura 3.12 ilustra o diagrama de estados a programar.

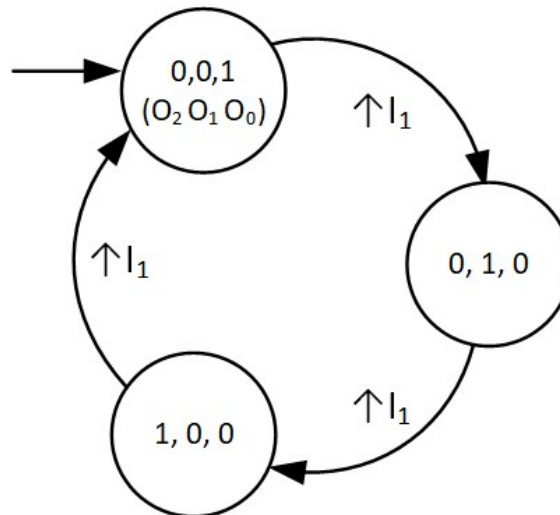


Figura 3.12 – Diagrama de estados do segundo problema.

Embora este exercício tenha um enunciado muito semelhante ao anterior, numa análise mais atenta, verifica-se que tal não é verdade. Existe, de facto, uma diferença que faz com que seja necessária uma estratégia de programação algo diferente. Isto deve-se ao facto de que, como se pode observar, as transições são iguais em todo o diagrama, ao contrário do que acontecia no problema anterior.

Dadas as referidas semelhanças entre este diagrama e o do problema anterior tentou-se aplicar as mesmas estratégias de programação. No entanto, numa primeira abordagem, concluiu-se que não seria possível fazê-lo com *latches*. Isto porque é necessário reter informação sobre a atuação de I_1 e isso teria de ser feito com recurso a variáveis auxiliares, como se poderá observar brevemente. Por isso, se se usassem estes FB, o código ficaria complicado de ler e confuso.

Tentou-se, então, aplicar o código do problema anterior que emprega a instrução `IF`. Para isso foi preciso adaptar o código ao presente problema, ou seja, teve-se de mudar as condições das transições. No entanto, verificou-se que ocorrem situações transitórias. Estas são situações em que dado as transições de dois ou mais estados consecutivos serem idênticas quando uma é verificada como `TRUE` todas são verificadas como tal no mesmo ciclo. Isto faz com que, nesse mesmo ciclo, os respetivos estados sejam ativados e desativados, dado que as suas condições de *set* e *reset* se verificam no mesmo ciclo. Isto leva a que as ações associadas a estes estados não sejam executadas.

Uma possível solução para programar o diagrama de estados deste problema, usando, então, como base o referido código do problema anterior, e evitar as situações transitórias é a utilização da instrução `ELSIF` no ciclo `IF` já existente. Isto permite garantir que, quando o programa executa o ciclo `IF`, só pode verificar uma das diferentes opções que constituem a instrução. Deste modo garante-se que o programa não tenha etapas fugazes e que este executa as todas as ações. Resulta, então, o seguinte código:

```

VAR

    R_In_1 : R_Trig;

    Mem_0 : BOOL;

    Mem_1 : BOOL;

END_VAR
  
```

```

R_In_1 (CLK := In_1);

IF NOT Mem_0 AND NOT Mem_1 AND R_In_1.Q THEN

    Mem_0 := TRUE;

ELSIF Mem_0 AND NOT Mem_1 AND R_In_1.Q THEN

    Mem_1 := TRUE;

ELSIF Mem_0 AND Mem_1 AND R_In_1.Q THEN

    Mem_0 := FALSE;

    Mem_1 := FALSE;

END_IF

Out_0 := NOT Mem_0 AND NOT Mem_1;

Out_1 := Mem_0 AND NOT Mem_1;

Out_2 := Mem_1;

```

Outra hipótese de programação do presente problema evita a utilização de instruções `IF` ou `CASE OF`. Para tal, é necessário definir as variáveis com que se irá trabalhar. Neste caso, visto que o uso de *latches* não é uma solução plausível, definiu-se variáveis de denominação `State_n` para registrar a transição que foi verificada. Além destas utilizaram-se ainda as variáveis de memória, já introduzidas no problema anterior, e que foram utilizadas no primeiro exemplo de programação deste problema – `Mem_n`. Estas têm as mesmas funcionalidades que no problema anterior. A codificação dos estados do sistema recorrendo a estas variáveis está resumida na Tabela 3.12.

Tabela 3.12 – Codificação dos estados do sistema do segundo problema.

Estado	Mem_0	Mem_1	O ₀	O ₁	O ₂
0	FALSE	FALSE	TRUE	FALSE	FALSE
1	TRUE	FALSE	FALSE	TRUE	FALSE
2	TRUE	TRUE	FALSE	FALSE	TRUE

As estruturas de decisão do tipo `State` retêm informação sobre a atuação do botão I_1 , fazendo com que o programa não tenha etapas fugazes. Isto é conseguido porque as variáveis codificadoras do estado do sistema – `Mem_0` e `Mem_1` – dependem das variáveis `State` para poderem ser colocadas a `TRUE`. Por sua vez, estas últimas estão sujeitas aos valores das variáveis de memória. Este ciclo fechado de funcionamento permite, como já foi referido, evitar a ocorrência de situações transitórias. À semelhança do que se verificava no primeiro problema, o estado 2 faz reset ao sistema, ou seja, coloca as variáveis `Mem_0` e `Mem_1` a `FALSE`. Na Tabela 3.13 estão resumidas as transições do diagrama de estados do segundo problema.

Tabela 3.13 – Transições do diagrama de estados do segundo problema.

Estado de partida	Estado atingido
	R_In_1.Q = TRUE
0	1
1	2
2	0

O programa desenvolvido com base nesta lógica é o seguinte:

VAR

R_In_1 : R_Trig;

Mem_0 : BOOL;

Mem_1 : BOOL;

State_0 : BOOL;

State _1 : BOOL;

State _2 : BOOL;

END_VAR

R_In_1 (CLK := In_1);

State _0 := NOT Mem_0 AND R_In_1.Q;

State _1 := Mem_0 AND NOT Mem_1 AND R_In_1.Q;

State _2 := Mem_1 AND R_In_1.Q;

Mem_0 := (State _0 OR Mem_0) AND NOT State _2;

Mem_1 := (State _1 OR Mem_1) AND NOT State _2;

Out_0 := NOT Mem_0;

Out_1 := Mem_0 AND NOT Mem_1 ;

Out_2 := Mem_1;

3.2.1.3 Terceiro Problema

Na Figura 3.13 pode-se ver o diagrama a programa para este problema.

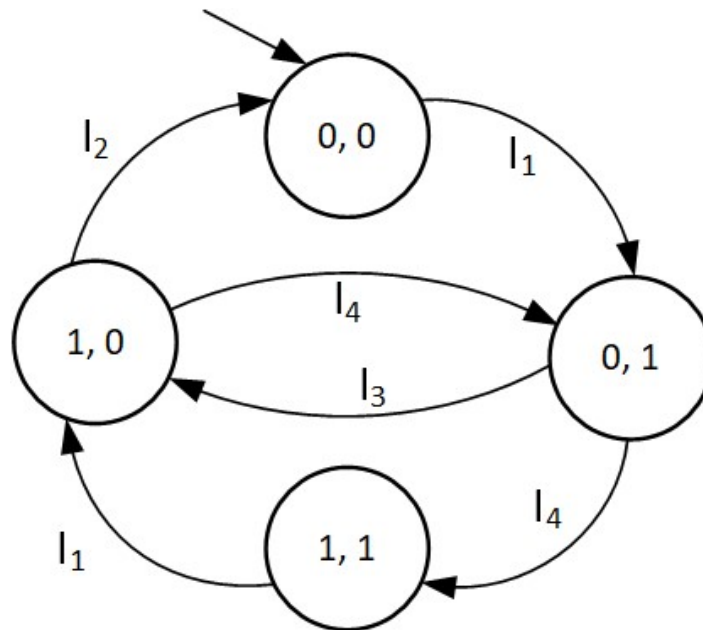


Figura 3.13 – Diagrama de estados do terceiro problema.

Como se pode observar, este problema apresenta um diagrama com 4 estados e mais do que uma trajetória possível, sendo que dependendo das condições que se verificam o sistema pode seguir trajetórias diferentes. Para programar este exercício é necessário, à semelhança dos anteriores, perceber que estados compõem o sistema e as respectivas condições de ativação e inativação de cada um.

Analisando o diagrama em questão, mais concretamente as variáveis de saída, constata-se que é necessário definir que posição corresponde a que saída. Defina-se, então, que o dígito à esquerda, em cada estado, representa a variável de saída O_1 e o da direita a variável O_0 . Deste modo, os estados podem ser codificados como a Tabela 3.14 resume.

Tabela 3.14 – Codificação dos estados do sistema do terceiro problema.

Estado	O_0	O_1
0	FALSE	FALSE
1	TRUE	FALSE
2	TRUE	TRUE
3	FALSE	TRUE

Assim, dada a complexidade do sistema apresentado, derivada da presença de estados com mais do que uma transição, utilizou-se a instrução `CASE OF` como base da solução deste exercício.

Esta permite construir uma lista de instruções que tenham a mesma variável como condição. Neste caso, essa variável é o estado em que o sistema se encontra. Para isso é, então, necessário criar uma variável que registre o estado do sistema. Neste exercício denominou-se essa variável, a título de exemplo, `State`. Dado o modo de funcionamento da instrução em causa esta variável deve ser definida como sendo do tipo `UINT`.

Assim sendo, o que esta instrução faz, mais concretamente, é comparar o valor atual da variável em causa - neste caso, `State` - com os valores listados e executar a instrução correspondente. Na presente situação essa instrução será, para qualquer caso, mudar de estado. No entanto, é preciso ter em atenção dois casos particulares, sendo eles os estados 1 e 3. Como já foi mencionado, dependendo da transição que é verificada, o estado subsequente é diferente, isto é, por exemplo, o estado 1 pode conduzir o sistema para o estado 2 ou para o estado 3. Isto

implica que para a situação em que a variável `State` vale 1 a instrução `CASE OF` deve, para esse mesmo valor, contemplar duas instruções – uma para cada estado possível. O mesmo é aplicado para o caso de a variável valer 3. No final do programa escrevem-se as variáveis de saída usando os estados em que estas são `TRUE`. A Tabela 3.15 resume as transições do diagrama de estados do problema em questão.

Tabela 3.15 – Transições do diagrama de estados do terceiro problema.

Estado de partida	Estado atingido			
	In_1 = TRUE	In_2 = TRUE	In_3 = TRUE	In_4 = TRUE
0	1	X	X	X
1	X	X	3	2
2	3	X	X	X
3	X	0	X	1

De seguida apresenta-se o código escrito como solução deste problema.

```

VAR

    State : UINT;

END_VAR

CASE State OF

0:

    IF In_1 THEN

        State := 1;

    END_IF

1:

    IF In_4 THEN

        State := 2;

    ELSIF In_3 THEN

        State := 3;

    END_IF

2:

    IF In_1 THEN

        State := 3;

    END_IF

3:

    IF In_2 THEN

```

```

        State := 0;

    ELSIF In_4 THEN

        State := 1;

    END_IF

END_CASE

Out_0 := State = 1 OR State = 2;

Out_1 := State = 2 OR State = 3;

```

Os programas apresentados como solução para os digramas de estados estudados foram testados no *soft*PLC *ST I/O* e todos foram executados sem qualquer problema ou alteração.

3.2.2 Leitura, interpretação e programação de grafkets

Tal como os diagramas de estado, os grafkets podem ser utilizados para representar, graficamente, o funcionamento desejado para um determinado sistema. Ambos os grafismos são constituídos por estados, ou etapas no caso do grafket, e condições de transição entre estes. No entanto, enquanto os primeiros são interessantes para situações com um número reduzido de estados, os grafkets permitem uma apresentação mais simples e clara de sistemas complexos constituídos por vários estados, possibilitando ainda a modelação de situações em que seja possível ter dois, ou mais, estados ativos ao mesmo tempo. Outra situação que os grafkets permitem modelar são as hierarquias, isto é, casos em que um grafket pode, como ação associada a uma etapa, ativar ou desativar outro grafket. Estas situações são exploradas nos problemas que se seguem.

3.2.2.1 Primeiro Problema

Para este problema é pedido que o aluno desenvolva um programa que resulte no comportamento especificado no grafket da Figura 3.14.

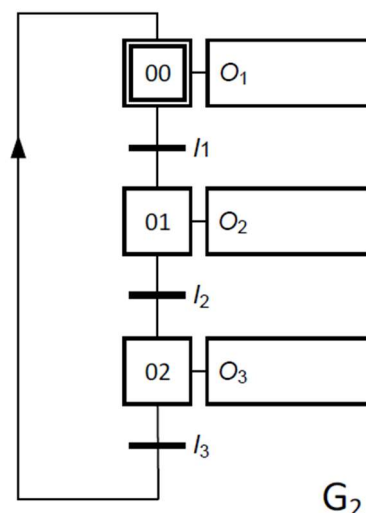


Figura 3.14 – Grafket a considerar para o primeiro problema.

A programação de graficets em texto estruturado pode ser feita de duas maneiras. A primeira consiste em definir cada etapa e transição como variáveis binárias.

Defina-se as variáveis `Etapa_n` como sendo as variáveis booleanas que codificam o estado de cada etapa do graficet, isto é, ativa ou inativa quando `TRUE` ou `FALSE`, respetivamente. Defina-se ainda `T_n` como sendo as variáveis booleanas associadas às transições. Quando esta variável for `TRUE` então a transição foi verificada.

É necessário, no entanto, definir as transições como sendo uma combinação da etapa que antecede a transição propriamente dita e a condição de transição. Isto é, para que a transição seja verificada, o sistema tem de estar na etapa imediatamente antes da transição em questão e a condição tem de ser `TRUE`. Para melhor ilustrar este ponto tome-se o exemplo da primeira transição, a que liga a etapa 00 à 01. Para o sistema mudar da etapa 00 para a 01 tem de ter a primeira etapa ativa, ou seja, a `TRUE` e a entrada `In_1` também tem de ser `TRUE` o que se traduz em: `T_1 := Etapa_00 AND In_1`.

Deste modo, quando se verifica a transição, coloca-se a variável da etapa em questão a `FALSE` – desativando essa etapa- e coloca-se a seguinte a `TRUE` – ativando a etapa seguinte.

A linguagem de programação *ladder*, como ilustrado na Figura 3.15, Figura 3.16 e Figura 3.17, só permite esta abordagem em que se utilizam apenas variáveis binárias. No entanto, a programação em ST permite diferentes abordagens graças a instruções `IF`, `ELSIF` e `CASE OF`, como já se pode observar em casos anteriores. Uma vantagem que esta linguagem de programação trás é que, através do uso destas instruções, é possível reduzir significativamente o número de variáveis, como se poderá constatar mais à frente.

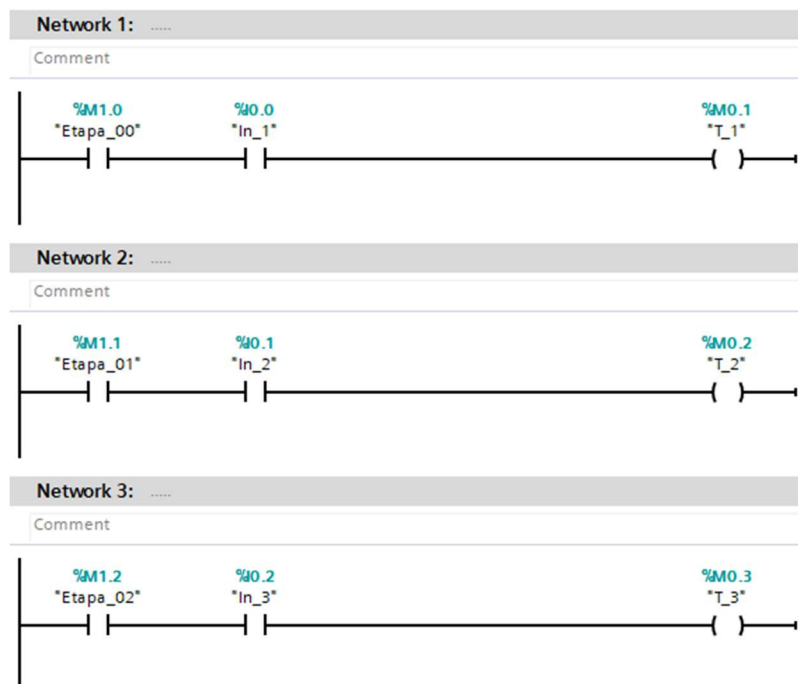


Figura 3.15 – Programa em *ladder* das transições do graficet G2 do primeiro problema.

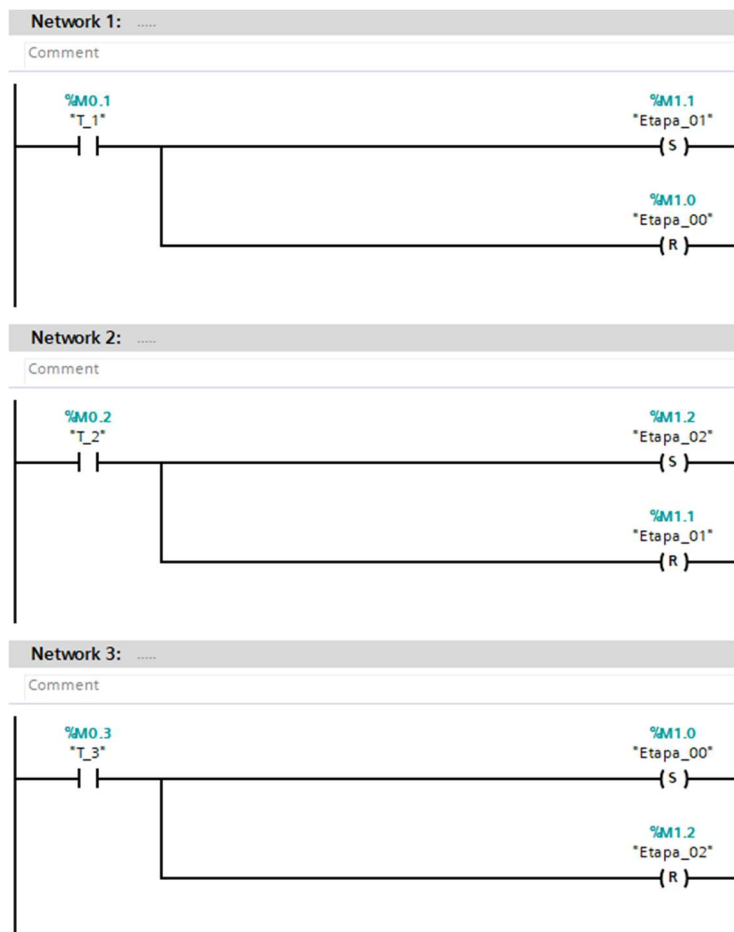


Figura 3.16 – Programa em *ladder* das etapas do grafcet G2 do primeiro problema.

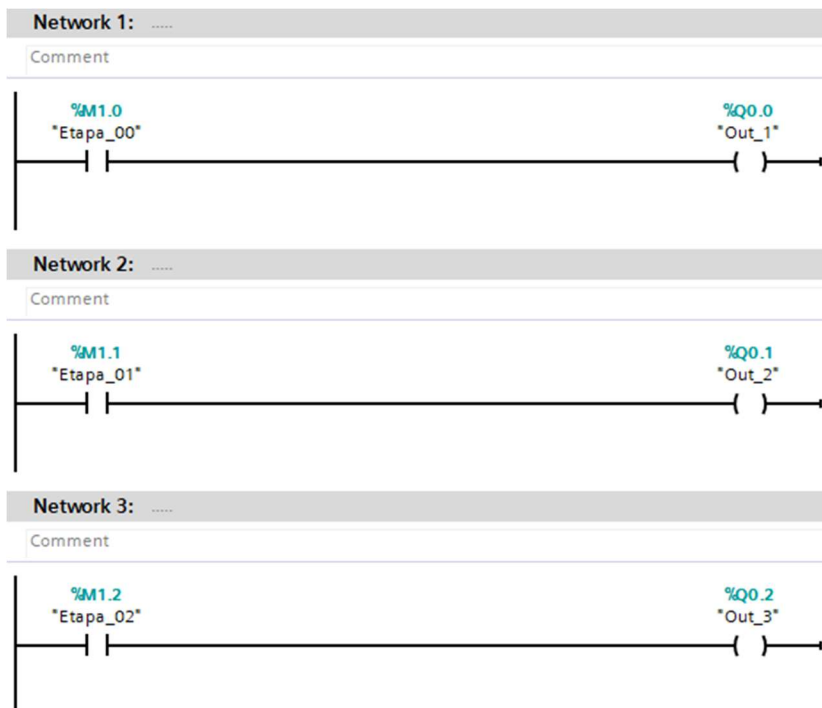


Figura 3.17 – Programa em *ladder* da escrita das saídas do grafcet G2 do primeiro problema.

No entanto, a mesma abordagem usando texto estruturado é algo extensa também, sendo necessária a utilização de várias instruções `IF`, como se pode observar no código apresentado de seguida.

```
VAR

    Etapa_00 : BOOL := TRUE;

    Etapa_01 : BOOL;

    Etapa_02 : BOOL;

    T_1 : BOOL;

    T_2 : BOOL;

    T_3 : BOOL;

END_VAR
```

```
T_1 := Etapa_00 AND In_1;
T_2 := Etapa_01 AND In_2;
T_3 := Etapa_02 AND In_3;

IF T_1 THEN

    Etapa_00 := FALSE;

    Etapa_01 := TRUE;

END_IF
```

```
IF T_2 THEN

    Etapa_01 := FALSE;

    Etapa_02 := TRUE;

END_IF
```

```
IF T_3 THEN

    Etapa_02 := FALSE;

    Etapa_00 := TRUE;

END_IF
```

```
Out_1 := Etapa_00;

Out_2 := Etapa_01;

Out_3 := Etapa_02;
```

Note-se a necessidade de inicializar a variável `Etapa_00` como `TRUE`, caso contrário o sistema não funciona porque começam todas as variáveis a `FALSE` e nunca se verifica nenhuma das condições existentes. Em contraste, o *ladder* omite esta necessidade de inicializar variáveis.

Além disso, caso haja etapas seguidas cujas transições são idênticas, esta abordagem pode levar a que ocorram etapas fugazes. Como já foi referido aquando da apresentação da mesma situação, mas no contexto dos diagramas de estado, este problema pode ser contornado utilizando um único ciclo `IF` com recurso à instrução `ELSIF`, como se pode ver no programa que se segue.

```
VAR

    Etapa_00 : BOOL := TRUE;

    Etapa_01 : BOOL;

    Etapa_02 : BOOL;

END_VAR

IF Etapa_00 AND In_1 THEN

    Etapa_00 := FALSE;

    Etapa_01 := TRUE;

ELSIF Etapa_01 AND In_2 THEN

    Etapa_01 := FALSE;

    Etapa_02 := TRUE;

ELSIF Etapa_02 AND In_3 THEN

    Etapa_02 := FALSE;

    Etapa_00 := TRUE;

END_IF

Out_1 := Etapa_00;

Out_2 := Etapa_01;

Out_3 := Etapa_02;
```

A outra abordagem possível é a aplicação da instrução `CASE OF`. Para isso, e como já foi referido, é preciso definir uma variável que retenha a informação sobre em que etapa é que o grafcet se encontra, por exemplo, `Etapa`, e esta deve ser do tipo `UINT`, pelas razões já apresentadas aquando da introdução da instrução em questão. O código resultante desta hipótese é apresentado de seguida.

```
VAR

    Etapa : UINT;

END_VAR
```

```
CASE Etapa OF
    00:
        IF In_1 THEN
            Etapa := 01;
        END_IF
    01:
        IF In_2 THEN
            Etapa := 02;
        END_IF
    02:
        IF In_3 THEN
            Etapa := 00;
        END_IF
END_CASE

Out_1 := Etapa = 00;
Out_2 := Etapa = 01;
Out_3 := Etapa = 02;
```

A utilização desta instrução permite evitar etapas fugazes, facilita a leitura e organização do código, e ainda a programação de graficets hierárquicos e divergências, como será demonstrado nos problemas que se seguem.

3.2.2.2 Segundo Problema

Neste problema é pedido que se programe o graficet da Figura 3.18.

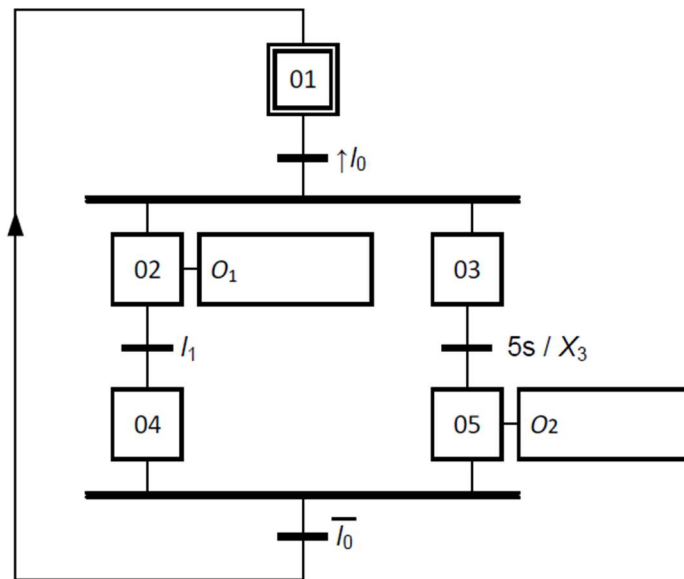


Figura 3.18 – Grafcet a considerar no segundo problema.

Como se pode constatar, este problema apresenta um grafcet com uma divergência E. Para traduzir e programá-lo, tem-se de “partir” o grafcet em dois. Isto porque o grafcet em questão prevê a ativação de duas etapas em simultâneo. No entanto, não é possível programar essa situação utilizando a instrução *CASE*, daí ser necessário fazer a divisão do grafcet. O objetivo é, então, ter dois grafkets semelhantes ao do primeiro problema, ficando, assim, mais fácil de programar. Assim sendo, esta partição deve ser feita de modo que as etapas comuns aos dois ramos, ou seja, as etapas fora da divergência, e um dos ramos constituam um grafcet. Deste modo o ramo que sobra será um grafcet por si só. Neste caso, resultaria, então, um grafcet - G1 - composto pelas etapas 01, 02 e 04, e outro - G2 - composto pelas etapas 03 e 05. O resultado desta divisão está ilustrado na Figura 3.19.

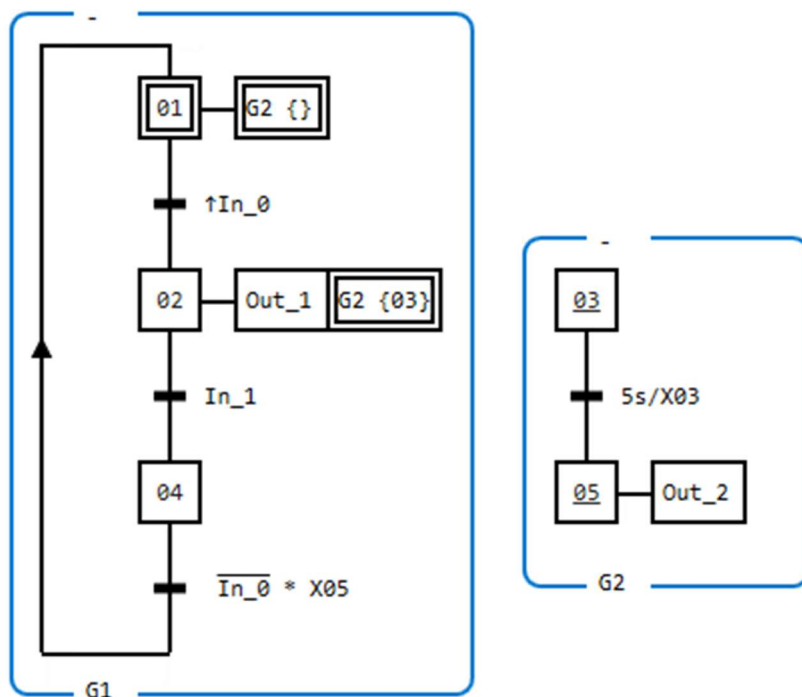


Figura 3.19 – Resultado da divisão do grafcet original.

Na programação deste problema ter-se-á então duas instâncias da instrução `CASE OF`, uma para cada grafcet. Ter-se-á, então, de definir variáveis, do tipo `UINT`, que permitam a identificação do estado do sistema. Sejam estas `State_G1`, para o grafcet G1, e `State_G2` para G2. A restante programação segue a lógica apresentada anteriormente, sendo que a diferença reside na programação da entrada e saída da divergência. Esta questão é, no entanto, de fácil resolução. A chave é a ativação e inativação do grafcet G2. Sendo que a ativação pode ser feita na etapa 01. Quando se verifica a transição, ou seja, quando se deteta uma transição ascendente de I_0 muda-se o estado de G1, passando da etapa 01 para a 02, e ativa-se a etapa 03 de G2. Por sua vez, a inativação é feita na etapa 04 de G1, seguindo a lógica aplicada na entrada da divergência, utilizando a transição associada, mas, agora, deve-se usar também, como condição, a etapa 05 de G2 estar ativa. Verificando-se as duas condições, G1 volta para o estado inicial e G2 é colocado num estado 0 que serve apenas o propósito de inativar G2. Esta situação representa uma hierarquia entre os dois grafcets, sendo que G1 pode ser visto como superior de G2, dado que tem a capacidade de ativar ou desativar este último. A seguir apresenta-se a resolução deste problema.

```
VAR

    My_TON : TON;

    R_In_0 : R_Trig;

    State_G1 : UINT := 01;

    State_G2 : UINT := 0;

END_VAR

R_In_0 (CLK := In_0);

CASE State_G1 OF

    01:

        IF R_In_0.Q THEN

            State_G1 := 02;

            State_G2 := 03;

        END_IF

    02:

        IF In_1 THEN

            State_G1 := 04;

        END_IF

    04:

        IF NOT In_0 AND State_G2 = 05 THEN

            State_G1 := 01;

            State_G2 := 0;

        END_IF

END_CASE

CASE State_G2 OF

    03:

        IF My_TON.Q THEN

            State_G2 := 05;

        END_IF

END_CASE

My_TON (IN := State_G2 = 03, PT := T#5S);
```

```
Out_1 := State_G1 = 02;
```

```
Out_2 := State_G2 = 05;
```

Note-se que, tal como se verificava no problema anterior, também neste se inicializou a variável *State_G1*, que codifica a etapa inicial do grafcet. Isto porque, por defeito, esta é inicializada como sendo zero. No entanto, foi definido que a etapa inicial é a 01, sendo esse o primeiro valor na instrução *CASE OF*, por isso é necessário inicializar *State_G1* como valendo 01, caso contrário o sistema não funciona.

3.2.2.3 Terceiro Problema

Para este problema o aluno deve desenvolver um programa para o grafcet da Figura 3.20.

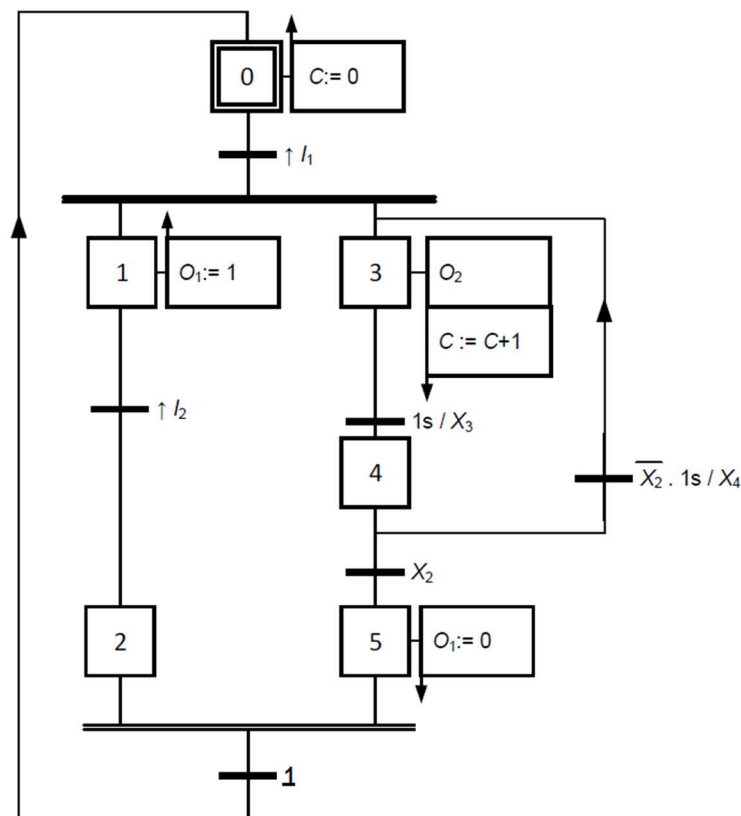


Figura 3.20 – Grafcto do terceiro problema.

Para este caso o raciocínio é em tudo semelhante ao apresentado para o problema anterior. Neste caso o grafcto G1 será composto pelas etapas 0, 1 e 2, e o grafcto G2 pelas etapas 3, 4 e 5.

Toda a programação é feita de acordo com o explicado. Declaram-se duas variáveis do tipo *UINT*, para serem utilizadas na instrução *CASE OF* e programa-se esta de acordo com o que o grafcto define como sendo o funcionamento deste sistema. A resolução deste problema é a seguinte:

```
VAR

    State_G1 : UINT;

    State_G2 : UINT;

    Counter : CTU;

    Timer_1 : TON;

    Timer_2 : TON;

    RTrig_Out_1 : R_Trig;

    RTrig_In_1 : R_Trig;

    RTrig_In_2 : R_Trig;

    RTrig_Counter : R_Trig;

    FTrig_Out_1 : F_Trig;

    FTrig_Counter : F_Trig;

    Latch : RS;

END_VAR


RTrig_Out_1 (CLK := State_G1 = 1);

RTrig_In_1 (CLK := In_1);

RTrig_In_2 (CLK := In_2);

RTrig_Counter (CLK := State_G1 = 0);

FTrig_Out_1 (CLK := State_G2 = 5);

FTrig_Counter (CLK := State_G2 = 3);


CASE State_G1 OF

    0:

        IF RTrig_In_1.Q THEN

            State_G1 := 1;

            State_G2 := 3;

        END_IF

    1:

        IF RTrig_In_2.Q THEN

            State_G1 := 2;

        END_IF

END_CASE
```

2:

```

    IF State_G2 = 5 THEN
        State_G1 := 0;
        State_G2 := 0;
    END_IF

```

END_CASE

CASE State_G2 OF

3:

```

    IF Timer_1.Q THEN
        State_G2 := 4;
    END_IF

```

4:

```

    IF NOT State_G1 = 2 AND Timer_2.Q THEN
        State_G2 := 3;
    ELSIF State_G1 = 2 THEN
        State_G2 := 5;
    END_IF

```

END_CASE

Counter (CU := FTrig_Counter.Q, Reset := State_G1 = 0);

Latch (set := RTrig_Out_1.Q, Reset1 := FTrig_Out_1.Q, Q1 => Out_1);

Timer_1 (IN := State_G2 = 3, PT := T#1S);

Timer_2 (IN := State_G2 = 4, PT := T#1S);

Out_2 := State_G2 = 3;

Chama-se a atenção para o facto de, neste problema, se ter omitido a inicialização da variável `State_G1`. Isto é possível, porque, como foi referido, esta é inicializada, por defeito, como sendo zero e, neste caso, é esse o primeiro valor da instrução `CASE OF`. Por isso o sistema funciona sem problemas.

Como se pode observar a estratégia é a mesma nos dois casos e facilmente aplicável a um qualquer número de divergências, sendo apenas necessário abordar cada ramo como um grafcet isolado que é ativado ou desativado quando necessário.

À semelhança dos restantes problemas, os programas apresentados para os problemas de tradução de grafkets foram testados no *soft*PLC *ST I/O* e foram executados sem problemas. No entanto, foi necessário fazer uma breve alteração no código do terceiro problema. Na instrução `CASE` de `G2`, para a situação em que `State_G2 = 4`, mudou-se a comparação da

variável `State_G1` de uma igualdade para uma diferença. Esta alteração é apresentada de seguida, estando a parte alterada destacada a vermelho.

```
CASE State_G2 OF
3:
    IF Timer_1.Q THEN
        State_G2 := 4;
    END_IF
4:
    IF State_G1 <> 2 AND Timer_2.Q THEN
        State_G2 := 3;
    ELSIF State_G1 = 2 THEN
        State_G2 := 5;
    END_IF
END_CASE
```

3.3 Síntese

Ao longo deste capítulo procurou-se abordar diferentes aspetos básicos da programação em texto estruturado.

Começou-se por apresentar e explicar os blocos funcionais mais simples que constituem as bases da programação em ST e de acordo com a norma. Abordaram-se todos os FB standard, tendo estes sido explicados com algum detalhe. Procurou-se tornar o mais claro possível como estes blocos funcionam e como as suas variáveis afetam esse mesmo funcionamento. Tentou-se ainda mostrar as diferentes instâncias de cada bloco, bem como um exemplo de aplicação dos mesmos.

De seguida, tratou-se de explicar a tradução e programação de máquinas de estado. Começou-se esta subsecção abordando diferentes métodos de programar diagramas de estados e grafets. Procurou-se expor casos particulares que demonstrem peculiaridades de programação, como é o caso das transições todas idênticas no diagrama de estados e da divergência no grafet.

Com os exercícios expostos é possível conhecer, com algum detalhe, não só os blocos funcionais básicos que constam na norma e ter uma compreensão geral de como estes se programam em texto estruturado, mas ainda ficar a conhecer diferentes abordagens para a tradução e programação de máquinas de estados.

Utilizando os blocos funcionais abordados, ou combinações dos mesmos, e aplicando as metodologias de programação apresentadas é possível resolver inúmeros problemas de programação de PLC.

Estando as questões fundamentais da programação apresentadas, é de todo o interesse avançar, agora, para questões de modelação de sistemas concretos. Esta capacidade será útil para estabelecer os requisitos de funcionamento de um sistema e expressá-los sob a forma de um diagrama de estados, ou grafcet, a partir do qual se poderá desenvolver um programa, aplicando o conhecimento exposto no presente capítulo. As questões relacionadas com a modelação de sistemas serão, então, tratadas no capítulo 4, onde serão apresentados mais alguns problemas a resolver.

4 Controlo de cenários de demonstração *Factory I/O* relativamente simples

O presente capítulo tem como objetivo o desenvolvimento estruturado de programas de controlo de cenários *Factory I/O* para fins de demonstração do produto. Tendo em conta que há aqui objetivos didáticos e interesse em testar o PLC que integrará o *Factory I/O*, procura-se, então, continuar e aprofundar a modelação de problemas lógicos em diagramas de estados ou graficets, a aplicação desses conceitos a casos de estudo selecionados e, por fim, a programação e teste do código desenvolvido no *soft*PLC *ST I/O*.

Para situar devidamente estes objetivos, importa começar por referir que a aplicação *Factory I/O* incorpora uma biblioteca de cenários que exemplificam as potencialidades do programa. No site da empresa Real Games é possível consultar os vídeos destes cenários em funcionamento, mas a solução de programação não é disponibilizada. Há, no entanto, um grande interesse, por parte de professores e estudantes, nessas soluções. Assim, neste capítulo, serão recuperados alguns desses cenários e ainda outros desenvolvidos exclusivamente no âmbito desta dissertação. Trata-se sempre de aplicações para “pôr na montra”, ou seja, de sistemas pequenos, relativamente inflexíveis, com pouca diversidade de elementos e que funcionam “em loop”. Por isso mesmo são sistemas que apresentam um baixo nível de interação tanto com o utilizador como entre os diferentes elementos que os constituem. Contudo, são do ponto de vista desta dissertação o ponto de partida para os cenários bastante mais ambiciosos a tratar no próximo capítulo, conforme se verá a seu tempo.

Para que se possa desenvolver um programa para controlo de um sistema deve-se definir os passos a dar, a aplicação de um método sistémico para o desenvolvimento de um programa pode permitir que esse desenvolvimento seja mais rápido. A abordagem aqui empregue baseia-se na que é apresentada no capítulo 13 do livro “Programmable Logic Controllers”[40]. Esta refere que se deve começar por definir os requisitos do sistema e os seus sinais de I/O, de seguida, deve ser feita uma modelação do sistema, depois, a partir dessa modelação, desenvolve-se um programa para o controlo do sistema em análise e, finalmente, testa-se o programa .

Assim, a abordagem que se apresenta para estes problemas inicia-se pela apresentação do sistema, fazendo uso de um breve enunciado onde são indicados os elementos que o constituem e qual o funcionamento desejado, seguida da modelação do problema, em diagrama de estados ou GRAFCET, como mencionado no capítulo 3, e, finalmente, programação da solução obtida fazendo uso dos elementos que foram apresentados no capítulo anterior.

Para a modelação feita em GRAFCET, de modo a poder validar os modelos obtidos neste capítulo, recorreu-se ao *software GRAFCET-Studio* [41]. Esta aplicação permite desenhar graficets de acordo com a norma IEC 60848 e ainda estabelecer ligação com o *Factory I/O*, através do driver MHJ, já incorporado neste último. Assim é possível simular o comportamento modelado, à semelhança do que acontece quando se utiliza o *Codesys*, a diferença é que quando se usa o *GRAFCET-Studio* o programa é o próprio graficet.

Chama-se à atenção para duas variáveis que têm origem na aplicação *Factory I/O* e que são utilizadas nos problemas apresentados. Estas são *Fact_Run* e *Fact_Rstd*. A primeira, como já foi mencionado no capítulo anterior, permite saber se a aplicação está em modo “Run” ou não, enquanto a segunda indica, quando *TRUE*, que o cenário de *Factory I/O* foi reiniciado. Salienta-se ainda a existência de dois elementos que o *Factory I/O* permite incorporar nos seus cenários e cujas funções são, na vida real, desempenhadas por outras entidades. Estes elementos são o emissor e o eliminador. O primeiro tem como função introduzir peça ou caixas no cenário, simulando um operador a colocar itens nos tapetes, por exemplo. O segundo, por sua vez, remove as referidas peças ou caixas, simulando um operador a retirar os itens do sistema. Estes elementos têm apenas dois estados, ligados ou desligados e serão mais explorados no decorrer do capítulo.

4.1 Primeiro cenário

O primeiro sistema a modelar e programar está apresentado na Figura 4.1 e, na Tabela 4.1, podem-se consultar os requisitos de funcionamento do sistema. Este cenário é o “*From A to B (Set and Reset)*” que pode ser encontrado na biblioteca de cenários do *Factory I/O*. A única diferença do que aqui é apresentado para o que consta nessa biblioteca é a presença de faróis.



Figura 4.1 – Primeiro cenário.

Tabela 4.1 – Requisitos de funcionamento do primeiro cenário.

Objetivo:	Transportar uma caixa plástica através dos dois tapetes que compõem o sistema até ao final do tapete 2.
Condições iniciais:	As observáveis na Figura 4.1
Funcionamento desejado:	O tapete de entrada liga quando se atua o botão de start e transporta a caixa até ao tapete seguinte, o tapete 2. Este liga quando deteta a caixa (sensor 1) e transporta a caixa até ao final, ou seja, até ao sensor 2. O tapete 1 desliga quando a caixa sai completamente do mesmo. O tapete 2 desliga quando a caixa é detetada pelo sensor 2. Os faróis sinalizam que os respetivos tapetes estão em movimento. O botão de comando do <i>Factory I/O</i> reinicia o cenário, repondo a caixa na posição inicial.
Atuações Manuais:	Dar início ao movimento do tapete de entrada por atuação do botão de start; reiniciar o cenário atuando o comando de reset do <i>Factory I/O</i> .
Condições a salvaguardar:	A atuação do botão start só tem efeito na situação inicial; a luz do botão de start só deve acender quando a atuação do mesmo tiver efeito.
Sinais I/O:	Entradas: Botão de start e comando de reset do <i>Factory I/O</i> (todos momentâneos, produzem <code>TRUE</code> quando premidos), sensores 1 e 2 (geram <code>FALSE</code> na situação de deteção). Saídas: Tapetes 1 e 2 (unidirecionais), faróis sinalizadores e luz do botão de start (todos energizados com <code>TRUE</code>).

Conhecendo a constituição do sistema e tendo informação sobre o funcionamento desejado pode-se, após uma análise dos mesmos, modelar o funcionamento pretendido para o sistema. Esta modelação, como já foi mencionado no capítulo anterior, pode ser feita em diagramas de estados ou em grafcet. Dado tratar-se de um sistema simples, constituído apenas por dois tapetes, o mais indicado será utilizar um diagrama de estados, visto que, como foi referido, são bons para modelar sistemas com poucos estados, como é o caso. Na Figura 4.2 e na Tabela 4.2 pode-se consultar o diagrama de estados resultante da análise do sistema, a sua legenda e aspetos combinacionais relacionados com o sistema em questão.

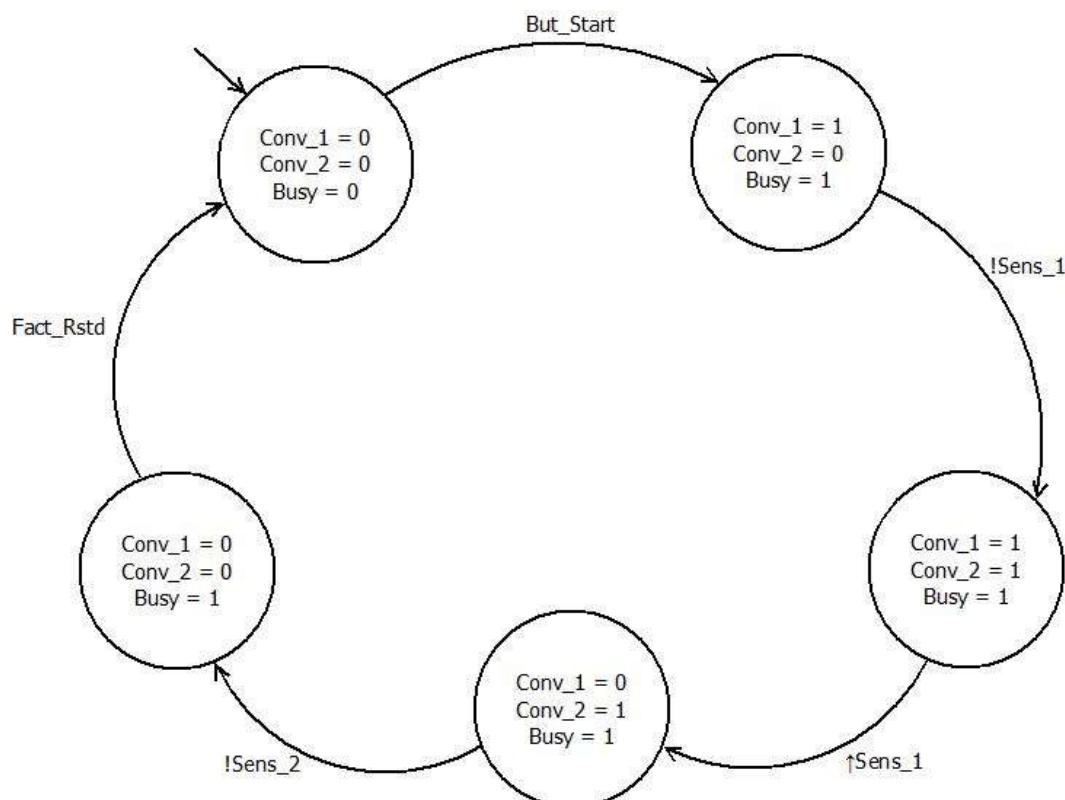


Figura 4.2 – Diagrama de estados do primeiro cenário.

Tabela 4.2 – Resumo da modelação do primeiro problema.

Aspetos combinacionais:	WLght_1 = Cnv_1; WLght_2 = Cnv_2; WLght_3 = Cnv_3; BLght_Start = !Busy * !But_Start
Legenda:	BLght_Start: Luz do botão de start; Busy: Busy (variável auxiliar); But_Start: Start; Conv_1: Tapete 1; Conv_2: Tapete 2; Fact_Rstd: Factory I/O (Reset); Sens_1: Sensor 1; Sens_2: Sensor 2; WLght_1: Luz do tapete 1; WLght_2: Luz do tapete 2.
Ações de resposta ao reset do factory:	Reinício do cenário.

Como se pode constatar, surge, na Figura 4.2, uma variável, denominada *Busy*, que não faz parte dos sinais de IO referidos na Tabela 4.1. Trata-se de uma variável auxiliar que tem como função registar se o botão de start foi atuado, inibindo, assim, o efeito do mesmo. É por isso que esta variável surge como sendo *FALSE* apenas no estado inicial, visto ser este o único em que a atuação do botão de start deve ter efeito.

É possível abordar a modelação deste sistema de uma outra maneira. Enquanto o diagrama de estados apresentado representa o funcionamento de todo o sistema num só grafo, a outra abordagem baseia-se em desenhar diagramas individuais para cada elemento e

sincronizá-los devidamente. Assim, em vez de se ter de pensar no sistema como uma única entidade pode-se abordar o mesmo como sendo várias partes que interagem para que se consiga o funcionamento desejado. Isto permite obter uma modelação mais fácil de expandir, isto é, é mais fácil acrescentar elementos ao cenário do sistema, como se verá brevemente. O resultado desta abordagem está ilustrado na Figura 4.3. Só se apresenta o diagrama de estados pois as variáveis e aspetos combinacionais não mudam, só a apresentação do diagrama é que é diferente, este continua a representar o sistema em análise.

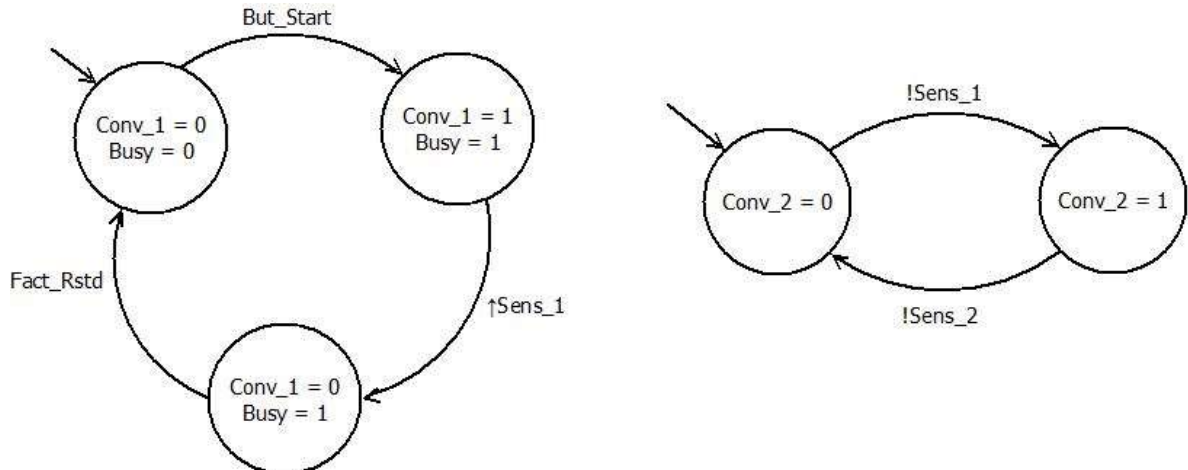


Figura 4.3 – Alternativa de modelação do primeiro problema.

Como se pode observar, esta alternativa é mais simples e permite uma leitura mais fácil do diagrama. Este último aspeto é de elevado interesse pois traduz-se numa programação mais imediata. Além disso, esta modelação facilita ainda a introdução de elementos no sistema, como será apresentado brevemente.

Tendo em conta que esta modelação é mais interessante do que a primeira que foi apresentada foi a partir destes diagramas que se desenvolveu o programa para controlar o sistema deste problema. Como se pode observar, analisando a Figura 4.3, os elementos deste cenário têm apenas dois estados cada um pelo que podem ser programados fazendo uso das *latches* apresentadas no capítulo anterior. Assim, resulta o seguinte código.

```

VAR_INPUT
    But_Start, Sens_1, Sens_2, Fact_Rstd : BOOL;

END_VAR

VAR
    RSens_1 : R_Trigger;

    RS_Conv_1, RS_Conv_2, RS_Busy : RS;

    Busy : BOOL;

END_VAR

VAR_OUTPUT
    Conv_1, Conv_2, WLght_Conv_1, WLght_Conv_2, BLght_Start : BOOL;

END_VAR
  
```

```

RSens_1 (CLK := Sens_1);

IF Fact_Rstd THEN

    Conv_1 := FALSE;

    Conv_2 := FALSE;

    WLght_Conv_1 := FALSE;

    WLght_Conv_2 := FALSE;

    Busy := FALSE;

ELSE

    BLght_Start := NOT Busy AND NOT But_Start;

    RS_Conv_1 (Set := But_Start AND NOT Busy, Reset1 := BLght_Start
OR RSens_1.Q, Q1 => Conv_1, Q1 => WLght_Conv_1);

    RS_Conv_2 (Set := NOT Sens_1 AND Busy, Reset1 := BLght_Start OR
NOT Sens_2, Q1 => Conv_2, Q1 => WLght_Conv_2);

    RS_Busy (Set := But_Start, Reset1 := BLght_Start, Q1 => Busy);

END_IF

```

Numa primeira análise do código apresentado verifica-se que o primeiro passo é a declaração das variáveis e blocos utilizados na programação do sistema, tal como foi abordado no capítulo 3. De seguida é feita a inicialização dos blocos que requerem que tal seja feito. Posto isto, pode-se então escrever o código propriamente dito.

Assim, pode-se observar a utilização de uma instrução `IF` cuja primeira função é verificar se foi feito reset ao cenário. Caso seja feito o reset ao *Factory I/O* então todos os elementos, incluindo a variável auxiliar, são colocados a `FALSE`, de modo a garantir que o programa é colocado na sua situação inicial. Na segunda parte deste `IF`, ou seja, na instrução `ELSE`, executa-se o programa que corresponde ao diagrama de estados. Ou seja, quando não se verifica o reset do cenário, executa-se o programa.

Em termos de blocos utilizados, pode-se observar que foram empregues *latches* e `R_Trig`.

De modo a poder explicar a questão da facilidade de expansão do sistema, mencionada anteriormente, considere-se, agora, o cenário da Figura 4.4 e respetivo enunciado, sendo que este se encontra resumido na Tabela 4.3.

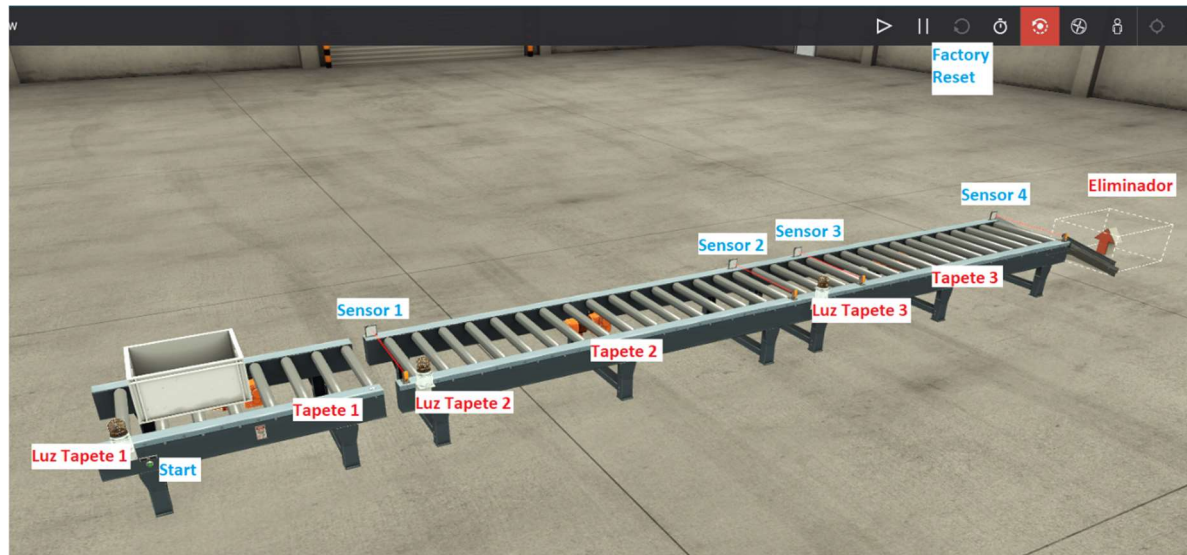


Figura 4.4 – Alternativa do primeiro cenário.

Tabela 4.3 – Requisitos de funcionamento da alternativa do primeiro cenário.

Objetivo:	Transportar uma caixa plástica através dos vários tapetes que compõem o sistema até à rampa de saída.
Condições iniciais:	As observáveis na Figura 4.4.
Funcionamento desejado:	O tapete de entrada liga quando se atua o botão de start e transporta a caixa até ao tapete 2. Este liga quando a deteta (sensor 1) e transporta-a até ao tapete 3, que, semelhantemente ao anterior, liga quando deteta a caixa (sensor 2) e encaminha-a até à rampa por onde esta sai. O tapete 1 desliga quando a caixa é completamente carregada no tapete 2, este por sua vez desliga quando a caixa passa para o tapete 3, ou seja, o sensor 3 é <i>FALSE</i> . O último tapete para quando a caixa entra na rampa. Os faróis indicam se os respetivos tapetes estão em movimento. O comando de reset do <i>Factory I/O</i> reinicia o cenário, colocando a caixa na posição inicial.
Atuações Manuais:	Dar início ao movimento do tapete de entrada por atuação do botão de start; reiniciar o cenário atuando o reset do <i>Factory I/O</i> .
Condições a salvaguardar:	A atuação do botão start só tem efeito na situação inicial; a luz do botão só acende quando a atuação do botão tiver efeito; o eliminador só liga quando necessário.
Sinais I/O:	Entradas: Botão de start e comando de reset do <i>Factory I/O</i> (ambos momentâneos, produzindo <i>TRUE</i> quando premidos), sensores 1 a 4 (geram <i>FALSE</i> na situação de deteção). Saídas: Tapetes 1 a 3 (unidirecionais), eliminador, faróis sinalizadores e luz do botão de start (todos energizados com <i>TRUE</i>).

Como se pode observar, as diferenças entre este sistema e o anterior resumem-se à adição de um tapete e do eliminador.

Tal como no caso anterior, através da análise da descrição do funcionamento do sistema e conhecendo os elementos que o constituem é possível modelar o sistema. Este é, como se

pode observar, muito semelhante ao anterior, pelo que também este deverá ser modelado utilizando diagramas de estados.

Em termos de diferenças para o primeiro caso estas prendem-se com o tapete e o eliminador. Este último tem um funcionamento muito semelhante ao dos tapetes, no sentido em que liga quando a caixa é detetada pelo sensor 4 e desliga ao fim de um determinado tempo. Ou seja, tem apenas dois estados, tal como os tapetes, o que facilita a sua inclusão.

Tendo em conta a modelação apresentada para o caso anterior, para este cenário é apenas necessário acrescentar dois diagramas, cada um com dois estados, um para o tapete 3, que será em tudo semelhante ao do tapete 2, e outro para o eliminador que, tal como foi referido, será também ele similar ao diagrama do tapete 2.

A modelação que resulta, tendo em conta estas considerações, está ilustrada na Figura 4.5. Na Tabela 4.4 encontra-se a legenda do mesmo e aspetos combinacionais do sistema.

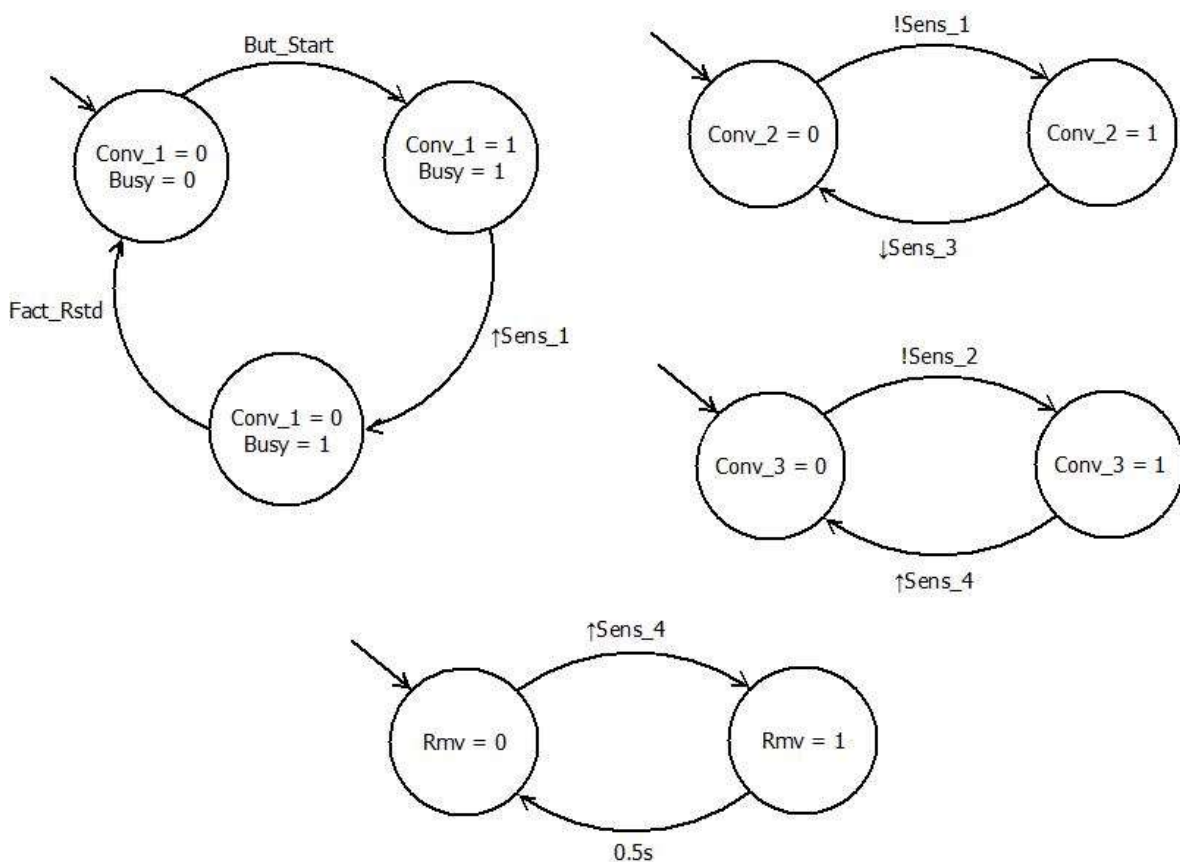


Figura 4.5 – Diagramas de estados da alternativa do primeiro cenário.

Tabela 4.4 – Resumo da modelação da alternativa do primeiro cenário.

Aspetos combinacionais:	WLght_1 = Cnv_1; WLght_2 = Cnv_2; WLght_3 = Cnv_3; BLght_Start = !Busy * !But_Start
Legenda:	BLght_Start: Luz do botão de start; Busy: Busy (variável auxiliar); But_Start: Start; Cnv_1: Tapete 1; Cnv_2: Tapete 2; Cnv_3: Tapete 3; Fact_Rstd: Factory I/O (Reset); Rmv: Eliminador; Sens_1: Sensor 1; Sens_2: Sensor 2; Sens_3: Sensor 3; Sens_4: Sensor 4; WLght_1: Luz do tapete 1; WLght_2: Luz do tapete 2; WLght_3: Luz do tapete 3.
Ações de resposta ao reset do factory:	Reinício do cenário.

Tal como foi referido, observa-se que, ao modelar o sistema com diagramas de estados separados, torna-se fácil a integração de novos elementos, sendo, como referido, necessário apenas desenhar o diagrama dos elementos adicionados. Como se pode verificar a inclusão do eliminador foi praticamente idêntica à do tapete, confirmando o que foi referido anteriormente.

Como foi mencionado, este sistema é muito semelhante ao anterior, visto que a única diferença é a adição do tapete 3 e do eliminador. Então, é de esperar que o código desenvolvido seja, também ele, semelhante ao já apresentado. A diferença no código será, tal como nos diagramas, a inclusão dos referidos elementos. Assim sendo, por uma questão de brevidade, será apresentado apenas o código que foi acrescentado e a declaração das variáveis que foram introduzidas.

À semelhança da modelação também na programação é fácil integrar os novos elementos, como se pode observar no programa que se segue.

```

VAR_INPUT

    Sens_3, Sens_4 : BOOL;

END_VAR

VAR

    RSens_4 : R_Trig;

    FSens_3 : F_Trig

    RS_Conv_3 : RS;

    TOF_Rmv : TOF;

```

```

END_VAR

VAR_OUTPUT
    Conv_3, Rmv, WLght_Conv_3 : BOOL;
END_VAR

FSens_3 (CLK := Sens_3);
RSens_4 (CLK := Sens_4);

IF Fact_Rstd THEN
    Rmv := FALSE;
    Conv_3 := FALSE;
    WLght_Conv_3 := FALSE;
ELSE
    TOF_Rmv (IN := RSens_4.Q, PT := T#0.5S, Q => Rmv);
    RS_Conv_3 (Set := FSens_2.Q AND Busy, Reset1 := BLght_Start OR
RSens_4.Q, Q1 => Conv_3, Q1 => WLght_Conv_3);
END_IF

```

Como se pode observar, tal como aconteceu na modelação do sistema, em que se desenhou um diagrama para cada elemento, também no desenvolvimento do código baseado nessa abordagem é mais fácil incluir estas novidades, sendo apenas necessário, neste caso, adicionar as *latches* dos novos elementos.

4.2 Segundo cenário

A Figura 4.6 ilustra o cenário a modelar. Este é precisamente o cenário “*Queue of Items (Counters)*” que é apresentado na biblioteca de cenários do *Factory I/O*. Na Figura 4.7 ilustram-se alguns dos estados intermédios do funcionamento do sistema, enquanto na Tabela 4.5 se podem consultar os requisitos do mesmo.



Figura 4.6 – Cenário do segundo problema.

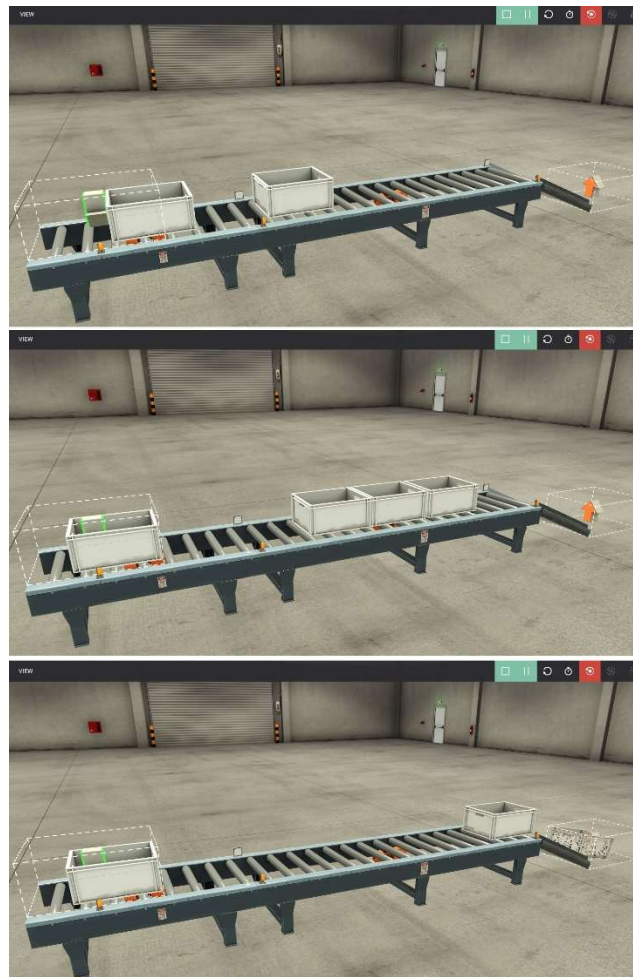


Figura 4.7 – Ilustração do funcionamento do segundo cenário.

Tabela 4.5 – Requisitos de funcionamento do segundo cenário.

Objetivo:	Agrupar 3 caixas no início do tapete 2 e depois transportá-las até à saída do mesmo.
Condições iniciais:	As observáveis na Figura 4.6.
Funcionamento desejado:	Quando se coloca o cenário em modo “Run” o emissor começa a emitir caixas com cerca de 3 segundos de intervalo entre si. Quando o sensor 1 deteta uma caixa ativa o tapete 1 e, quando a primeira caixa chega ao sensor 2, isto é, entra no tapete 2, arranca o mesmo. Quando a caixa passa o sensor 2 o tapete fica imobilizado à espera da próxima caixa. Isto repete-se até que se tenha um grupo de 3 caixas no tapete 2. Verificada esta condição, o tapete 1 desliga e o tapete 2 arranca de modo a transportar as caixas até à saída. O tapete 2 desliga quando a última caixa sai. A atuação do comando do <i>Factory I/O</i> reinicia o cenário.
Atuações Manuais:	Colocação do cenário do <i>Factory I/O</i> em modo “Run” por atuação do comando “Factory Run”. Reset do cenário por atuação do comando “Factory Reset”.
Condições a salvaguardar:	O emissor e o eliminador estão sempre ligados.
Sinais I/O:	Entradas: Comando de reset do <i>Factory I/O</i> (momentâneo, produz TRUE quando atuado) e sensores 1 a 3 (geram FALSE na situação de deteção). Saídas: Tapetes 1 e 2 (unidirecionais).

À semelhança do que se verificou para o problema anterior, também este trata de um sistema simples e com poucos estados, pelo que deverá ser, preferencialmente, modelado por um diagrama de estados. Sendo que a Figura 4.8 ilustra o diagrama de estados deste cenário. Na Tabela 4.6 pode-se consultar, tal como nos casos anteriores, o diagrama de estados obtido e a sua legenda.

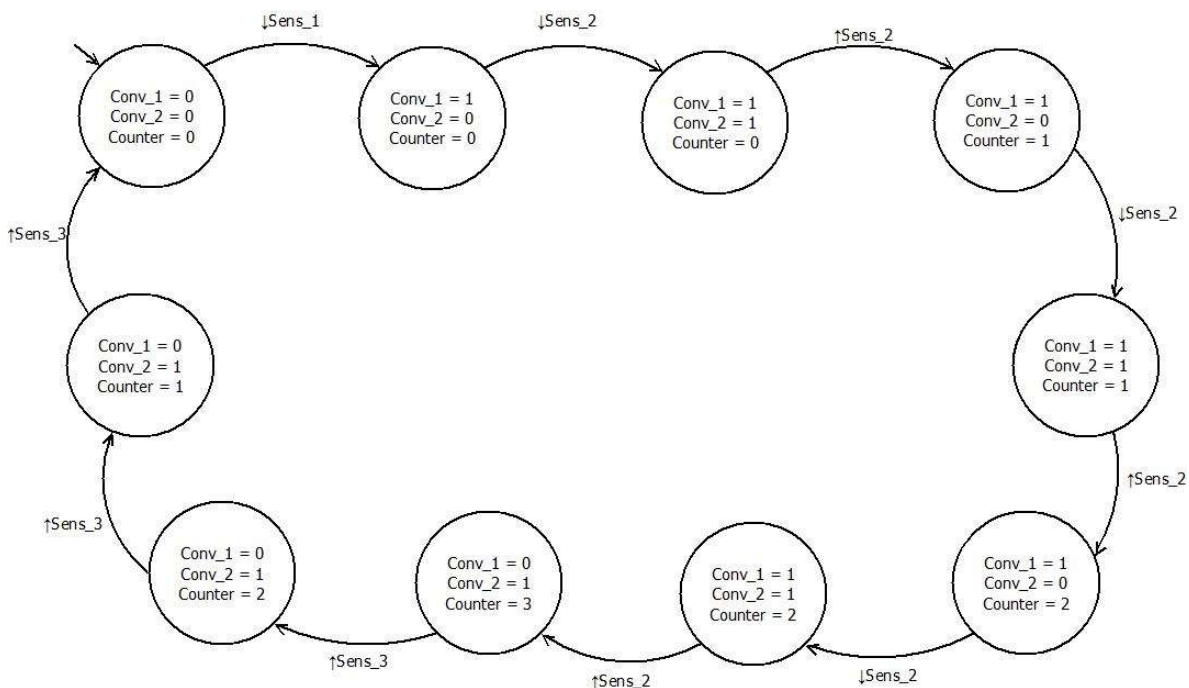


Figura 4.8 – Diagrama de estados do segundo cenário.

Tabela 4.6 – Resumo da modelação do sistema do segundo cenário.

Legenda:	Conv_1: Tapete 1; Conv_2: Tapete 2; Counter: Contador; Sens_1: Sensor 1; Sens_2: Sensor 2; Sens_3: Sensor 3.
Ações de resposta ao reset do factory:	Reinício do cenário.

Note-se que, embora seja de facto um sistema simples e com poucos estados, quando se elabora o diagrama de estados do mesmo, por causa do contador, torna-se num diagrama confuso e com muitos estados. No entanto, tal como foi mencionado nos casos anteriores, também neste problema se pode modelar o sistema utilizando diagramas de estados individuais, em vez de um único diagrama para o sistema todo, dando origem a um diagrama de estados mais simples de compreender. A aplicação desta abordagem resulta na modelação apresentada na Figura 4.9.

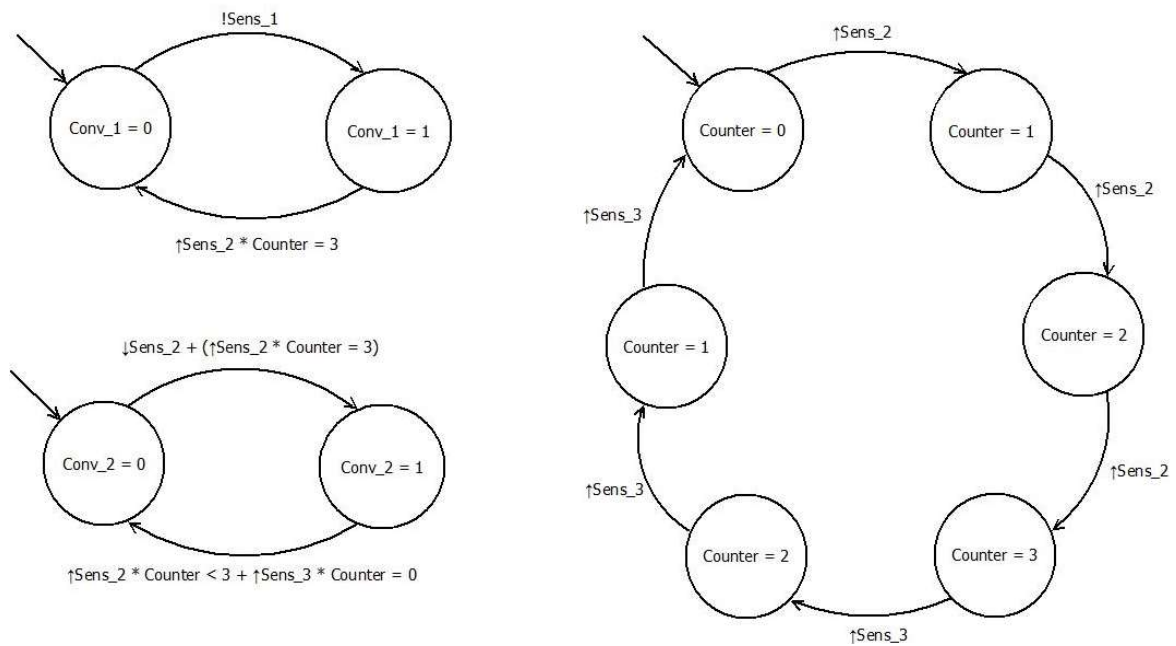


Figura 4.9 – Alternativa de modelação do segundo cenário.

Como se pode observar, tal como acontecia nos casos apresentados para o primeiro problema, também neste caso a separação dos diagramas permite uma leitura e programação mais fácil. Sendo que o programa que resulta desta modelação é apresentado a seguir. Note-se que, por uma questão de brevidade e de modo a evitar repetição de informação, será omitida a declaração de variáveis, visto que esta foi apresentada no capítulo anterior.

```

RSens_3 (CLK := Sens_3);

Counter (CU := Sens_2, CD := Sens_3, PV := 3);

IF Fact_Rstd THEN

    Conv_1 := FALSE;

```

```

    Conv_2 := FALSE;

    Counter.RESET := TRUE;

    RS_Conv_2_Unloading.RESET1 := TRUE;

ELSE

    Counter.RESET := FALSE;

    RS_Conv_1 (Set := Fact_Run AND NOT Sens_1 AND NOT Conv_2,
Reset1 := Counter.QU, Q1 => Conv_1);

    SR_Conv_2 (Set1 := NOT Sens_2 OR RS_Conv_2_Unloading.Q1, Reset
:= (Sens_2), Q1 => Conv_2);

    RS_Conv_2_Unloading (Set := Counter.QU, Reset1 := Counter.QD);

END_IF

```

Mais uma vez, embora esteja omitida neste caso, inicia-se a programação pela declaração de variáveis e blocos funcionais, seguida da inicialização dos FB que assim o requerem.

Tal como no problema anterior, também o programa deste começa com uma instrução `IF`. Esta tem a mesma função que no caso anterior que é a de verificar se o cenário de *Factory I/O* foi reiniciado e, caso tal se verifique, fazer reset do programa, de modo que este esteja na sua situação inicial, tal como o cenário. Caso não se verifique o reinício do cenário então o código que corresponde ao funcionamento do sistema é executado.

Os blocos funcionais utilizados são, como se pode observar, *latches* e contadores, sendo que não surge qualquer elemento de programação que não tenha sido já apresentado.

4.3 Terceiro cenário

O terceiro cenário a programar está ilustrado na Figura 4.10. Na

Tabela 4.7 estão resumidos os requisitos de funcionamento para este sistema. Este cenário resulta de uma adaptação do cenário “*Buffer Station*” da biblioteca do *Factory I/O*. A única alteração feita foi a substituição do segundo tapete pela balança.

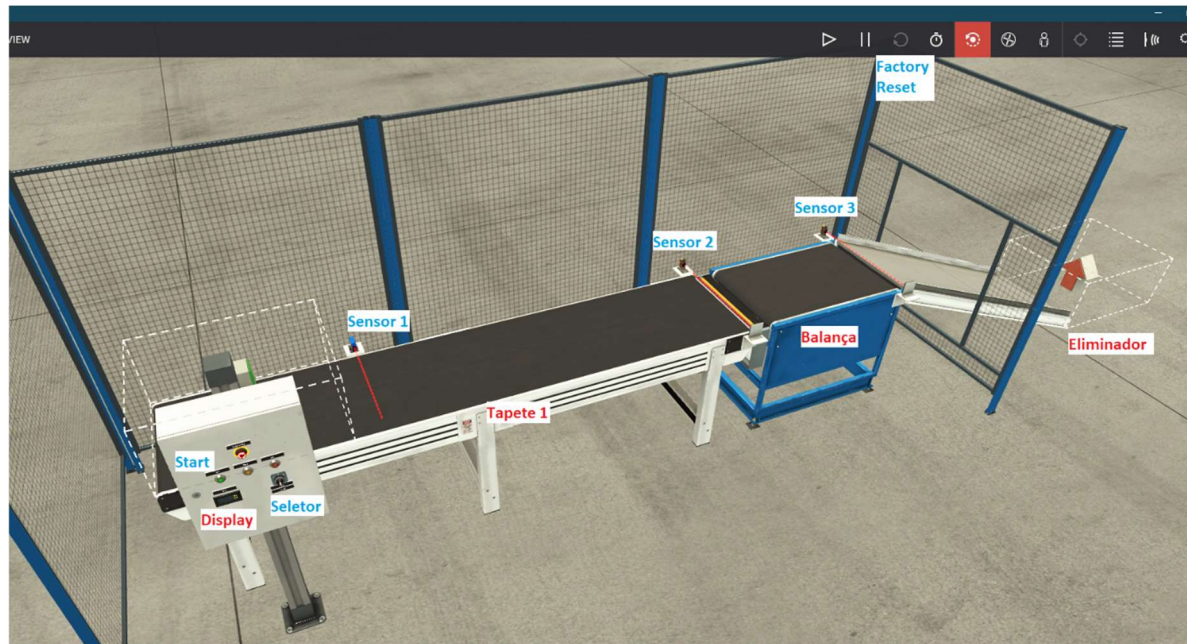


Figura 4.10 – Terceiro cenário.7

Tabela 4.7 – Requisitos de funcionamento do terceiro cenário.

Objetivo:	Acumular peças no tapete 1 e pesar uma de cada vez na balança.
Condições iniciais:	As observáveis na Figura 4.10.
Funcionamento desejado:	Quando se atua o botão de start, liga-se o emissor e o tapete 1, este deve parar quando uma peça chegar ao sensor 2. Se a balança estiver livre, então a barreira baixa para deixar passar uma caixa. O tapete da balança arranca para receber a caixa e desliga-se quando a caixa está completamente carregada. Quando a caixa está carregada a barreira sobe para impedir que mais caixas entrem. Quando a peça está pesada sai da balança, permitindo a pesagem de outra peça. A atuação do reset do <i>Factory I/O</i> reinicia o cenário.
Atuações Manuais:	Atuação do botão de start para ligar o emissor e o tapete de entrada.
Condições a salvaguardar:	A atuação do botão start só tem efeito na situação inicial; o emissor e o eliminador só devem ligar quando necessário; a caixa deve ficar imobilizada algum tempo para que seja corretamente pesada; a luz do botão só deve acender quando a atuação do botão tem efeito.
Sinais I/O:	Entradas: Botão de start e o comando de reset do <i>Factory I/O</i> (ambos momentâneos, produzem <code>TRUE</code> quando atuados), seletor das unidades de peso (duas posições – kilos ou libras) e sensores de 1 a 3 (geram um sinal <code>FALSE</code> na situação de deteção). Saídas: Tapetes (unidirecionais) de entrada e da balança, barreira, emissor, eliminador e luz do botão (todos energizados com <code>TRUE</code>), balança (que devolve um valor em tensão entre 0 e 10V; definida para um peso máximo de 20kg) e <i>display</i>, para mostrar o peso da caixa que está a ser pesada (definido como <code>INT</code>).

Analisando o sistema apresentado e a descrição do seu funcionamento constata-se que, ao contrário dos problemas apresentados anteriormente, este trata-se de um sistema mais complexo e com mais estados. Assim, e tendo em conta o que foi mencionado no capítulo anterior, o mais indicado para modelar este sistema é um grafcet. No entanto, tal como nos casos já apresentados neste capítulo, também é possível modelar os elementos separados, em grafcet. Também nesta situação tem vantagens como facilidade de leitura, programação mais imediata e também a facilidade em integrar novos elementos, tal como se observou no primeiro problema.

Repare-se ainda que neste cenário é introduzido um elemento novo: o *display*. A sua função é a de mostrar valores numéricos. Como se pode observar na

Tabela 4.7 este elemento é definido como *INT*, no entanto, este pode apresentar outras definições que são passíveis de ser alteradas no *Factory I/O*, sendo que estas afetam o intervalo de valores e o tipo de variável.

Posto isto, na Figura 4.11 pode-se consultar o grafcet desenvolvido para este sistema e, na Tabela 4.8, a sua legenda.

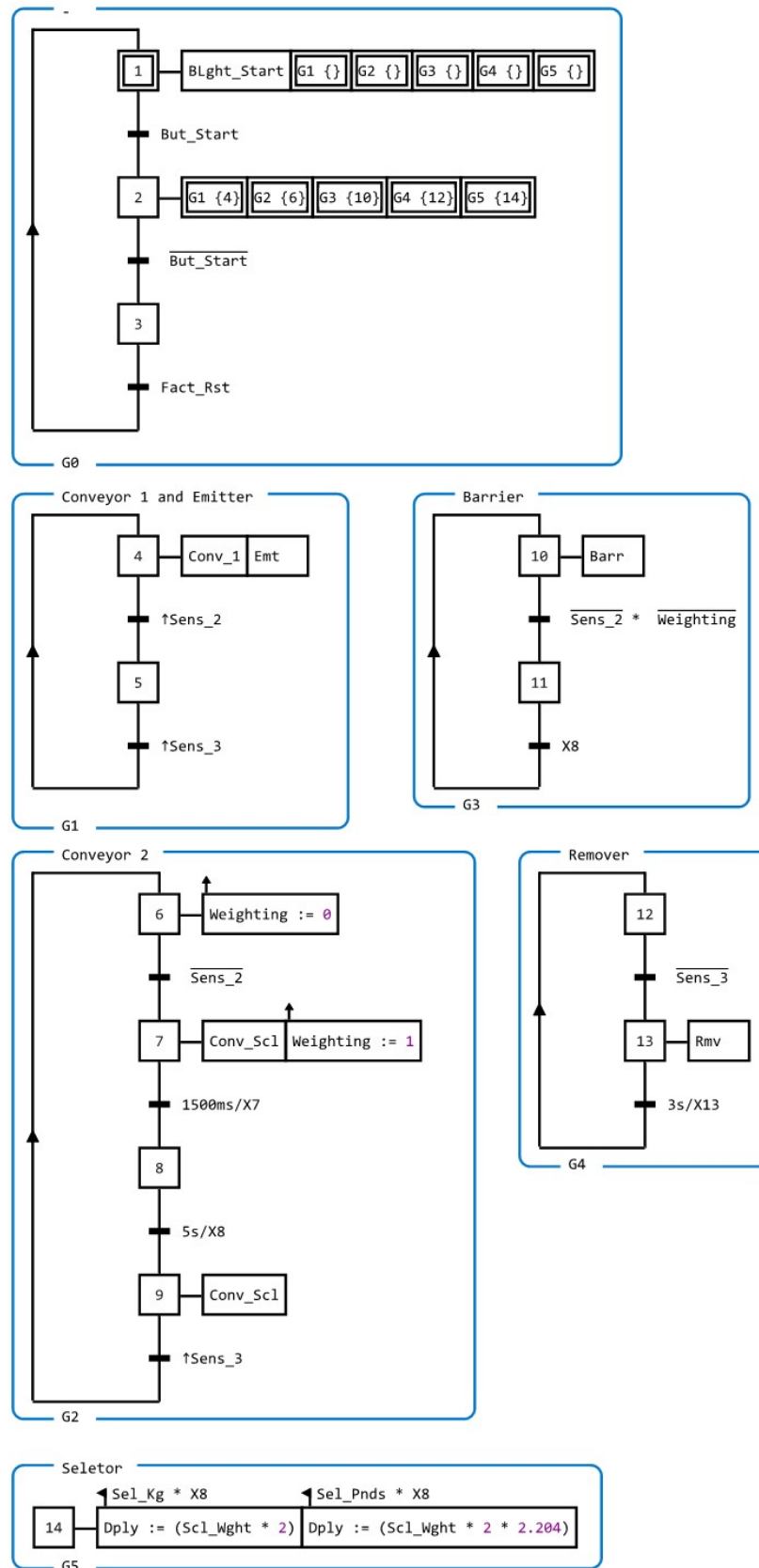


Figura 4.11 – Grafset do terceiro cenário.

Tabela 4.8 – Resumo da modelação do terceiro problema.

Legenda:	Barr: Barreira; BLght_Start: Luz do botão de start; But_Start: Botão de start; Conv_1: Tapete 1; Conv_Scl: Tapete da balança; Counter: Contador; Dply: Display; Emt: Emissor; Fact_Rstd: Factory I/O (Reset); Loading: Loading (variável auxiliar); Rmv: Eliminador; Scl_Wght: Peso lido pela balança; Sens_1: Sensor 1; Sens_2: Sensor 2; Sens_3: Sensor 3; Weighting: Weighting (variável auxiliar).
Ações de resposta ao reset do factory:	Reinício do cenário.

Tal como se apresentou no capítulo 3 pode-se, a partir do grafcet apresentado, desenvolver o programa para controlar o sistema apresentado. Resultando o seguinte programa. À semelhança do que foi mencionado para o cenário anterior, também neste caso é omitida a declaração de variáveis.

```

RSens_2 (CLK := Sens_2);

RSens_3 (CLK := Sens_3);

RFact_Run (CLK := Fact_Run);

IF Fact_Rstd THEN

    Emt := FALSE;

    Conv := FALSE;

    Conv_Scl := FALSE;

    Barr := FALSE;

    Rmv := FALSE;

    Weighting := FALSE;

    Loading := FALSE;

    Dply_Wght := 0;

    BLght_Start := TRUE;

ELSE

    TON_Pesagem (IN := Weighting, PT := T#5S);

    TON_Conv_Scl (IN := Loading, PT := T#0.25S);

```

```

    TOF_Rmv (IN := NOT Sens_3, PT := T#1S, Q => Rmv);

    RS_BLght_Start (SET := RFact_Run.Q, RESET1 := But_Start, Q1 =>
BLght_Start);

    RS_Emt (SET := But_Start OR Conv, RESET1 := BLght_Start OR NOT
Conv, Q1 => Emt);

    RS_Conv (SET := But_Start OR NOT Sens_3, RESET1 := BLght_Start
OR RSens_2.Q, Q1 => Conv);

    RS_Conv_Scl (SET := NOT Sens_2 OR TON_Pesagem.Q, RESET1 :=
BLght_Start OR TON_Conv_Scl.Q OR RSens_3.Q, Q1 => Conv_Scl);

    RS_Barr (SET := But_Start OR TON_Conv_Scl.Q, RESET1 :=
BLght_Start OR NOT Sens_2, Q1 => Barr);

    RS_Weighting (SET := RSens_2.Q, RESET1 := TON_Pesagem.Q, Q1 =>
Weighting);

    RS_Loading (SET := RSens_2.Q, RESET1 := TON_Conv_Scl.Q, Q1 =>
Loading);

    END_IF

    IF Sel_Kg THEN

        Dply_Wght := 2* Scl_Wght;

    ELSIF Sel_Pnds THEN

        Dply_Wght := (2* Scl_Wght)*2.204;

    END_IF

```

Tal como nos casos anteriormente apresentados, embora não seja explicitada, também neste se inicia pela declaração de variáveis e blocos funcionais, seguida da inicialização dos blocos que têm de ser inicializados.

Numa primeira análise do código apresentados verifica-se que foram utilizadas duas instruções `IF`. A primeira, como se pode observar, é muito semelhante à empregue nos problemas anteriores, servindo o mesmo propósito: verificar se foi feito reset ao cenário ou não, e fazer reset ou executar ao programa de acordo com a condição verificada.

A segunda instrução `IF` tem como objetivo verificar qual a posição do seletor, fazer a conversão do valor do peso lido pela balança para a unidade escolhida e escrever esse valor na variável associada ao *display*.

No que toca a blocos funcionais empregues pode-se constatar que, mais uma vez, não é utilizado nenhum que não tenha sido exposto no capítulo anterior. Já que foram utilizados apenas *latches* e temporizadores.

4.4 Quarto cenário

O quarto cenário a modelar e programar está ilustrado na Figura 4.12. Ao contrário dos cenários anteriores, este não consta da biblioteca do *Factory I/O*, tendo sido desenvolvido no âmbito desta dissertação. A Tabela 4.9 resume os requisitos de funcionamento deste sistema.

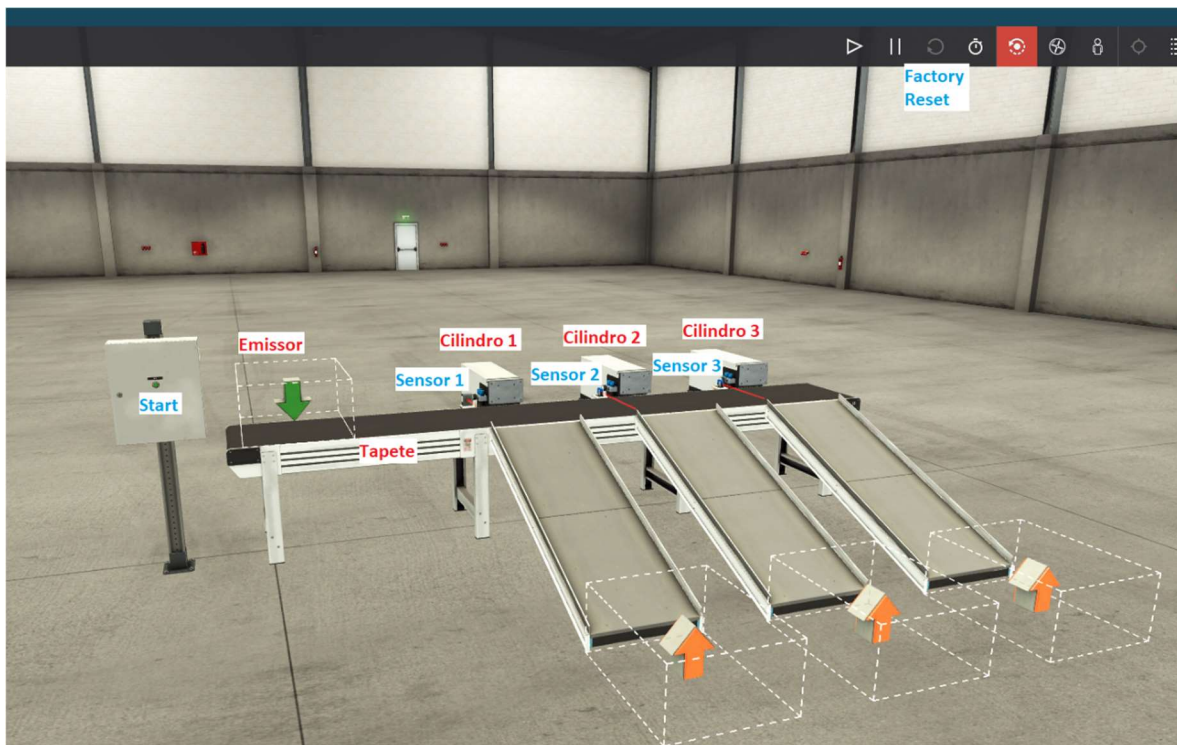


Figura 4.12 – Quarto cenário.

Tabela 4.9 – Requisitos de funcionamento do quarto cenário.

Objetivo:	Enviar peças para as rampas.
Condições iniciais:	As observáveis na Figura 4.12.
Funcionamento desejado:	Quando se atua o botão de start o emissor e o tapete ligam. O primeiro cilindro empurra todas as peças metálicas. Para isso, o sensor 1 trata-se de um sensor indutivo que devolve TRUE sempre que deteta uma peça metálica, fazendo com que o cilindro 1 seja atuado sempre que este sensor for TRUE . Por sua vez os cilindros 2 e 3 alternam entre si a empurrar peças. Ou seja, o cilindro 2 é atuado sempre que deteta uma peça (sensor 2) e é a sua vez de atuar. Já o cilindro 3 atua sempre que deteta uma peça (sensor 3). O comando de reset do <i>Factory I/O</i> reinicia o cenário.
Atuações Manuais:	Atuação do botão de start para ligar o emissor e o tapete.
Condições a salvar:	A atuação do botão start só tem efeito na situação inicial; os eliminadores estão sempre ligados; a luz do botão de start só acende quando a atuação do botão tem efeito.
Sinais I/O:	Entradas: Botão de start e comando de reset do <i>Factory I/O</i> (momentâneos, geram TRUE quando atuados), o sensor 1 (gera um sinal TRUE quando deteta uma peça metálica), os sensores 2 e 3 (geram um sinal TRUE na situação de detecção) e os sensores que indicam se o cilindro está avançado ou recuado dos respectivos cilindros 1 a 3 (também geram um sinal TRUE na situação de detecção). Saídas: Tapete (unidirecional), cilindros 1 a 3, emissor e luz do botão de start (todos energizados com TRUE).

Depois de analisar os requisitos do sistema e a sua constituição pode-se, então, modelar o mesmo. O sistema deste problema, embora relativamente simples, trata-se, à semelhança do caso anterior, de um sistema com vários estados sendo, por isso, indicada a utilização de um grafset para modelar o mesmo. Na Figura 4.13 está ilustrado o grafset deste cenário e na Tabela 4.10 encontra-se a respetiva legenda.

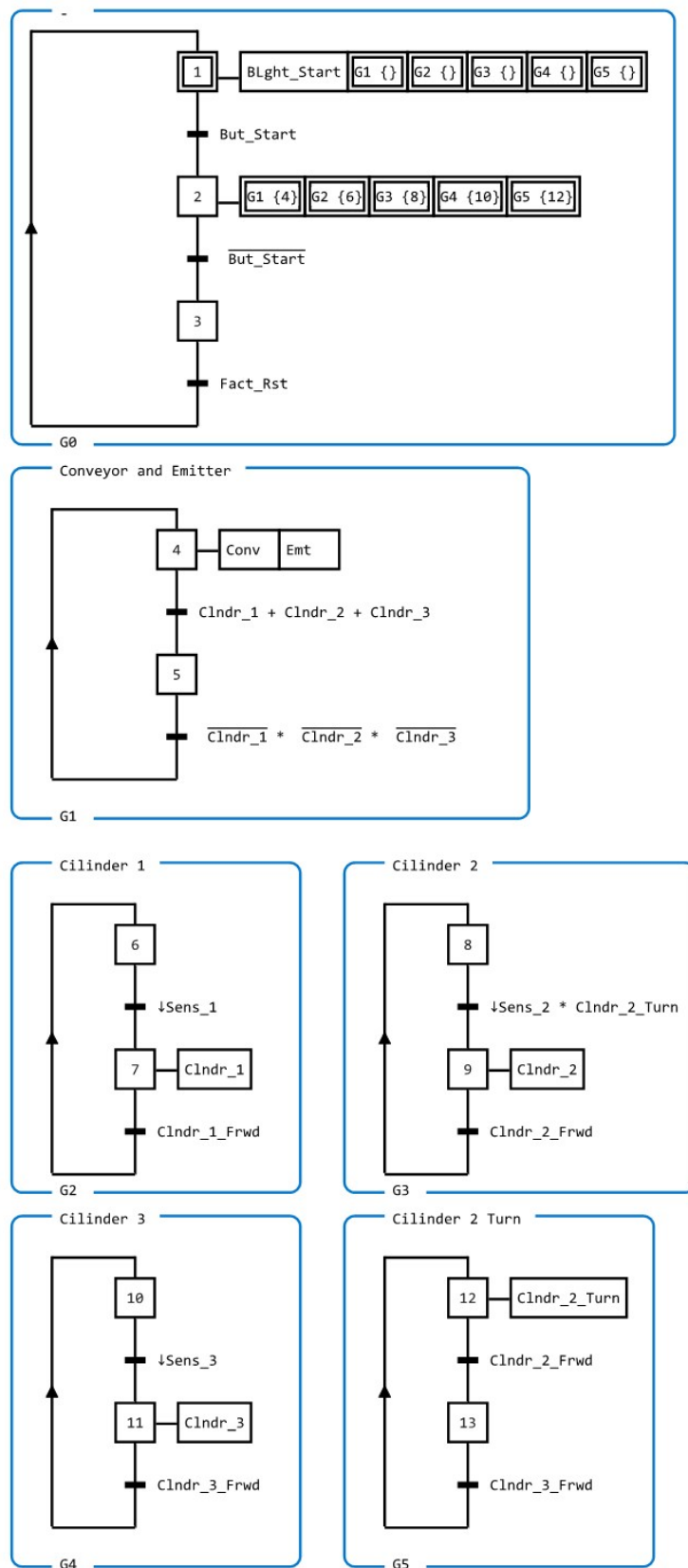


Figura 4.13 – Grafcet do quarto cenário.

Tabela 4.10 – Resumo da modelação do quarto problema.

Legenda:	But_Start: Botão de start BLght_Start: Luz do botão de start Clndr_1: Cilindro 1 Clndr_2: Cilindro 2 Clndr_3: Cilindro 3 Clndr_1_Frwd: Cilindro 1 avançado Clndr_2_Frwd: Cilindro 2 avançado Clndr_3_Frwd: Cilindro 3 avançado Clndr_2_Turn: Variável auxiliar, indica se o cilindro 2 deve ser atuado Conv: Tapete Emt: Emissor Fact_Rstd: Factory I/O (Reset) Sens_1: Sensor 1 Sens_2: Sensor 2 Sens_3: Sensor 3
Ações de resposta ao reset do factory:	Reinício do cenário

Tal como em casos anteriores, e como foi mencionado no capítulo anterior, pode-se desenvolver o programa para controlar este cenário a partir do grafcet apresentado. Resulta, então, o seguinte programa. Também neste caso se omitiu a declaração de variáveis, pelas razões já apresentadas.

```

FSens_1 (CLK := Sens_1);
FSens_2 (CLK := Sens_2);
FSens_3 (CLK := Sens_3);

IF Fact_Rstd THEN
    Emt := FALSE;
    Conv:= FALSE;
    Clndr_1 := FALSE;
    Clndr_2 := FALSE;
    Clndr_3 := FALSE;
    RS_Clndr_2_Turn.RESET1 := TRUE;
ELSE
    BLght_Start := NOT Emt AND NOT But_Start;
    RS_Emt (Set := But_Start, RESET1 := BLght_Start, Q1 => Emt);
    RS_Conv (Set := NOT BLght_Start AND NOT Clndr_1 AND NOT Clndr_2
AND NOT Clndr_3, Reset1 := BLght_Start OR Clndr_1 OR Clndr_2 OR Clndr_3, Q1
=> Conv);

```

```
RS_Clnr_2_Turn (Set := But_Start OR Clnr_3_Fwrd, Reset1 :=  
BLght_Start OR Clnr_2_Fwrd);  
  
RS_Clnr_1 (Set := FSens_1.Q, Reset1 := BLght_Start OR  
Clnr_1_Fwrd, Q1 => Clnr_1);  
  
RS_Clnr_2 (Set := (FSens_2.Q AND RS_Clnr_2_Turn.Q1), Reset1  
:= BLght_Start OR Clnr_2_Fwrd, Q1 => Clnr_2);  
  
RS_Clnr_3 (Set := FSens_3.Q, Reset1 := BLght_Start OR  
Clnr_3_Fwrd, Q1 => Clnr_3);  
  
END_IF
```

Mais uma vez, e como mencionado nos problemas até aqui apresentados, começa-se por declarar as variáveis e os blocos funcionais utilizados no programa, seguindo-se a inicialização dos FB que o requerem.

Analisando o programa observa-se novamente a utilização da instrução `IF` com a função de verificar se é necessário fazer reset ao programa ou se se deve executar este conforme se tenha, ou não, reiniciado o cenário *Factory I/O*.

Em termos de blocos funcionais utilizados, pode-se constatar que foram empregues *latches* e `F_Trig` para a programação do sistema, e que, mais uma vez, não surgem elementos que não tenham sido já referidos anteriormente.

4.5 Teste no novo PLC

Relembre-se, rapidamente, que os programas apresentados ao longo deste documento foram desenvolvidos em *Codesys*, pelas razões previamente apresentadas.

À semelhança do capítulo anterior, e tal como é objetivo desta dissertação, os programas desenvolvidos a partir dos graficets apresentados neste capítulo foram testados no *soft*PLC *ST I/O*, tendo surgido algumas questões.

Os códigos apresentados ao longo do presente capítulo, foram testados no novo PLC e foram executados sem qualquer problema.

Em termos de alterações feitas ao código, foi preciso, como visto no capítulo anterior, alterar algumas variáveis de entrada de alguns FB, nomeadamente, dos blocos `SR` e `RS` de `Set`, `Set1`, `Reset` e `Reset1` para `S`, `S1`, `R` e `R1`, e nos contadores foi preciso passar `Reset` para `R`.

Outro aspeto que deve ser chamado à atenção é a ligação de variáveis aos respetivos elementos no *Factory I/O*, mais particularmente, o *display* do terceiro cenário. Este é definido como `INT`, ou seja, tem 16 bits sendo, por isso, uma `WORD`, pelo que quando é feita a ligação ao *display* do cenário deve-se definir o seu endereço como sendo do tipo `QW` de modo a indicar que se trata de uma `WORD`. No *Codesys* não é necessário ter este detalhe da ligação em conta visto que a ligação é feita de um modo gráfico na secção dos drivers do *Factory I/O*.

4.6 Síntese

Ao longo deste capítulo foram apresentados e analisados diferentes cenários desenvolvidos na aplicação *Factory I/O*. Procurou-se, ao longo destas páginas, apresentar a sua modelação e desenvolvimento de programas, a partir dessa modelação, que imponham o funcionamento descrito.

Através desta abordagem foi possível demonstrar a utilidade dos blocos funcionais e das instruções apresentadas no capítulo anterior. Além disso, ao programar soluções para estes cenários, desenvolveram-se soluções para alguns dos cenários que constituem a biblioteca do *Factory I/O*, podendo estas, agora, ser utilizadas como material didático, tanto por professores, como por alunos, ou até indivíduos que estejam a aventurar-se na programação de PLCs em texto estruturado.

Por fim testaram-se os programas desenvolvidos no *soft*PLC *ST I/O*.

Tendo observado a aplicação da modelação e dos aspetos fundamentais da programação em ST, desde os blocos funcionais até às instruções `IF` e `CASE OF` no desenvolvimento de código para controlo de sistemas relativamente simples, é, agora, de todo o interesse escalar a dificuldade dos sistemas a controlar. Assim, no capítulo 5 procura-se expor, analisar e criar programas para controlar sistemas mais complexos, compostos por elementos mais diversos e que apresentam uma maior interação tanto entre si como com o utilizador, ao contrário do que foi observado nos cenários apresentados neste capítulo.

5 Controlo de estações de produção em *Factory I/O*

Este capítulo tem como objetivo o desenvolvimento de soluções de programação que contemplam a divisão de sistemas em subsistemas e o estabelecimento de intercomunicação entre estes, incluindo ainda a comunicação com uma interface homem-máquina – HMI. Não descorando, no entanto, o teste das soluções no *soft*PLC *ST I/O*.

Como foi mencionado no capítulo anterior, o *Factory I/O* tem uma biblioteca de cenários de demonstração que vão desde sistemas simples para transportar uma caixa de um ponto A para um ponto B até sistemas mais complexos como o armazém automático. Também no capítulo anterior foram apresentadas soluções para alguns desses cenários, tendo-se, no entanto, cingido aos casos mais simples. Assim, neste capítulo procura-se abordar alguns dos casos mais complexos presentes nessa biblioteca e desenvolver soluções para os mesmos, sempre procurando aplicar os conceitos expostos no capítulo 3.

Ao longo do capítulo serão, então, apresentados dois cenários da referida biblioteca; mais concretamente, o “*Pick and Place*” e o “*Assembler*”.

Para tal, começa-se por introduzir o problema, ilustrando o cenário e descrevendo o funcionamento desejado para o sistema em questão. É apresentada, de seguida, uma análise do sistema, onde se introduzirá a divisão do sistema em subsistemas e das respetivas tarefas de controlo. Segue-se, à semelhança do que foi feito no capítulo 4, a modelação do problema sob a forma de grafets, que, tal como no capítulo anterior, foram desenhados na aplicação *GRAFCET-Studio*, e, a partir destes, desenvolve-se uma solução de programação. Por fim, testa-se o programa desenvolvido no *soft*PLC *ST I/O*.

No entanto, o principal realce não será dado a estes tópicos, visto que já foram abordados com alguma profundidade nos capítulos anteriores, mas procurar-se-á antes explorar a questão da comunicação entre subsistemas bem como a interação com o utilizador, por meio de uma interface homem-máquina.

Tendo ainda em conta que as questões mais básicas da programação, como declaração de variáveis e inicialização de blocos funcionais, já foram abordadas em capítulos anteriores com algum detalhe, neste capítulo os códigos não serão apresentados na íntegra. Serão apresentadas apenas as secções de interesse, ou seja, as que demonstram as questões de comunicação entre subsistemas, visto ser esse o foco deste capítulo.

5.1 Primeiro cenário

O primeiro sistema a controlar é o “*Pick and Place*” da biblioteca do *Factory I/O*, e está ilustrado na Figura 5.1. Os requisitos para o seu funcionamento estão resumidos na Tabela 5.1 e a Figura 5.2 ilustra o funcionamento deste sistema.

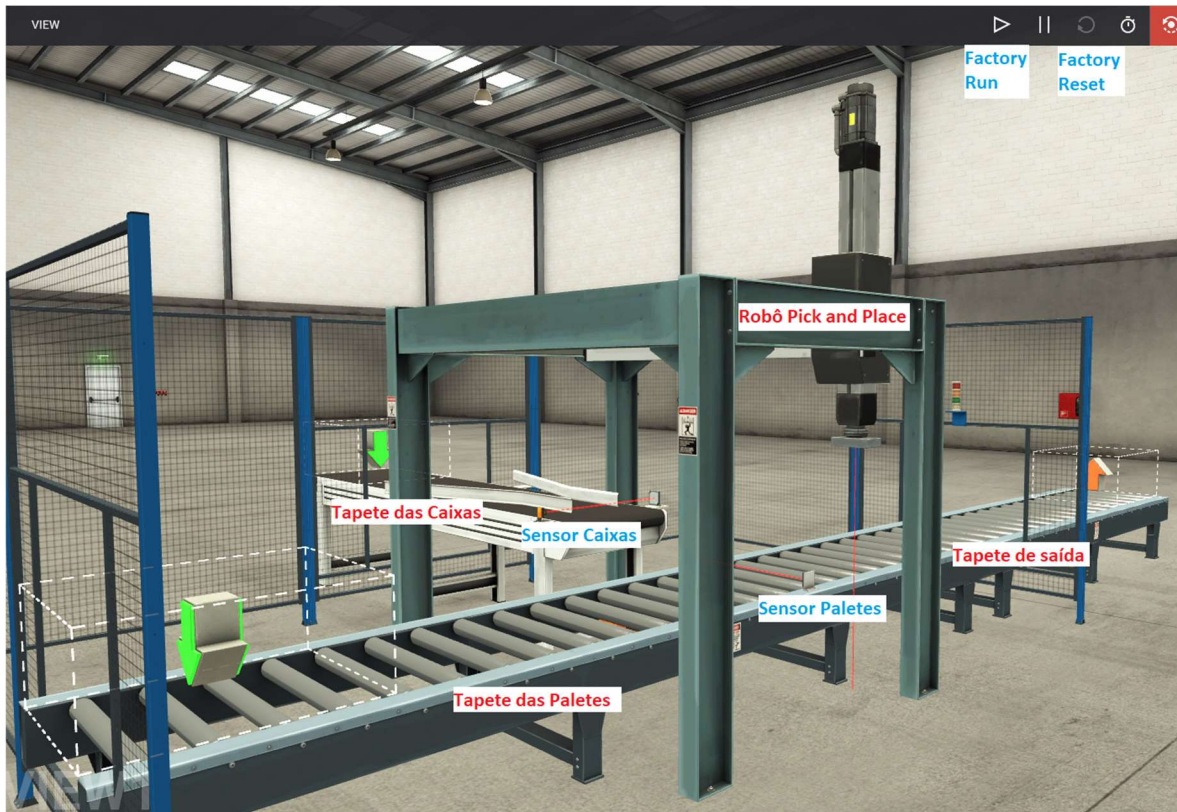


Figura 5.1 – Primeiro cenário.

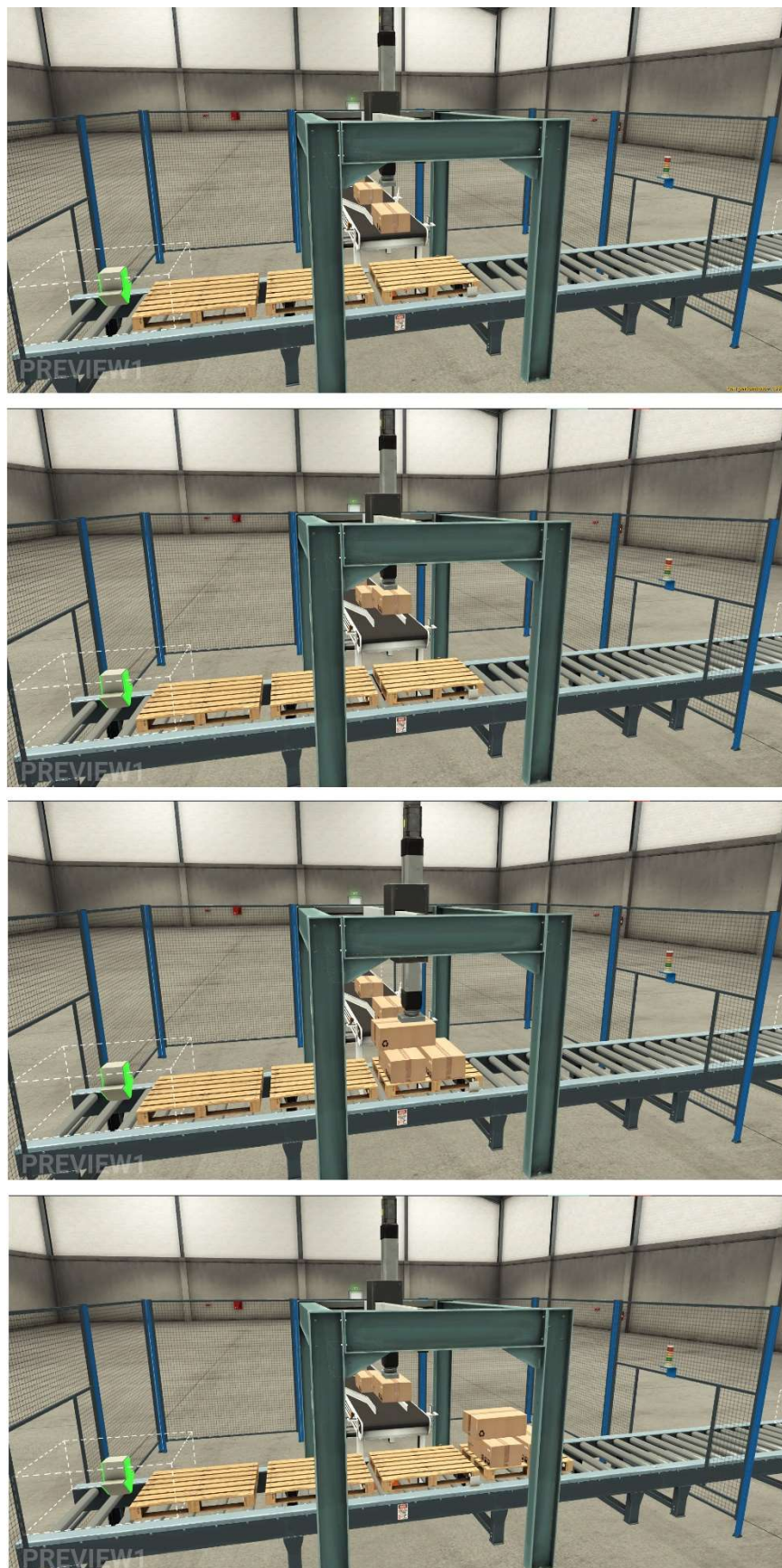


Figura 5.2 – Ilustração do funcionamento do primeiro sistema.

Tabela 5.1 – Requisitos de funcionamento do primeiro cenário.

Objetivo:	Empilhar três caixas em cada palete.
Condições iniciais:	As observáveis na Figura 5.1.
Funcionamento desejado:	Quando se coloca o <i>Factory I/O</i> em modo “Run”, os emissores e os tapetes ligam. Quando uma paleta chega ao sensor das paletes, os tapetes das paletes e de saída param. Por sua vez, o tapete das caixas para quando uma caixa é detetada pelo respetivo sensor. Quando houver uma paleta e uma caixa em posição, o robô vai buscar a caixa e colocá-la na paleta. O robô deve criar uma pirâmide com três caixas. Sempre que uma caixa é removida pelo robô o tapete das caixas liga para colocar outra em posição. Quando a referida pirâmide estiver feita o tapete das paletes e o de saída ligam até que haja uma paleta vazia em posição e o processo repete-se. O comando de reset do <i>Factory I/O</i> reinicia o cenário.
Atuações Manuais:	Dar início ao funcionamento do sistema colocando o <i>Factory I/O</i> em modo “Run”; reiniciar o cenário atuando o comando de reset do <i>Factory I/O</i> .
Sinais I/O:	Entradas: Comando para colocar o <i>Factory I/O</i> em modo Run e comando de reset do <i>Factory I/O</i> (todos momentâneos, produzem <code>TRUE</code> quando premidos), sensores das caixas e das paletes (geram <code>FALSE</code> na situação de deteção) e sensores de posicionamento do robô (variáveis do tipo <code>INT</code>). Saídas: Tapetes das paletes, de saída e das caixas (unidirecionais, todos energizados com <code>TRUE</code>), robô (atuado por variáveis do tipo <code>INT</code> para definir a sua posição), ventosa do robô e rotação da ventosa (ambos energizados com <code>TRUE</code>).

Tal como visto no capítulo anterior, agora que se conhece o sistema que se quer controlar, e se sabe como é que este se deve comportar, é agora necessário, antes de programar, modelar o comportamento pretendido para o sistema.

Como se pode observar, o sistema em questão é complexo, com alguma diversidade de elementos, cada um com vários estados. Assim sendo, como foi referido no capítulo 3, a modelação mais adequada para esta situação será através de um grafcet.

Numa análise mais cuidada, verifica-se que este sistema pode ser dividido em três subsistemas relativamente independentes, sendo eles, o subsistema de controlo das paletes, constituído pelo tapete das paletes e pelo de saída, bem como o sensor das paletes; o subsistema de controlo das caixas, constituído pelo tapete das caixas e respetivo sensor e, finalmente, o subsistema de controlo do robô, constituído pelo robô e os seus sensores. Esta separação facilita não só a modelação, mas também, como será apresentado brevemente, o estabelecimento de comunicação entre estes. É de referir ainda, que o funcionamento do robô foi separado em dois ciclos. Isto é, dado que o robô desempenha determinadas funções de um modo repetitivo, mais concretamente, os movimentos de deslocamento até à posição onde vai buscar a caixa, bem como os de subida e descida para agarrar ou largar a caixa, este foram colocados no seu próprio ciclo, que deve ser ativado quando necessário. Deste modo, o grafcet do sistema fica mais simples e fácil de ler. O grafcet que resulta desta análise está ilustrado na Figura 5.3 e, na Tabela 5.2, encontra-se a legenda do mesmo.

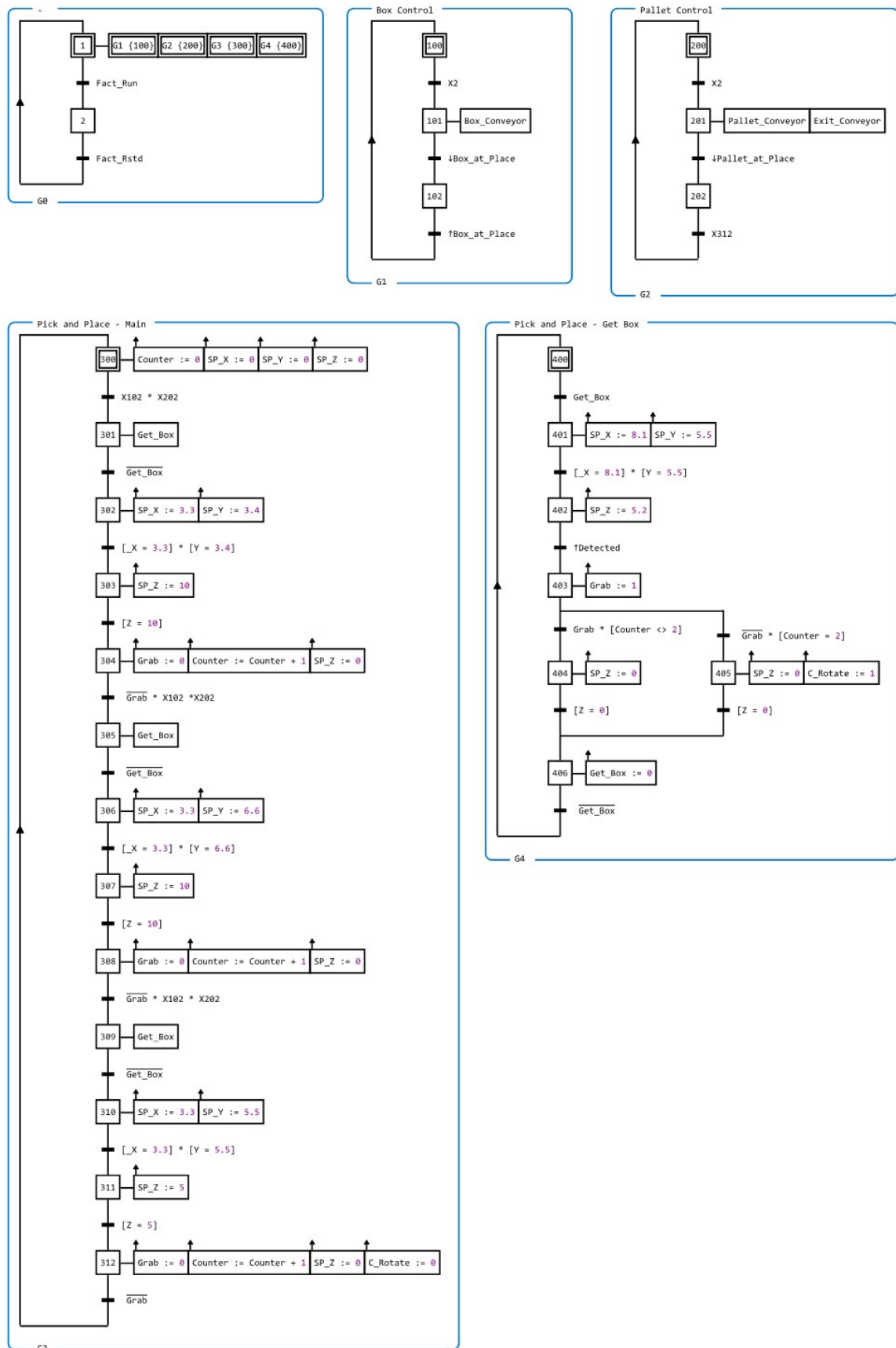


Figura 5.3 – Grafcet do primeiro problema.

Tabela 5.2 – Legenda do grafcet do primeiro cenário.

Legenda:	Box_at_Place: Sensor das caixas; Box_Conveyor: Tapete das caixas; C_Rotate: Rotação da ventosa do robô; Counter: Contador; Detected: Sensor na ponta do braço do robô; Exit_Conveyor: Tapete de saída; Fact_Run: Factory I/O (Running); Fact_Rstd: Factory I/O (Reset); Get_Box: Variável auxiliar; Grab: Ventosa do robô; Pallet_at_Place: Sensor das paletes; Pallet_Conveyor: Tapete das paletes; SP_X: Variável de posicionamento do robô em X; SP_Y: Variável de posicionamento do robô em Y; SP_Z: Variável de posicionamento do robô em Z; X: Posição do robô em X; Y: Posição do robô em Y; Z: Posição do robô em Z.
-----------------	---

Como foi mencionado, o sistema pode ser dividido em três subsistemas, no entanto, na Figura 5.3, observam-se cinco grafcets. Isto acontece porque, além dos grafcets dos subsistemas de controlo das caixas e das paletes, G1 e G2 respetivamente, inclui-se um grafcet G0, que procura representar o controlo do sistema, e ainda G3 e G4 que modelam o funcionamento do robô em dois ciclos, como mencionado previamente.

O grafcet G0 procura, ainda que de um modo simples, representar o controlo do sistema, como será explicado brevemente. Relativamente aos grafcets do robô, G3, controla o posicionamento o braço do robô sobre a palete e a colocação da caixa na mesma, e G4, controla o processo de obtenção da caixa.

No entanto, para que o sistema tenha o funcionamento desejado estes devem comunicar entre si. Nos próximos parágrafos procura-se explicar esta questão com mais detalhe.

Antes de mais, como já foi referido, além dos grafcets para os subsistemas definidos, existe um grafcet G0, detalhado na Figura 5.4, que tem apenas como função indicar se o sistema está, ou não, em funcionamento e, quando a variável *Fact_Rstd* é TRUE, reiniciar o programa. Deste modo, caso ocorra algum problema no decorrer do funcionamento do sistema, atuando apenas o comando de reset do *Factory I/O*, consegue-se reiniciar o cenário e o programa, sem ter de desligar e ligar o PLC.

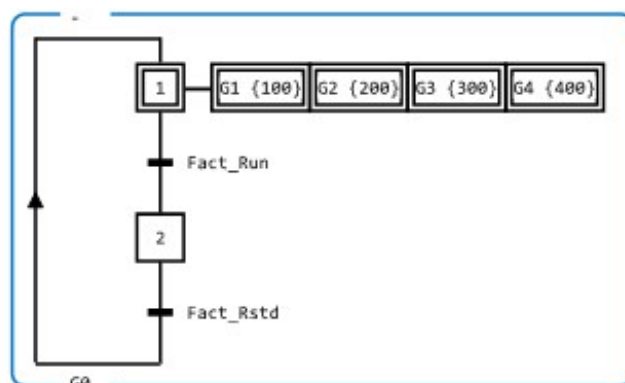


Figura 5.4 – Grafcet G0 do primeiro cenário.

Relativamente aos subsistemas de controlo das caixas e das paletes, como se pode observar no grafcet da Figura 5.3, estes começam nas suas etapas iniciais, onde aguardam que o cenário seja colocado em modo “Run”. Na Figura 5.5 detalha-se como estes grafkets estabelecem contacto com o grafcet G0, que tem a função, ainda que de um modo genérico e muito simples, de controlar o funcionamento do sistema. Assim, quando a etapa 2 de G0 está ativa, ou seja, a variável `Fact_Run` é `TRUE`, então os subsistemas de controlo das caixas e das paletes podem iniciar o seu funcionamento, visto que as suas transições de saída das etapas iniciais são verificadas.

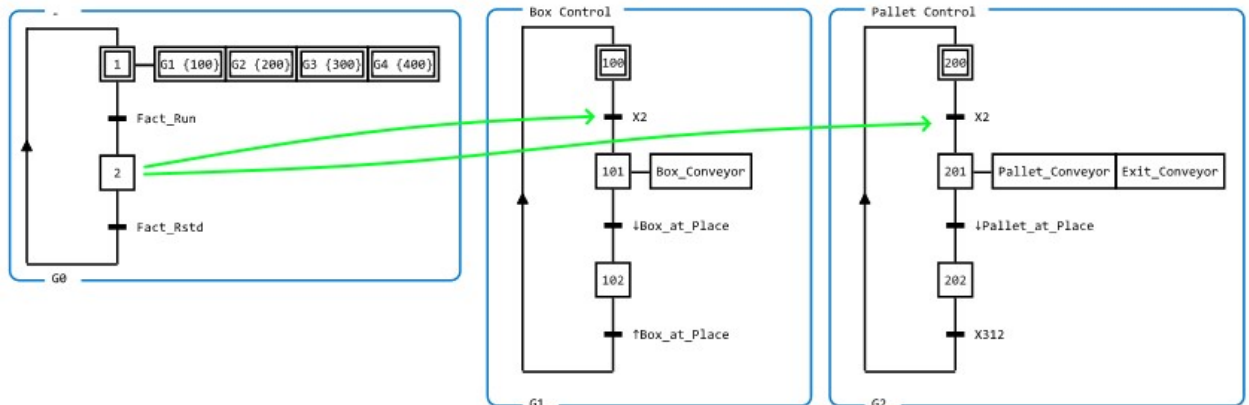


Figura 5.5 – Comunicação entre o grafcet G0 e os grafkets dos subsistemas de controlo das caixas e das paletes, G1 e G2.

Quanto ao robô, este só deve ser atuado quando houver uma caixa e uma paleta em posição, então, é necessário que este receba essa informação dos respetivos subsistemas. Essa troca de informação é feita usando as etapas dos subsistemas de controlo das caixas e das paletes. Ou seja, quando os subsistemas detetam uma caixa e uma paleta através dos respetivos sensores e, por tanto, avançaram das etapas 101 e 201 para as etapas 102 e 202, respetivamente, é verificada a transição de saída da etapa inicial do robô, fazendo com que este inicie o processo de ir buscar uma caixa. A Figura 5.6 salienta essa comunicação.

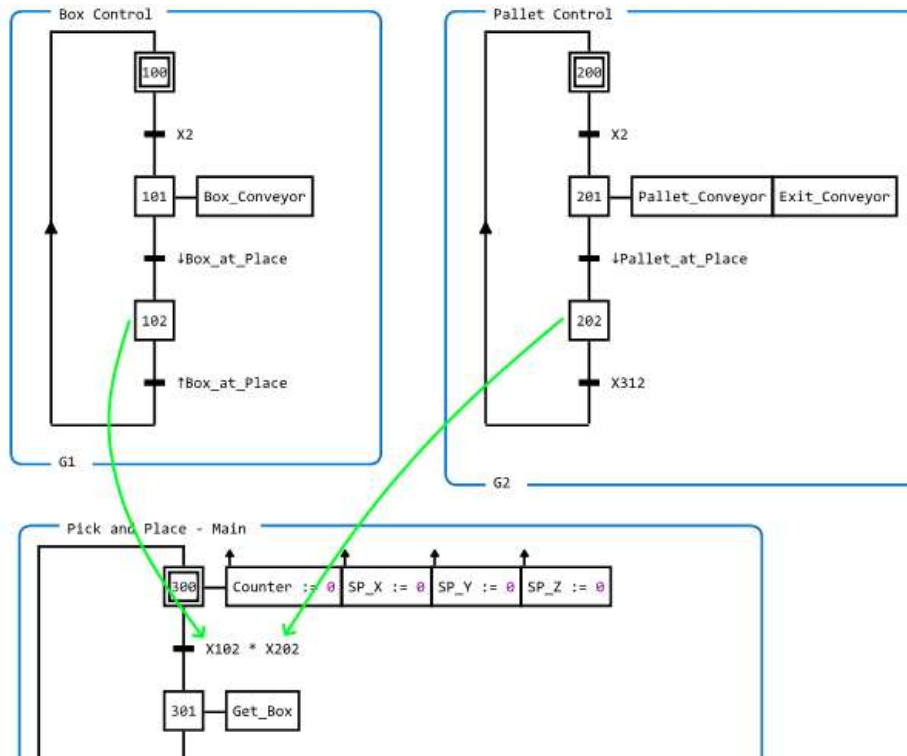


Figura 5.6 – Comunicação entre os subsistemas de controle das caixas e das paletes e o subsistema principal do robô.

Como já foi mencionado e se pode observar no grafcet da Figura 5.3, o sistema de controle do robô apresenta dois grafcets. O grafcet G4 tem como função controlar o ciclo de carga e descarrega da caixa, enquanto o grafcet G3 controla o restante funcionamento do robô. Estes dois ciclos de funcionamento deste subsistema têm de comunicar entre si, de modo a garantir que o robô desempenha o comportamento desejado. Assim sendo, quando o ciclo principal, G3, verifica que é necessário ir buscar ou deixar uma caixa, “chama” o G4. Essa comunicação é feita por meio de uma variável denominada `Get_Box`, que, quando `TRUE`, faz com que se verifique a primeira transição de G4, informando este que deve executar o seu papel e, quando este termina, ou seja, `Get_Box` é `FALSE`, informa G3 que pode continuar a sua função. Esta troca de informação está detalhada na Figura 5.7.

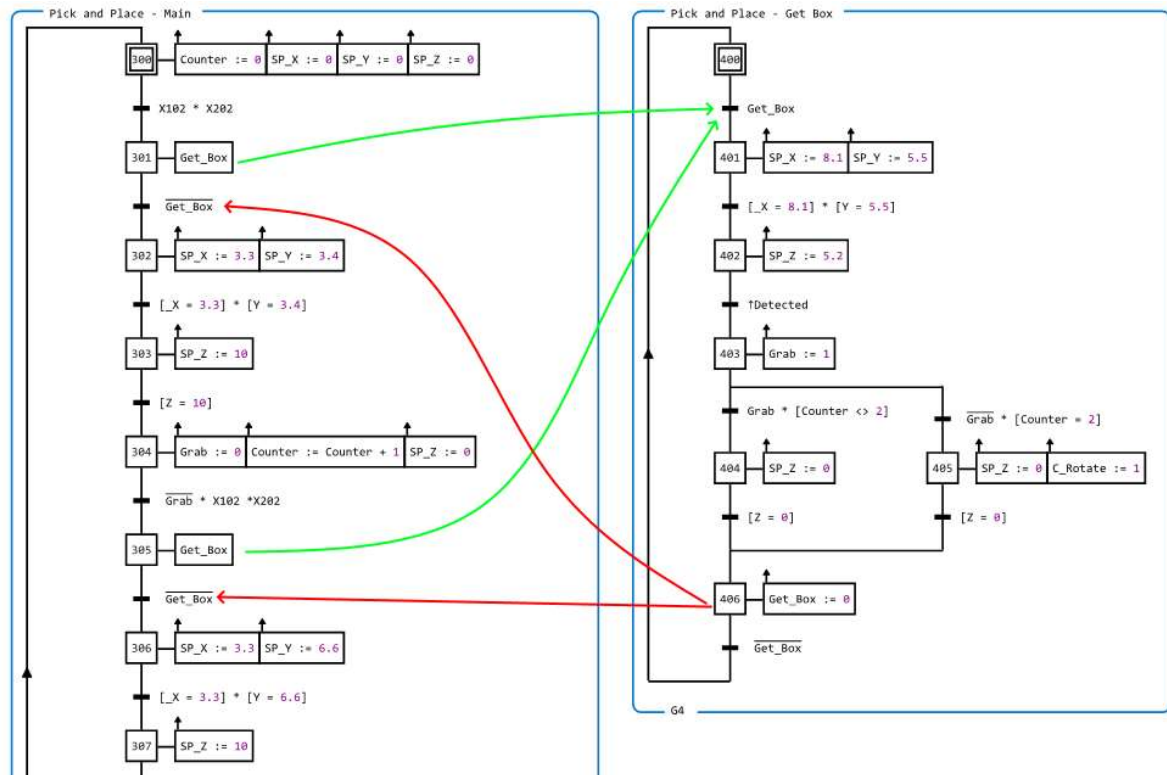


Figura 5.7 – Comunicação entre os subsistemas principal e o Get Box do robô.

Finalmente, o subsistema de controlo das paletes, tem de saber quando é que a paleta está completamente carregada, de modo a poder proceder à remoção da mesma e à colocação de uma paleta vazia em posição para ser carregada. Para isso, o subsistema das paletes vai ficar à espera de que o robô termine o seu ciclo principal, que acaba quando este tiver colocado a última das três caixas na paleta. Assim, quando a etapa 312 do graficet G3 é ativada, a última transição do subsistema de controlo das paletes é verificada e este volta ao início, ligando o tapete até que uma nova paleta esteja em posição. A Figura 5.8 ilustra esta questão.

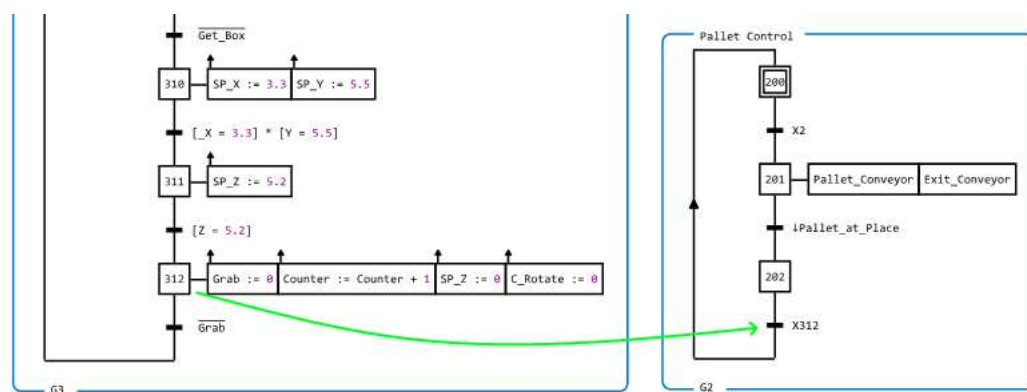


Figura 5.8 – Comunicação entre o subsistema principal do robô e o subsistema de controlo das paletes.

À semelhança daquilo que foi apresentado nos capítulos anteriores, agora que se tem a modelação completa do sistema, pode-se desenvolver o programa para controlar o cenário. De seguida apresenta-se, como foi referido inicialmente, as secções que ilustram as questões abordadas.

```
IF Fact_Rstd THEN
```

```
(...)  
  
ELSE  
  
    RS_Pallet_Conv (SET := R_Fact_Run.Q OR State_G3 = 312, RESET1  
:= F_Pallet_at_Place.Q, Q1 => Pallet_Conveyor, Q1 => Exit_Conveyor);  
  
    RS_Box_Conv (SET := R_Fact_Run.Q OR Box_at_Place, RESET1 := NOT  
Box_at_Place, Q1 => Box_Conveyor);  
  
    CASE State_G3 OF  
  
        300:  
  
            SP_X := 0;  
  
            SP_Y := 0;  
  
            SP_Z := 0;  
  
            C_Rotate := FALSE;  
  
            Grab := FALSE;  
  
  
            IF NOT Pallet_at_Place AND NOT Box_at_Place THEN  
  
                Get_Box := TRUE;  
  
                State_G3 := 301;  
  
            END_IF  
  
        301:  
  
            IF NOT Get_Box THEN  
  
                State_G3 := 302;  
  
            END_IF  
  
        (...)  
  
    END_CASE  
  
    CASE State_G4 OF  
  
        400:  
  
            IF Get_Box THEN  
  
                State_G4 := 401;  
  
            END_IF  
  
        (...)  
  
        406:  
  
            Get_Box := FALSE;
```

```

        IF NOT Get_Box THEN

            State_G4 := 400;

        END_IF

    END_CASE

END_IF

```

Como se pode observar utilizou-se, mais uma vez, uma instrução `IF` para separar a situação de reset do programa e a execução normal do mesmo. Relativamente à parte do código que corresponde à execução do programa propriamente dito, verifica-se que, contrariamente ao que foi apresentado no capítulo 3, são apresentadas apenas duas instruções `CASE OF` e não uma para cada grafcet. Isto acontece porque, analisando com cuidado os grafkets dos subsistemas de controlo das caixas e das paletes, constata-se que estes representam nada mais do que *latches*, o que permite simplificar o código.

Relativamente à comunicação entre subsistemas e, começando do topo do código, verifica-se na *latch* do subsistema de controlo de paletes que a condição de set é definida como sendo a transição ascendente da variável `Fact_Run`, ou seja, o momento de arranque do cenário liga os tapetes deste sistema, e ainda o estado 312 do grafcet G3, ou seja, quando a paleta está cheia o tapete liga para a retirar, tal como referido anteriormente.

Quanto ao subsistema de controlo das caixas, a comunicação apresentada anteriormente mencionava, tal como para o subsistema das paletes, que o cenário *Factory I/O* estar em modo “Run” dava início ao funcionamento deste subsistema. Por isso surge, como condição de set deste sistema, a transição ascendente da variável `Fact_Run`, tal como visto para o subsistema das paletes.

Finalmente, falta apenas mencionar a comunicação entre os dois ciclos de funcionamento do robô. Tal como foi modelado nos grafkets apresentados, quando o ciclo principal do robô verifica a sua primeira transição, a variável `Get_Box` é colocada a `TRUE` para que o ciclo de carga e descarga da caixa seja ativado - G4. Assim, na transição da etapa 300 para a 301 esta variável é colocada a `TRUE` e, na etapa 400, verifica-se a condição de transição, tal como definido. Quando G4 termina a sua função, coloca a variável `Get_Box` a `FALSE` permitindo a G3 que continue o seu funcionamento, avançando para a etapa 302. Esta comunicação é feita deste modo sempre que necessário, como definido no grafcet.

Note-se ainda que na condição de transição da etapa 300 para a 301, não se utilizaram as etapas de G1 e G2, com definido no grafcet, porque, como referido, estes foram programados com recurso a *latches*, pelo que, para a condição de transição, fez-se uso dos sensores dos respetivos subsistemas, que codificam a mesma situação.

5.2 Segundo cenário

O segundo cenário que se quer controlar é o “*Assembler*” que consta na biblioteca do *Factory I/O*. Este será apresentado em duas partes. Numa primeira abordagem controla-se o sistema sem recurso a uma interface com o utilizador, que neste caso será feita através de um quadro elétrico, e, posteriormente, incorpora-se essa mesma interface. Tendo em conta que o cenário é o mesmo para as duas situações sendo que a única diferença reside na questão de usar,

ou não, o referido quadro elétrico, a Figura 5.9 serve de ilustração do cenário em estudo para as duas situações expostas. Na Figura 5.10 ilustra-se o funcionamento do sistema.



Figura 5.9 – Segundo cenário.

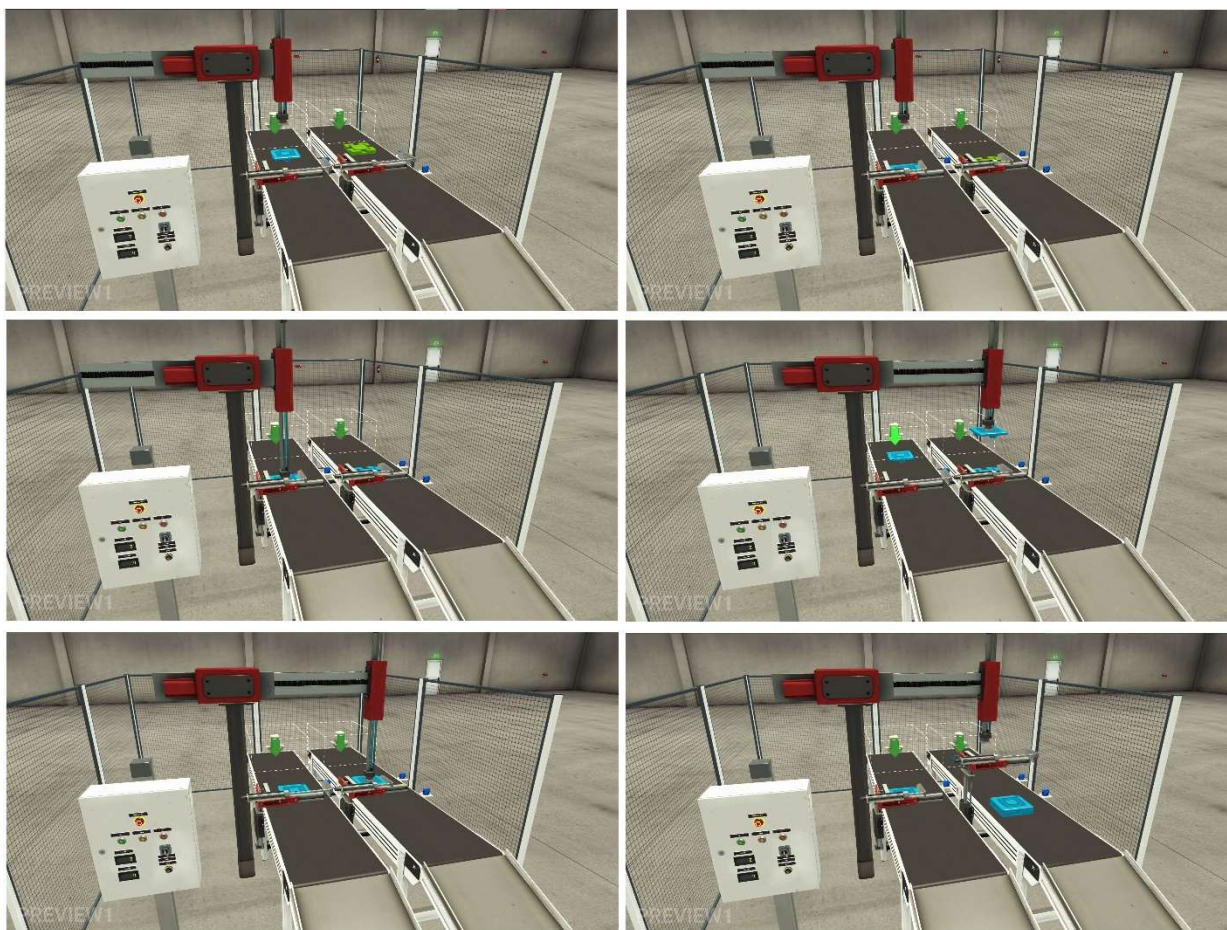


Figura 5.10 – Ilustração do funcionamento do segundo cenário.

5.2.1 Sem interface homem-máquina

Na Tabela 5.3 estão resumidos os requisitos de funcionamento deste sistema.

Tabela 5.3 – Resumo dos requisitos do segundo cenário, sem interface.

Objetivo:	Colocar tampas em bases.
Condições iniciais:	As observáveis na Figura 5.9.
Funcionamento desejado:	Quando se coloca o <i>Factory I/O</i> em modo “Run”, os emissores e os tapetes ligam. Quando uma tampa encosta à garra das tampas, ou seja, o sensor das tampas volta a FALSE, a garra aperta a tampa de modo que esta última esteja na posição correta para que o robô a agarre. Estando a tampa em posição, o braço do robô desce e agarra-a. Depois sobe, avança e, havendo uma base em posição, desce e larga a tampa, montando-a na base. Relativamente ao tapete das bases, este funciona de maneira similar à do tapete já descrito, com a diferença que, quando a tampa é montada na base, a garra deste tapete deve largar a peça e subir e o tapete deve ligar de modo a transportar a peça para a rampa de saída. O comando de reset do <i>Factory I/O</i> reinicia o cenário.
Atuações Manuais:	Dar início ao funcionamento do sistema colocando o <i>Factory I/O</i> em modo “Run”; reiniciar o cenário atuando o comando de reset do <i>Factory I/O</i> .
Sinais I/O:	Entradas: Comando para colocar o <i>Factory I/O</i> em modo Run e comando de reset do <i>Factory I/O</i> (todos momentâneos, produzem TRUE quando premidos), sensores das tampas, das bases e de saída (geram TRUE na situação de detecção) e sensores de posicionamento do robô (variáveis do tipo BOOL). Saídas: Tapetes das tampas e das bases (unidirecionais, todos energizados com TRUE), robô (atuado por variáveis do tipo BOOL) e ventosa do robô (energizada com TRUE).

Tal como visto anteriormente, conhecendo a constituição do sistema a controlar e sabendo o comportamento desejado, é agora necessário modelar o mesmo.

Como se pode observar, à semelhança do cenário anterior, também este se trata de um sistema complexo, com alguma diversidade de elementos, e vários estados. Assim sendo, também este será modelado em GRAFCET.

À semelhança do primeiro sistema também este pode ser dividido em subsistemas de modo a facilitar a modelação e respetiva programação. A divisão deste pode ser feita em três subsistemas e da seguinte maneira: o subsistema de controlo das tampas, composto pelo emissor, o tapete e a garra das tampas; o subsistema de controlo das bases, constituído pelo emissor, o tapete e a garra das bases e, finalmente, o último subsistema é o de controlo do robô.

Viu-se, no caso anterior, que dado que o robô apresentava comportamentos repetitivos estes tinham sido separados e condensados num grafcet à parte do grafcet principal do robô. Neste sistema também é possível identificar uma situação semelhante, pelo que também este robô apresenta dois grafcets, um para o ciclo principal do robô e outro para descer o braço do robô, agarrar, ou largar, a tampa e subir o braço.

Assim obtém-se um grafcet mais simples e fácil de ler. A modelação que resulta desta análise está ilustrado na Figura 5.11. A Tabela 5.4 contém a legenda para o grafcet desenvolvido.

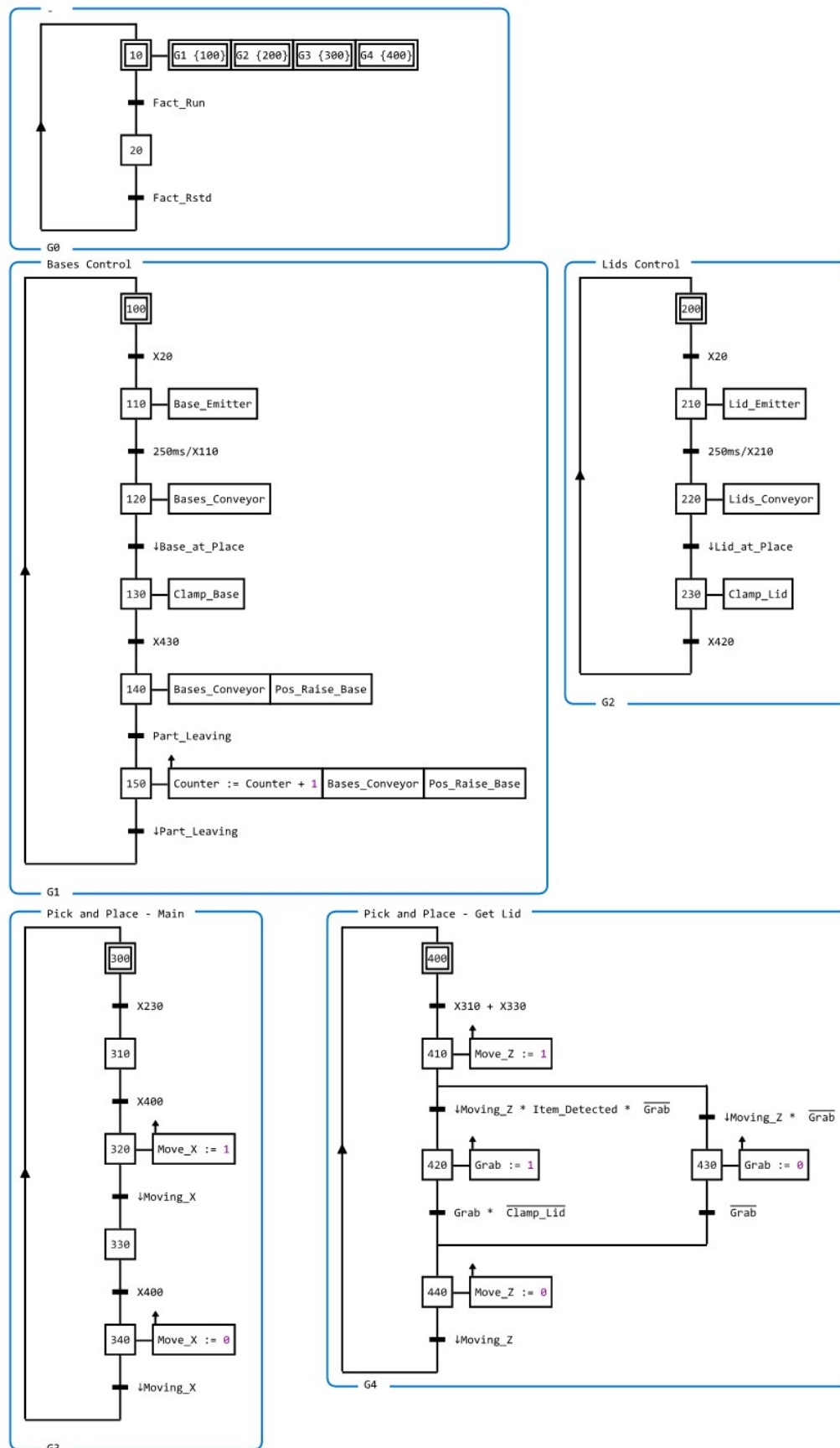


Figura 5.11 – Grafcet do segundo cenário, sem interface homem-máquina.

Tabela 5.4 – Legenda do grafcet do segundo cenário, sem interface.

Legenda:	Base_at_Place: Sensor das bases; Bases_Conveyor: Tapete das bases; Base_Emitter: Emissor de bases; Clamp_Base: Garra das bases; Clamp_Lid: Garra das tampas; Counter: Contador; Fact_Rstd: Factory I/O (Reset); Fact_Run: Factory I/O (Running); Grab: Ventosa do robô; Item_Detected: Sensor na ponta do braço do robô; Lid_at_Place: Sensor das tampas; Lids_Conveyor: Tapete das tampas; Lid_Emitter: Emissor das tampas; Move_X: Variável de movimento do robô em X; Move_Z: Variável de movimento do robô em Z; Moving_X: Sensor de movimento do robô em X; Moving_Z: Sensor de movimento do robô em Z; Pos_Raise_Base: Atuador para levantar a garra.
-----------------	--

Tal como no caso anterior, também neste se dividiu o sistema em três subsistemas, no entanto, observam-se na Figura 5.11 cinco grafcets. Isto acontece porque, à semelhança do cenário anterior, além dos grafcets de cada subsistema, inclui-se um grafcet G0 que representa o controlo do sistema, e ainda dois grafcet, G3 e G4, que modelam o funcionamento do robô em dois ciclos, tal como referido anteriormente.

Assim, G3, controla o movimento do braço do robô na direção horizontal, e G4 controla o processo de obtenção e deposição da tampa.

Estes subsistemas, à semelhança do que foi apresentado para o primeiro cenário, também precisam comunicar entre si de modo que o sistema funcione como um todo e de acordo com os requisitos estabelecidos. Assim, de seguida, explica-se como essa comunicação é feita.

À semelhança do sistema anterior, também o grafcet G0 tem a função de exercer controlo sobre os restantes subsistemas, ainda que de uma forma muito simples. Este está detalhado na Figura 5.12.

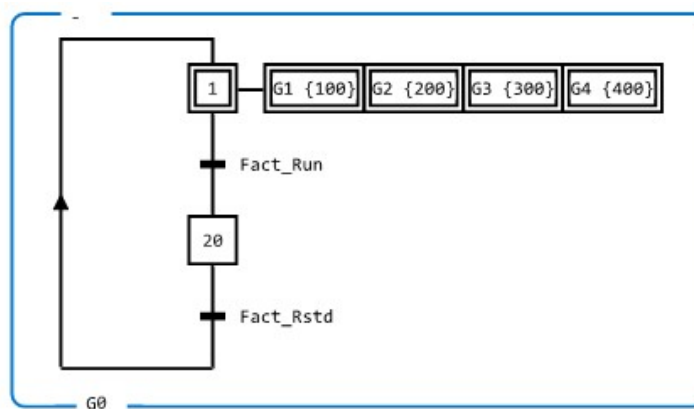


Figura 5.12 – Grafcet G0 do segundo cenário.

O grafcet G0, como foi mencionado, tem como função iniciar ou terminar o funcionamento do sistema. Assim, quando este determina que o sistema deve ser ativado, ou seja, a variável `Fact_Run` é `TRUE`, deve comunicar essa informação aos subsistemas que compõem o cenário. À semelhança do que acontecia no primeiro cenário, também neste, essa comunicação é feita usando a ativação de etapas como condição de transição de outros grafcets. Assim, quando a etapa 20 de G0 é ativada verifica-se a primeira transição dos subsistemas de controlo das tampas e das bases, como ilustrado na Figura 5.13.

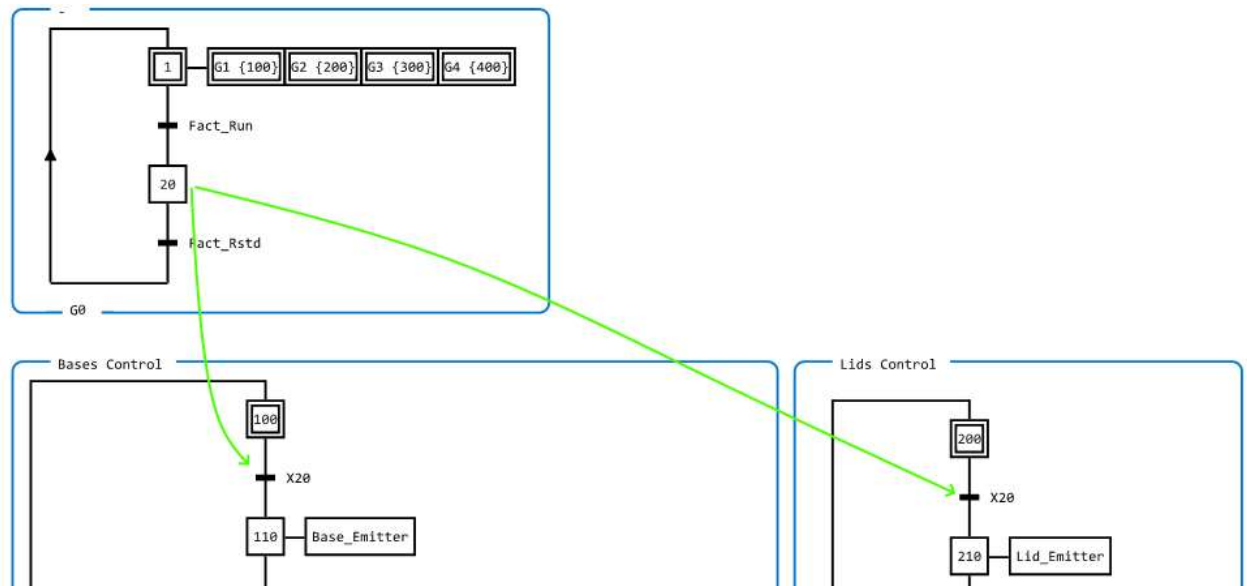


Figura 5.13 – Detalhe da comunicação entre G0 e os subsistemas de controlo das tampas e das bases.

Como referido no funcionamento desejado para o sistema em causa, o robô só deve descer o seu braço quando houver uma tampa para agarrar. Assim sendo, o subsistema de controlo das tampas deve informar o robô quando tiver uma tampa em posição para este último iniciar o seu movimento. O subsistema do robô começa, como os restantes, na sua etapa inicial onde espera que as condições necessárias para o seu funcionamento sejam verificadas, neste caso particular, espera que o subsistema das tampas esteja na sua etapa 230, onde tem a tampa presa pela garra. Quando a referida etapa está ativa verifica-se a transição do robô fazendo com que este inicie o seu movimento. Na Figura 5.14 está destacada esta comunicação entre os subsistemas referidos.

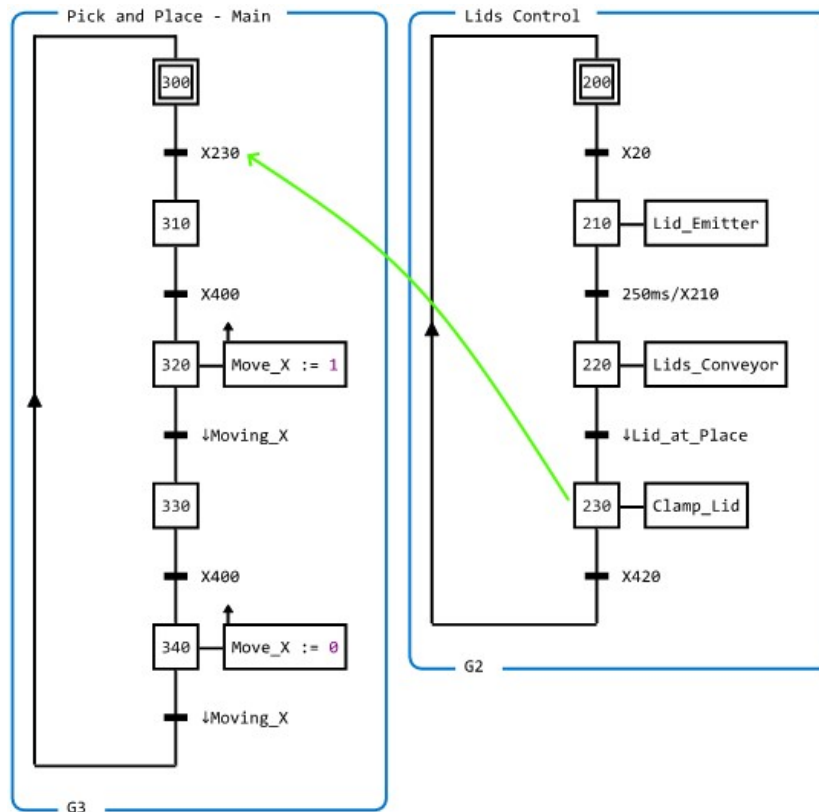


Figura 5.14 – Comunicação entre o subsistema de controle das tampas e o ciclo principal de funcionamento do robô.

Como referido, o subsistema de controlo do robô apresenta dois grafcets, um para cada ciclo de funcionamento do mesmo. Tal como foi apresentado no cenário anterior, estes dois grafcets têm de estabelecer uma comunicação entre si de modo que o robô desempenhe o comportamento desejado. Essa comunicação é feita do mesmo modo que a comunicação entre G0 e os subsistemas de controlo das tampas e das bases, fazendo uso das etapas. Assim, quando a etapa 310, ou 330, de G3 está ativa verifica-se a primeira transição de G4, permitindo que este execute o seu papel de carga, ou descarga, da tampa. Por sua vez, quando G4 termina a sua função, ou seja, agarra ou larga a tampa, e regressa à sua etapa inicial, etapa 400, verifica-se a transição de G3 e este continua o seu funcionamento. A Figura 5.15 ilustra esta troca de informação entre os dois grafcets.

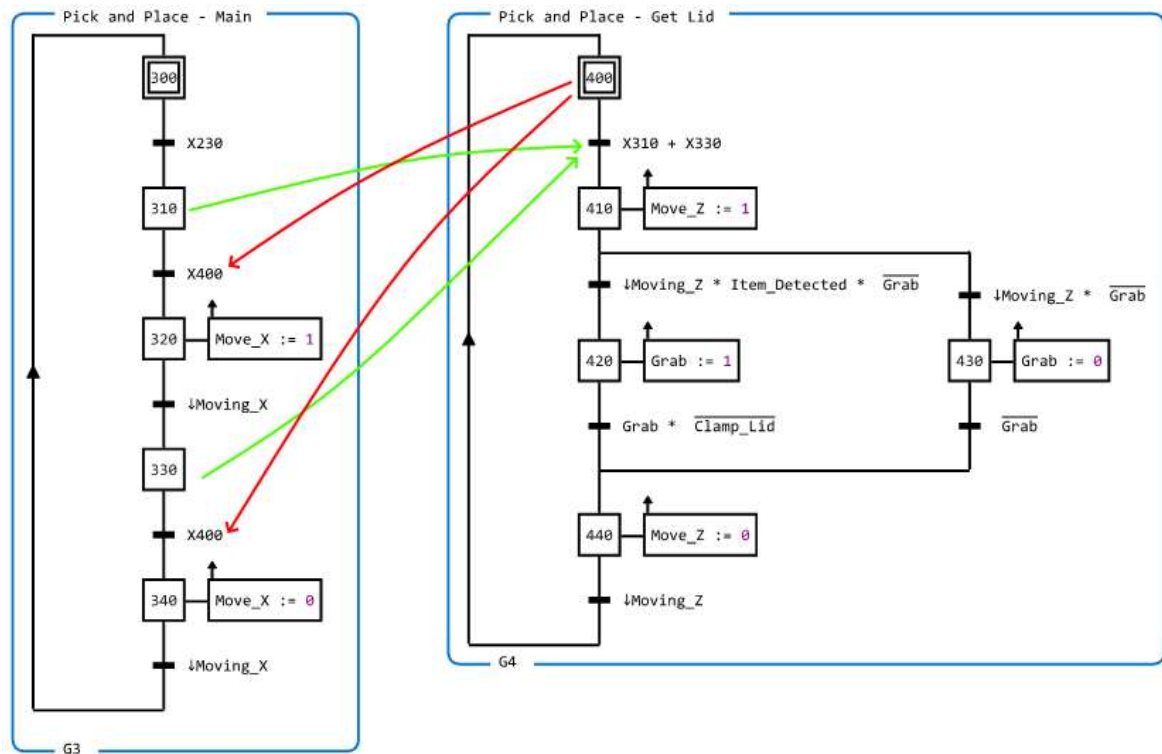


Figura 5.15 – Comunicação entre os dois ciclos de funcionamento do robô.

Como foi referido quando há uma tampa em posição o robô desce o seu braço e agarra a mesma. Quando a tampa é retirada pelo robô o subsistema de controlo das tampas pode, então, emitir e colocar em posição uma nova tampa. Para isso, este subsistema tem de receber a informação de que o robô já removeu a tampa anterior. Esta comunicação é feita recorrendo, mais uma vez, às etapas. Ou seja, quando o robô agarra a tampa, isto é, a etapa 420 de G4 é ativada, verifica-se a última transição de G2, fazendo com que o subsistema de controlo das tampas regresse à sua etapa inicial, repetindo o seu funcionamento. Esta comunicação está ilustrada na Figura 5.16.

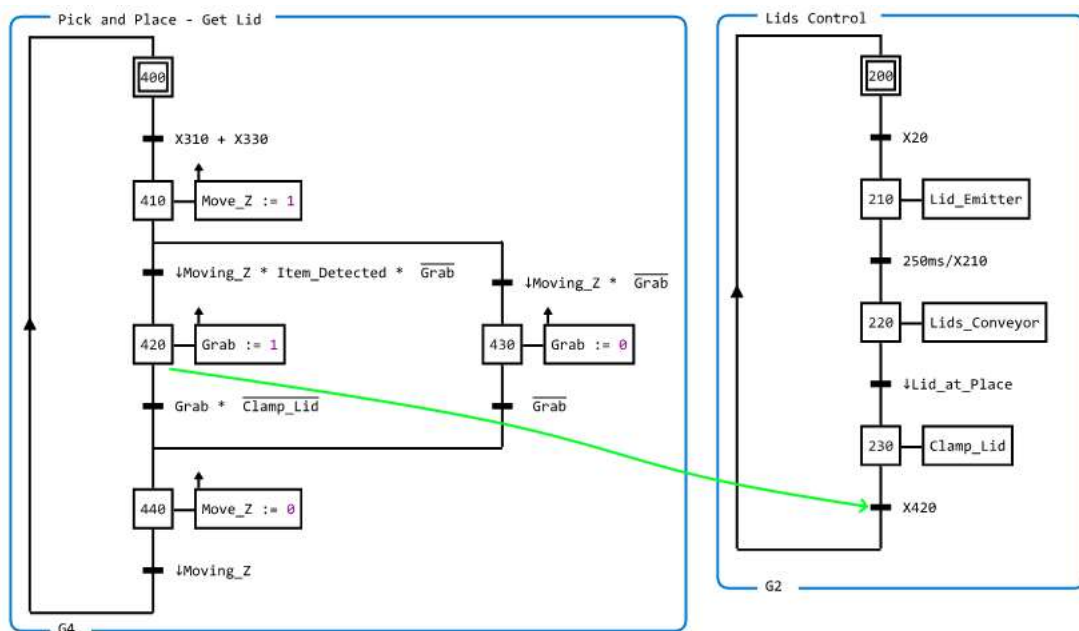


Figura 5.16 – Comunicação entre o ciclo de carga e descarga das tampas e o subsistema de controlo das tampas.

Finalmente, falta apenas referir a comunicação entre o subsistema de controlo das bases e o robô. O subsistema das bases tem de saber quando é que a tampa foi colocada na base disponibilizada, de modo que possa retirar a peça, que está agora completa, e disponibilizar uma nova base. Para isso, quando a etapa 430 de G4 é ativada, ou seja, o robô largou a tampa na base, verifica-se a quarta transição de G1, fazendo com que o tapete das bases ligue e a garra das bases seja levantada de modo que a peça seja removida do sistema. Esta comunicação está detalhada na Figura 5.17.

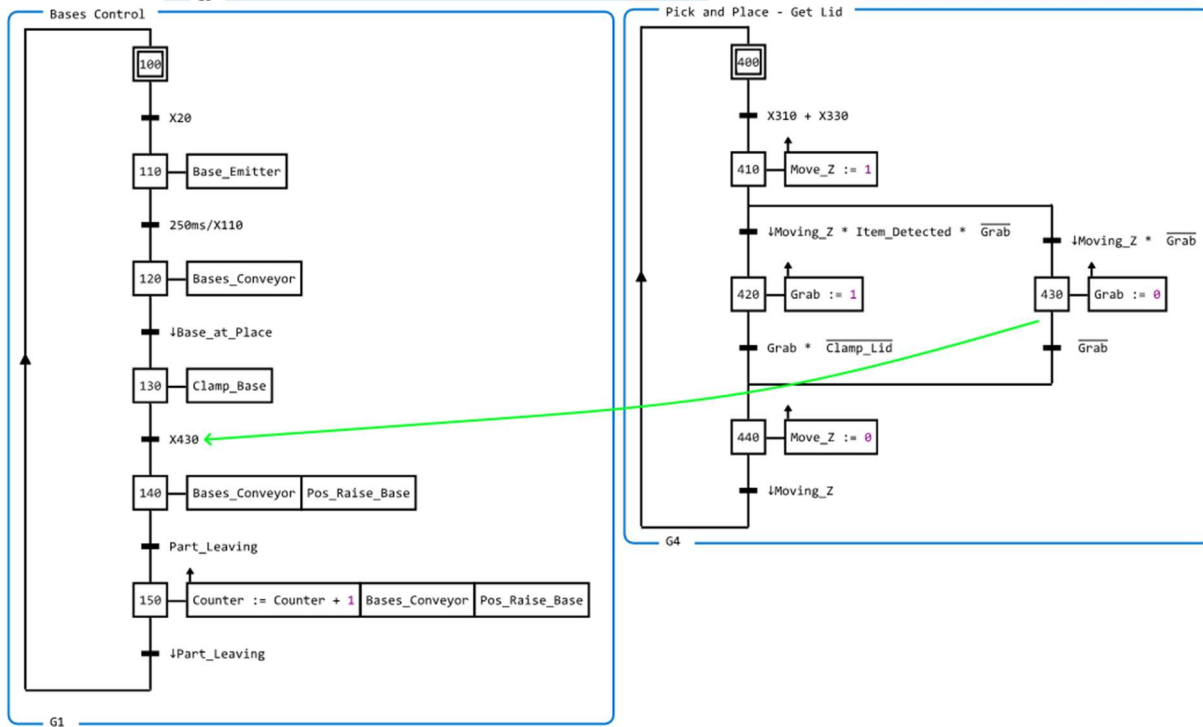


Figura 5.17 – Comunicação entre o ciclo de carga e descarga das tampas e o subsistema das bases.

```

IF Fact_Rstd THEN

    (...)

ELSE

    RS_Base_Conv (SET := R_Fact_Run.Q OR State_G4 = 430, RESET1 :=
F_Base_at_Place.Q, Q1 => Base_Conveyor);

    RS_Lid_Conv (SET := Lid_Emitter, RESET1 := F_Lid_at_Place.Q, Q1
=> Lid_Conveyor);

    RS_Pos_Raise_B (SET := State_G4 = 430, RESET1 :=
F_Part_Leaving.Q, Q1 => Pos_Raise_Base);

    RS_Clamp_Base (SET := F_Base_at_Place.Q, RESET1 := State_G4 =
430, Q1 => Clamp_Base);

```

```

        RS_Clamp_Lid (SET := F_Lid_at_Place.Q, RESET1 := State_G4 =
420, Q1 => Clamp_Lid);

        CASE State_G3 OF

            300:

                IF Lid_Clamped THEN

                    State_G3 := 310;

                END_IF

            310:

                IF State_G4 = 400 THEN

                    State_G3 := 320;

                END_IF

            (...)

        END_CASE

        CASE State_G4 OF

            400:

                IF State_G3 = 310 OR State_G3 = 330 THEN

                    State_G4 := 410;

                END_IF

            (...)

        END_CASE

    END_IF

```

Tal como nos casos anteriores, também neste se utilizou uma instrução `IF` para separar o reset e a execução do programa. Relativamente ao código que corresponde à execução do programa, verifica-se que, tal como no primeiro cenário, são apresentadas apenas duas instruções `CASE OF`. Isto acontece porque, mais uma vez, verifica-se que os subsistemas de controlo das tampas e das bases podem ser controlados por *latches*.

Começando a análise do excerto de programa apresentado pelas *latches*, verifica-se que, no código do tapete das bases surge com condição de set a etapa 430 de G4. Esta condição surge ainda, como era de esperar, como set da variável que faz levantar a garra e ainda como condição de reset da mesma. Tudo de acordo com a modelação apresentada. Quanto ao subsistema das tampas, verifica-se que na *latch* da garra das tampas surge, como condição de reset, a etapa 420 de G4, também conforme o que foi modelado.

Relativamente à comunicação entre o subsistema das tampas e o robô, dado que se programou o primeiro recorrendo a uma *latch* e não a uma instrução `CASE OF`, não é possível usar a etapa 230 de G2 como condição de transição de G3 no programa. Para contornar essa

questão utilizou-se a variável que indica se a garra está ou não atuada, visto que a referida etapa 230 é a única em que a garra está atuada.

Finalmente, falta apenas mencionar a comunicação entre os dois ciclos de funcionamento do robô. Tal como é apresentado nos graficets, quando a etapa 310 é ativada G4 verifica a sua primeira transição, por isso a mencionada etapa surge como condição para mudar de etapa na instrução *CASE OF* de G4. Quando este último regressa à sua etapa inicial, etapa 400, G3 verifica, por sua vez, a sua transição e avança no seu funcionamento.

5.2.2 Inclusão de uma interface homem-máquina

Dado que, devido à integração da interface, os requisitos de funcionamento do sistema são ligeiramente diferentes dos apresentados anteriormente, na Tabela 5.5 resumem-se os requisitos que diferem dos anteriormente apresentados.

Tabela 5.5 – Resumo dos requisitos do segundo cenário, com interface.

Funcionamento desejado:	Quando se atua o botão de start os emissores e os tapetes ligam. Quando uma tampa encosta à garra das tampas, ou seja, o sensor das tampas volta a <i>FALSE</i> , a garra aperta a peça de modo que esta última esteja na posição correta para que o robô a agarre. Estando a tampa em posição, o braço do robô desce e agarra-a. Depois sobe, avança e, havendo uma base em posição, desce e larga a tampa, montando-a na base. Relativamente ao tapete das bases, este funciona de maneira similar à do tapete já descrito, com a diferença que, quando a tampa é montada na base, a garra deste tapete deve largar a peça e subir e o tapete deve ligar de modo a transportar a peça para a rampa de saída. Quando se atua o botão de stop o sistema deve terminar o ciclo que estiver a fazer e depois parar. O comando de reset do <i>Factory I/O</i> reinicia o cenário.
Atuações manuais:	Dar início ao funcionamento do sistema atuando o botão de start; parar o sistema atuando o botão de stop.
Sinais I/O:	Entradas: Botões de start e stop e comando de reset do <i>Factory I/O</i> (todos momentâneos, produzem <i>TRUE</i> quando premidos), sensores das tampas, das bases e de saída (geram <i>TRUE</i> na situação de deteção) e sensores de posicionamento do robô (variáveis do tipo <i>BOOL</i>).

À semelhança do que aconteceu com os cenários até aqui apresentados, dado que é conhecido o sistema segue-se a modelação.

Como foi referido, o cenário agora apresentado é exatamente o mesmo que estudado anteriormente sendo a única diferença a incorporação de uma interface homem-máquina, que neste caso se trata de um quadro elétrico.

Assim sendo, o graficet que modela este sistema será muito semelhante ao anterior, sendo a grande diferença no G0 que agora contempla mais funções, como será explicado em breve. Outra diferença que surge é nos graficets G1 e G2 que agora devem incluir a possibilidade de funcionar em loop, como o caso anterior, e ainda a possibilidade de ter de terminar o ciclo em curso e regressar à sua etapa inicial onde aguardam nova atuação do botão de start. Quanto aos graficets que modelam o funcionamento do robô, estes não sofrem alterações visto que só

são “chamados” quando necessário pelo subsistema de controlo das tampas. Este último é que deve estabelecer contacto com o G0, como já foi referido.

Na Figura 5.18 pode-se ver o grafcet deste cenário e a sua legenda encontra-se na Tabela 5.6.

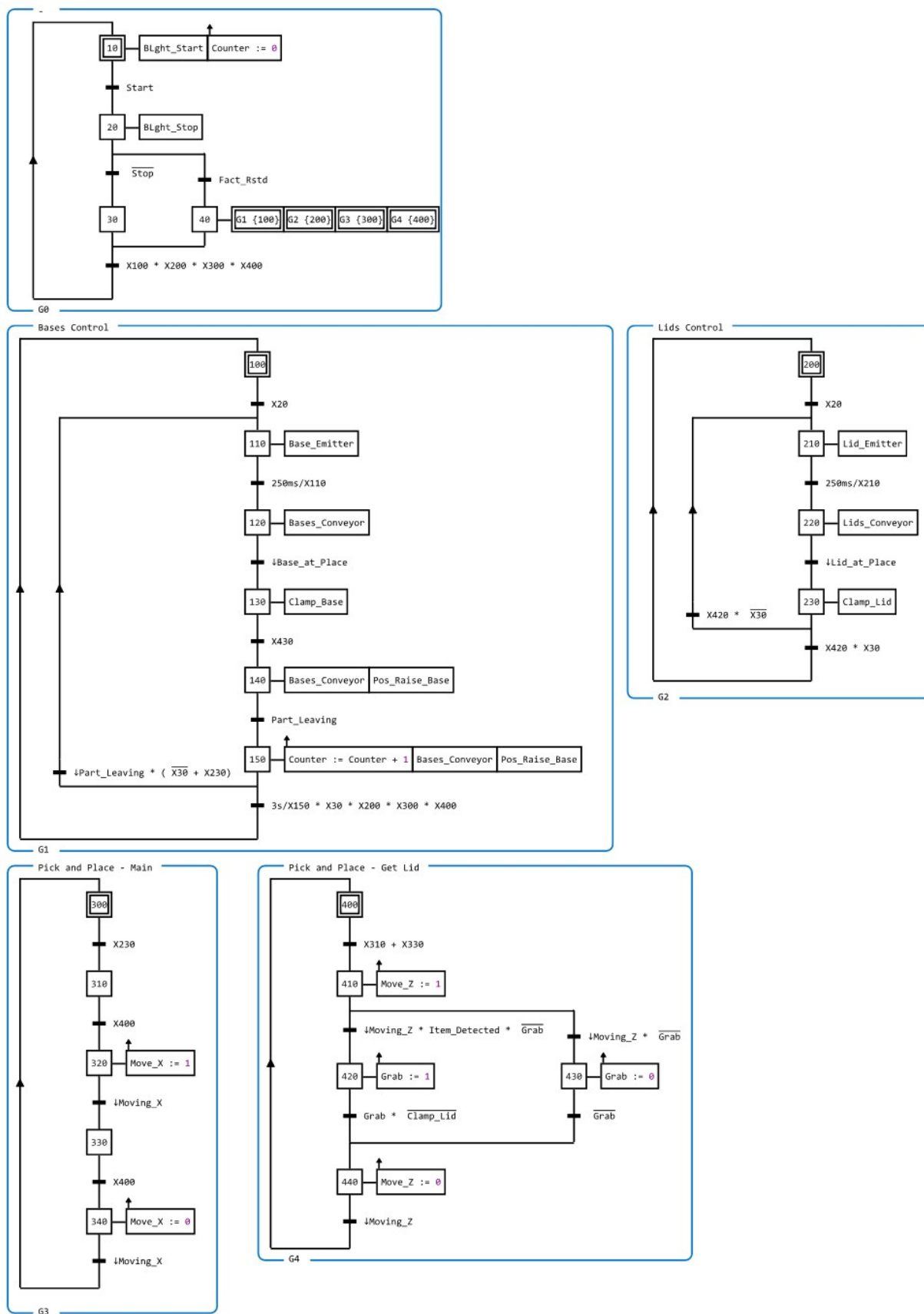


Figura 5.18 – Grafcet do segundo cenário, com interface homem-máquina.

Tabela 5.6 – Legenda do grafcet do segundo cenário, com interface.

Legenda:	Base_at_Place: Sensor das bases; Bases_Conveyor: Tapete das bases; Base_Emitter: Emissor de bases; BLght_Start: Luz o botão de start; BLght_Stop: Luz do botão de stop; Clamp_Base: Garra das bases; Clamp_Lid: Garra das tampas; Counter: Contador; Fact_Rstd: Factory I/O (Reset); Grab: Ventosa do robô; Item_Detected: Sensor na ponta do braço do robô; Lid_at_Place: Sensor das tampas; Lids_Conveyor: Tapete das tampas; Lid_Emitter: Emissor das tampas; Move_X: Variável de movimento do robô em X; Move_Z: Variável de movimento do robô em Z; Moving_X: Sensor de movimento do robô em X; Moving_Z: Sensor de movimento do robô em Z; Pos_Raise_Base: Atuador para levantar a garra; Start: Botão de start; Stop: Botão de stop.
-----------------	--

Como foi mencionado, a grande diferença entre a modelação deste cenário e a do anterior, encontra-se no grafcet G0. No caso anterior este tinha como função iniciar ou terminar o funcionamento do sistema. No presente cenário essa função não mudou, mas opera de uma maneira algo diferente. Como foi referido, agora, para iniciar o funcionamento do sistema deve-se atuar o botão de start e, para parar o mesmo, deve-se atuar o botão de stop. Isto faz com que se deixe de precisar da variável *Fact_Run*. No entanto, a função de reset do programa mantém-se para que, caso algo corra mal do decorrer do funcionamento, se possa facilmente reiniciar o programa. Posto isto, o quadro elétrico tem de ser modelado como mostra a Figura 5.19.

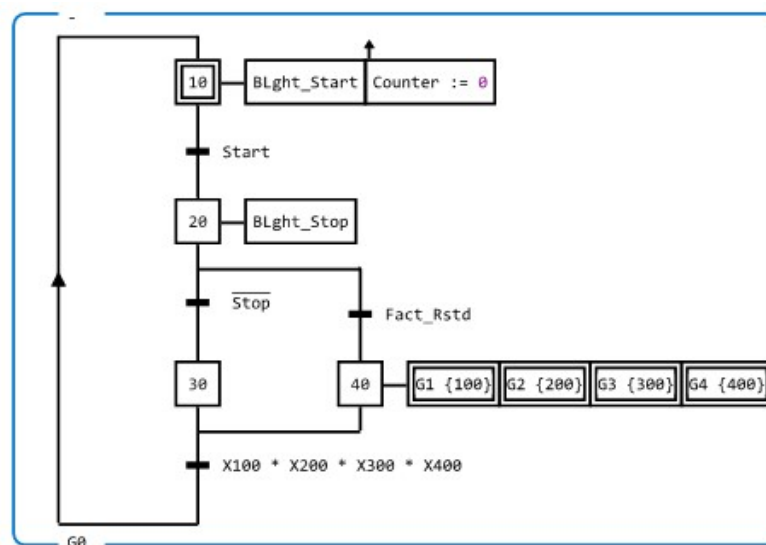


Figura 5.19 – Grafcet da interface do segundo cenário.

Repare-se que, quando se atua o botão de stop, G0 ativa a etapa 30 e só a desativa quando os restantes graficets que constituem o sistema regressarem às suas etapas iniciais. Isto permite que os outros subsistemas saibam que o botão de stop foi atuado e devem, por isso, terminar o ciclo em curso e depois parar.

Assim, quando os subsistemas de controlo das tampas e das bases chegam ao fim de um ciclo verificam se o botão de stop foi ou não premido e atuam de acordo, como se pode ver na Figura 5.20.

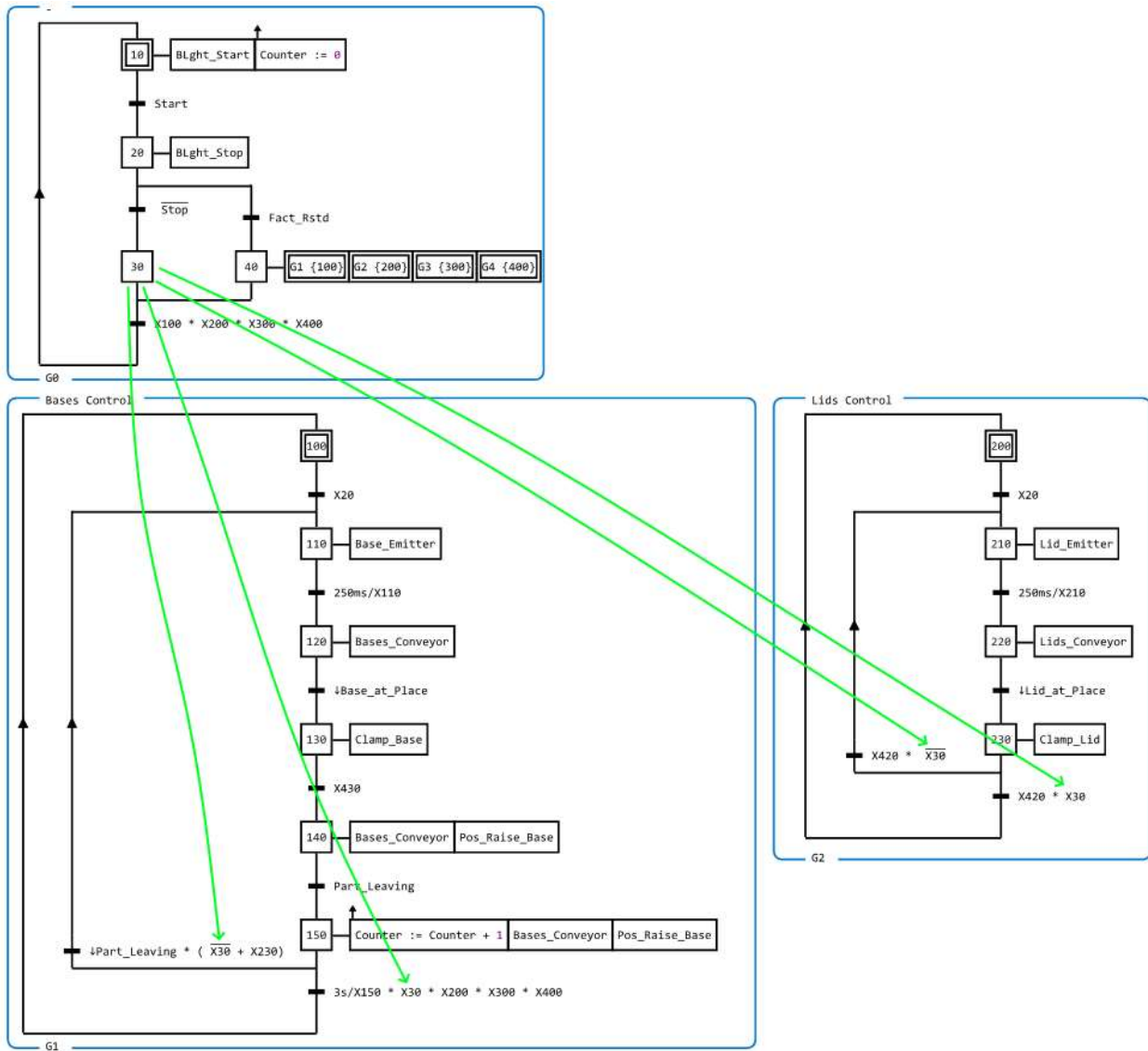


Figura 5.20 – Comunicação entre a interface e os subsistemas de controlo das tampas e das bases.

```

IF Fact_Rstd THEN
    (...)
ELSE
    CASE State_G0 OF
        (...)
        30:

```

```

        IF State_G1 = 100 AND State_G2 = 200 AND State_G3 =
300 AND State_G4 = 400 THEN

            State_G0 := 10;

        END_IF

    END_CASE

CASE State_G1 OF

    100:

        IF State_G0 = 20 THEN

            State_G1 := 110;

        END_IF

        (...)

    130:

        IF State_G4 = 430 THEN

            State_G1 := 140;

        END_IF

        (...)

    150:

        IF F_Part_Leaving.Q AND (State_G0 <> 30 OR
Lid_Clamped) THEN

            State_G1 := 110;

            ELSIF Timer_Base_Conv.Q AND State_G0 = 30 AND
State_G2 = 200 AND State_G3 = 300 AND State_G4 = 400 THEN

                State_G1 := 100;

            END_IF

        END_CASE

CASE State_G2 OF

    200:

        IF State_G0 = 20 THEN

            State_G2 := 210;

        END_IF

        (...)

    230:

```

```

        IF State_G4 = 420 AND State_G0 <> 30 THEN

            State_G2 := 210;

        ELSIF State_G4 = 420 AND State_G0 = 30 THEN

            State_G2 := 200;

        END_IF

    END_CASE

    (...)

END_IF

Start_Light := State_G0 = 10;

Stop_Light := State_G0 = 20;

Base_Conveyor := State_G1 = 120 OR State_G1 = 140 OR State_G1 = 150;

Clamp_Base := State_G1 = 130;

Pos_Raise_Base := State_G1 = 140 OR State_G1 = 150;

Lid_Conveyor := State_G2 = 220;

Clamp_Lid := State_G2 = 230;

Counter_Display := Counter.CV;

```

Mais uma vez, tal como se viu nos casos anteriores, foi incluída uma instrução `IF` para fazer a separação entre o reset e a execução do programa.

Quanto ao programa propriamente dito, embora este não seja apresentado na sua íntegra, este é essencialmente o mesmo que foi apresentado para a primeira versão deste cenário, visto que a única diferença é a incorporação da interface homem-máquina.

Assim, verifica-se que foram acrescentadas três instruções `CASE OF`, uma para o quadro elétrico, uma para subsistema de controlo das tampas e uma para o subsistema de controlo das bases. Isto porque estes dois últimos subsistemas já não se podem resumir a *latches* visto que devem, como referido, contemplar a possibilidade de terminar o ciclo em curso ou continuar o funcionamento em loop.

Como se pode observar na instrução `CASE OF` do G0, no caso deste estar na etapa 30, só transita da mesma quando todos os outros subsistemas regressarem às suas etapas iniciais, como modelado no grafcet. Verifica-se ainda que no código do subsistema de controlo das bases, G1, se encontra como condição de parar ou continuar, a etapa 30 de G0. O mesmo se verifica para o subsistema de controlo das tampas, G2, nunca esquecendo as outras condições que já haviam sido definidas na situação sem quadro elétrico. Sempre de acordo com o que foi modelado.

Não se apresentam os códigos relativos ao robô, visto que estes se mantêm iguais relativamente ao apresentado anteriormente.

No final do código escrevem-se as variáveis de saída, como se pode observar.

5.3 Teste no novo PLC

Numa nota rápida, lembre-se que, à semelhança dos capítulos anteriores e pelas mesmas razões apresentadas, os códigos que são presentes neste capítulo também foram desenvolvidos no *Codesys*.

Tal como sucedeu anteriormente, e como já foi referido no presente capítulo, os programas apresentados foram testados no *soft*PLC *ST I/O*.

Todos os programas apresentados ao longo do presente capítulo foram executados sem qualquer problema no novo PLC, sendo que as únicas alterações feitas foram, como já referido anteriormente, a mudança de `RESET` para `R` nos contadores.

5.4 Síntese

Ao longo do presente capítulo procurou-se expor, analisar, modelar e programar dois dos muitos cenários que o *Factory I/O* disponibiliza na sua biblioteca.

Começou-se por apresentar os cenários de *Factory I/O*, indicando os elementos atuadores e sensores que os compõem e seguiu-se a apresentação da sua modelação, a partir da qual se desenvolveram os programas apresentados. Procurou-se ainda salientar a divisão do sistema principal em subsistemas de modo a obter uma modelação, e respetiva programação, mais simples. Os códigos foram apresentados de uma forma sucinta procurando salientar como se programa a referida comunicação entre os diferentes subsistemas que foram definidos, visto que a restante programação já foi abordada com algum detalhe nos capítulos anteriores. Por fim, testaram-se os programas desenvolvidos, no *soft*PLC *ST I/O*.

6 Conclusões e trabalhos futuros

A presente dissertação iniciou-se pela apresentação do seu objetivo: a produção de material de teste para o *soft*PLC *ST I/O* em desenvolvimento pela empresa Real Games, indicando ainda que um outro propósito para esse material seria a sua aplicação num contexto didático, nomeadamente, enquanto documentação de acompanhamento do referido PLC.

Assim, contextualizou-se a importância da automação no mundo atual e, por consequência, a relevância dos PLCs enquanto peça central da automação de processos, referindo a necessidade de profissionais qualificados com as competências necessárias para programarem autómatos e ainda os problemas logísticos associados à preparação desses mesmo profissionais, nomeadamente, questões de segurança perante a ocorrência de potenciais erros de programação. Isto é, os estudantes podem cometer erros de programação que, no momento da implementação e teste do código, podem desencadear comportamentos incompatíveis com o funcionamento do equipamento a controlar, levando a acidentes. No entanto, a componente prática do ensino, principalmente quando o tópico em questão tem aplicações maioritariamente práticas na vida real, não deve ser desvalorizada ou ignorada.

Prosseguiu-se, assim, com uma análise ao estado atual do ensino de PLC, onde se analisaram as matérias lecionadas em diferentes cursos universitários, técnico e profissionais e ainda a abordagem adotada pelos autores de livros técnicos sobre PLCs. Concluiu-se então que, frequentemente, a vertente prática do ensino de programação de PLCs apresenta lacunas. Estas prendem-se maioritariamente, como já referido, com questões económicas e de segurança. No seguimento desta questão foram apresentados os *software* de simulação que permitem a criação e controlo de sistemas industriais, sendo dado ênfase à aplicação da Real Games, o *Factory I/O*, onde será implementado o *soft*PLC atualmente em desenvolvimento. Surgiu, no entanto, a questão de como esse controlo é feito. O *Factory I/O* permite a ligação com vários PLCs comerciais e inclui um controlador próprio, o *Control I/O*, no entanto, este apresenta várias limitações. Assim, apresentou-se a motivação que levou à decisão, por parte da Real Games, de desenvolver um novo *soft*PLC para integrar no seu *software* de simulação.

Concluída a contextualização do objetivo da dissertação, seguiu-se a apresentação dos aspetos fundamentais da programação em Texto Estruturado. Nesta secção introduziu-se, por meio de exercícios, os blocos funcionais mais simples, e as suas diferentes instâncias, que constituem as bases da programação em ST. Através da resolução dos exercícios procurou-se apresentar uma explicação clara das características e do funcionamento de cada bloco estudado.

Ainda no âmbito das questões fundamentais da programação em Texto Estruturado, abordou-se o desenvolvimento de programas a partir de máquinas de estado, apresentando estas sob a forma de diagramas de estados e grafets. Os casos expostos foram selecionados com o intuito de expor questões particulares de programação, como é o caso das transições todas idênticas no diagrama de estados e da divergência no grafet.

A aplicabilidade destes conceitos é demonstrada ao longo dos capítulos 4 e 5. No primeiro são expostos cenários relativamente simples que integram a biblioteca do *Factory I/O*, são apresentados os requisitos funcionais dos sistemas a partir dos quais se desenvolve a modelação dos mesmos e, finalmente, é apresentado o programa que impõe o comportamento desejado.

O capítulo 5, por sua vez, expõe cenários, também da biblioteca do *Factory I/O*, mas mais complexos do que os anteriores. O objetivo, à semelhança do capítulo anterior, é demonstrar a aplicabilidade dos conceitos fundamentais da programação em Texto Estruturado, mas desta vez num contexto mais complexo, demonstrando assim, as potencialidades desta linguagem de programação de PLCs.

Conforme se progrediu na dissertação também se procedeu ao teste do *soft*PLC *ST I/O* fazendo uso dos programas apresentados tendo-se verificado que este os executou de forma correta e de acordo com a norma IEC 61131-3, como expectável. Desta forma, é possível concluir que, embora o PLC ainda se encontre numa fase embrionária de desenvolvimento, já apresenta grande potencial tendo preenchido os requisitos estipulados para esta sua etapa de desenvolvimento, permitindo concluir que a presente dissertação alcançou os objetivos delineados no início.

No que se refere à experiência pessoal da autora, pode-se afirmar que o desenvolvimento da presente dissertação se provou um desafio de valor. Graças a este projeto foi possível a aprendizagem mais detalhada de um tópico que esta considera de elevado interesse: a programação de sistemas de automação em Texto Estruturado. Este trabalho permitiu um aprofundamento de conhecimentos previamente adquiridos em Unidades Curricular da área e ainda o desenvolvimento das suas capacidades de programação tendo aprendido a fazer uso de um vasto leque de ferramentas.

6.1 Trabalhos futuros

Tendo em consideração o objetivo didático da presente dissertação, sugere-se, como seguimento do projeto, o desenvolvimento de mais material com o mesmo objetivo, no qual se aprofunde a programação em ST e se aumente o nível de dificuldade. Uma vez que a aplicação *Factory I/O* apresenta uma vasta biblioteca de cenários cuja grande maioria ainda não tem soluções de programação, propõe-se, por exemplo a sua resolução e disponibilização na documentação que acompanha o *soft*PLC.

Visto que o *ST I/O* se encontra, aquando da escrita do presente documento, em desenvolvimento propõe-se ainda a criação de mais testes que sejam mais complexos. Sugere-se, por exemplo, a criação de um cenário no *Factory I/O* complexo que se aproxime tanto quanto possível de uma linha de produção e cuja programação seja desenvolvida com recurso a funções e procure aplicar a programação orientada a objetos, de modo a testar o PLC em questões de programação mais avançadas.

Referências

- [1] K. L. S. Sharma, "1.1 Introduction," in *Overview of Industrial Process Automation (2nd Edition)*: Elsevier, 2017.
- [2] P. Ackerman, *Industrial Cybersecurity Efficiently secure critical infrastructure systems*. Packt Publishing, 2017.
- [3] IEC 61131-3:2013 standard. *Programmable controllers - Part 3: Programming languages*. IEC, 2013.
- [4] Real Games. [Online]. Available: <https://realgames.co/company/> [Accessed 25-Jun-2021].
- [5] IEC 60848:2013 standard. *GRAFCET specification language for sequential function charts*. IEC, 2013.
- [6] Faculdade de Engenharia da Universidade do Porto, *Automação e Segurança Industrial (Mestrado Integrado em Engenharia Mecânica)*. [Online]. Available: https://sigarra.up.pt/feup/pt/ucurr_geral.ficha_uc_view?pv_ocorrencia_id=399467 [Accessed: 10-Feb-2021].
- [7] Faculdade de Engenharia da Universidade do Porto, *Computação Industrial (Mestrado Integrado em Engenharia Mecânica)*. [Online]. Available: https://sigarra.up.pt/feup/pt/ucurr_geral.ficha_uc_view?pv_ocorrencia_id=435390 [Accessed: 10-Feb-2021].
- [8] Faculdade de Engenharia da Universidade do Porto, *Sistemas e Automação (Mestrado Integrado em Engenharia Electrotécnica e de Computadores)*. [Online]. Available: https://sigarra.up.pt/feup/pt/ucurr_geral.ficha_uc_view?pv_ocorrencia_id=461249 [Accessed: 10-Feb-2021].
- [9] Faculdade de Engenharia da Universidade do Porto, *Automação (Mestrado Integrado em Engenharia Electrotécnica e de Computadores)*. [Online]. Available: https://sigarra.up.pt/feup/pt/ucurr_geral.ficha_uc_view?pv_ocorrencia_id=461235 [Accessed: 10-Feb-2021].
- [10] Universidade do Minho, *Mestrado Integrado em Engenharia Electrónica Industrial e Computadores*. [Online]. Available: https://www.eng.uminho.pt/pt/Estudar/_layouts/15/UMinho.PortaisUOEI.UI/Pages/CatalogoCursoDetail.aspx?itemId=3683&catId=11 [Accessed: 24-Jun-2021].
- [11] Universidade do Minho, *Mestrado em Engenharia Mecatrónica*. [Online]. Available: https://www.eng.uminho.pt/pt/Estudar/_layouts/15/UMinho.PortaisUOEI.UI/Pages/CatalogoCursoDetail.aspx?itemId=3924&catId=12 [Accessed: 24-Jun -2021].
- [12] Universidade de Aveiro, *Automação Industrial*. [Online]. Available: <https://www.ua.pt/pt/uc/13465> [Accessed: 10-Feb-2021].
- [13] Universidade de Aveiro, *Automação (Licenciatura em Engenharia Electrotécnica)*. [Online]. Available: <https://www.ua.pt/pt/uc/11786> [Accessed: 10-Feb-2021].
- [14] Universidade de Aveiro, *Automação e Controlo (Licenciatura em Engenharia e Gestão Industrial)*. [Online]. Available: <https://www.ua.pt/pt/uc/2232> [Accessed: 10-Feb-2021].
- [15] Universidade de Aveiro, *Automação I (Licenciatura em Engenharia Mecânica)*. [Online]. Available: <https://www.ua.pt/pt/uc/2233> [Accessed: 10-Feb-2021].
- [16] Universidade de Coimbra, *Automação Industrial (Licenciatura em Engenharia Mecânica)*. [Online]. Available: https://apps.uc.pt/courses/PT/unit/8444/20901/2021-2022?common_core=true&type=ram&id=8801 [Accessed: 10-Feb-2021].
- [17] Universidade de Coimbra, *Automação e Controlo Industrial (Licenciatura em Engenharia Mecânica)*. [Online]. Available: https://apps.uc.pt/courses/PT/unit/8459/17501/2020-2021?common_core=true&type=ram&id=324 [Accessed: 10-Feb-2021].
- [18] Universidade de Coimbra, *Automação Industrial (Mestrado Integrado em Engenharia Electrotécnica e de Computadores)*. [Online]. Available:

- https://apps.uc.pt/courses/PT/unit/8593/19325/2020-2021?common_core=true&type=ram&id=359 [Accessed: 10-Feb-2021].
- [19] Universidade Beira Interior, *Automação Industrial (Mestrado em Engenharia e Gestão Industrial)*. [Online]. Available: <https://www.ubi.pt/Disciplina/11700> [Accessed: 10-Feb-2021].
- [20] Universidade Nova de Lisboa, *Automação (Mestrado Integrado em Engenharia Mecânica)*. [Online]. Available: <https://guia.unl.pt/pt/2020/fct/program/982/course/10493#subject> [Accessed: 10-Feb-2021].
- [21] Siemens, *SIMATIC Programming 1 in the TIA Portal (TIA-PRO1)*. [Online]. Available: <https://www.sitrain-learning.siemens.com/DE/en/rw89904/SIMATIC-Programming-1-in-the-TIA-Portal> [Accessed: 11-Feb-2021].
- [22] Siemens, *SIMATIC S7-1200 Advanced Course (TIA-MICRO2)*. [Online]. Available: <https://www.sitrain-learning.siemens.com/DE/en/rw40867/SIMATIC-S7-1200-Advanced-Course> [Accessed: 11-Feb-2021].
- [23] Siemens, *SIMATIC S7-1200 Basic Course (TIA-MICRO1)*. [Online]. Available: <https://www.sitrain-learning.siemens.com/DE/en/rw44789/SIMATIC-S7-1200-Basic-Course> [Accessed: 11-Feb-2021].
- [24] OMRON, *Serviços de formação*. [Online]. Available: <https://industrial.omron.pt/pt/services-support/training> [Accessed: 11-Feb-2021].
- [25] Escola Técnica Vale de Cambra, *Curso de Programação de Autómatos*. [Online]. Available: https://www.dual.pt/course/show/7759.html?course_type=formacao-modular [Accessed: 11-Feb-2021].
- [26] UDEMY, *Automação Industrial em Ambiente Virtual*. [Online]. Available: <https://www.udemy.com/course/automacao-industrial-em-ambiente-virtual/> [Accessed: 11-Feb-2021].
- [27] Dual: Qualificação Profissional, *Autómatos programáveis - Linguagens de Programação*. [Online]. Available: https://www.dual.pt/course/show/7187.html?course_type=formacao-modular [Accessed: 11-Feb-2021].
- [28] Dual: Qualificação Profissional, *Autómatos Programáveis - Aplicações Industriais*. [Online]. Available: https://www.dual.pt/course/show/7759.html?course_type=formacao-modular [Accessed: 11-Feb-2021].
- [29] Cenertec, *Programação de Autómatos*. [Online]. Available: https://cenertec.pt/?p=cursos&slug=programacao-de-automatos&utm_content=buffer00ded&utm_medium=social&utm_source=facebook.com&utm_campaign=buffer [Accessed: 11-Feb-2021].
- [30] Master D, *Curso de Programação de Automação Industrial*. [Online]. Available: <https://www.educaedu.com.pt/curso-de-programacao-de-automacao-industrial-cursos-33134.html> [Accessed: 11-Feb-2021].
- [31] A. Francisco, *Autómatos Programáveis*. Edições Técnicas e Profissionais, 2015.
- [32] J. R. C. Pinto, *Técnicas de Automação*. Edições Técnicas e Profissionais, 2010.
- [33] P. Oliveira, *Curso de Automação Industrial*. Edições Técnicas e Profissionais, 2009.
- [34] A. P. d. Magalhães, *Práticas de Automação Industrial*. Real Games, 2010.
- [35] W. Bolton, "Programmable Logic Controllers (5th Edition)," ed: Elsevier, 2015.
- [36] A. P. d. Magalhães, "Exercícios de Programação de PLCs," ed, 2020.
- [37] Codesys, *CODESYS Online Help, "TON"*. [Online]. Available: <https://help.codesys.com/webapp/ton;product=codesys;version=3.5.11.0> [Accessed: 10-Jun-2021].

- [38] Codesys, *CODESYS Online Help, "TOF"*. [Online]. Available: <https://help.codesys.com/webapp/tof;product=codesys;version=3.5.11.0> [Accessed: 10-Jun-2021].
- [39] Codesys, *CODESYS Online Help, "TP"*. [Online]. Available: <https://help.codesys.com/webapp/tp;product=codesys;version=3.5.11.0> [Accessed: 10-Jun-2021].
- [40] W. Bolton, "Chapter 13 - Designing Systems," in *Programmable Logic Controllers (Sixth Edition)*, W. Bolton, Ed. Boston: Newnes, 2015, pp. 295-339.
- [41] MHJ-Software, *GRAFCET-Studio*. [Online]. Available: <https://www.mhj-tools.com/?page=grafcet-studio> [Accessed: 27-Jun-2021].

Todas as referências em página *web* continuavam ativas a 27 de junho de 2021, data de entrega da versão provisória da dissertação.